Lecture 1

What is an algorithm?

a well-defined computational procedure that takes an input and returns an output.

(Computational) Problem

⟨ Input - Output ⟩

spec. spec.

e.g. Sorting Problem

Input: A list $A = [a_1, \ldots, a_n]$

Output: A permutation $A' = [a_1', \ldots, a_n']$

s.t. $a_1' \leq a_2' \leq \cdots \leq a_n'$

Problem vs. Its Instances

generic specific

e.g. $A = [a_1 \ldots a_n]$ $A = [1, 4, 2, 5]$

e.g. Sorting Problem

Input: A list $A = [a_1, \ldots, a_n]$

Output: A permutation $A' = [a_1', \ldots, a_n']$

s.t. $a_1' \leq a_2' \leq \cdots \leq a_n'$

Question: How many instances?

How many instances with $n = 3$?

Note: $A = [a_1, a_2, a_3]$

How many instances if

$A = [a_1, \ldots, a_5]$ and $a_i \in \{1, \ldots, 100\}$?

Lecture 2

(Read Ch 1 & 2)

Problem vs Its Instances.

Input: A list $A = [a_1, \ldots, a_n]$ $a_i \in \mathbb{Z}$.

Output: A rearrangement $[a_1', \ldots, a_n']$ of A

s.t. $a_1' \leq a_2' \leq \ldots \leq a_n'$

$A = [1, 3, 5, 7, 2]$

$\Big\{$ Alg A

$A' = [1, 2, 3, 5, 7]$

RAM Computational Model

- Primitive:
  - Arithmetic ops (+, *, /, -, $\lfloor \rfloor$, $\lceil \rceil$, ...)
  - Assignment = e.g. $x = y + z$
  - Function calls e.g. $x = f(a, b)$
  - Memory access (load, copy, store, ...)
- Nonprimitive:
  - Loops (for, while)
  - Functions / subroutines
  - return
- Deterministic (no randomness, no coin tossing)
- Sequential (no parallelism, no concurrency)
- Offline (i.e. whole input is given at the beginning)

Remark:

- An algorithm should terminate.
- An algorithm should return a correct answer for each input. (Correctness)

Proof of Correctness of Algorithms

↓

proof of correctness of loop invariant(s) of algorithms.

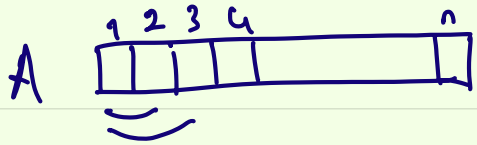
Example:

INSERTION-SORT(A, n)

for $j = 2$ **to** n
  $key = A[j]$
  // Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.
  $i = j - 1$
  while $i > 0$ and $A[i] > key$
    $A[i+1] = A[i]$
    $i = i - 1$
  $A[i+1] = key$

Exercise:

Run on
$A = [2, 1, 4, 3]$



"an incremental algorithm"

Loop invariant: "At the beginning of each iteration j, the subarray $A[1, \ldots, j-1]$ is already sorted"

How to prove loop invariants (L.I.)?

Step 1: Prove that the L.I. is true/correct at the beginning of the loop.

Step 2: Prove that the L.I. remains true after each iteration of the loop.

Step 3 Prove that the L.I. is true after the loop ends.

Proofs

Proof by induction:

Let C be statement indexed by natural numbers.

e.g. $S(n) = 1+2+\dots+n = \frac{n(n+1)}{2}$ for every n.

To prove C:

- Prove that C is true for basis case(s).
- Write an inductive hypothesis (I.H.) and prove C in general using the I.H.

Remark:

- Strong induction:
  - Assume C for every $k < n$.
  - Prove C for n.
- Weak induction
  - Assume C for n
  - Prove C for $n+1$.

Example: Prove $S(n) = 1 + 2 + \dots + n = \frac{n(n+1)}{2}$ by ind.

Proof: by induction

Basis: $n = 1 \Rightarrow S(1) = 1 = \frac{1(1+1)}{2} = 1$ ✓

I.H. Assume $S(k) = \frac{k(k+1)}{2}$ for $k < n$.

Prove: $S(n) = \frac{n(n+1)}{2}$

$$S(n) = \underbrace{1 + 2 + \dots + (n-1)}_{S(n-1)} + n$$

$$= S(n-1) + n \quad \text{by def.}$$

$$= \frac{(n-1)(n)}{2} + n \quad \text{by I.H.}$$

$$= \frac{n^2 - n}{2} + \frac{2n}{2} = \frac{n^2 - n + 2n}{2} = \frac{n^2 + n}{2} = \frac{n(n+1)}{2}$$ ✓

Exercise: Prove $S(n) = \frac{n(n+1)}{2}$ using weak induction.

Example: Prove that $11^n - 6$ is divisible by 5 for $n \geq 1$.

Proof: by induction

Basis: $n = 1 \Rightarrow 11^1 - 6 = 5$ ✓

I.H.: $11^k - 6 = 5t$ for some $t \in \mathbb{Z}$ for every $k < n$. for an arbitrary n.

Prove: $11^n - 6$ is divisible by 5.

$11^n - 6 = 11 \cdot 11^{n-1} - 6$

[**Note:** $11^{n-1} - 6$ is divisible by 5 thanks to I.H.

i.e. $11^{n-1} - 6 = 5m$ for some m.]

$\Rightarrow 11^{n-1} = 5m + 6$

$$\begin{aligned} 11^n - 6 &= 11 \cdot 11^{n-1} - 6 \\ &= 11 \cdot (5m+6) - 6 \\ &= 55m + 66 - 6 \\ &= 55m + 60 \\ &= 5(11m + 12) \end{aligned}$$ //

Exercise: Prove that $F_n < \left(\frac{5}{3}\right)^n$ for all n. (F_n : n-th fibonacci number)

Lecture 3

Proof by contradiction:

Example: Prove that for all $a, b \in \mathbb{Z}$

$$a^2 - 4b \neq 2.$$

Proof: by contradiction

- Assume: For some $a, b \in \mathbb{Z}$, $a^2 - 4b = 2$.

$\Rightarrow a^2 = 4b + 2$

$\Rightarrow a^2 = 2(2b+1)$

$\Rightarrow a^2$ is even (by def.)

$\Rightarrow a$ is even

$\Rightarrow a = 2k$ for some $k \in \mathbb{Z}$. (by def.)

$\Rightarrow a^2 = (2k)^2 = 4b+2$

$\Rightarrow 4k^2 = 4b+2$

$\Rightarrow 2k^2 = 2b+1$

Contradiction because LHS is even and RHS is odd. ∎

Example: If $\underbrace{a^2 \text{ is even}}_{p}$, then $\underbrace{a \text{ is even}}_{q}$.

Proof: by contradiction

- Assume: a^2 is even
- Assume: a is odd

$\Rightarrow a = 2k+1$ for some $k \in \mathbb{Z}$.

$\Rightarrow a^2 = (2k+1)^2 = 4k^2 + 4k + 1$

$= 2(2k^2+2k) + 1$ ... odd

But we assumed that a^2 is even!

Contradiction.

Exercise: Prove each by contradiction:

- If a^2 is odd, a is odd.
- There are infinitely many prime numbers.
- $\sqrt{2}$ is irrational.

Proof by counterexample:

To disprove a claim starting with "for all..." it is enough to give (at least) one counterexample

Example: Disprove $x^n + x^n = x^{2n}$ for all $x, n \in \mathbb{Z}$.

Take $x=1$, $n=1$. Then we get $1^1 + 1^1 \neq 1^{2\cdot 1}$

$\Rightarrow 1 \neq 2$.

Direct Proof:

To prove p, show that p is true.

To prove $p \to q$, assume p is true and show that q is true.

Example: Prove that $\sum_{i=1}^{n} (n-i) = \frac{n(n-1)}{2}$.

Proof:

$$\sum_{i=1}^{n} (n-i) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i$$

$$= n \cdot n - \left(\frac{n(n+1)}{2}\right)$$

$$= \frac{2n^2}{2} - \left(\frac{n^2+n}{2}\right)$$

$$= \frac{2n^2 - n^2 - n}{2}$$

$$= \frac{n^2 - n}{2}$$

$$= \frac{n(n-1)}{2} \quad //$$

$a \in \mathbb{Z}$.

Example: Prove: if a^2 is even, a is even.

Proof: Assume a^2 is even. ... p.

$\Rightarrow$ $a^2 = 2k$.

Fact: a is a root of a^2.

Case 1: a is odd.

Then $a = 2b+1$ for some $b \in \mathbb{Z}$.

$\Rightarrow$ $a^2 = 4b^2+4b+1 = 2(2b^2+2b)+1$

$\Rightarrow$ a^2 is odd! Contradiction.

Case 2: a is even

Then $a = 2b$ for some $b \in \mathbb{Z}$.

$\Rightarrow$ $a^2 = 4b^2 = 2(2b^2)$ ... even.

Therefore, a is even. $\square$.

For a given problem, there may be multiple correct algorithms.

Q: How to compare algorithms?

Analyze w.r.t. Time / Running Time
or Space

Running Time: The number of steps taken by an algorithm on a given input.

Running Space: The number of memory cells used by an algorithm on an input.

Performance
↳ Running Time
↳ Input Size n

Running Time is a function $T(n)$ i.e. it is a function of input size.

e.g. Problem P_1

Alg. A_1 with $T_1(n) = 2^n + 3$ .. better

Alg. A_2 with $T_1(n) = n!$

⟹ The slower $T(n)$ grows, the better the algorithm is.

Lecture 5

Big-Oh Notation cont...

- $f(n) = o(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

  note: no other constant than 0 !!

- $f(n) = \omega(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Examples:

1) $n = o(n)$? No. $n = O(n)$? YES

$n = o(n \lg n)$? YES $n \lg n = \omega(n)$? YES

$n^2 = \omega(n^2)$? NO

Remark: Note that

1) $o \Rightarrow O$ e.g. $n = o(n \lg n) \Rightarrow n = O(n \lg n)$

but not $O \Rightarrow o$ e.g. $n^2 = O(n^2)$ but $n^2 \neq o(n^2)$

2) $\omega \Rightarrow \Omega$ but not $\Omega \Rightarrow \omega$

3) $\Theta \Rightarrow O$ but not necessarily $O \Rightarrow \Theta$

4) $\Theta \Rightarrow \Omega$ but " " $\Omega \Rightarrow \Theta$

Exercise: 1) Is O reflexive?

I.e. is it true that $f(x) = O(f(x))$ for all f?

2) Is Ω reflexive?

3) Is Θ reflexive?

4) Is o reflexive?

5) Is ω reflexive?

6) How about transitivity of O, Ω, Θ, o and ω?

Divide and Conquer Technique

... An algorithm design technique.

... If it works, there are three parts/phases

1) Divide: Form subproblem, i.e, partition the input into equal size chunks.
2) Conquer: Solve subproblems recursively.
3) Combine: Use the solutions of subproblems to get a solution for the original problem.

Example: Merge Sort ... a divide-and-conquer alg.

Input: An array A of n items

Divide: Find the middle index q in A and get subproblems $L = A[1, \ldots, q]$ and $R = A[q+1, \ldots, n]$

Conquer: Solve L and R recursively.

Combine: Merge the sorted subarrays L and R.

Careful! e.g. $L = [1,2,3,4]$, $R = [5,6,7,8]$

e.g. $L = [1,3,5,7]$, $R = [2,4,6,8]$

```
MERGE-SORT(A, p, r)                                   T(n)
if p < r
    q = ⌊(p + r)/2⌋                                   Divide  O(1)
    MERGE-SORT(A, p, q)                               ...  T(n/2)
    MERGE-SORT(A, q + 1, r)                           T(n/2)
    MERGE(A, p, q, r)                                 ....  O(n)
```

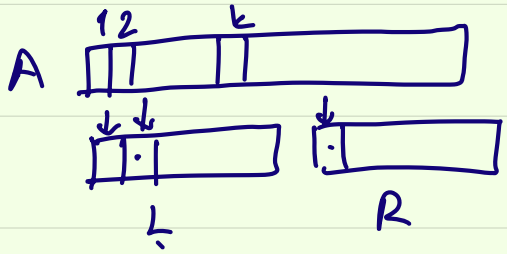
Initial call: Merge-Sort $(A, 1, n)$

```
MERGE(A, p, q, r)
n1 = q − p + 1                                        O(1)
n2 = r − q                                            O(1)
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1                                       } O(n1)
    L[i] = A[p + i − 1]                               }
for j = 1 to n2                                       } O(n2)
    R[j] = A[q + j]                                   }
L[n1 + 1] = ∞                                         O(1)
R[n2 + 1] = ∞                                         O(1)
i = 1
j = 1
for k = p to r                                        } O(n)
    if L[i] ≤ R[j]                                    }
        A[k] = L[i]                                   }
        i = i + 1                                     }
    else A[k] = R[j]                                  }
        j = j + 1                                     }
```

$O(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$T(1) = O(1)$$



Example: Matrix Multiplication

Input: Two square matrices of dimension $n \times n$

i.e. $A_{n \times n}$ and $B_{n \times n}$

e.g. $A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}_{2 \times 2}$ $B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{2 \times 2}$

Note: Two matrices $A_{m \times n}$ and $B_{p \times q}$ are multiplicable if $n = p$ and the result is a matrix $C_{m \times q}$

Divide:

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]_{n \times n} \qquad B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]_{n \times n}$$

Note: $A_{n \times n}$

$A_{11}\ (\frac{n}{2} \times \frac{n}{2})$

$$C = \left[\begin{array}{c|c} C_{11}^{?} & C_{12}^{?} \\ \hline C_{21}^{?} & C_{22}^{?} \end{array}\right]$$

Conquer: Solve $A_{11} \cdot B_{11}$, $A_{12} \cdot B_{21}$, $A_{12} \cdot B_{12}$ .... $A_{22} \cdot B_{22}$ recursively.

Combine:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = ?$$

$$C_{22} = ?$$

$$T(n) = O(1) + 8T(\tfrac{n}{2}) + O(n^2)$$

Lecture 6

D-C Algorithm for Matrix Multiplication

Running time?

$$T(n) = 8T(\tfrac{n}{2}) + O(n^2)$$

$$T(1) = 1$$

Q: Can we get a better D-C algorithm?

YES. See Section 4.2 and the handout.

Analysis of Divide and Conquer Algorithms.

- D-C Algorithms are recursive.

  ⇒ The running time functions are represented as recurrence relations.

  ↓ A recurrence is a function that is described by applying itself to a smaller input.

  e.g. $T(n) = 8T(\tfrac{n}{2}) + O(n^2)$

  $f(n) = n \cdot f(n-1)$

but not $f(n) = 2f(n^2) + 1$

Analysis: Find the running time $T(n)$

↳ Solve $T(n)$, a recurrence

.... asymptotic soln. e.g. $T(n) = \Theta(n^2)$

... exact soln. e.g. $T(n) = 2n^2 + 5$

Need: To solve recurrences (systematically)

Three methods to solve recurrences

1) The Substitution Method

2) The Recursion Tree Method

3) The Master Method

The Substitution Method

Two steps: Given a recurrence

Step 1: Guess a solution (asymptotic or exact)

e.g. $T(n) = O(n^3)$ $\quad T(n) = \Omega(n^3)$

$T(n) = 3n^3 + 4$ $\quad T(n) = \Theta(n^3)$

Step 2: Verify the guess using mathematical induction.

Basis: (Skip if no base case is given in the definition of the recurrence.)

I.H.: Assume that $T(k) \leq \ldots$ $\geq \ldots$ for all $k < n$.

Prove for n.

Example: $T(n) = 8T(\frac{n}{2}) + O(n^2)$. Prove $T(n) = O(n^3)$

Guess: $T(n) \leq cn^3$ for some c.

Verify:

Basis: Skip. no basis case

I.H.: $T(k) \leq ck^3$ for $k < n$.

Prove: $T(n) \leq cn^3$

$$T(n) = 8T(\tfrac{n}{2}) + O(n^2)$$

$$= 8T(\tfrac{n}{2}) + dn^2$$

$$\leq 8\left(c\tfrac{n^3}{8}\right) + dn^2 \quad \text{by I.H.}$$

$$= cn^3 + dn^2$$

$$\Rightarrow T(n) \leq cn^3 + dn^2$$

note: $T(\frac{n}{2}) \leq c(\frac{n}{2})^3$ by I.H.

Mistake: We didn't prove that $T(n) \leq cn^3$!

To fix this we need to make a new guess.

New Guess: $T(n) \leq cn^3 - en^2$

Verify:

I.H: $T(k) \leq ck^3 - ek^2$ for $k < n$.

Prove for n. $T(n) \leq cn^3 - en^2$

$$T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$$

$$= 8T\left(\frac{n}{2}\right) + dn^2$$

$$\leq 8\left(c\frac{n^3}{8} - e\frac{n^2}{4}\right) + dn^2$$

$$= cn^3 - 2en^2 + dn^2$$

$$= cn^3 - en^2 - (e-d)n^2$$

$$\leq cn^3 - en^2 \quad \text{if } e > d.$$

$\Rightarrow T(n) \leq cn^3 - n^2$

$\Rightarrow T(n) = O(n^3)$.

Example: $T(n) = 2T(\frac{n}{2}) + n$ and $T(1) = 1$

Prove: $T(n) = n \lg n + n$

Guess: $T(n) = n \lg n + n$

Verify:

Basis: $T(1) = 1 = 1 \cdot \lg 1 + 1$ ✓

I.H.: $T(k) = k \lg k + k$ for $k < n$

Prove for n:

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$= 2\left[\tfrac{n}{2}\lg(\tfrac{n}{2}) + \tfrac{n}{2}\right] + n \quad \text{by. I.H.}$$

$$= n(\lg n - \lg 2) + n + n$$

$$= n \lg n - n + n + n$$

$$= n \lg n + n \;//$$

$$\Longrightarrow T(n) = n \lg n + n \;\blacksquare$$

Exercise: $T(n) = 2T(\frac{n}{2}) + n$ and $T(1) = 1$

Prove: $T(n) = O(n \lg n)$

Guess: $T(n) \leq cn \lg n$

Strassen's Algorithm:

(See Section 4.2 in CLRS)

Given: Two $n \times n$ matrices

Step 1: Divide into $\frac{n}{2} \times \frac{n}{2}$ matrices, as before

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$O(n^2)$

Step 2: Define 10 new matrices $S_1, S_2, \ldots S_{10}$ where

$O(n^2)$

$$S_1 = B_{11} - B_{22} \quad \ldots \quad n/2 \times n/2$$
$$S_2 = A_{11} + A_{12} \qquad \text{(not recursive!)}$$
$$\vdots$$
$$S_{10} = ?$$

Step 3: Compute the following matrices recursively

$P_1, P_2, \ldots, P_7$ where

(7 recursive calls)

$$P_1 = A_{11} \cdot S_1 \qquad n/2 \times n/2$$
$$\vdots$$
$$P_7 = ?$$

Step 4: Compute C_{11}, C_{12}, C_{21} and C_{22} as follows

$$C_{11} = P_3 + P_4 - P_2 + P_6$$
$$C_{12} = ? \qquad \text{--}$$
$$C_{21} = ? \qquad C_{22} = ?$$

$O(n^2)$

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$$

$$T(1) = 1$$

Lecture 7

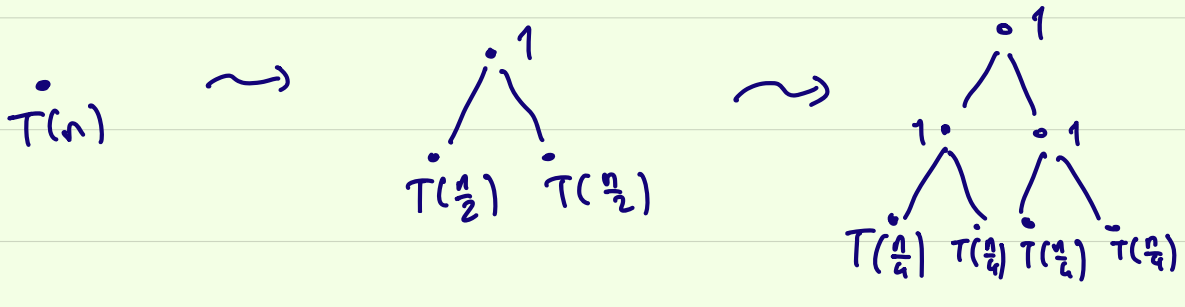
Solving recurrences cont...

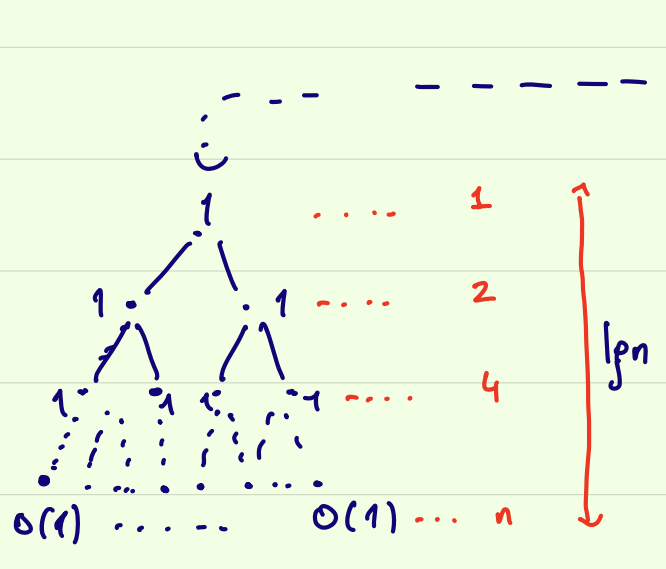
Recursion Tree Method

Idea: For a given recurrence, construct a tree, in which nodes are labeled with terms (without $T(n)$ or its versions), and sum the labels of all nodes.

Example: $T(n) = 2T(\frac{n}{2}) + 1$

The (recursion) tree is constructed gradually:





- $T(n) = \Theta(n \lg n)$

$\Rightarrow$ **Exercise:**

Find $\underbrace{1+2+4+\dots+n}_{\lg n \text{ terms}} = ?$

$n \to \frac{n}{2} \to \frac{n}{4} \to \cdots \to 1$

$\lg n$

$n \to \frac{n}{3} \to \frac{n}{9} \to \cdots \to 1$

$\log_3 n$

Exercise: Solve $T(n) = 2T(\frac{n}{2}) + n$?

Remark:

i) Height of the recursion tree: $\log_b n$

$T(n) = aT(\frac{n}{b}) + f(n)$ ...

$(a > 0, \ b > 0)$

ii) The tree may not be balanced

e.g. $T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + n$

iii) Number of children for each inner node: a

The Master Method

... applied to recurrences of the following form:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where $a > 0$, $b > 1$ and $f(n) > 0$

Master theorem:

- Compare $n^{\log_b a}$ and $f(n)$ w.r.t their growth rates.

Case 1: $f(n)$ is slower

i.e. $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for some $\varepsilon > 0$

then, $T(n) = \Theta\left(n^{\log_b a}\right)$.

Case 2: "Same growth rate"

i.e. $f(n) = \Theta\left(n^{\log_b a}\right)$

Then, $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

Case 3: $f(n)$ grows faster

i.e. $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ for some $\varepsilon > 0$

then $T(n) = \Theta(f(n))$

But we need to check one more condition in Case 3: we must have

$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ for some $0 < c < 1$

the regularity condition.

Example: $T(n) = 8\,T\left(\frac{n}{2}\right) + \Theta(n^2)$

$a = 8 \qquad b = 2 \qquad \log_2^8 = 3$

$n^{\log_2^8} = n^3 \qquad f(n) = \Theta(n^2)$

$\Rightarrow$ Case 1 applies

$\Rightarrow$ $T(n) = \Theta(n^3)$

Example: $T(n) = 7\,T\left(\frac{n}{2}\right) + \Theta(n^2)$

$a = 7 \qquad b = 2 \qquad \Rightarrow \log_b^a = \log_2^7 = 2.87\ldots$

$f(n) = \Theta(n^2) \qquad n^{\log_b^a} = n^{2.87\ldots}$

$\Rightarrow$ $n^{\log_b^a}$ grows faster

$\Rightarrow$ Case 1 applies

$\Rightarrow$ $T(n) = \Theta(n^{2.87\ldots})$

Example: • $T(n) = 9T(\frac{n}{3}) + n$

$n^{\log_b a} = n^{\log_3 9} = n^2$ $\qquad f(n) = n$

$\Rightarrow$ Case 1

$\Rightarrow$ $T(n) = \Theta(n^2)$

• $T(n) = 9T(\frac{n}{3}) + n^2$

$n^{\log_b a} = n^{\log_3 9} = n^2$ $\qquad f(n) = n^2$

$\Rightarrow$ Case 2

$\Rightarrow$ $T(n) = \Theta(n^2 \lg n)$

• $T(n) = 9T(\frac{n}{3}) + n^3$

$n^{\log_b a} = n^{\log_3 9} = n^2$ $\qquad f(n) = n^3$

$9 \cdot (\frac{n}{3})^3 < c \cdot n^3$

$\Rightarrow$ $\frac{n^3}{3} < c n^3$ $\Rightarrow$ $c > \frac{1}{3}$ //

$\Rightarrow$ Case 3 applies

$\Rightarrow$ $T(n) = \Theta(n^3)$

Remark: Either exactly one case applies or none of them applies.

Example: $T(n) = 2T(\frac{n}{2}) + 1$

$a = 2 \qquad b = 2 \qquad n^{\log_2 2} = n \qquad f(n) = 1 = n^0$

$\Rightarrow$ Case 1

$\Rightarrow T(n) = \Theta(n)$

Lecture 8

Example: $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1$, $b = \frac{3}{2}$ $\quad f(n) = 1$

$n^{\log_{3/2} 1} = n^0 = 1$

$\Rightarrow$ Case 2 applies

$\Rightarrow$ $T(n) = \Theta(\lg n)$

Example: $T(n) = 16\,T\left(\frac{n}{4}\right) - n^2 \lg n$.

M.T. does not apply because $f(n) < 0$.

Example: $T(n) = 3\,T\left(\frac{n}{4}\right) + n$

$a = 3$, $b = 4$ $\quad n^{\log_4 3}$ vs n

$\Rightarrow$ Case 3 might apply.

$3 \cdot \frac{n}{4} \leq cn$ for some $c < 1$.

$\Rightarrow$ $c \geq \frac{3}{4}$

$\Rightarrow$ Case 3 applies

$\Rightarrow$ $T(n) = \Theta(n)$

Example: $T(n) = 2^n T\left(\frac{n}{4}\right) + n$

$a = 2^n$ ... not a constant

$\Rightarrow$ M.T. does not apply because a must be constant!

Example: $T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \lg n$

$a = \sqrt{2} \quad b = 2 \quad n^{\log_2 \sqrt{2}} = n^{\log_2 2^{1/2}} = n^{1/2} = \sqrt{n}$

$\sqrt{n}$ vs $\lg n$

$\Rightarrow$ Case 1 applies

$\Rightarrow T(n) = \Theta(\sqrt{n})$

Example: $T(n) = 2\, T\left(\frac{n}{2}\right) + \frac{n}{\lg n}$

$n^{\log_2 2} = n$ vs $n/\lg n$

$\Rightarrow$ M.T. does not apply because n grows faster than $n/\lg n$ by a factor of $\lg n$, which is logarithmic but not polynomial.

Example: $T(n) = T\left(\frac{n}{0.9}\right) + n^2$

M.T. doesn't apply because $b = 0.9 < 1$.

Probability Theory - A short review

Definition: An **experiment** is an activity that has a set of possible outcomes.

e.g. flip a coin .. H, T
flip two coins HH, HT, TH, TT

Definition: An **elementary event** is a possible outcome of an experiment.

e.g. flip a coin .. H, T .. 2 elem. events
flip two coins HH, HT, TH, TT 4 elem. events

Definition: The set of all elementary events is called **sample space**.

e.g. flip a coin $S = \{H, T\}$
flip two coins $S = \{HH, HT, TH, TT\}$
throw a dice once $S = \{1, 2, 3, 4, 5, 6\}$

Definition: An **event** is a subset of a sample space.

e.g. throw a dice .. $S = \{1, \ldots, 6\}$ 2^6 events

Definition: An **event X occurs** if the outcome of the underlying experiment is y s.t. $y \in X$.

e.g.

throw a dice .. $S = \{1, \dots, 6\}$

$X = \{2, 4, 6\}$ outcome $= 4$

$\Rightarrow$ X occurs since $4 \in X$.

$Y = \{1, 2, 5\}$

- $X = S$ ... certain event
- $X = \emptyset$ ... null event

Definition: Two events X and Y are **mutually exclusive** if $X \cap Y = \emptyset$.

e.g. throw a dice .. $S = \{1, \dots, 6\}$

$X = \{2, 4, 6\} \cap Y = \{1, 3, 5\} = \emptyset$

$T = \{1\} \cap Z = \{3\} = \emptyset$

Definition: A **probability distribution** on a sample space S is a function Pr from S to the real number interval $[0,1]$

i.e. $Pr: S \longrightarrow [0,1]$

$Pr(x \in S) = \frac{1}{|S|}$

Let $A \subseteq S$ be an event.

Then $Pr(A) = \sum_{x \in A} Pr(x)$

$Pr(A)$: the probability that event A occurs.

e.g. throw a dice .. $S = \{1, \dots, 6\}$

$X = \{2, 4, 6\}$ $\quad$ $Y = \{1, 3, 5\}$

$Pr(X) = Pr(2) + Pr(4) + Pr(6)$

$= \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{3}{6}$

Remark:

- $Pr(A) \geq 0$ for every $A \subseteq S$.
- $Pr(S) = 1$

  $\Rightarrow Pr(A) \leq 1$. for every $A \subseteq S$
- $Pr(A \cup B) = Pr(A) + Pr(B)$ if A and B are mut. exclusive.

- In general, $\Pr\left(\bigcup_i A_i\right) = \sum_i \Pr(A_i)$ if A_i and A_j are mut. exclusive for every i and j.

$$\Pr(A) = \frac{\#\text{ of elementary events in } A}{|S|}$$

$$= \frac{|A|}{|S|}$$

Definition: Two events A and B are **independent** if $\Pr(A \cap B) = \Pr(A) \cdot \Pr(B)$

Definition: A **random variable** X is a function from the sample space S to $\mathbb{R}$, the real numbers.

$$X : S \longrightarrow \mathbb{R}$$

probability that "$X = r$" occurs

$$\text{"}X = r\text{"} = \{ s \in S \mid X(s) = r \} \subseteq S$$

e.g. $\text{"}X = 5\text{"} = \{ s \in S \mid X(s) = 5 \}$

$$Pr(X=r) = \sum_{\substack{s \in S \\ s.t.\ X(s)=r}} Pr(s)$$

Example: Throw two die

$$S = \{(1,1), (1,2), \ldots, (6,6)\} \qquad |S| = 36$$

X = max. of two outcomes

e.g. $X(1,2) = 2$ or $X(3,3) = 3$.

$$Pr(X=5) = Pr((1,4)) + Pr((2,3)) + Pr((4,1)) + Pr((3,2))$$

$$= \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} = \frac{4}{36}$$

Exercise: Y = the sum of the two outcomes

- $Pr(Y=8)$?
- $Pr(Y=5 \cup Y=3)$?

Definition: Expected value of a rand. var. X:

$$E[X] = \sum_{\substack{s \in S \\ X(s)=r}} r \cdot Pr(X(s)=r)$$

$$= \sum r \cdot Pr(X=r)$$

Example: Throw two die

$$S = \{(1,1), (1,2), \ldots, (6,6)\} \quad |S| = 36$$

X = max. of two outcomes

e.g. $X(1,2) = 2$ or $X(3,3) = 3$.

$E[X] = ?$

$Pr(X=1) = \frac{1}{36}$

$Pr(X=2) = \frac{3}{36}$

$Pr(X=3) = \frac{5}{36}$

$Pr(X=4) = \frac{7}{36}$

$Pr(X=5) = \frac{9}{36}$

$Pr(X=6) = \frac{11}{36}$

$$E[X] = 1 \cdot \frac{1}{36} + 2 \cdot \frac{3}{36} + \ldots + 6 \cdot \frac{11}{36}$$

$$= \frac{161}{36}$$

Remark: Linearity of expectation.

$$E[X+Y] = E[X] + E[Y]$$

$$E[aX] = a\,E[X] \quad \text{-- } a \text{ is a constant}$$

$$E[aX+Y] = a\,E[X] + E[Y]$$

$$E[X \cdot Y] = E[X] \cdot E[Y] \quad \text{if}$$

X and Y are independent.

Indicator Random Variables

Let A be an event. We define an indicator random variable $I(A)$ as follows:

$$I(A) = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{owise} \end{cases}$$

e.g. $S = \{H, T\}$ flip a coin

$$Pr(\{H\}) = Pr(\{T\}) = \frac{1}{2}$$

$$I(H) = \begin{cases} 1 & \text{if the outcome is } H \\ 0 & \text{owise.} \end{cases}$$

$$E(I(H)) = 1 \cdot Pr(I(H) = 1) + 0 \cdot Pr(I(H) = 0)$$

$$= Pr(I(H) = 1)$$

$$= \frac{1}{2}$$

Example: Hat Check Problem

- N people, each with a hat, at a party
- Hats are mixed
- Each person picks a hat after the party.
- Observe: Probability that a person picks their own hat is $1/N$.
- Question: What is the expected number of people who get their own hats?

  i.e. Define $R :=$ number of people who ... and find $E[R]$.

- $E[R] = 1 \cdot Pr(R=1) + 2 \cdot Pr(R=2) + \dots + N \cdot Pr(R=N)$.

  .... Hard to compute in general.

- Instead, define an indicator random variable:

$$R_i = \begin{cases} 1 & \text{if } i\text{-th person picks their own hat} \\ 0 & \text{otherwise} \end{cases}$$

$i = 1, 2, \dots, N$

$$\Rightarrow R = R_1 + R_2 + \dots + R_N$$

$$\Rightarrow E[R] = E[R_1] + E[R_2] + \dots + E[R_N] \quad \text{by linearity}$$

$$E[R_i] = 0 \cdot Pr(R_i = 0) + 1 \cdot Pr(R_i = 1)$$
$$= 0 + 1 \cdot \frac{1}{N} = \frac{1}{N}$$

$$\Rightarrow E[R] = \underbrace{\frac{1}{N} + \frac{1}{N} + \dots + \frac{1}{N}}_{N} = 1$$

Lecture 9

Analysis of Divide and Conquer Algorithms

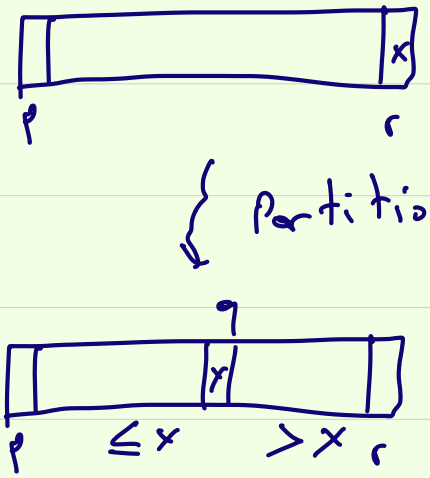
Example: Quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]                    ... pivot
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

$O(n)$

Initial Call: Quicksort $(A, 1, n)$



Worst-case occurs when

- x is min $T(n) = T(n-1) + \Theta(n) = O(n^2)$
- x is max " "
- A is already sorted " "

Best case occurs when subarrays are balanced

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

$$= \Theta(n \lg n)$$

Exercise:

- $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \lg n)$
- $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$

Prove both by using the substitution method.

To avoid the worst-case, we can randomize the algorithm.

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q − 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

$\Theta(n)$

$\Longrightarrow$ $T(n)$ ... a random variable.

Expected value of $T(n)$?

(expect running time of Rand. Quicksort)

Two sources of cost:

i) partitions

ii) Recursive calls

Question: How many recursive calls for input size n?

Theorem: For input size n, Quicksort makes $2n+1$ recursive calls.

Proof: by induction

Base: $n = 0 \Rightarrow 2 \cdot 0 + 1 = 1$.

I.H.: For $k < n$, $2k+1$ calls.

Show: For n, $2n+1$ calls are made.

There must be a partition $a : n-1-a$

By I.H., For size a, $2a+1$ calls

For size $n-1-a$, $2(n-1-a)+1$ calls

Total number of calls will then be

$$1 + (2a+1) + (2(n-1-a)+1)$$

$$= \cancel{1} + 2a + \cancel{1} + 2n - \cancel{2} - \cancel{2a} + 1$$

$$= 2n+1. \quad \checkmark$$

Note: $2n+1 = O(n)$

Cost of Partitions

- Partition runs in $O(n)$ time for size n.
- Partition uses $n-1$ comparisons for size n.

  $\Rightarrow$ Calculate cost of partitions by counting the number of comparisons throughout the execution.

- Define: v_i = i-th element in the input

  (Assume: All elements are distinct)

- Define a new random variable C_{ij} as "the number of times that v_i and v_j are compared"

Observe:

- v_i and v_j are compared only if one of them is a pivot. Note that all comparisons are made against pivot.

- If v_i and v_j are compared once, they are never compared again.

$\Rightarrow$ C_{ij} is indeed an indicator random variable:

$$C_{ij} = \begin{cases} 1 & \text{if } v_i \text{ and } v_j \text{ are compared} \\ 0 & \text{otherwise.} \end{cases}$$

$$E[C_{ij}] = 0 \cdot Pr(C_{ij} = 0) + 1 \cdot Pr(C_{ij} = 1)$$

$$= Pr(C_{ij} = 1)$$

$$= Pr(v_i \text{ and } v_j \text{ are compared})$$

$$= Pr(\underline{v_i \text{ is pivot}} \text{ or } \underline{v_j \text{ is pivot}})$$

$$= Pr(v_i \text{ is pivot}) + Pr(v_j \text{ is pivot})$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

$$= \frac{2}{j-i+1}$$

• Define X = total number of comparisons.

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij}$$

$$E[X] = E\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij}\right]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[c_{ij}]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Note:

$j = i+1 \dots n$

$j - i = 1, \dots, n-i$

let $j - i = k$

$\Rightarrow k = 1, \dots, n-i$

$$= \sum_{i=1}^{n} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$\leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k+1}$$

$$= n \cdot \left(\sum_{k=1}^{n} \frac{2}{k+1}\right)$$

$$= 2n \cdot \sum_{k=1}^{n} \frac{1}{k+1}$$

$$\leq 2n \left(\sum_{k=1}^{n} \frac{1}{k}\right) \longrightarrow$$ n-th Harmonic number: $O(\lg n)$

$$= 2n \cdot O(\lg n)$$

$$= O(n \lg n)$$

$$\Longrightarrow E[X] = O(n \lg n)$$

Expect Running Time of Randomized Quicksort

- Cost of rec. calls : $O(n)$
- Cost of partitions : $O(n \lg n)$

$+$

$O(n \lg n)$.

Lecture 10

Lower Bounds

Consider the sorting problem

... Insertion Sort $O(n^2)$

... Merge Sort $O(n \lg n)$

... Quick Sort $O(n^2)$

... comparison-based sorting algorithms

Comparisons: Given two items a_i and a_j in the input, we have

$a_i \leq a_j$, $a_i < a_j$, $a_i = a_j$, $a_i \geq a_j$, $a_i > a_j$

Assume: All elements are distinct in the input

$\Rightarrow$ $a_i = a_j$ is useless

$\Rightarrow$ $a_i \leq a_j$ and $a_i < a_j$ ... equivalent

$\Rightarrow$ $a_i \geq a_j$ and $a_i > a_j$ ... equivalent

$\Rightarrow$ Using $a_i \leq a_j$ suffices!

Decision Tree Model

* a fully binary tree: every has either 0 or 2 children

inner nodes: 2 children

leaves: 0 "

e.g. 

✓ ✗

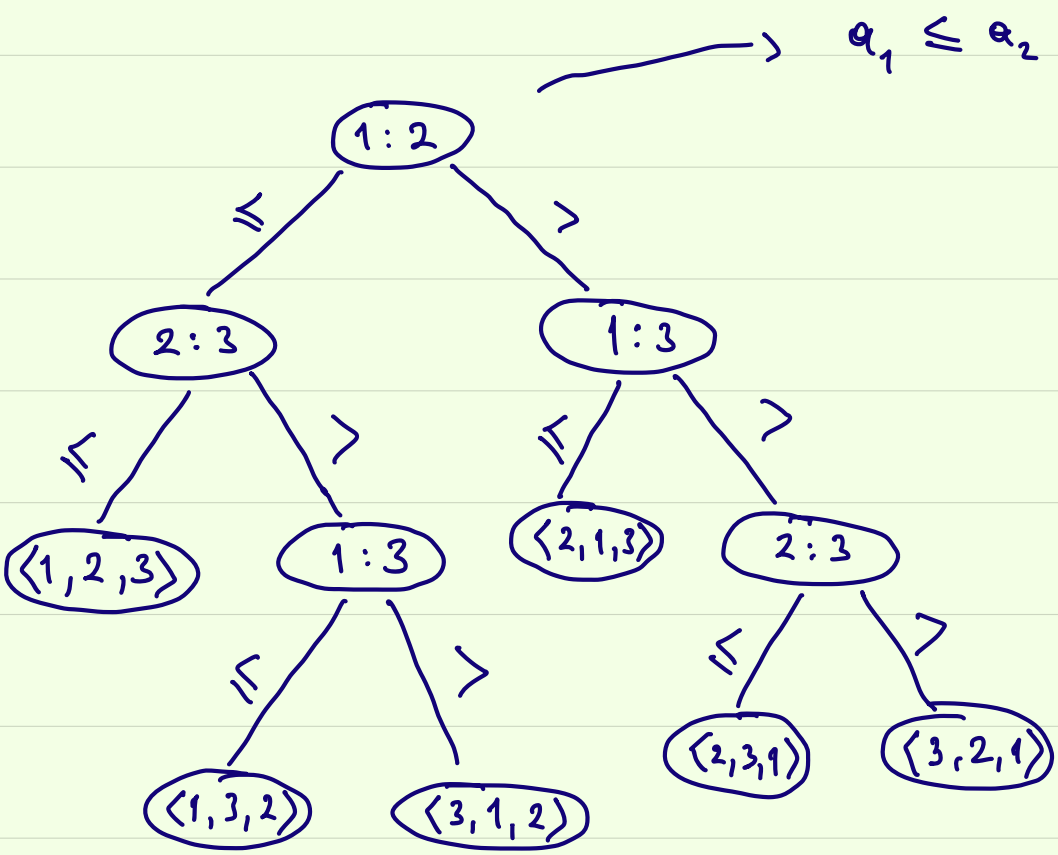
* Used to represent all the comparisons that an algorithm performs on an input.

- inner nodes: labeled with "$i:j$" for $1 \leq i, j \leq n$, meaning that the node has the label "$a_i \leq a_j$"

- leaves: labelled with permutations of the input

... $n!$ permutations

Example: $A = [a_1, a_2, a_3]$



Exercise: DTM for $A = [a_1, a_2, a_3, a_4]$

Remarks:

- An execution of the algorithm corresponds to a path from the root to a node.

- Lower bound for the worst-case is equivalent to
- the maximum number of comparisons

which is equivalent to

- the length of the longest path in DTM

which is equivalent to

- the height of the DTM.

Lemma: For a binary tree, let h be the height and ℓ be the number of leaves reachable from the root. Then $\ell \leq 2^h$.

Proof: by induction on h

Basis: $h = 0$ .. a single node

$\ell = 1 \leq 2^0$ ✓

I.H.: Assume the claim for $k < h$.

Show: $\ell \leq 2^h$

Observe: Assume that we increase the height from $h-1$ to h by adding as many new leaves as possible. Now all leaves at height $h-1$ become inner nodes with two children.

Then, # of leaves at height h

$= 2 \cdot (\#$ of leaves at heigh $h-1)$

$\leq 2 \cdot 2^{h-1}$ (by I.H.)

$= 2^h$

$\Rightarrow \ell \leq 2^h$.

Theorem: Any comparison-based sorting algorithm requires $\Omega(n \lg n)$ comparisons in the worst-case. (For size n)

Proof:

- n ... input size

  $\Rightarrow n!$ leaves in D.T.

- Let h = height of the D.T.
- Let ℓ = # of reachable leaves in DT.

$\Rightarrow n! \leq \ell \leq 2^h$

$\Rightarrow n! \leq 2^h$

$\Rightarrow \lg n! \leq \lg 2^h = h$

$\Rightarrow \quad h \geq \lg n! = \Theta(n \lg n)$

$\Rightarrow \quad h = \Omega(n \lg n)$

$$\lg n! = \lg(n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1)$$

$$= \lg n + \lg(n-1) + \cdots + \lg(2) + \lg 1$$

$$\leq \lg n + \lg n + \cdots + \lg n + \lg n$$

$$\leq n \lg n \quad \square$$

<u>Exercise:</u> Prove $\lg n! = \Omega(n \lg n)$

$\lg n! \geq c\, n \lg n$ for all $n \geq n_0$

1) Solve $T(n) = 2T(\sqrt{n}) + 1$

Change variables ...

Let $m = \lg n$. Then $n = 2^m$. Note $\sqrt{n} = 2^{m/2}$!

$T(n) = T(2^m) = 2T(2^{m/2}) + 1$

Let $S(m) = T(2^m)$.

$\Rightarrow S(\frac{m}{2}) = T(2^{m/2})$

$\Rightarrow S(m) = 2S(m/2) + 1$

By M.T. case 1, we get $S(m) = \Theta(m)$.

$T(n) = T(2^m) = S(m) = \Theta(m) = \Theta(\lg n)$.

$\Rightarrow T(n) = \Theta(\lg n)$.

2) Is $O((x+y)^2) = O(x^2) + O(y^2)$

Let's take $x \leq y$ w.l.o.g.

$$(x+y)^2 = x^2 + 2xy + y^2 \leq 4y^2$$

$$\Rightarrow (x+y)^2 = O(y^2) \quad \text{by def. big-O.}$$

$$\Rightarrow (x+y)^2 = O(x^2) + O(y^2)$$

$$\Rightarrow O((x+y)^2) = O(O(x^2) + O(y^2))$$
$$= O(O(x^2)) + O(O(y^2))$$
$$= O(x^2) + O(y^2)$$

3) Maximum Subarray Problem.

Input: $A = [1 \ldots n]$ of integers.
Output: Indices $i, j \in \{1 \ldots n\}$ s.t. $A[i \ldots j]$ has the greatest sum.
($A[i, \ldots, j]$ is nonempty, contiguous)

Solve using Divide and Conquer approach.

Given: $A[p \ldots r]$ p: lower index
r: higher index

Small: $n = 1 \Rightarrow i = j = 1$, return $A[1]$.

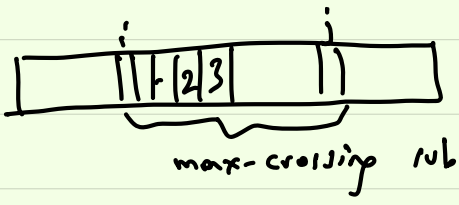
Divide: Find the middle index $q = \lfloor \frac{p+r}{2} \rfloor$ and obtain

$L = A[p \ldots q]$ and $R = A[q+1, \ldots, r]$

Conquer: Solve L and R recursively.

i.e. Find max-sum-subarrays in L and R.

Combine:



- Find max-subarray that crosses the middle index
  q: cross ... (low-end, high-end, sum)

- Compare solutions:
  - From L, - From R and - cross
- Return the one maximum sum.

4) Search problem:

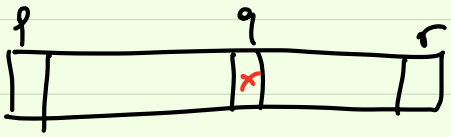
Input: An array $A[1....n]$ which is sorted.
And an integer z

Output: Yes, if z appears in A. No, otherwise.

Goal: Apply divide and conquer approach

BinSearch (A, p, r, z)

if $p \geq r$
return No.
else $q = \lfloor \frac{p+r}{2} \rfloor$ ← divide
if $z = A[q]$
return YES
else if $z < A[q]$
BinSearch (A, p, q, z)
else BinSearch $(A, q+1, z)$



$T(n) = T(\frac{n}{2}) + O(1) = \Theta(\log n)$ (M.T. case 2.)

5) Selection Problem

Input: $A[1...n]$ of distinct numbers. And $i \in [1...n]$

Output: The i-th smallest element.

Note: If $i=1$, then it is asking for min.
If $i=n$, then " " " max.

An obvious alg: - Sort A ($O(n \lg n)$ time) } overall $O(n \lg n)$
- Return $A[i]$ ($O(1)$) }

Select(A, i):

1.) Divide A into $\lfloor \frac{n}{5} \rfloor$ groups of 5 elements each and at most one group of the remaining $n \bmod 5$ elements. — $\Theta(n)$

e.g. $n=12$ 2 groups with 5 elements
1 group with 2 elements

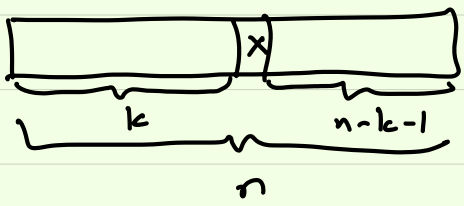
2.) Find median of each of $\lceil \frac{n}{5} \rceil$ groups. — $\Theta(n)$
(e.g. use insertion-sort and pick the 3rd element).

3.) Find median-of-medians and call it x. — $T(\lceil \frac{n}{5} \rceil)$ $O(n)$

4.) Use x as pivot to partition A into three subarrays:
- one containing elements smaller than x

- one " x
- one " elements larger than x

$O(1)$ 5.) Compute the number of elements smaller than x and call it k.

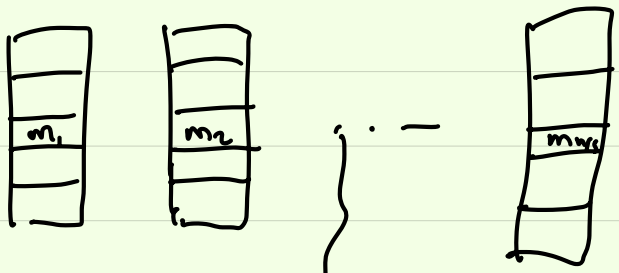


6.) If $i = k+1$, return x
If $i < k+1$, call Select$(A[1,\dots,k], i)$
if $i > k+1$, call Select$(A[k+2,\dots n], i-k-1)$.

$T(\frac{7n}{10}+6)$

Running Time $= O(n)$

A:



median of medians
= median of A.

Lecture 11

COUNTING SORT

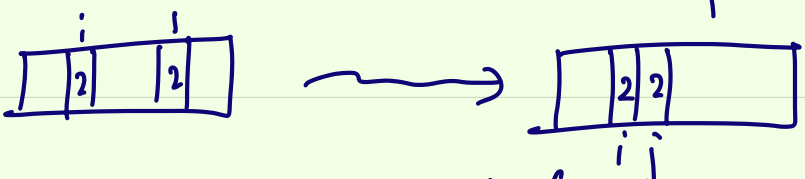
Suppose we want to sort n elements from an interval $[0, k]$.

Input: $A[1, \ldots, n]$ s.t. $0 \leq A[i] \leq k$

Output: $B[1, \ldots, n]$ .. sorted

- This algorithm uses an auxiliary array $C[0, \ldots, k]$

- This is a stable sorting algorithm:
  Input elements with the same value preserve their relative order in the output.

  e.g. 

  ie. $A[i]$ appears before $A[j]$ in the output.

COUNTING-SORT(A, B, n, k)

let $C[0..k]$ be a new array
for $i = 0$ **to** k — $O(k)$
  $C[i] = 0$
for $j = 1$ **to** n — $O(n)$
  $C[A[j]] = C[A[j]] + 1$
for $i = 1$ **to** k — $O(k)$
  $C[i] = C[i] + C[i-1]$
for $j = n$ **downto** 1 — $O(n)$
  $B[C[A[j]]] = A[j]$
  $C[A[j]] = C[A[j]] - 1$

Example:

A =

1	2	3	4	5	6	7
0	4	2	0	1	5	2

$n=7$, $k=5$

1) C

0	1	2	3	4	5
0	0	0	0	0	0

2) C

0	1	2	3	4	5
2	1	2	0	1	1

3) C

0	1	2	3	4	5
2	3	5	5	6	7

4) C

0	1	2	3	4	5
~~2~~ 1	~~3~~ 2	~~5~~ 4	5	~~6~~ 5	~~7~~ 6

B

1	2	3	4	5	6	7
0	0	1	2	2	4	5

* $j = 7$ $A[7] = 2$
  $C[A[7]] = C[2] = 5$
  $B[5] = 2$

* $j = 6$ $A[6] = 5$
  $C[5] = 7$
  $B[7] = 5$

* $j = 5$ $A[5] = 1$
  $C[1] = 3$
  $B[3] = 1$

* $j = 4$ $A[4] = 0$
  $C[0] = 2$
  $B[2] = 0$

* $j = 3$ $A[3] = 2$
  $C[2] = 4$
  $B[4] = 2$

$j = 2$ $A[2] = 4$
  $C[4] = 6$
  $B[6] = 4$

* $j = 1$ $A[1] = 0$
  $C[0] = 1$
  $B[1] = 0$

Running Time : $O(n+k)$

Note: $O(n)$ if $k = O(n)$

Radix Sort

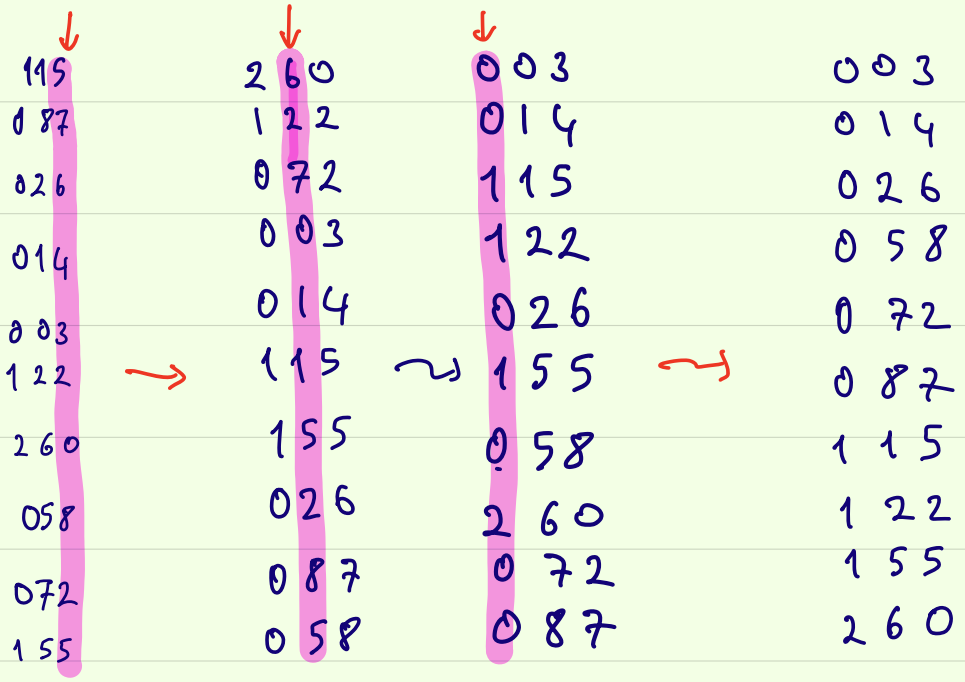
Goal: Sort n numbers each of which has d digits.

for $i = 1$ to d

use a <u>stable sort</u> to sort the input w.r.t. digit i.

(Note: digit 1 is the rightmost one)

We can apply Counting-sort with $[0, 9]$ at each iteration



Running Time: $d \cdot O(n+k)$

since $k = 9$ above, we get $O(d \cdot n)$

Lecture 12

Bucket Sort

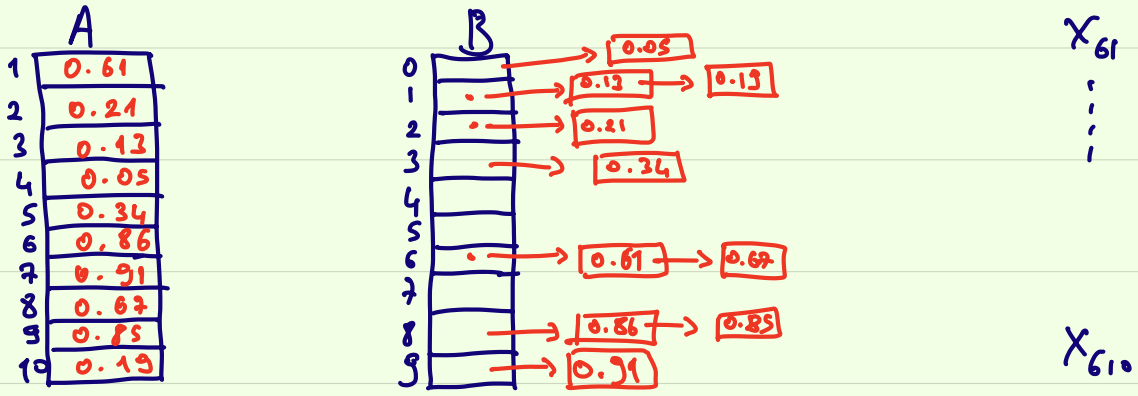
BUCKET-SORT(A, n)

let $B[0..n-1]$ be a new array
for $i = 0$ **to** $n-1$ $\quad \Theta(n)$
    make $B[i]$ an empty list
for $i = 1$ **to** n $\quad \Theta(n)$
    insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$
for $i = 0$ **to** $n-1$ $\quad \Theta(n_0^2) + \Theta(n_1^2) + \dots + \Theta(n_{n-1}^2)$
    sort list $B[i]$ with insertion sort
concatenate lists $B[0], B[1], \dots, B[n-1]$ together in order
return the concatenated lists

Input: $A[1, \dots, n]$ s.t. $0 \le A[i] < 1$

Uses $B[0, \dots, n-1]$ -- a list of buckets

Example:



$B[\lfloor n \cdot A[i] \rfloor] = A[i]$

$n = 10 \quad A[1] = 0.61 \Rightarrow \lfloor 10 \cdot 0.61 \rfloor = \lfloor 6.1 \rfloor = 6$

$\Rightarrow B[6] = 0.61$

Remark: Worst-case running time is $O(n^2)$

- Let m_i = the number of elements in $B[i]$

  $m_i = 0, 1, \ldots\ldots, n$

  $\Longrightarrow$ m_i is a random variable!

- $T(n) = \Theta(n) + \sum_{i=0}^{n-1} \Theta(m_i^2)$

  $\Longrightarrow$ $T(n)$ is a random variable!

  $E[T(n)] = ?$

- $E[T(n)] = E\left[\Theta(n) + \sum_{0}^{n-1} \Theta(m_i^2)\right]$

  $= E[\Theta(n)] + E\left[\sum_{i=0}^{n-1} \Theta(m_i^2)\right]$

  $= \Theta(n) + \sum_{i=0}^{n-1} E[\Theta(m_i^2)]$

  $= \Theta(n) + \sum_{i=0}^{n-1} \Theta(\underbrace{E[m_i^2]}_{?})$

- Define $X_{ij} = \begin{cases} 1 & \text{if } A[j] \text{ is in } B[i] \\ 0 & \text{owise} \end{cases}$

  e.g. $X_{61} = 1$

  $X_{68} = 1$ $\Longrightarrow$ $X_{61} + X_{62} + \cdots \times X_{610} = 2$

  $X_{63} = 0$ $\qquad\qquad = m_6$

  $\vdots$

In general $m_i = \sum_{j=1}^{n} X_{ij}$

$$\Rightarrow \quad E[m_i^2] = E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right]$$

$$\left(\sum_{j=1}^{n} X_{ij}\right)^2 = \left(\sum_{j=1}^{n} X_{ij}\right) \cdot \left(\sum_{j=1}^{n} X_{ij}\right)$$

$$= \left(\sum_{j=1}^{n} X_{ij}\right) \cdot \left(\sum_{k=1}^{n} X_{ik}\right)$$

$$= \sum_{j=1}^{n} \sum_{k=1}^{n} X_{ij} \cdot X_{ik}$$

$$= \sum_{j=1}^{n} X_{ij} X_{i1} + X_{ij} \cdot X_{i2} + \cdots + X_{ij} \cdot X_{in}$$

$$= X_{i1} \cdot X_{i1} + X_{i2} \cdot X_{i1} + \cdots + X_{in} \cdot X_{i1}$$
$$+ X_{i1} \cdot X_{i2} + X_{i2} \cdot X_{i2} + \cdots + X_{in} \cdot X_{i2}$$
$$+ \cdots$$
$$+ \cdots$$
$$+ X_{i1} \cdot X_{in} + X_{i2} \cdot X_{in} + \cdots + X_{in} \cdot X_{in}$$

$$= \sum_{j=1}^{n} X_{ij}^2 + \sum_{1 \le j \le n} \sum_{\substack{1 \le k \le n \\ j \neq k}} X_{ij} \cdot X_{ik}$$

- $E[X_{ij}^2] = 1^2 \cdot Pr(X_{ij}^2 = 1) + 0^2 \cdot Pr(X_{ij} = 0)$

$$= 1 \cdot \frac{1}{n} = \frac{1}{n}$$

- $E[X_{ij} \cdot X_{ik}] = E[X_{ij}] \cdot E[X_{ik}]$

$$= \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$$

$$E[m_i^2] = E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right]$$

$$= E\left[\sum_{j=1}^{n} \sum_{k=1}^{n} X_{ij} \cdot X_{ik}\right]$$

$$= E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1 \le j \le n} \sum_{\substack{1 \le k \le n \\ j \ne k}} X_{ij} \cdot X_{ik}\right]$$

$$= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{j} \sum_{\substack{k \\ k \ne j}} E[X_{ij} \cdot X_{ik}]$$

$$= \sum_{j=1}^{n} \frac{1}{n} + \sum_{j} \sum_{k} \frac{1}{n^2}$$

$$= n \cdot \frac{1}{n} + n \cdot (n-1) \cdot \frac{1}{n^2}$$

$$= 1 + \frac{n^2 - n}{n^2} = \frac{n^2 + n^2 - n}{n^2} = 2 - \frac{1}{n}$$

Recall:

$$E[T(n)] = E\left[\Theta(n) + \sum_{0}^{n-1} \Theta(m_i^2)\right]$$

$$= E[\Theta(n)] + E\left[\sum_{i=0}^{n-1} \Theta(m_i^2)\right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} E[\Theta(m_i^2)]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} \Theta(E[m_i^2])$$

?

$$= \Theta(n) + \sum_{i=0}^{n-1} \Theta\left(2 - \frac{1}{n}\right)$$

$$= \Theta(n) + n \cdot \Theta\left(2 - \frac{1}{n}\right)$$

$$= \Theta(n) + \Theta(2n - 1)$$

$$= \Theta(n)$$

Lecture 13

DYNAMIC PROGRAMMING

- An algorithm design technique

- **Main Idea:** Solve each subproblem only once and store its solution to use later when necessary.

Example: Fibonacci numbers

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$F_1 = 1$ $\qquad F_n = F_{n-1} + F_{n-2}$ for $n \geq 3$

$F_2 = 1$

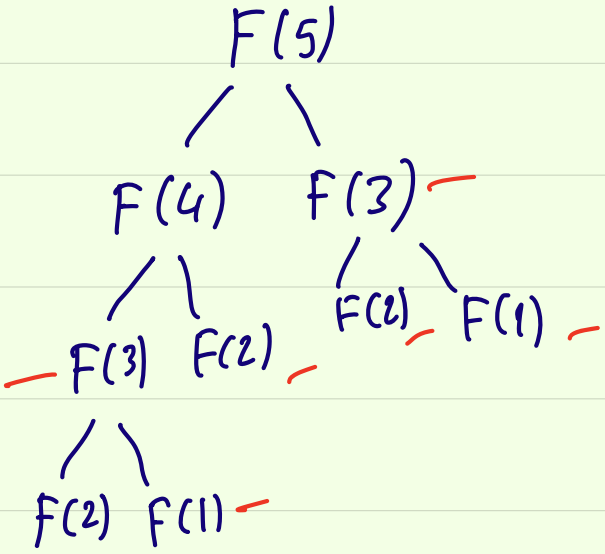
Problem: Find the n-th Fibonacci number.

i.e. Input: n $\qquad$ Output: F_n

```
F(n) {
  if (n==1 or n==2)
     return 1
  else return F(n-1)+F(n-2)
}
```

Recursive Algorithm

Take $n = 5$



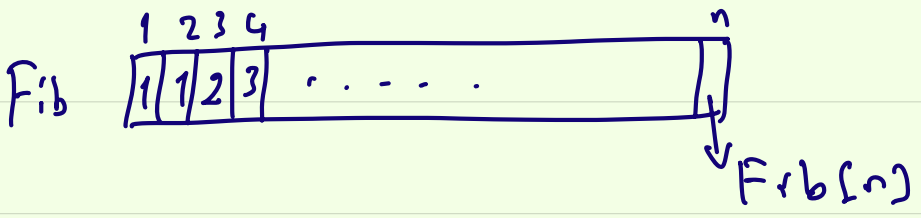
Challenge:

It recomputes the same subproblems

$\Rightarrow$ inefficiency.

Bottom-up DP Algorithm

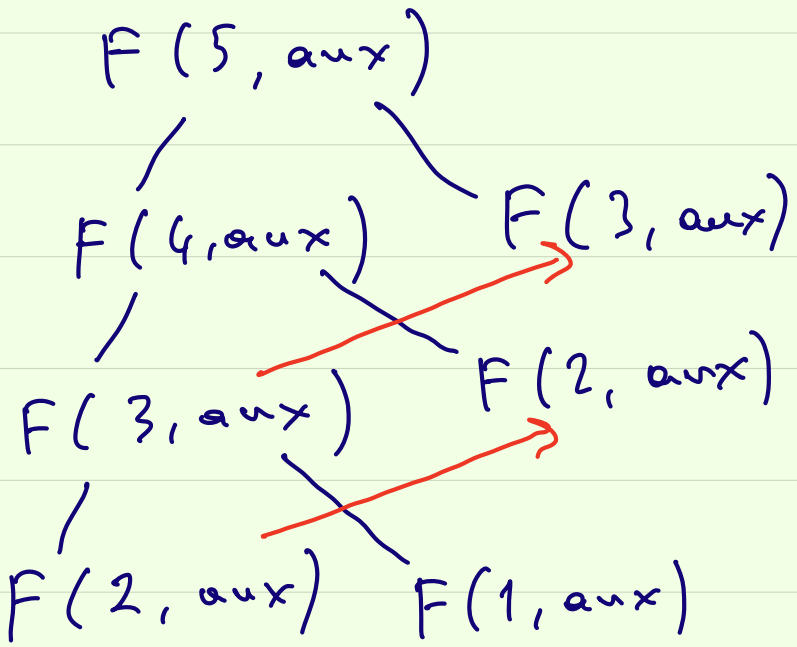
```
F(n) {
  Fib[1] = 1
  Fib[2] = 1
  for ( i = 3 to n ){
      Fib[i] = Fib[i-1] + Fib[i-2]
  }
  return Fib[n]
}
```



$O(n)$ - time

Top-Down DP Algorithm

```
F(n, aux) {
  if ( aux[n] is defined)
        return aux[n]
  if (n==1 or n==2)
        return 1
  else
        aux[n] = F(n-1, aux) + F(n-2, aux)
  return aux[n]
}
```



Lecture 14

Dynamic Programming cont.

Matrix Chain Multiplication Problem

Input: A chain of matrices $\langle A_1, A_2, \ldots, A_n \rangle$ with dimensions $p_{i-1} \times p_i$ for $i = 1, \ldots, n$.

e.g. $A_1 : p_0 \times p_1$

$A_2 : p_1 \times p_2$

$A_3 : p_2 \times p_3$

Output: A full parenthesization of $\langle A_1 \ldots A_n \rangle$ in such a way that the required number scalar multiplications is minimum.

e.g. $$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}_{3\times 2} \cdot \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}_{2\times 2} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}_{3\times 2}$$

we needed $3 \cdot 2 \cdot 2 = 12$ sc. mult.

Remark: For dim. $p \times q$ and $q \times r$, we need $p \cdot q \cdot r$ sc. multiplications.

Example: A_1: 10×100 A_2: 100×5 A_3: 5×50

The min. number of sc. mult for $\langle A_1, A_2, A_3 \rangle$?

- $(A_1 \cdot A_2) \cdot A_3$

$10 \cdot 100 \cdot 5 = 5000$

$10 \cdot 5 \cdot 50 = 2500$

7500

- $A_1 \cdot (A_2 \cdot A_3)$

$100 \cdot 5 \cdot 50 = 25000$

$10 \cdot 100 \cdot 50 = 50000$

75000

Definition: A product of matrices is fully parenthesized if it is either a single matrix or a product two fully parenthesized product of matrices.

i.e. - A_1

- $(A_1 \dots A_k)(A_{k+1} \dots A_n)$

$P(n)$: The number of possible full parent. for a sequence $\langle A_1, \dots, A_n \rangle$

$$P(n) = \begin{cases} 0 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)\, P(n-k) & \text{if } n > 1 \end{cases}$$

Exercise: • $P(n) = \Omega\left(\frac{4^n}{n\sqrt{n}}\right)$. Prove

• $P(100) = ?$

DP Solution: -- bottom up.

• Let $A_{ij} = \langle A_i, \ldots, A_j \rangle$ ... a subproblem

• Note: If the solution of A_{ij} is nontrivial, i.e. length of $A_{ij} \geq 2$, then there must be an integer $k \in [i, j)$ such that A_{ij} is split as $A_{ik} = \langle A_i, \ldots, A_k \rangle$ and $A_{(k+1)j} = \langle A_{k+1}, \ldots, A_j \rangle$.

• Then, cost of A_{ij} is equal to sum of
  - cost of A_{ik} $p_{i-1} \times p_k$
  - cost of $A_{k+1 j}$ $p_k \times p_j$
  - cost of multiplying A_{ik} and $A_{k+1 j}$ $p_{i-1} \cdot p_k \cdot p_j$

$p_{i-1} \times p_i, p_i \times p_{i+1}, \ldots, p_{k-1} \times p_k$

- Let $m[i,j]$: min cost of A_{ij}

  - $m[i,i] = 0$
  - $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$

    where $k \in [i,j)$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j \} & \text{if } i < j \end{cases}$$

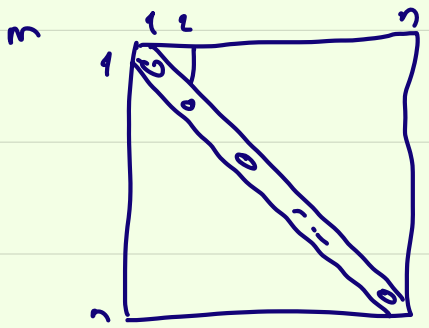
the recursive solution.

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n                    // chain length is 1.
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

$A_1 A_2$, $A_2 A_3 \dots A_{n-1} A_n$

$A_1 A_2 A_3$, $A_2 A_3 A_4 \dots$ —

$A_1 \dots A_1$ $A_2 \dots A_5$



$O(n^3)$

solution: $m[1,n]$

Longest Common Subsequence Problem

Input: $X = [x_1, \ldots, x_m]$ $\quad |X| = m$

$Y = [y_1, \ldots, y_n]$ $\quad |Y| = n$

Output: A subsequence common to both X and Y with maximum length.

e.g. X: springtime "pine"

Y = pioneer

- **Brute-force solution**

For every subsequence of X, check if it is a subsequence of Y.

- 2^m possible subsequences of X
- Each subsequence requires $O(n)$ time to check.

Total cost: $O(n \cdot 2^m)$

DP Solution:

- Let $X_i = [x_1, \ldots, x_i]$ i-th prefix $X = [x_1, \ldots, x_m]$

$Y_j = [y_1, \ldots, y_j]$ j-th prefix $Y = [y_1, \ldots, y_n]$

- Let $Z = [z_1, \ldots, z_k] = LCS(X, Y)$

  - If $x_m = y_n$, then
    $z_{k-1} = LCS(X_{m-1}, Y_{n-1})$

  - If $x_m \neq y_n$, then
    $Z = LCS(X, Y_{n-1})$ or
    $Z = LCS(X_{m-1}, Y)$

- Let $c[i,j] = LCS(X_i, Y_j)$

$$c[i,j] = \begin{cases} 0 & i=0 \text{ or } j=0 \\ 1 + c[i-1, j-1] & \text{if } i,j>0,\ x_i = y_j \\ \max\{c[i-1,j], c[i,j-1]\} & \text{if } i,j>0,\ x_i \neq y_j \end{cases}$$

```
LCS-LENGTH(X, Y, m, n)
  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
  for i = 1 to m
      c[i, 0] = 0
  for j = 0 to n
      c[0, j] = 0
  for i = 1 to m
      for j = 1 to n
          if x_i == y_j
              c[i, j] = c[i − 1, j − 1] + 1
              b[i, j] = "↖"
          else if c[i − 1, j] ≥ c[i, j − 1]
                  c[i, j] = c[i − 1, j]
                  b[i, j] = "↑"
          else c[i, j] = c[i, j − 1]
                  b[i, j] = "←"
  return c and b
```

$O(m \cdot n)$

Example: Rod Cutting Problem

Input: A rod of length n.
(i, p_i) $i = 1, \ldots, n$.

Output: max revenue r_n for length n.

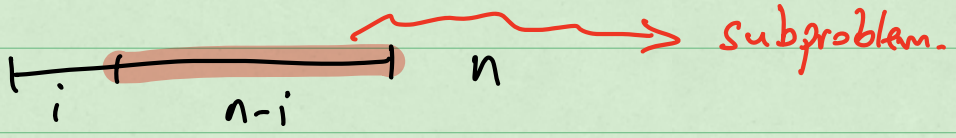
e.g. $n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	30

$r_1 = 1$	with decomposition	$1 = 1$ (no cut)
$r_2 = 5$	" "	$2 = 2$ (no cut)
$r_3 = 8$	" "	$3 = 3$ (no cut)
$r_4 = 10$	" "	$4 = 2+2$
$r_5 = 13$	" "	$5 = 3+2$
$r_6 = 17$	" "	$6 = 6$
$r_7 = 18$	" "	$7 = 6+1$
$r_8 = 22$	" "	$8 = 6+2$
$r_9 = 25$	" "	$9 = 6+3$
$r_{10} = 30$	" "	$10 = 10$ (no-cut)

Challenge: There are 2^{n-1} partitions of n.

- Given n, a partition has two parts



$$n = i + (n-i)$$

$$r_n = \max_i \{r_i + r_{n-i}\} \quad \text{for } i = 1, \dots n$$

$$\Downarrow$$

$$r_n = \max\{p_n, p_1 + r_{n-1}, p_2 + r_{n-2}, \dots, p_{n-1} + r_1\}$$

r_n --- $T(n)$

r_{n-i} --- $T(n-i)$ for $i = 1, \dots, n$

$$\Rightarrow T(n) = 1 + T(0) + T(1) + \dots + T(n-1)$$

$$= 1 + \sum_{j=0}^{n-1} T(n-j)$$

Claim: $T(n) = O(2^n)$ .... (Running Time for Recursive Soln.)

Proof: Use the substitution method.

Guess: $T(n) \leq c \cdot 2^n$

Verify:

I.H.: $T(k) \leq c \cdot 2^k \quad k < n$

$$T(n) = 1 + T(0) + T(1) + \cdots + T(n-1)$$

$$\leq 1 + c \cdot 2^0 + c \cdot 2 + c \cdot 2^2 + \cdots + c \cdot 2^{n-1}$$

$$= 1 + c\left(1 + 2 + \cdots + 2^{n-1}\right)$$

$$= 1 + c \cdot \frac{2^n - 1}{2 - 1}$$

$$= 1 + c(2^n - 1)$$

$$= c \cdot 2^n + 1 - c$$

$$\leq c \cdot 2^n \quad \text{if } (1 - c) \leq 0$$

$$\Rightarrow c \geq 1.$$

$$\Rightarrow T(n) = O(2^n).$$

```
CUT-ROD(p, n)
if n == 0
    return 0
q = -∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n - i))
return q
```

$O(2^n)$

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

$O(n^2)$

Lecture 15

Optimal Binary Search Trees (BST)

Input: A sequence of keys $K = [k_1, k_2, \ldots, k_n]$

... distinct, sorted

.. for each k_i, define p_i to be the prob. that k_i is searched.

Output: A binary search tree T that is constructed using keys in K and has the minimum expected search cost.

Note:

* For a key k_i, we define

$$\text{cost}(k_i) = 1 + \text{depth}(k_i)$$

* Let C be the search cost of a tree T

$$E[C] = \sum_{i=1}^{n} p_i \cdot cost(k_i)$$

$$= \sum_{i=1}^{n} p_i \cdot (1 + depth(k_i))$$

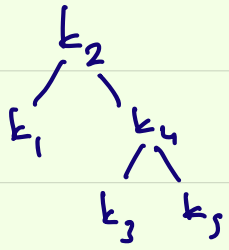
$$= \sum_{i=1}^{n} p_i \cdot 1 + \sum_{i=1}^{n} p_i \cdot depth(k_i)$$

$$= 1 + \sum_{i=1}^{n} p_i \cdot depth(k_i)$$

Example:

i	1	2	3	4	5
p_i	0.25	0.2	0.05	0.2	0.3

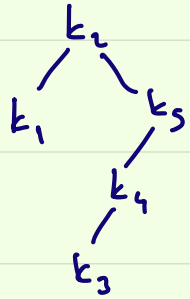
T_1:



i	$depth(k_i)$	$p_i \cdot depth(k_i)$
1	1	0.25
2	0	0
3	2	0.1
4	1	0.2
5	2	0.6
		+
		1.15

$\Rightarrow E[C] = 1 + 1.15 = 2.15$

T_2:



i	$depth(k_i)$	$p_i \cdot depth(k_i)$
1	1	0.25
2	0	0
3	3	0.15
4	2	0.40
5	1	+0.30
		1.10

$\Rightarrow E[C] = 1 + 1.10 = 2.10$

Remark:

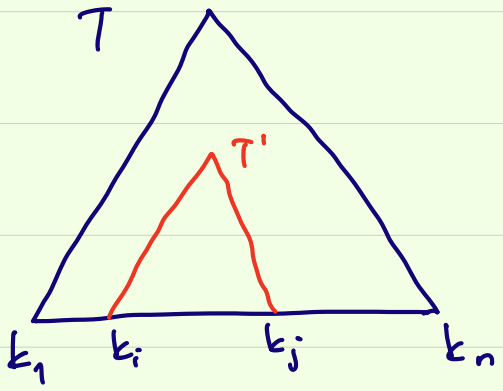
* # of BST for $k_1, \ldots, k_n$: $\Omega\left(\frac{4^n}{n\sqrt{n}}\right)$

* An optimal BST may not have the smallest depth.

* An optimal BST may not have the key with the highest probability at its root.

DP Solution:

* Consider the following BST T for keys $k_1, \ldots, k_n$:



i.e. • $k_i, \ldots, k_j$ is indeed a subproblem in this case.

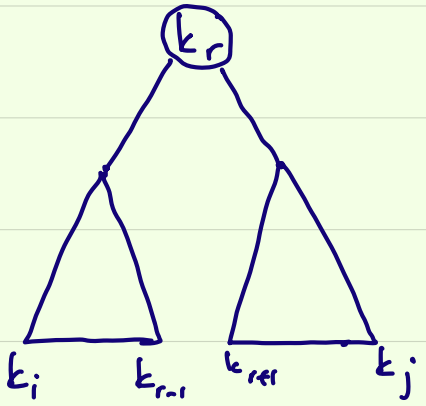
• T' is a subtree of T constructed using $k_i, \ldots, k_j$.

Claim: If T above is an optimal BST for $k_1, \ldots, k_n$, then T' above is an optimal BST for $k_i, \ldots, k_j$.

Proof: by cut-and-paste argument.

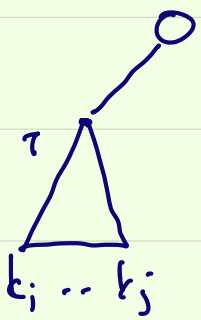
- Assume that T' is not optimal for $k_i, \ldots, k_j$
- Then, there must be another BST T'' which is optimal for $k_i \ldots k_j$.
- Now, we cut out T' and paste in T'' in T.
- But we assumed that T is optimal for $k_1, \ldots, k_n$. After replacing T' with T'', the total cost of T decreases obviously. Contradiction because the optimal solution can't get better.

* Given keys $k_i, \ldots, k_j$, one of them must be the root of their optimal BST. Let k_r (where $r \in [i,j]$) be the root.



* To find an optimal BST for $k_i, \ldots, k_j$
  - We must try all possible roots k_r
  - For each candidate k_r, we must construct optimal BSTs for $k_i, \ldots, k_{r-1}$ and $k_{r+1}, \ldots, k_j$

* Define $e[i,j]$ to be the expected search cost for an opt. BST of $k_i, \ldots, k_j$
  - if $i=j$, $e[i,j] = 0$
  - if $i<j$, we need to make the following observations first:



Note: If a subtree T becomes a subtree under a node, then depth of every node in T increases by 1.

Therefore, the expected search cost of T increases as well.

The extra cost: $w(i,j) = \sum_{l=i}^{j} p_l$

(Recall: $E[C] = 1 + \sum_{i=1}^{n} p_i \cdot depth(k_i)$)

After increasing the depth by 1, we get

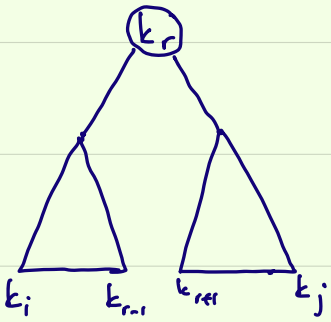
$$E[C] = 1 + \sum_{i=1}^{n} p_i \cdot (depth(k_i) + 1)$$

$$= 1 + \sum_{i=1}^{n} p_i \cdot depth(k_i) + \sum_{i=1}^{n} p_i$$

$$= 1 + \sum_{i=1}^{n} p_i \cdot depth(k_i) + w(1,n)$$

* Assume k_r is the root among $k_i, \cdots, k_j$

$$e[i,j] = p_r + e[i, r-1] + w(i, r-1) + e[r+1, j] + w(r+1, j)$$



Note: $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$

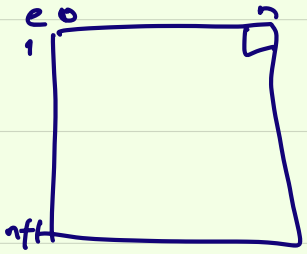
$\Rightarrow e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$

The above formula assumes that k_r is known.

To adjust this, we write

$$e[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \le r \le j} \{ e[i,r-1] + e[r+1,j] + w(i,j) \} & \text{if } i<j \end{cases}$$

```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
for i = 1 to n+1
    e[i, i-1] = 0
    w[i, i-1] = 0
for l = 1 to n
    for i = 1 to n-l+1
        j = i+l-1
        e[i, j] = ∞
        w[i, j] = w[i, j-1] + p_j
        for r = i to j
            t = e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```



Solution: $e[1,n]$

DP Solution Template

Input	
Output	
Subproblem space	Describe here how subproblems looks and what happens the first choice is made.
Recursive formula	Describe recursive formula from which a (bottom-up) DP algorithm will be extracted.
DP algorithm	Write here the pseudocode of the DP algorithm derived from the recursive formula.

Lecture 16

Fundamentals of Dynamic Programming

1) Phases of DP approach

See DP Soln. Template

1: Characterize the structure of optimal solution

2: Recursively define an optimal solution

3: Compute the optimal solution in a bottom-up or top-down fashion.

Ch 15 in textbook

4: Construct an optimal solution using the information gained from the implementation of recursive solution.

2) A DP solution requires a choice in the input and that choice leads to one or more subproblems

e.g. MCMP $A_i \dots A_j$

- 2 subproblems
- $j-i$ choices

Exercise: Optimal BST, rod-cutting?

3) We need to try all the choices, and solve all the subproblems after each choice. and compare their solutions to find an optimal solution.

4) Set of subproblems

e.g. MCMP ... $\{\langle A_i \dots A_j \rangle \mid 1 \le i, j \le n\}$

Opt. BST ... $\{\langle k_i \dots k_j \rangle \mid 1 \le i, j \le n\}$

Let us instead take $k_1, \dots, k_j$ for opt. BST. Then, there must be a key k_r, s.t. $1 \le r \le j$, and k_r is the root of opt. BST for $k_1, \dots, k_j$. After k_r is selected we get two subproblems:

$k_1, \dots, k_{r-1}$ and $k_{r+1}, \dots, k_j$

Yet, $k_{r+1}, \dots, k_j$ is not a valid subproblem since $r+1 \not\equiv 1$.

As a result, we expand the set of subproblems by making both indices vary.

5) DP is not applicable to every problem!

A given problem should exhibit "optimal substructure"

Example: Consider shortest and longest path problems.

Shortest Path (SP)

Input: An unweighted $G = (V, E)$

Output: A path $u \rightsquigarrow v$ with the least number of edges for any pair of nodes (u, v)

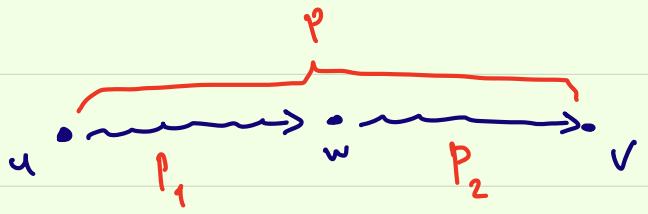
Longest Path (LP)

Input: An unweighted $G = (V, E)$

Output: Longest path between any pair of nodes.

(note: cycles are not allowed)

Remark: SP has optimal substructure.



$|p| = |p_1| + |p_2|$

Claim: Suppose p is a shortest path from u to v. Then p_1 is a shortest path from u to w.

Proof: by cut-and-paste argument.

- Suppose p_1 is not a shortest path from u to w.
- Then, there exists a path p_1' which is a shortest path from u to w.
- We replace p_1 with p_1' then.

Yet, this leads to a contradiction:

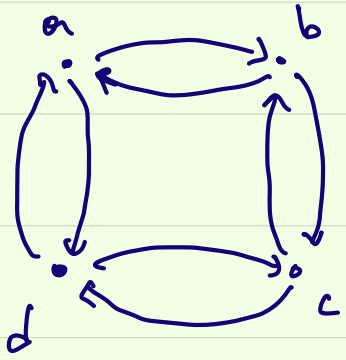
Note that p is assumed to be optimal and we get $|p_1'| + |p_2| < |p_1| + |p_2|$ which means $|p|$ decreases.

Exercise: Prove the same for p_2.

Remark: LP does not have optimal substructure.

We will show this by using a counterexample.

G:



- A longest path from a to c:

  $a \to b \to c$

- A longest path from a to b:

  $a \to d \to c \to b$

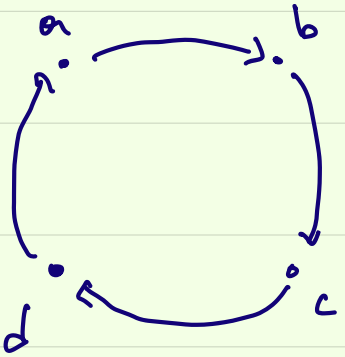
- A longest path from b to c:

  $b \to a \to d \to c$

Note that combining $a \to d \to c \to b$ and $b \to a \to d \to c$ will not give $a \to b \to c$!

For some instances, solving subproblems optimally can help.

e.g.



Exercise: Show that MCMP and LCS have opt. substructure.

6) Running time:

= # of subproblems × # of choices

e.g. mcmp:

- $O(n^2)$ subproblems
- $O(n)$ choices

} $O(n^3)$ time

LCS::

- $O(mn)$ subproblems
- ≤ 2 choices

} $O(m \cdot n)$ time

Opt. BST:

- $O(n^2)$ subproblems
- $O(n)$ choices

} $O(n^3)$ time.

Lecture 17

Greedy Method

- An algorithm design technique

- Similarities to DP
  - Optimal substructure (not enough always)
  - Applicable to optimization problem

- Main Idea: When making a choice, choose the option that looks best at the moment.

- A greedy may not always return an optimal solution.

- Main Challenge: To prove that the greedy algorithm always returns an optimal solution.

- Designing a greedy algorithm requires to find a greedy strategy.

See the handout "coin-change"

Activity Selection Problem

Input: $S = \{a_1, a_2, \ldots, a_n\}$ a_i ... activities

a_i ... s_i start time

f_i finish time ($f_i > s_i$)

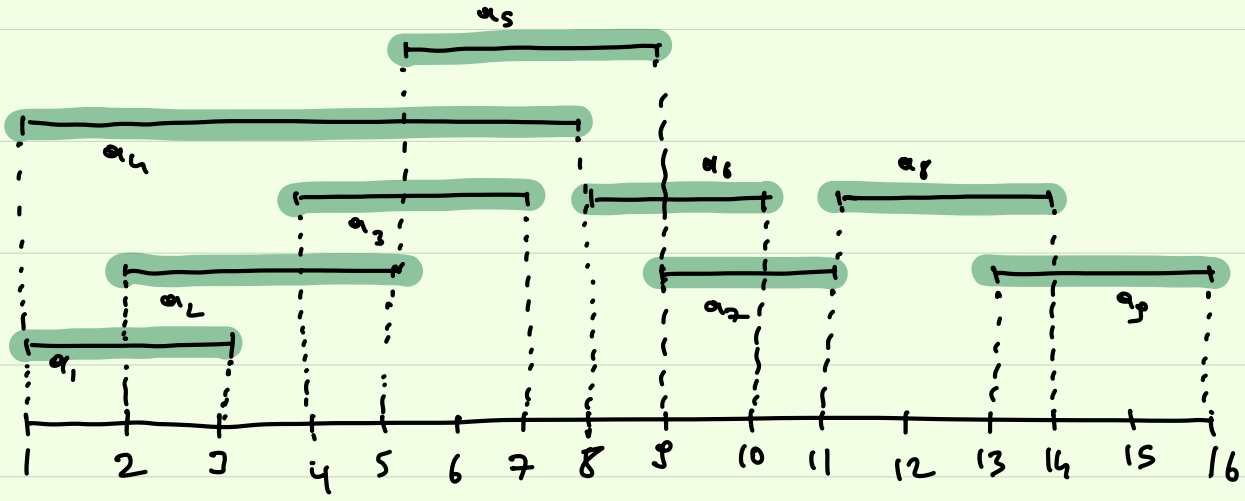
- Assume S is sorted w.r.t f_i.
- Activities should use a common source exclusively.

  $\Rightarrow$ Activities in the solution should not overlap.

Goal: Find a maximum size subset of nonoverlapping activities.

e.g.

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16



$A_1 = \{a_1, a_3, a_6, a_8\}$ $\quad A_3 = \{a_2, a_5, a_7, a_9\}$

$A_2 = \{a_1, a_3, a_6, a_9\}$ $\quad A_4 = \{a_2, a_5, a_7, a_8\}$

A DP Solution:

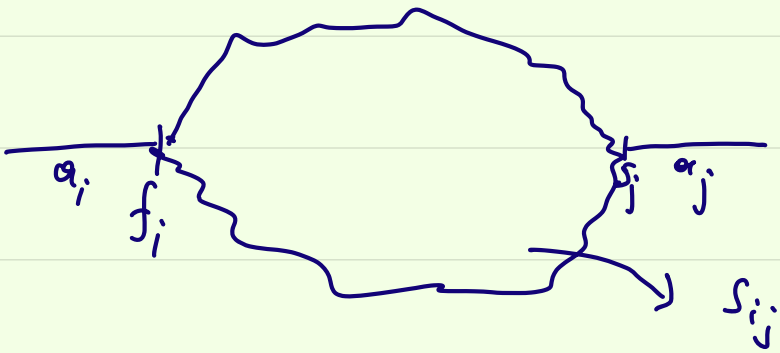
- $S = \{a_1, a_2, \ldots, a_n\}$

a_i (s_i, f_i)
a_j (s_j, f_j)

- $S_{ij} = \{a_k \in S \mid f_i \leq s_k < f_k \leq s_j\}$

Note: $S_{ij} \subseteq S$

e.g. $S_{28} = \{a_5, a_6, a_7\}$ $\quad S_{39} = \{a_6, a_7\}$

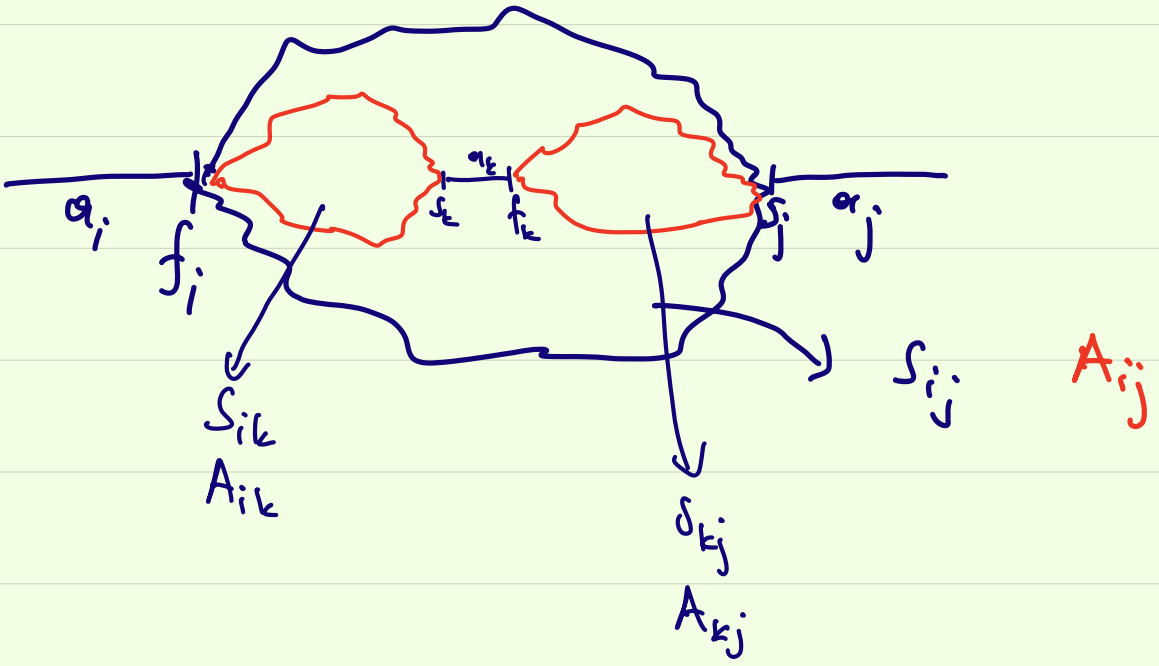


- Let A_{ij} be an optimal solution for S_{ij}.

$$A_{ij} \subseteq S_{ij} \subseteq S$$

- Let $a_k \in A_{ij}$.

Then, this choice leads to two subproblems



- Note that we get

$$A_{ij} = A_{ik} \cup A_{kj} \cup \{a_k\}$$

$$|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$$

- $A_{ik} = A_{ij} \cap S_{ik}$

- $A_{kj} = A_{ij} \cap S_{kj}$

Exercise: Prove these.

- Let $c[i,j]$: size of an opt. soln. of S_{ij}

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max_{i \le k < j} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

- This problem has optimal substructure:

Claim: An optimal solution A_{ij} for S_{ij} includes A_{ik}, i.e., $A_{ik} \subseteq A_{ij}$.

Proof: by cut-and-paste argument.

- Let A_{ik} is not in A_{ij}.

Then, there must be a better set A'_{ik} included in A_{ij}. Note: $|A'_{ik}| > |A_{ik}|$

- But we get that

$$|A'_{ik}| + |A_{kj}| + 1 > |A_{ik}| + |A_{kj}| + 1$$

which means $|A_{ij}|$ increases. Contradiction because A_{ij} is assumed to be optimal.

Coin Changing Problem

Input: A set of n coin denominations

e.g. $C = \{1, 5, 10, 25\}$ $n = 4$

Output: Pay an amount X using the minimum number of coins.

e.g. $C = \{1, 5, 10, 25\}$

$X = 9 = 5 + 1 + 1 + 1 + 1$ 5 coins

or $9 = 1 + 1 + \dots + 1$ 9 coins

Greedy strategy: Pick the highest coin value at each step.

Cashier Alg(X, C)

- Sort C
- Set $S = \emptyset$

while ($X > 0$) {

$k = \max c_k$ s.t. $c_k \leq X$

if no such k

break;

else $S = S \cup \{c_k\}$

$X = X - c_k$

$S = S \cup \{c_k\}$

}

return S.

Q: Does the alg. return an opt. soln. for every input?

$C = \{10, 25, 30\}$ Opt. Soln.: $X = 25 + 25$

$X = 50$

Alg. above: $X = 30 + 10 + 10$

Exercise:

1) Show that Coin Changing Problem has optimal substructure.
2) Design a DP algorithm for the Coin Changing Problem.

Lecture 18

Activity Selection Problem -- cont.

Greedy Solution:

Idea: Choose an activity that leaves the source available for the maximum possible number of activities.

Q: How to choose such an activity?

A: Pick an activity with earliest finish time.

"greedy choice" (strategy)

$\Rightarrow$ Since activities are sorted w.r.t finish times, we always start by picking a_1.

Also, note that each choice leads to only ONE SUBPROBLEM!

Assume a_1 is chosen. Then let us define S_1 as follows:

$$S_1 = \{ a_k \in S \mid s_k \geq f_1 \}$$

In general, we get $S_i = \{ a_k \in S \mid s_k \geq f_i \}$ after picking a_i.

Correctness

We need to prove two things:

(i) Prove that the problem exhibits optimal substructure. Already done for Act. Sel. problem. See Lec. Note 17.

(ii) Prove that the problem has greedy choice property. See below for Act. Sel. problem:

Claim: If S_k is nonempty and $a_m \in S_k$ has the earliest finish time in S_k. Then a_m is in an opt. solution of S_k.

Proof. Let A_k be an optimal solution for S_k and $a_j \in A_k$ has the earliest finish time in A_k.

Case 1: $a_j = a_m$. We are done.

Case 2: $a_j \neq a_m$. That is $a_m \notin A_k$.

Then we construct another opt. solution A_k' for S_k.

Let $A_k' = (A_k - \{a_j\}) \cup \{a_m\}$

i.e cut out a_j, paste in a_m

Note: 1) $f_m < f_j$

$\Rightarrow$ if a_j does not overlap with other activities in $(A_k - \{a_j\})$, then so does a_m!

$\Rightarrow$ A_k' is a subset of nonoverlapping activities

2) $|A_k'| = |A_k|$ by construction.

Therefore, A_k' is an opt. soln. for S_k and it includes a_m. Hence we are done.

Job Scheduling Problem:

Input: A set of jobs

job i ... t_i processing time

d_i deadline

latency $\ell_i = \max\{0, f_i - d_i\}$

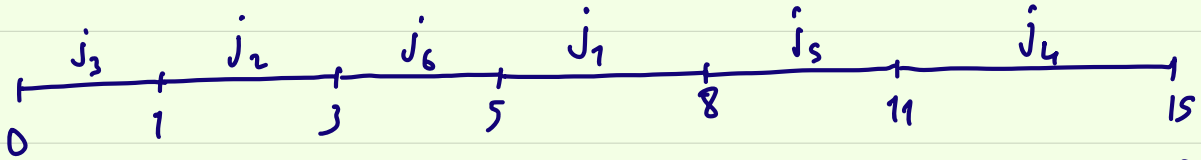
Define $L = \max \ell_i$.

Goal: Find (s_i, f_i) for each job i s.t. L is min.

Example:

i	1	2	3	4	5	6
f_i	3	2	1	4	3	2
d_i	6	8	9	9	14	15

Greedy strategy 1: Choose the smallest t_i first.



$\ell_1 = 2$ $\quad$ $\ell_4 = 6$

$\Rightarrow L = 6$ //

Greedy strategy 2: Choose the earliest d_i first.



$l_4 = 1$

$\Rightarrow L = 1$

Greedy Alg. for strategy 2:

- Sort w.r.t d_i
- $t = 0$
- for $i = 1$ to n

  Assign job i to $[t, t_i]$

  $s_i = t$

  $f_i = t + t_i$

  $t = t + t_i$
- return $[s_1, f_1], \ldots, [s_n, f_n]$

Greedy vs DP

- Both can apply. For instance, the act. selection problem. But DP can take longer:

Act. Sel. Problem:

- S_{ij} ... DP, two subproblems
- S_k .. Greedy, one subproblem

- Greedy appr. can be applied directly. No need to apply DP first.

- Greedy Choice Property: It holds if the globally optimal solution is obtained by locally optimal solutions.

- DP: Solve the subproblems and then pick the best

  Greedy: First choose, then solve the subproblems

- DP: bottom-up (in general)
  Greedy: top-down

- Not every problem with optimal substructure has the greedy choice property.

Example:

0-1 Knapsack Problem: (KP)

Input: A collection of n items and $W \in \mathbb{N}$.

item i v_i ... value

w_i - ... weight.

Goal: Find a max. value subset of items with total weight $\leq W$.

(One cannot pick a fraction of an item.)

Fractional Knapsack Problem: (KP)

Same 0-1 KP but one can now pick a fraction of an item.

- Both problems have optimal substructure But only fractional KP has the greedy choice property. A greedy algorithm for fractional KP chooses items w.r.t highest ratio $\frac{v_i}{w_i}$. This strategy does not work for 0-1 KP, though.

i	1	2	3
v_i	60	100	120
w_i	10	20	30
$\frac{v_i}{w_i}$	6	5	4

$W = 50$

Greed soln: 1, 2 Total value: $60 + 100 = 160$

Total weight: $10 + 20 = 30$

Optimal soln: 2, 3 Total value: $100 + 120 = 220$

" weight: $20 + 30 = 50$

Note: A soln. for fractional KP:

1, 2 and $\frac{2}{3}$ of 3 value: $60 + 100 + 80 = 240$

weight: 50

Lecture 19

Graph Algorithms

A graph $G = (V, E)$

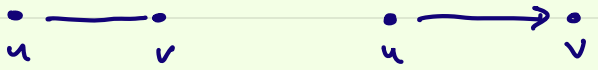
V ... set of nodes / vertices

E ... set of edges

$E \subseteq V \times V$

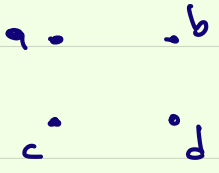
(u, v) .. an edge between u and v — undirected

an edge from u to v — directed



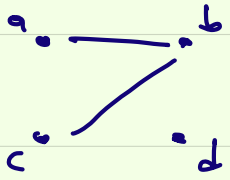
e.g. $V = \{a, b, c, d\}$

G_1:

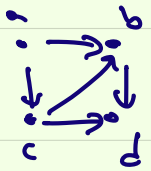


$E = \phi$

G_2:



G_3:



directed graph: not necessarily symmetric

undirected graph: symmetric

Review: degree / neighbors, path, cycle, loop

in-degree / out-degree, connectedness

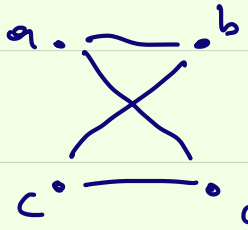
, strongly connectedness

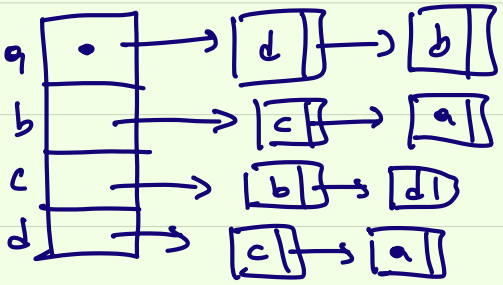
?
a : loop.

Adjacency Lists

Define an array for nodes and associate each entry with a list of neighbors of the node.

Example:



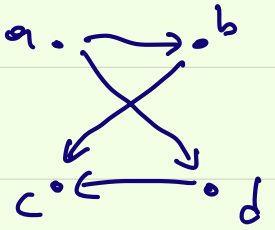


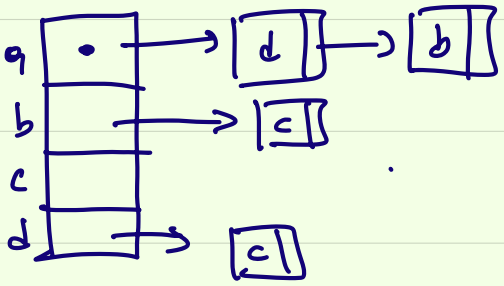
Space: $|V| + 2|E| = O(|V| + |E|)$

To check if $(u, v) \in E$ takes $O(|V|)$

or $O(\text{degree}(u))$ or $O(\text{degree}(v))$

Example:



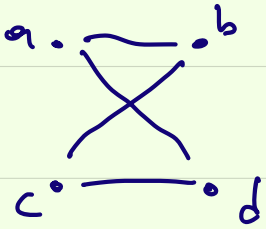


Adjacency Matrix

Define a matrix $A = (a_{ij})$ s.t.

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{owise.} \end{cases}$$

Example:



	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

4×4

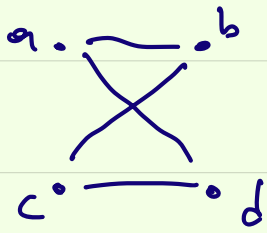
Space: $O(|V|^2)$

Time to check if $(i,j) \in E$: $O(1)$

List of edges:

Represent a graph as a list of its edges

Example:



$\{(a,b), (b,c), (c,d), (a,d)\}$

Breadth-First Search (BFS)

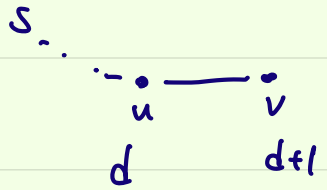
Input: $G = (V, E)$ and a source node s

Output: $v.d$ for every node $v \in V$.

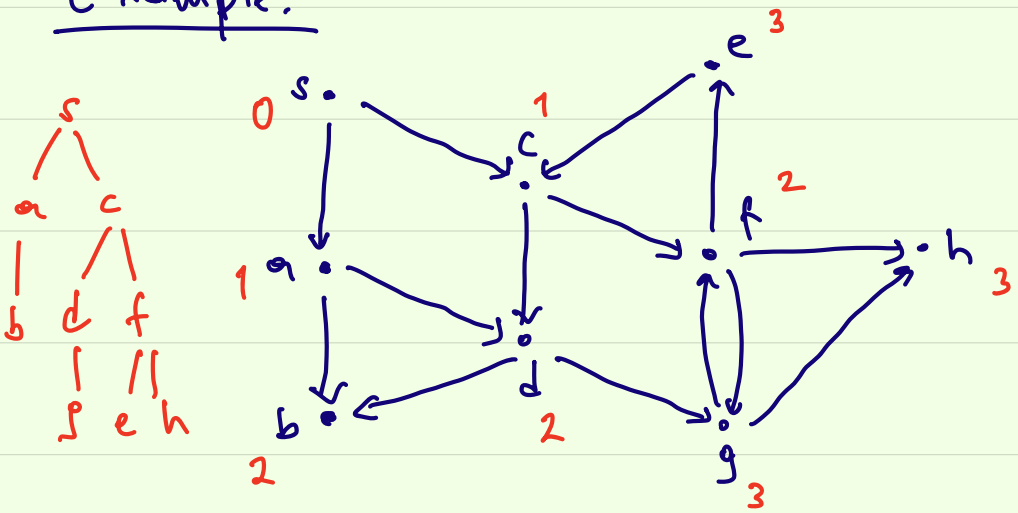
$v.d$: distance of v from s

↳ the number of nodes on a minimum length path from s to v.

```
BFS(V, E, s)
for each u ∈ V − {s}
    u.d = ∞
s.d = 0
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.d == ∞
            v.d = u.d + 1
            ENQUEUE(Q, v)
```



Example:



Q

- s
- c, a
- a, f, d
- f, d, b
- d, b, e, h, g
- b, e, h, g
- e, h, g
- h, g
- g
-

Depth-First Search (DFS)

Input: $G=(V,E)$

Output: $(v.d, v.\pi)$ for each $v \in V$

$v.d$: discovery time of v

$v.\pi$: finish time of v

Three colors are used:

White: Undiscovered

Gray: Discovered but not fully explored

Black: Discovered and fully explored

(i.e. every node reachable from it has been finished)

```
DFS(G)
  for each u ∈ G.V
      u.color = WHITE
  time = 0
  for each u ∈ G.V
      if u.color == WHITE
          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
  time = time + 1
  u.d = time
  u.color = GRAY
  for each v ∈ G.Adj[u]
      if v.color == WHITE
          DFS-VISIT(v)
  u.color = BLACK
  time = time + 1
  u.f = time
```

Edit Distance

Input: $X = [x_1, x_2, \ldots, x_m]$ $\quad |X| = m$

$Y = [y_1, y_2, \ldots, y_n]$ $\quad |Y| = n$

Output: Min # of edits to convert X to Y.
(edit distance)

e.g. X = car $\quad Y$ = fair

car $\longrightarrow$ cair $\longrightarrow$ air $\longrightarrow$ fair

DP Solution:

Let $X_i = [x_1 \ldots x_i]$ $\quad$ i-th prefix of X

$Y_j = [y_1 \ldots y_j]$ $\quad$ j-th prefix of Y.

Let $d(i,j)$: Edit distance between X_i and Y_j.

Goal: Find $d(m,n)$ $\quad \sim X_m, Y_n$
(" ")
$X \quad Y$

X_i ... what happens to the last element x_i
when calculating $d(i,j)$

a) x_i is deleted. Then $d(i,j) = d(i-1, j) + 1$

e.g. tank $\longrightarrow$ tan

b) x_i remains but is not the last element any more because a new one is inserted.

e.g. swan $\longrightarrow$ swank

$\Longrightarrow d(i,j) = d(i, j+1) + 1$

c) $x_i = y_j$. Then x_i remains as rightmost char in X_i.

$$\Longrightarrow d(i,j) = d(i-1, j-1) + 2 \cdot 1_{\{x_i \neq y_j\}}$$

where $1_{\{x_i \neq y_j\}} = \begin{cases} 1 & \text{if } x_i \neq y_j \\ 0 & \text{if } x_i = y_j \end{cases}$

d) If $|Y_j| = 0$, then $d(i,j) = d(i,0) = i$.

e) If $|X_i| = 0$, then $d(i,j) = d(0,j) = j$.

$$d(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \max\left\{ d(i,j-1)+1,\ d(j,i-1)+1,\ d(i-1,j-1) + 2 \cdot 1_{\{x_i \neq y_j\}} \right\} \end{cases}$$

Coin Changing:

Input: a set C coins, $|C| = m$, n

Output: min # of coins to pay n cents.

DP Solution:

- $D(n)$: min # of coins to pay n cents.
- From C, pick a coin with value ℓ to use.

  $\Rightarrow$ the rest to pay $= n - \ell$

  $\Rightarrow$ $D(n) = D(n-\ell) + 1$ Q: which $\ell \in C$ gives min?

  (Note: Making a choice leads to one subproblem.)

- If $n = 0$, then $D(n) = D(0) = 0$.

- $$D(n) = \begin{cases} 0 & \text{if } n = 0 \\ \min\limits_{\substack{\ell \in C \\ \text{s.t.} \\ (n-\ell) \geq 0}} \{ D(n-\ell) + 1 \} & \text{for } n \geq 1. \end{cases}$$

Alg.

$D = [0, \infty, \ldots, \infty]$ $\quad |D| = n+1$

```
for k=1 to n {
  for l ∈ C {
      if (l ≤ k) {
          if ( D(k-l) +1 < D(k) )
              D(k) = D(k-l) +1
      }
  }
}
Return D[n].
```

Subset-Sum

Input: A set $A = \{a_1, \dots, a_n\}$, $a_i \in \mathbb{Z}^+$, $k \in \mathbb{Z}$

Output: True if $\exists S \subseteq A$ s.t. S adds up to k.

False owise.

e.g. $A = \{1, 2, 3\}$ $k = 4$ $\Rightarrow$ True $1+3=4$.

$A = \{2, 3, 4\}$ $k = 8$ $\Rightarrow$ False.

$A = \{2, 3, 4\}$ $k = 0$ $\Rightarrow$ True $\emptyset \subseteq A$.

$A = \{1, 2, 3\}$ $k = -5$ $\Rightarrow$ False

$A = \emptyset$ $k = 5$ $\Rightarrow$ False

DP Solution:

subproblem: A_i

$A[i, \dots, n]$ ... subset including i-th thru n-th elements

$M[i, k]$ = True if some subset of $A[i, \dots, n]$

$\Rightarrow$ Want: $M[1, k]$.

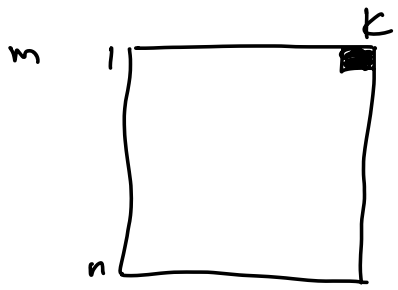
- If $k = 0$, then $M[i, k]$ = true since $\emptyset \subseteq A_i$
- If $k < 0$ or $k \neq 0$ but $A = \emptyset$, then $M[i, k]$ = false
- If $k < A[i]$, discard $A[i]$ and try $M[i+1, k]$

- Generally, let $x \in A$. There is a subset X of A that sums to k iff one of the following is true:
  - $x \in X$ and X adds up to k.
  - $x \notin X$ and X adds up to k.

  $\Rightarrow$ $M[i+1, k] \vee M[i+1, k - A[i]]$

$$M[i,k] = \begin{cases} \text{True} & \text{if } k = 0 \\ \text{false} & \text{if } i > n \\ M[i+1,k] & \text{if } k < A[i] \\ M[i+1,k] \vee M[i+1, k-A[i]] & \text{otherwise.} \end{cases}$$

Return. $M[1,k]$.



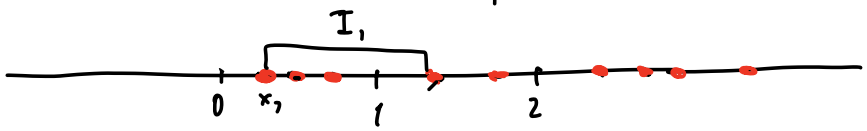
Greedy Examples.

Unit Intervals

Input: $X = \{x_1, \ldots, x_n\}$ of n points on the real line.

Goal: Smallest set of unit-length intervals that contains all points in X.

e.g.



Algorithm:

- Sort X and re-label the points so that $x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_n$
- $S = \emptyset$ ... solution set, i.e. the intervals.
- For each $x_k \in X$ (while $X \neq \emptyset$)
  - $S = S \cup \{I_k\}$ (where $I_k = [x_k, x_{k+1}]$
  - $X = X \setminus \{x_i \mid x_i \in I_k\}$
- Return S.

Gives an optimal solution.

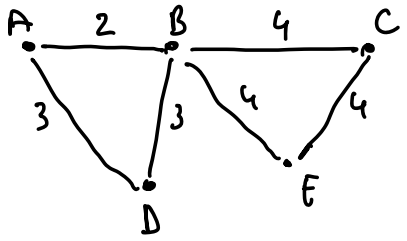
Traveling Salesman.

Input: A set of cities and distances.

Goal: A salesman should visit all cities by covering least possible distance.

Assume: there is an initial city A.

Greedy choice: Visit the next nearest city.



A $\xrightarrow{2}$ B $\xrightarrow{3}$ D $\xrightarrow{3}$ B $\xrightarrow{4}$ C $\xrightarrow{4}$ E

16

Not optimal!

Optimal: A $\xrightarrow{3}$ D $\xrightarrow{3}$ B $\xrightarrow{4}$ C $\xrightarrow{4}$ E

14 //

Alg. does not always return an opt. solution.

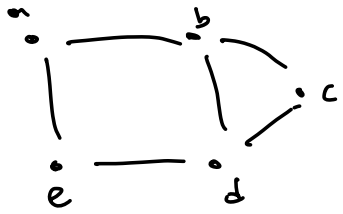
Maximum Independent Set.

Input: $G = (V, E)$

Output: An independent set of nodes whose size is maximum.

(A subset $C \subseteq V$ is independent if no two nodes $u, v \in C$ are adjacent.)

e.g.



$C = \{a, c\}$

Greedy choice: Select the node with least degree.

(1) Select a vertex v_1 with least degree and add it to the solution set C.

(2) Delete v_1 and its neighbors and the edges between them from the graph.

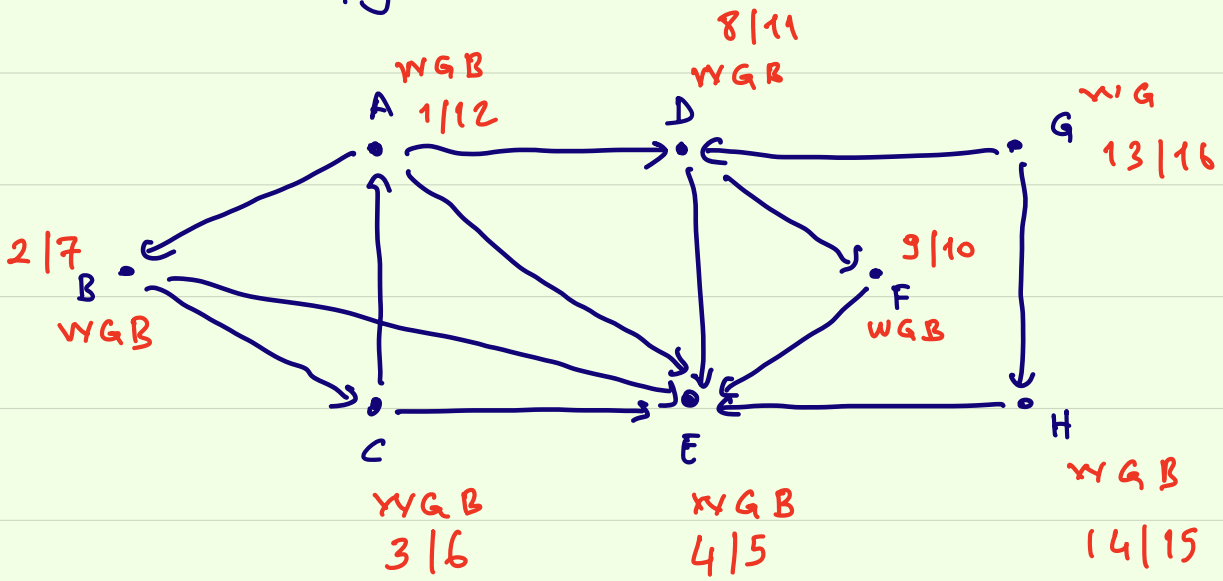
(3) Repeat (1) and (2) until no node remains.

(4) Return C.

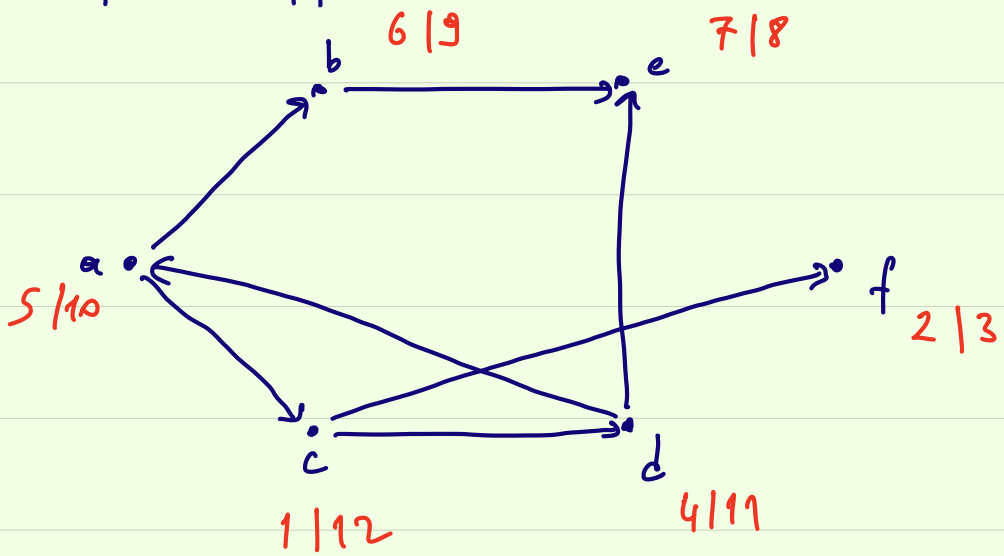
Lecture 20

DFS cont...

Example: Apply DFS



Example: Apply DFS



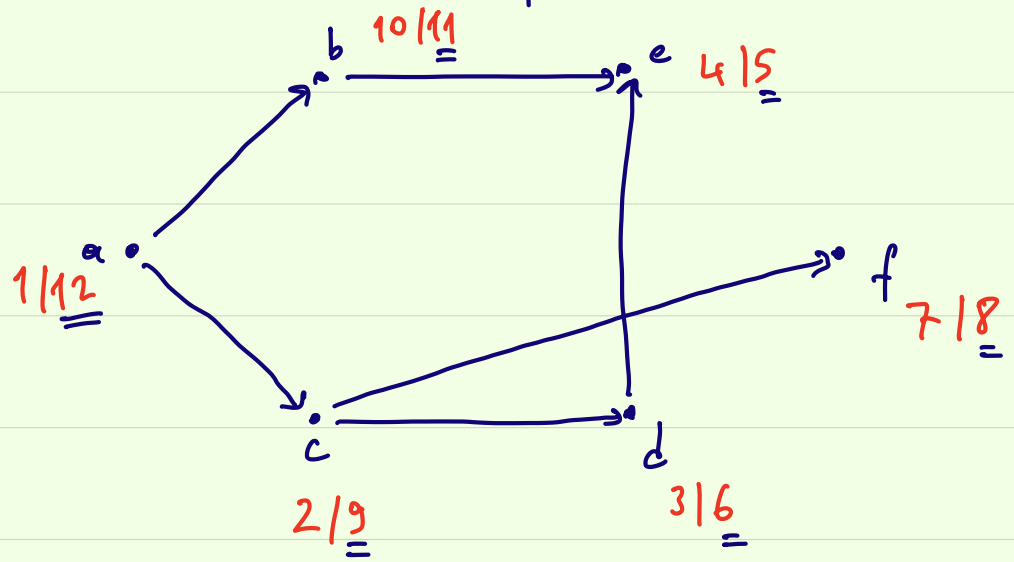
Topological Sort

Assume: The input is directed acyclic graph

Top. Sort (G)

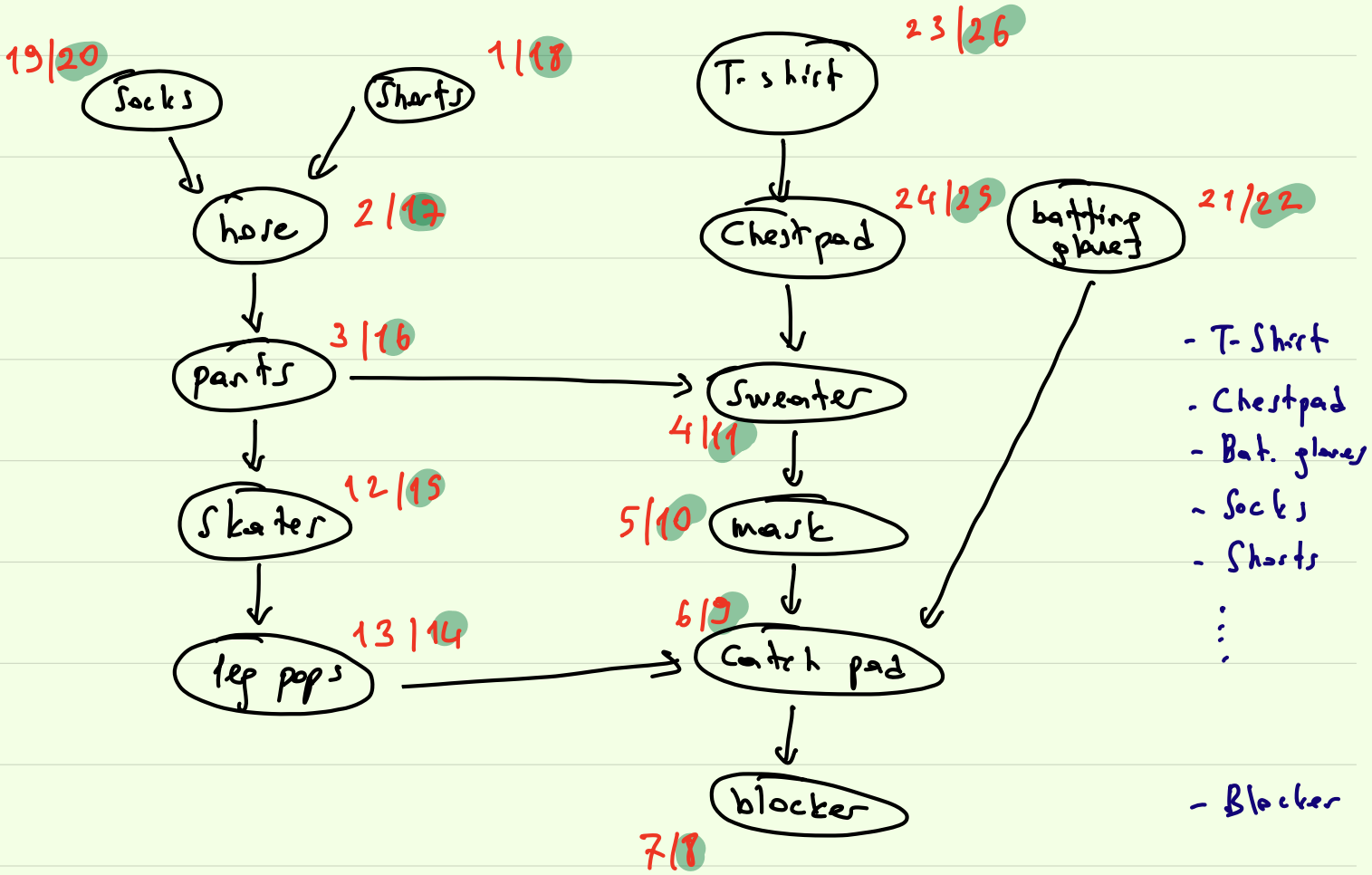
- DFS(G)
- Output the nodes in decreasing w.r.t. finish times.

Example: Apply Top. Sort



Output: a, b, c, f, d, e

Example:

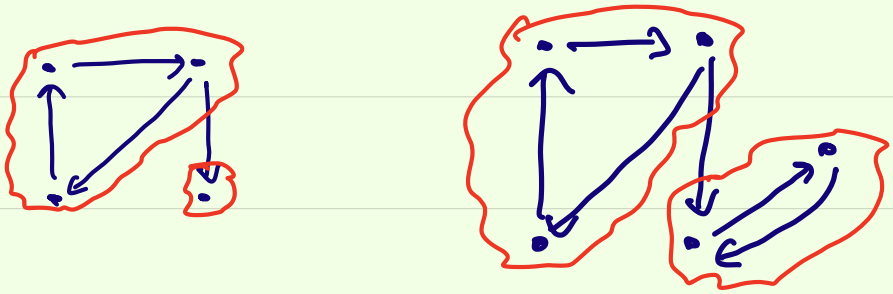


Strongly Connected Components (SCCs)

- A directed graph $G=(V,E)$ is "strongly connected" if for every pair of nodes $u, v \in V$ there is a (directed) path from u to v and vice versa.
- Given a directed graph $G=(V,E)$, a SCC of G is a maximal subgraph G' of G which is strongly connected.

i.e. for every node u, v in G', there is a directed path from u to v and vice versa.

Example:



- Given a directe graph $G = (V, E)$, we define $G^T = (V, E^T)$ where

$$E^T = \{(u,v) \mid (v,u) \in E\}.$$

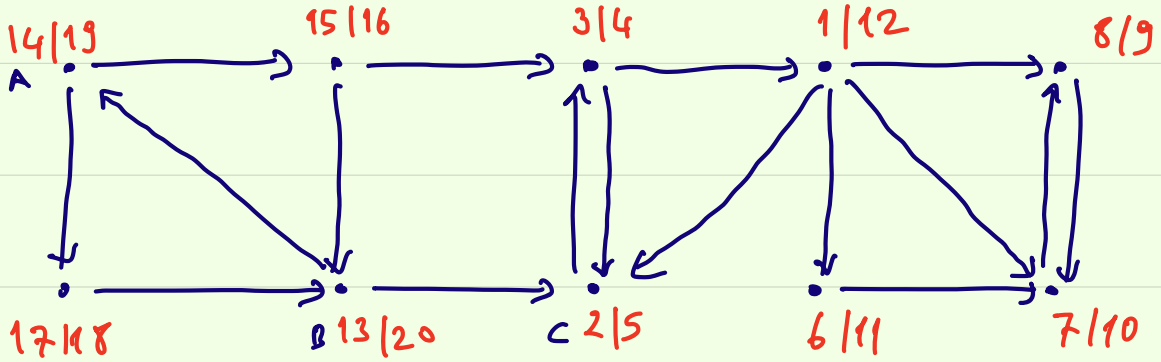
- The Algorithm:

SCC(G)

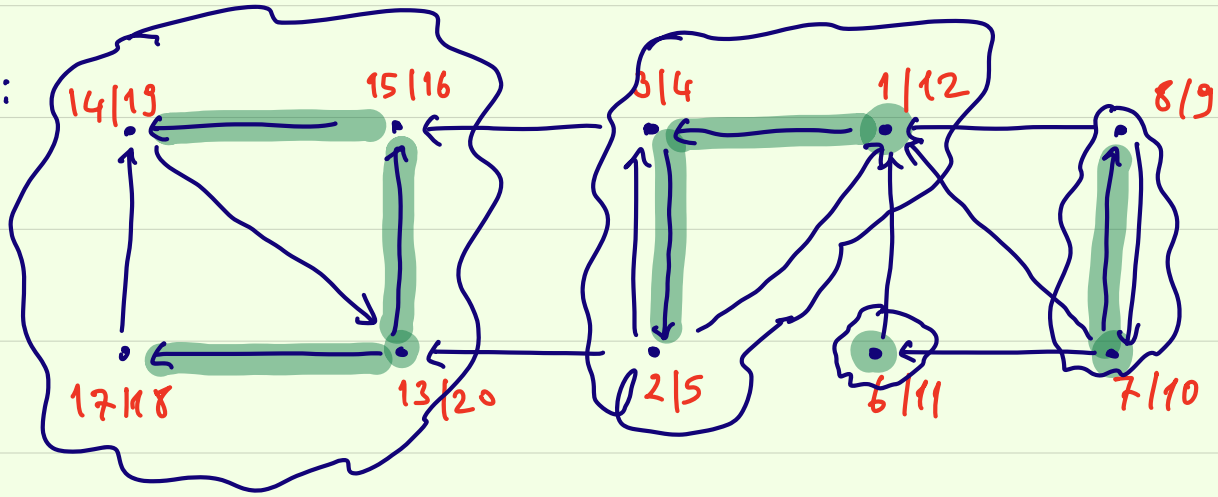
- DFS(G)
- Construct G^T
- DFS(G^T), by picking the node with highest finish time and visiting all the node that are reachable from it.

Example:

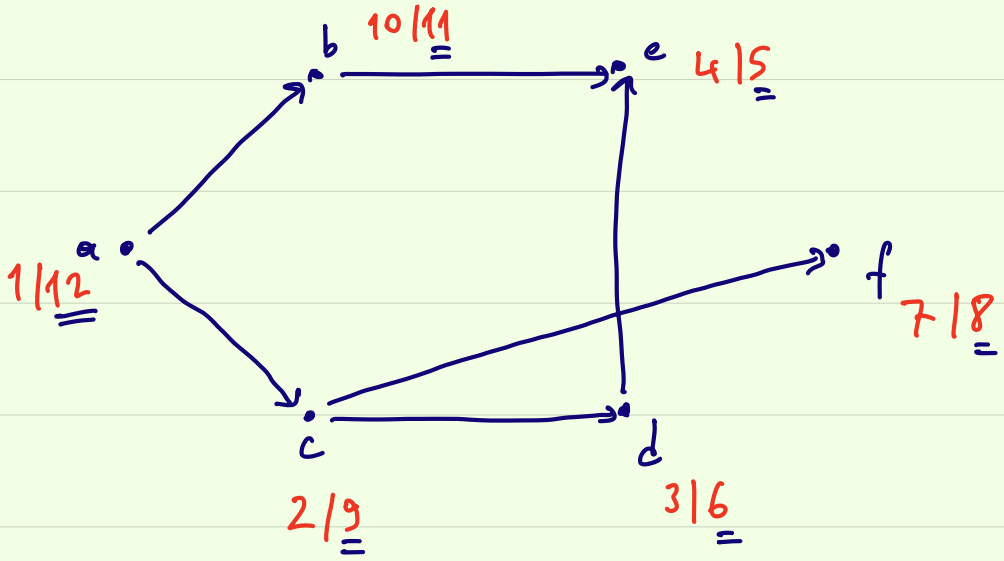
G:



G^T:



Exercise: Apply SCC Alg.



Lecture 21

Minimum Spanning Trees (MST)

Input: An undirected graph $G = (V, E)$ and a weight function $w: E \longrightarrow R$

Output: $T \subseteq E$ such that

- T connects all nodes in V. (i.e. T spans the graph G.)
- $w(T) = \sum_{e \in T} w(e)$ is minimum

- A spanning tree with minimum possible weight over all spanning trees is called a minimum spanning tree.

Example:



Some properties of MSTs:

- An MST connects all nodes
- An MST has no cycles.
- An MST has $|V|-1$ edges.
- For a given graph $G=(V,E)$, there may be more than one MST.

Generic MST Alg:

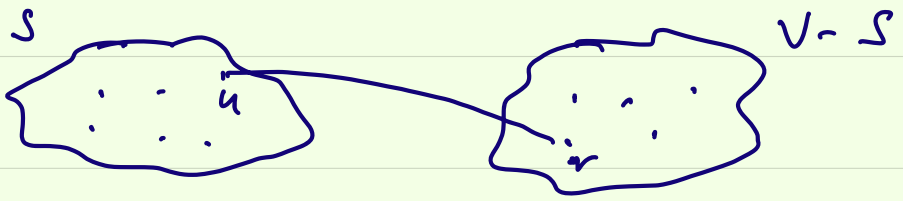
- Builds a set of edges A s.t. $A \subseteq E$.
  - Initially $A = \emptyset$
  - Loop variant: A is a subset of an MST.
  - Adds only <u>safe</u> edges to A.
    (An edge (u,v) is safe iff $A \cup \{(u,v)\}$ is a subset of some MST.)

Generic_MST(G, w)

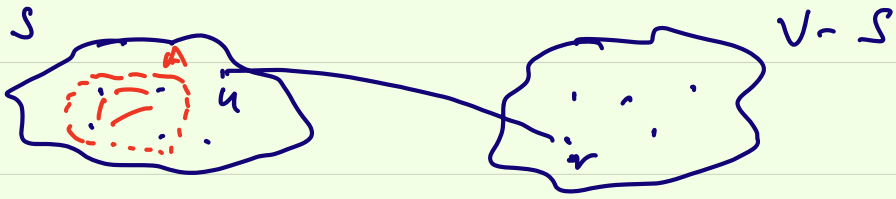
- $A = \emptyset$
- while (A is not a spanning tree)
  - Find a safe edge (u,v) for A
  - $A = A \cup \{(u,v)\}$
- return A

- Let $S \subseteq V$ and $A \subseteq E$.
- A cut $(S, V-S)$ is a partition of V into two disjoint subsets. (Note: $S \cap (V-S) = \emptyset$)
- An edge (u,v) crosses a cut $(S, V-S)$ if its endpoints u and v are on different sides of the cut.

e.g.

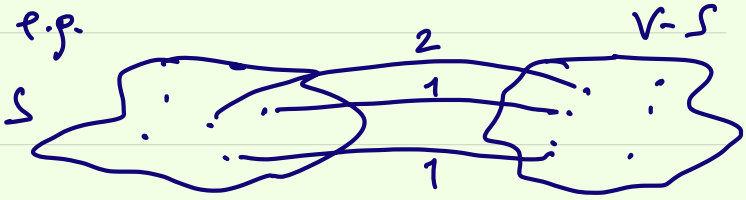


- A cut $(S, V-S)$ <u>respects</u> a set A of edges, $A \subseteq E$, if no edge in A crosses it.



- A <u>light edge</u> (u,v) is a crossing with minimum weight.

e.g.

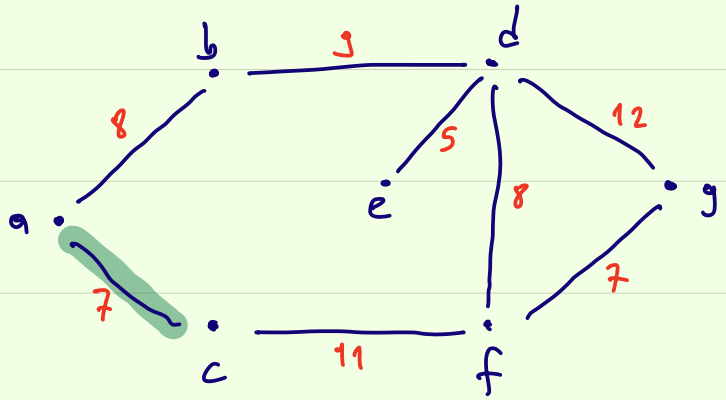


Theorem: Let

- A be a subset of some MST,
- $(S, V-S)$ be a cut that respects A
- (u,v) be a light edge crossing $(S, V-S)$.

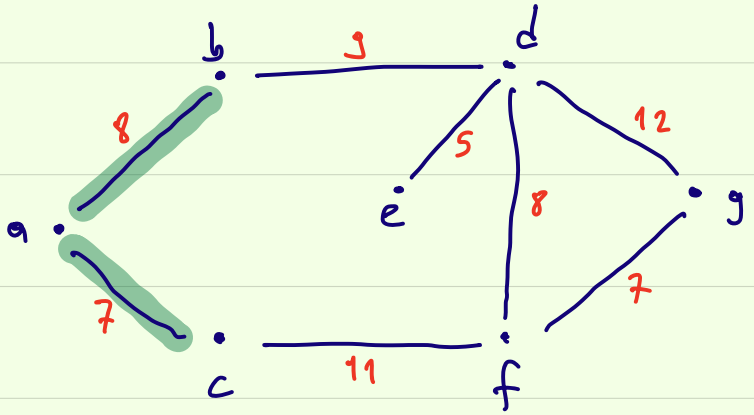
Then (u,v) is safe for A.

Example:



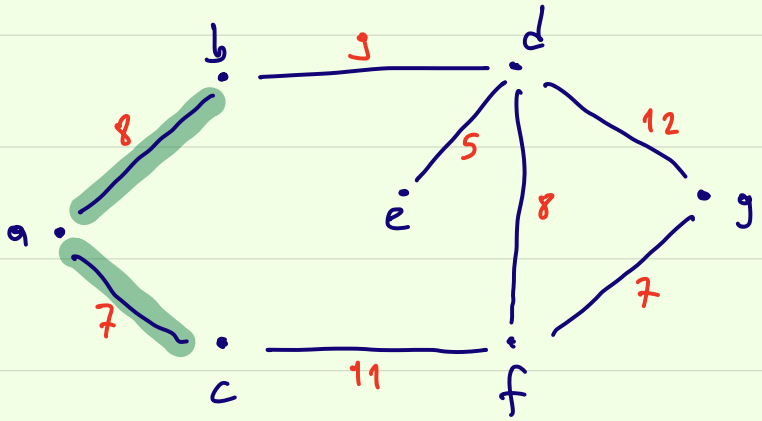
$A = \emptyset$

$S = \{a\}$



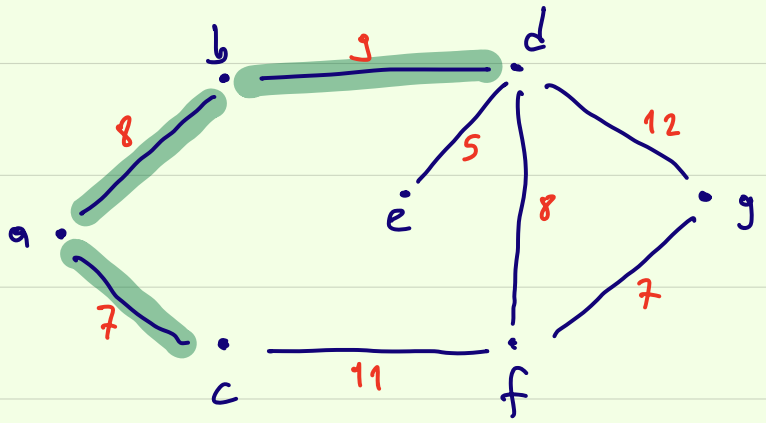
$A = \{(a,c)\}$

$S = \{a, c\}$



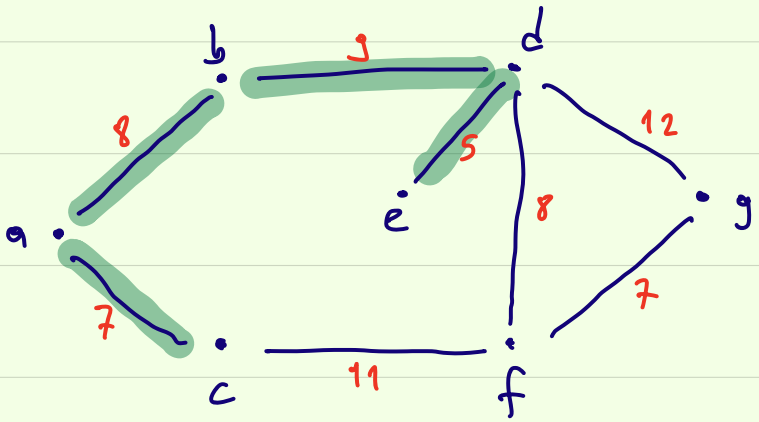
$A = A \cup \{(a,b)\}$

$S = \{a, c, b\}$



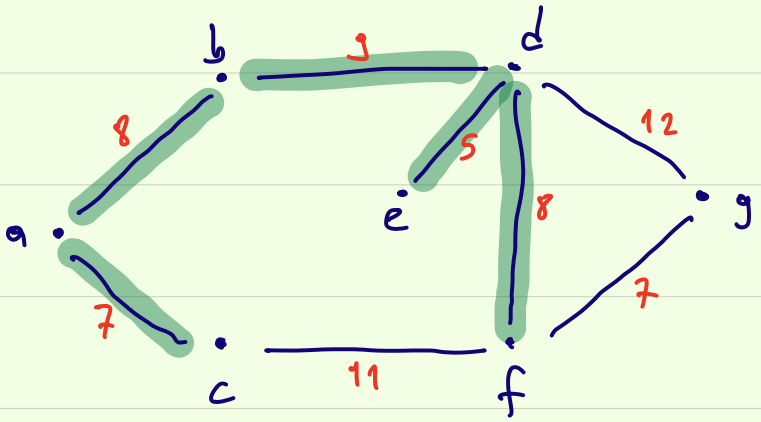
$A = A \cup \{(b,d)\}$

$S = \{a, c, b\}$



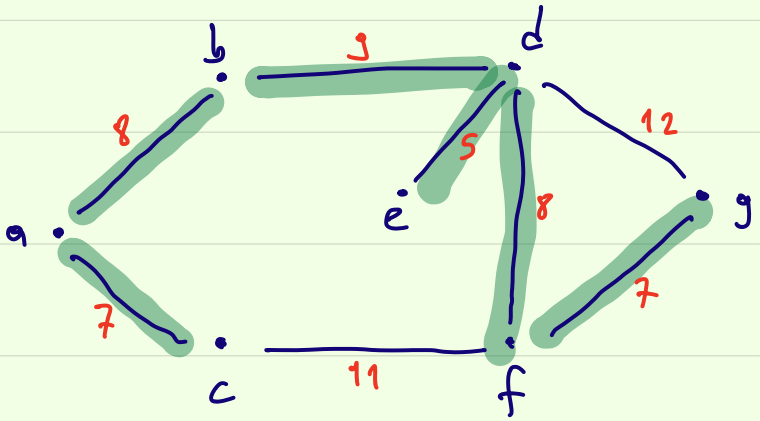
$A = A \cup \{(d,e)\}$

$S = \{a, c, b, d\}$



$A = A \cup \{(f,d)\}$

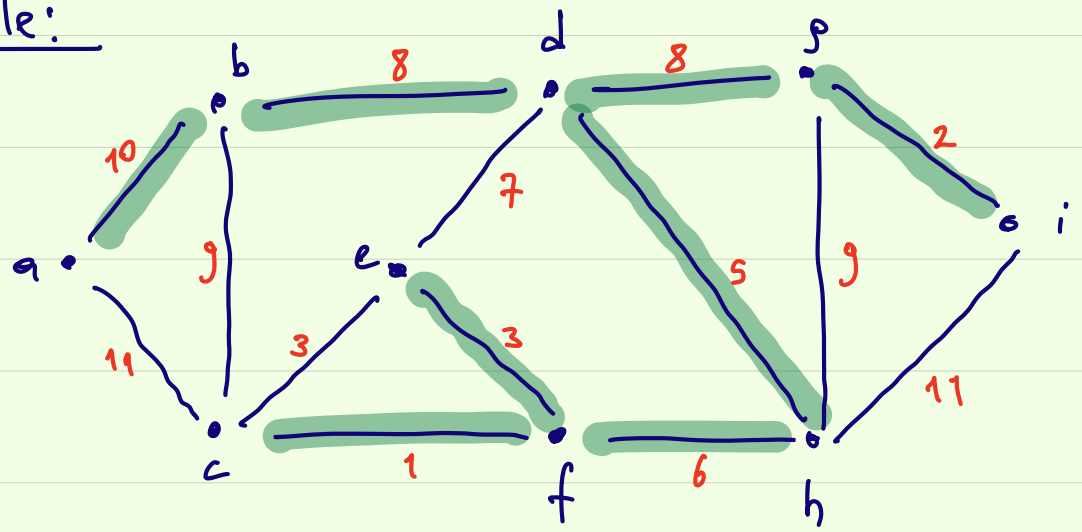
$S = \{a, c, b, d, e, f\}$



$A = A \cup \{(f,g)\}$

$S = \{a, c, b, d, e, f, g\}$

Example:



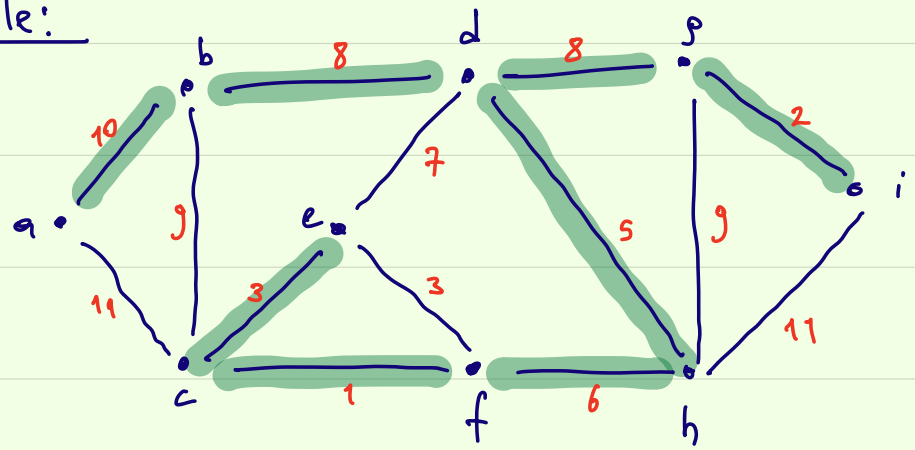
- $A = \emptyset$ $S = \{e\}$
- $A = A \cup \{(e,f)\}$ $S = \{e,f\}$
- $A = A \cup \{(c,f)\}$ $S = \{e,f,c\}$
- $A = A \cup \{(h,f)\}$ $S = \{e,f,c,h\}$
- $A = A \cup \{(d,h)\}$ $S = \{e,f,c,h,d\}$
- $A = A \cup \{(b,d)\}$ $S = \{e,f,c,h,d,b\}$
- $A = A \cup \{(d,g)\}$ $S = \{e,f,c,h,d,b,g\}$
- $A = A \cup \{(g,i)\}$ $S = \{e,f,c,h,d,b,g,i\}$
- $A = A \cup \{(a,b)\}$ $S = \{e,f,c,h,d,b,g,i,a\}$

Exercise:
start with
$S = \{a,b\}$
instead.

Kruskal's Algorithm:

- Sort all edges in a nondecreasing order w.r.t their weights.
- Add each edge as long as it does not cause a cycle.

Example:



(c, f)
(g, i)
(c, e)
(e, f)
(d, h)
(f, h)
(d, e)
(b, d)
(d, g)
(b, c)
(g, h)
(a, b)
(a, c)
(h, i)

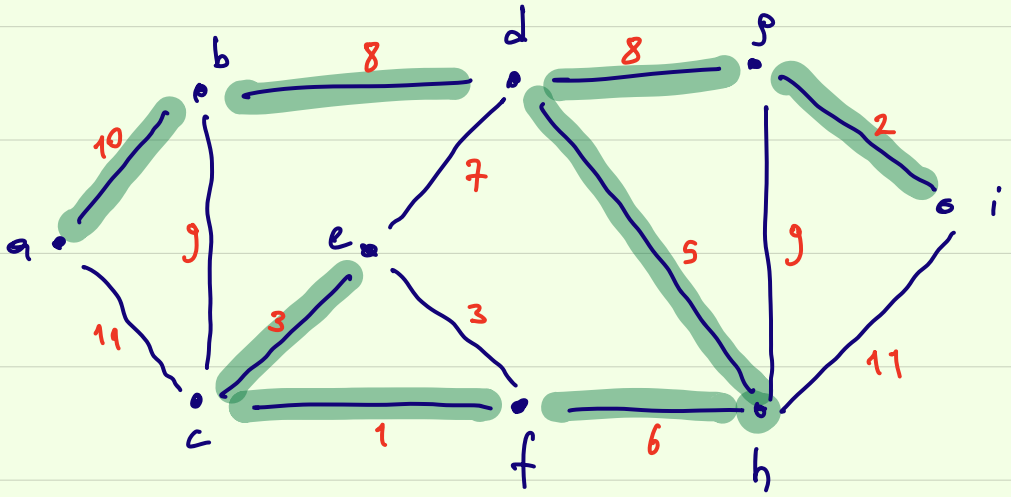
Lecture 22

Minimum Spanning Trees (MST) cont.

Prim's Algorithm

- Start with a node u and greedily grow an MST T.
- Add to T an edge with minimum weight s.t. the added edge has exactly one endpoint in T
- Repeat until $|T| = |V| - 1$.

Example:



Log	$\lvert T\rvert$
- Pick h	0
- Add (d,h)	1
- Add (f,h)	2
- Add (c,f)	3
- Add (c,e)	4
- Add (d,g)	5
- Add (g,i)	6
- Add (b,d)	7
- Add (a,b)	8

Exercise: A DP Solution for MST Problem?

Single Source Shortest Path Problem

Input: - A directed graph $G = (V, E)$

- A weight function $w: E \to \mathbb{R}$
- Source node $s \in V$

- Weight of a path $\langle v_0, v_1, \ldots, v_k \rangle$: $\sum_{i=1}^{k} w(v_{i-1}, v_i)$

- Weight of a shortest path from u to v:

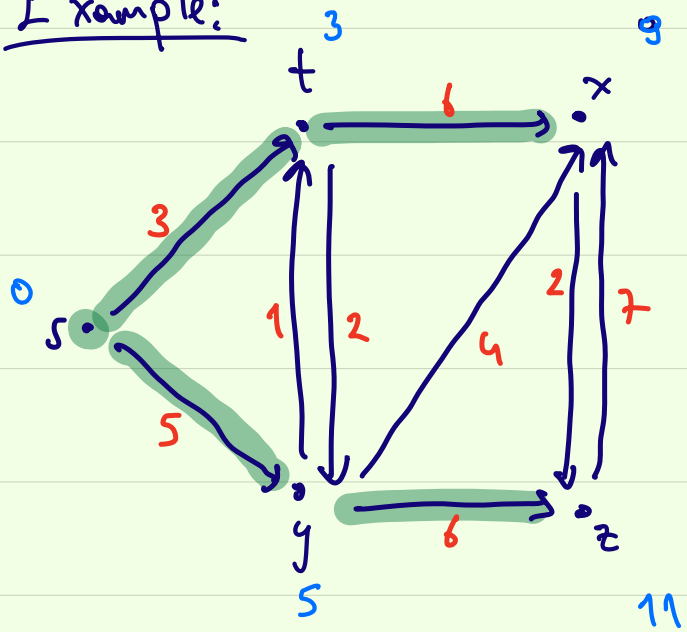
$$\delta(u, v) = \begin{cases} \min_p \{ w(p) \mid p \text{ is a path from } u \text{ to } v \} & \text{if } \exists\, p: u \rightsquigarrow v \\ \infty & \text{otherwise} \end{cases}$$

Output: For every node u,

- $u.d$: $\delta(s, u)$
- $u.\pi$: Predecessor of u on a shortest path from s to u.

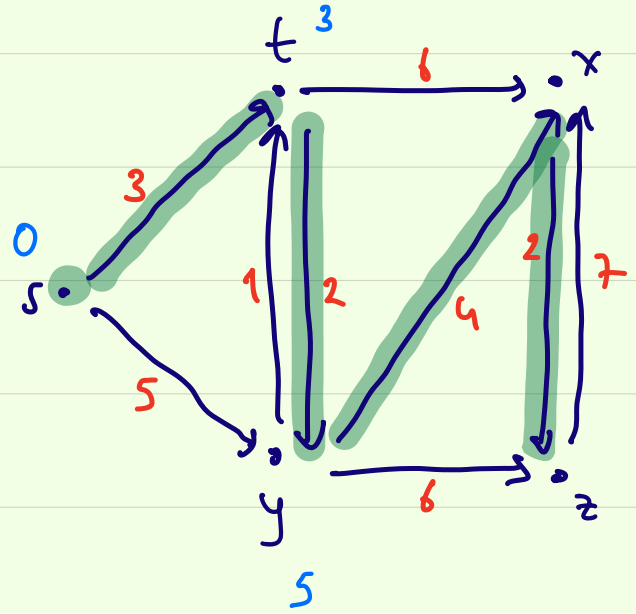
(note: $s.\pi = nil$)

Example:



v	s	t	x	y	z
v.d	0	3	9	5	11
v.π	nil	s	t	s	y



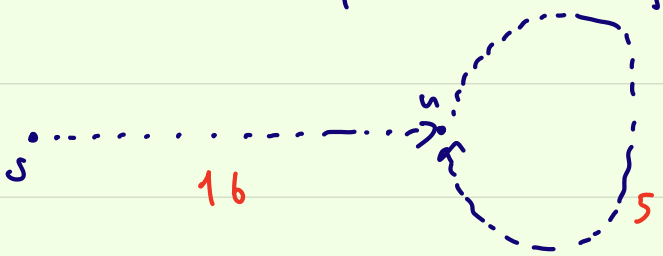


v	s	t	x	y	z
v.d	0	3	9	5	11
v.π	nil	s	y	t	x



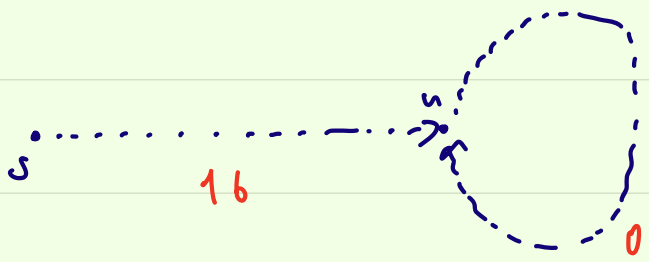
Remark: Shortest path don't contain cycles.

- Cycles with positive weight:



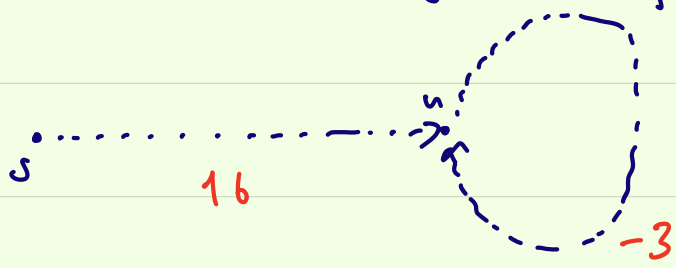
just ignore.

- Cycles with zero weight:



just ignore!

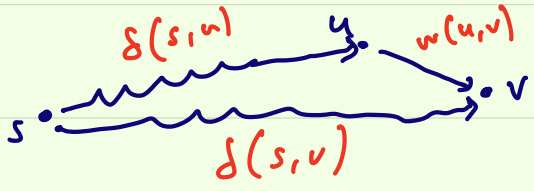
- Cycles with negative weight:



no solution!

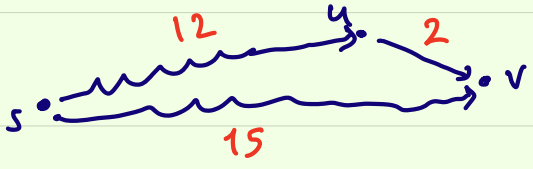
Edge Relaxation:

- Let $(u,v) \in E$
- Assume we know (current) $\delta(s,u)$ and $\delta(s,v)$

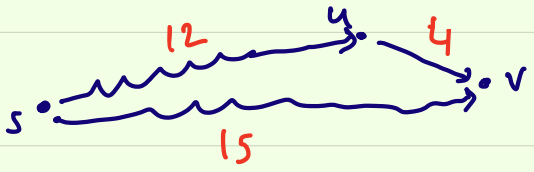


Question: Can we improve $\delta(s,v)$ if reach v by going thru u instead?

e.g.



YES!



NO!

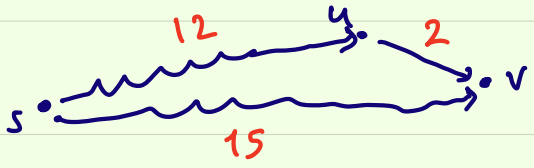
Relax (u, v, w)

if (v.d > u.d + w(u,v))

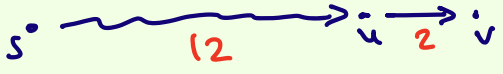
v.d = u.d + w(u,v)

v.π = u

e.g.



|| Relax (u,v)



Lecture 23

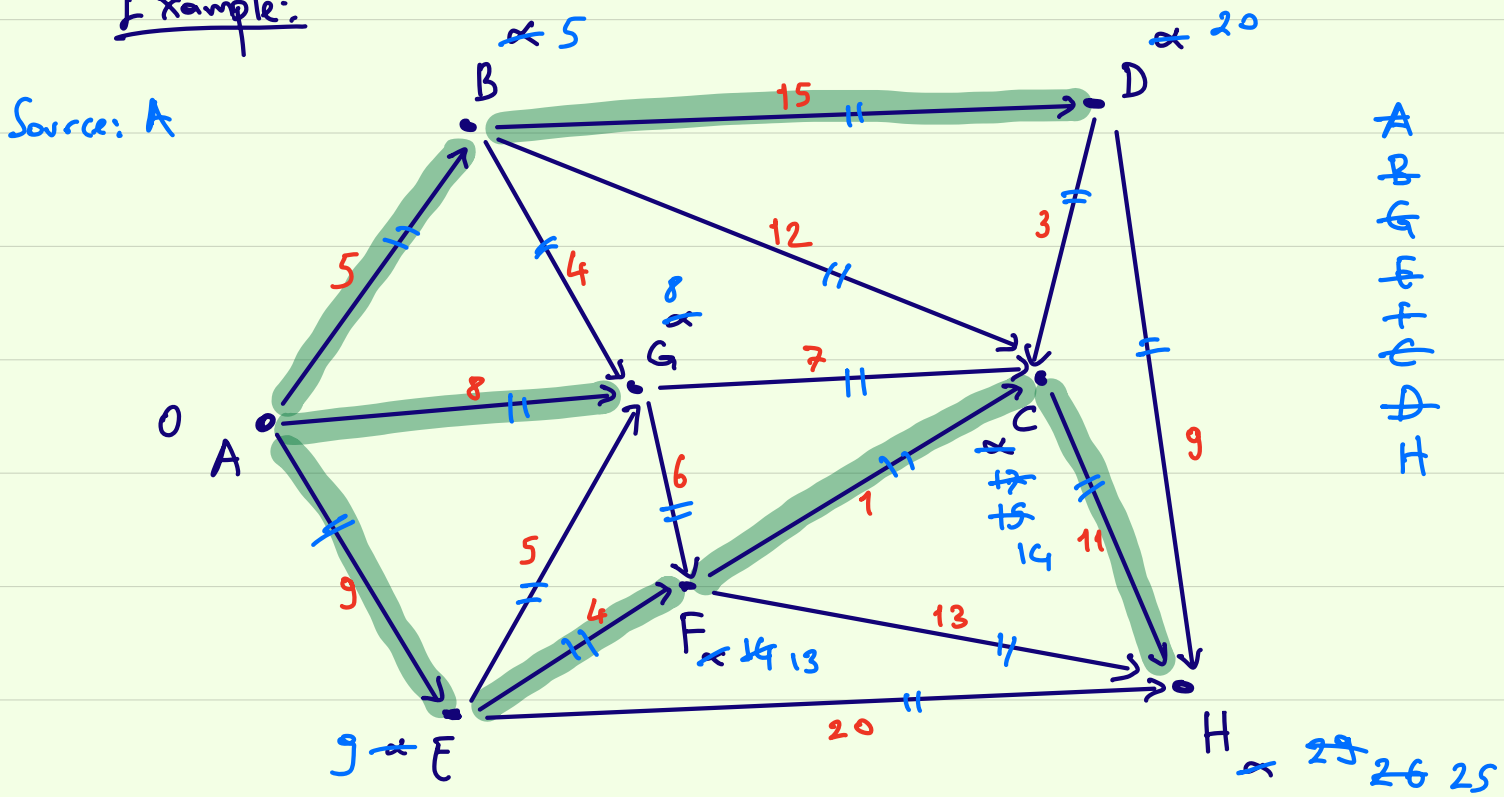
Dijkstra's Algorithm

- Consider the nodes in a nondecreasing order of distance from the source.
- Add a node and <u>relax</u> all edges going out of it.

Remark:

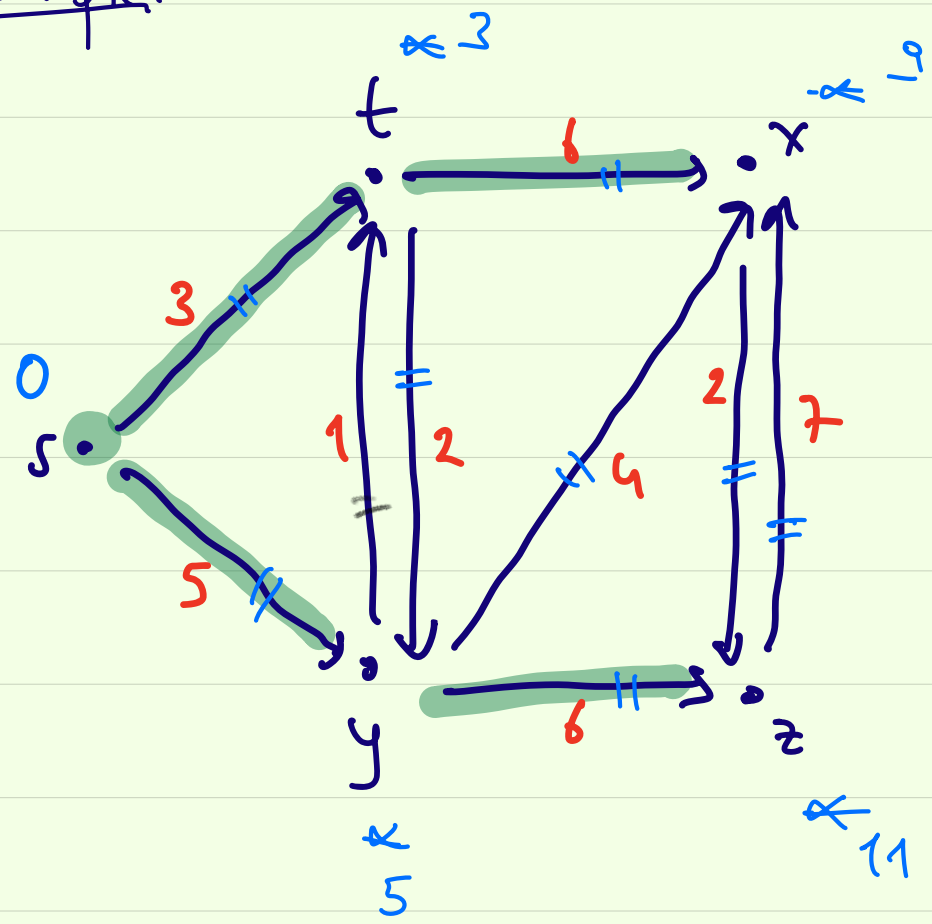
- It visits all nodes
- It relaxes all edges.
- Running time: $O(|V| + |E|)$

Example:



v	A	B	C	D	E	F	G	H
v.d	0	5	14	20	9	13	8	25
v.π	nil	A	F	B	A	E	A	C

Example:



Bellman-Ford Algorithm

- It allows negative weights. (Note: Dijkstra's Alg. does not.)

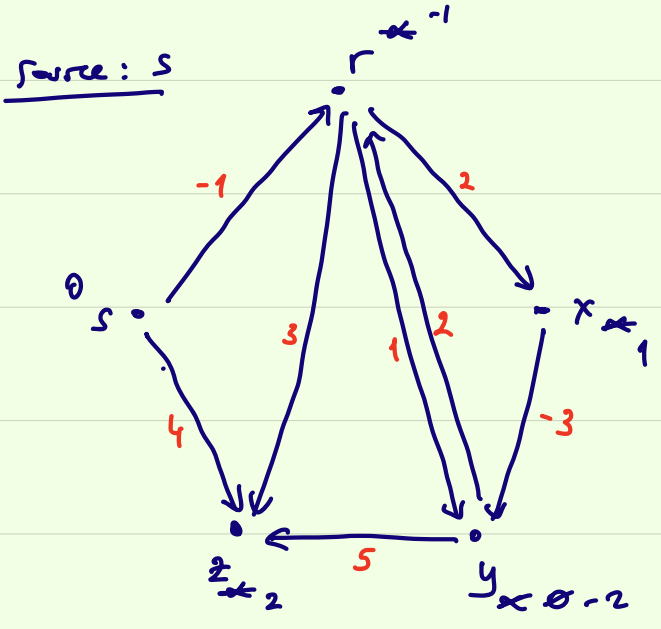
- It returns false if a negative weight cycle is reachable from the source.

- Idea:

  1) Pick an arrangement of edges
  2) Repeat the following $|V|$ times:

     Relax all edges in the order you picked.

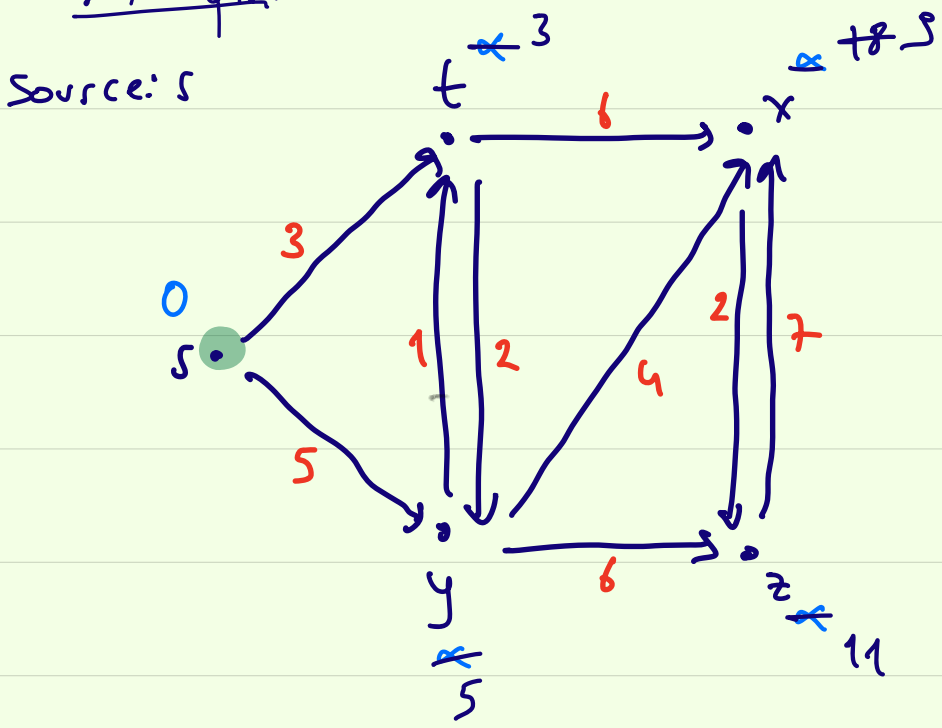
Example:



(s,r) ⌐ ⌐ ⌐
(y,z) ⌐ ⌐ ⌐
(r,z) ⌐ ⌐ ⌐
(r,y) ⌐ ⌐ ⌐
(y,r) ⌐ ⌐ ⌐
(s,z) ⌐ ⌐ ⌐
(x,y) ⌐ ⌐ ⌐
(r,x) ⌐ ⌐ ⌐

Note: Stopped after iter. 3 because no change was observed.

Example:



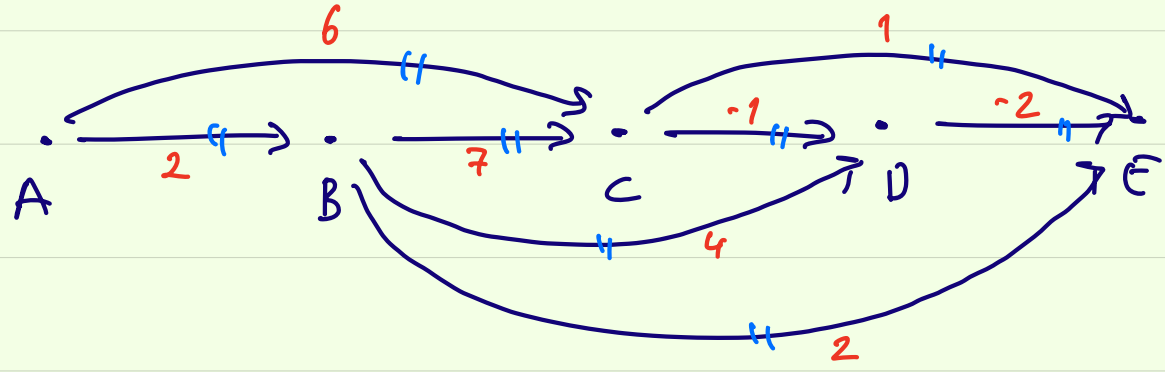
(y, x)	✓	✓	✓
(t, x)	✓	✓	✓
(s, t)	✓	✓	✓
(t, y)	✓	✓	✓
(y, t)	✓	✓	✓
(y, z)	✓	✓	✓
(x, z)	✓	✓	✓
(z, x)	✓	✓	✓
(s, y)	✓	✓	✓

Note: Stopped after iter. 3 because no change was observed.

DAG-Shortest Path Algorithm

It works if the input is a directed acyclic graph (dag). We can proceed by pick nodes w.r.t. their topological order.

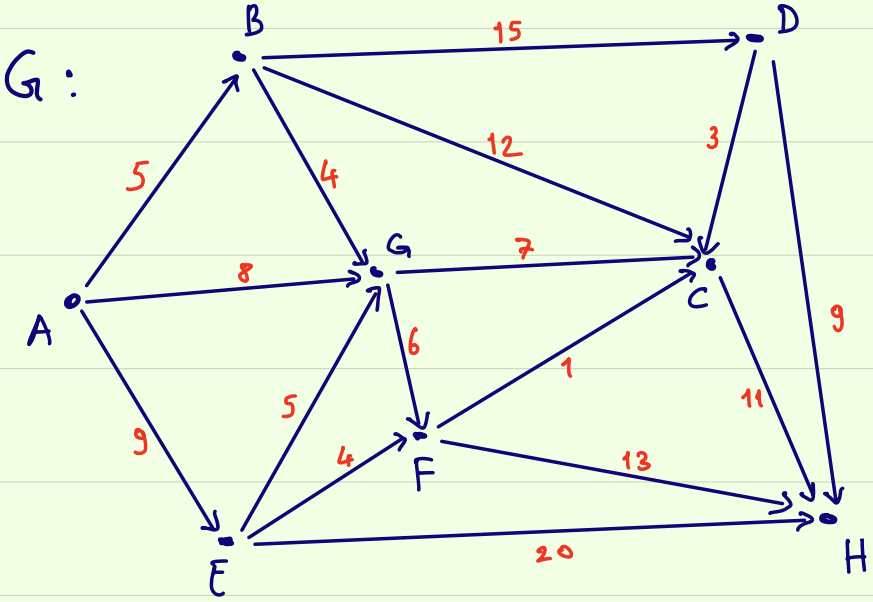
Example:



0 2 6 ~~6~~ 5 ~~4~~ ~~3~~ 1

Exercise:

G:



- Pick node A
- Sort G topologically
- Apply DAG-Sh. P. Alg.

Lecture 24

All-Pairs Shortest Path Problem

Input: - $G = (V, E)$ ... directed

- $w: E \to R$

- Nodes are labeled as $1, 2, \ldots, n$

Output: An $n \times n$ matrix $D = (d_{ij})$ ($i = 1, \ldots, n$, $j = 1, \ldots, n$) of shortest path distances s.t. $d_{ij} = \delta(i, j)$

Recall:

$$\delta(u,v) = \begin{cases} \min_p \{ w(p) \mid p \text{ is a path from } u \text{ to } v \} & \text{if } \exists\, p: u \rightsquigarrow v \\ \infty & \text{otherwise} \end{cases}$$

Remark:

- Could use B-F alg to solve this problem. Simply call it $|V|$ times. $O(|V|^2|E|)$
- Could use Dijkstra's Alg. $O(|V| \cdot |E| \cdot \lg |E|)$ if we use binary min. heap.

Goal: Solve in $O(|V|^3)$ w/out using fancy data structures.

Floyd-Warshall Algorithm.

- A bottom-up DP algorithm. $O(|V|^3)$ time.

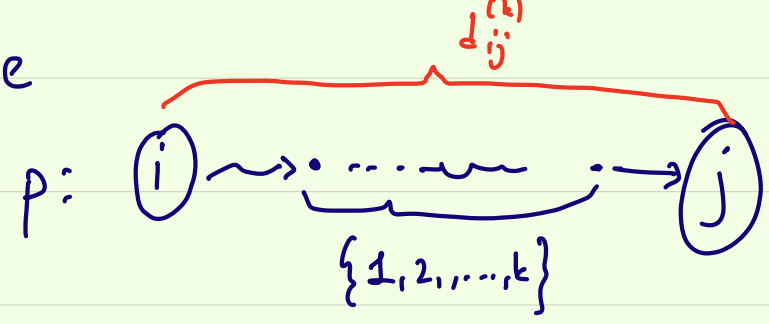
- Let $p = \langle v_1, v_2, \ldots, v_\ell \rangle$ be a path. $|p| = \ell - 1$

  v_1, v_ℓ : end nodes

  $v_2, v_3, \ldots, v_{\ell-1}$: "intermediate nodes"

- Let $d_{ij}^{(k)}$ = the weight of the shortest path distance from i to j with all intermediate nodes from the set $\{1, 2, \ldots, k\}$

  i.e



Consider the following cases:

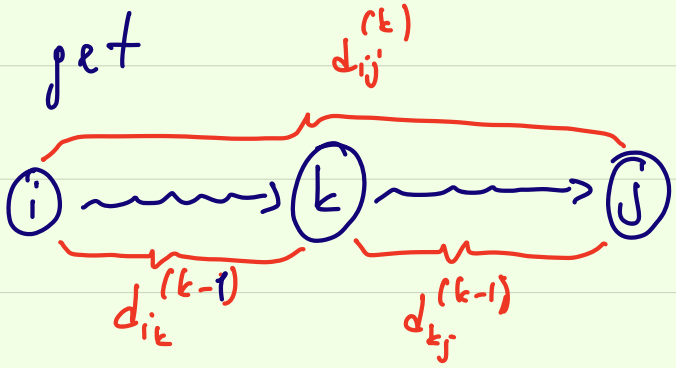
Case 1: Node k is not an intermediate node.

$\Rightarrow$ the set of all int. nodes is $\{1, \ldots, k-1\}$

$\Rightarrow$ $d_{ij}^{(k)} = d_{ij}^{(k-1)}$

Case 2: Node k is an intermediate node.

Then we get



$$\Rightarrow \quad d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

Case 3: There is no int. node on the path p from i to j.

i.e. $i \xrightarrow{w(i,j)} j$ i.e. $\{1, \dots, 0\}$

$$d_{ij}^{(k)} = \begin{cases} w(i,j) & \text{if } k = 0 \\ \min\left\{ d_{ij}^{(k-1)}, \; d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\} & \text{otherwise.} \end{cases}$$

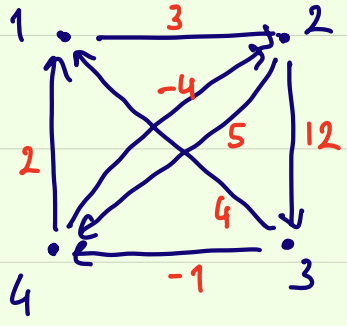
Note:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & \text{if } (i,j) \in E \\ \alpha & \text{if } (i,j) \notin E \end{cases}$$

The DP Algorithm

- Start with $D^{(0)} = W = (w_{ij})$
- Compute $D^{(1)}, D^{(2)}, \dots, D^{(n)}$ where $D^{(i)}$ is obtained from $D^{(i-1)}$ by using the formula of $d_{ij}^{(k)}$.
- Return $D^{(n)}$.

Example:



$$D^{(0)} = W = \begin{bmatrix} 0 & 3 & \infty & \infty \\ \infty & 0 & 12 & 5 \\ 4 & \infty & 0 & -1 \\ 2 & -4 & \infty & 0 \end{bmatrix}$$

$\{1\}$

$$D^{(1)} = \begin{bmatrix} 0 & 3 & \infty & \infty \\ \infty & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & \infty & 0 \end{bmatrix}$$

$$d_{32}^{(1)} = \min\{d_{32}^{(0)}, d_{31}^{(0)} + d_{12}^{(0)}\}$$

$$= \min\{\infty, 4+3\}$$

$$= 7$$

$$d_{42}^{(1)} = \min\{d_{42}^{(0)}, d_{41}^{(0)} + d_{12}^{(0)}\}$$

$$= \min\{-4, 2+3\} = -4$$

$\{1,2\}$

$$D^{(2)} = \begin{bmatrix} 0 & 3 & 15 & 8 \\ \infty & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

$$d_{13}^{(2)} = \min\left\{ d_{13}^{(1)} , d_{12}^{(1)} + d_{23}^{(1)} \right\}$$

$$= \min\{ \infty , 3 + 12 \}$$

$$= 15 //$$

Justify $d_{14}^{(2)} = 8$?

$d_{43}^{(2)} = 8$?

$\{1,2,3\}$

$$D^{(3)} = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 16 & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

$$d_{21}^{(3)} = \min\left\{ d_{21}^{(2)} , d_{23}^{(2)} + d_{31}^{(2)} \right\}$$

$$= \min\{ \infty , 12 + 4 \}$$

$$= 16$$

$\{1,2,3,4\}$

$$D^{(4)} = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 7 & 0 & 12 & 5 \\ 1 & -5 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

$$d_{31}^{(4)} = \min\left\{ d_{31}^{(3)} , d_{34}^{(3)} + d_{41}^{(3)} \right\}$$

$$= \min\{ 4 , -1 + 2 \}$$

$$= 1$$

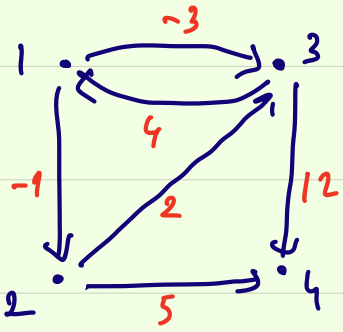
$$d_{32}^{(4)} = \min\left\{ d_{32}^{(3)} , d_{34}^{(3)} + d_{42}^{(3)} \right\}$$

$$= \min\{ 7 , (-1) + (-4) \}$$

$$= -5 //$$

Justify $d_{21}^{(4)} = 7$?

Exercise:



Lecture 26

Maximum Flow Problem

Idea: Use graphs to model the material flow through conduit.

- Each edge corresponds to a conduit and has a capacity.

(capacity: an upper bound on the flow rate.)

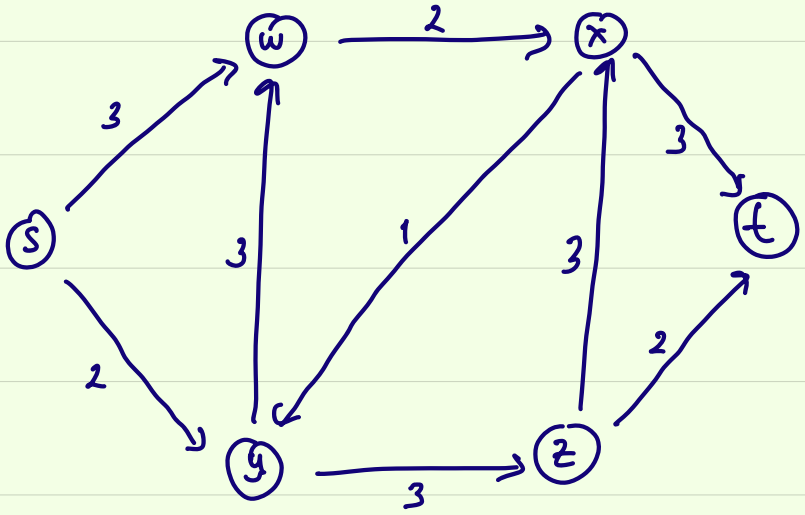
Input: $G = (V, E)$ ... directed, edge-labeled

$c: E \to R$ such that for every edge $(u,v) \in E$, $c(u,v) \geq 0$.

Notes:

- If $(u,v) \in E$, then $(v,u) \notin E$.
- If $(u,v) \notin E$, $c(u,v) = 0$.
- Two special nodes: Source s, Sink t.
- For all $u \in V - \{s,t\}$, there is a path $s \rightsquigarrow u \rightsquigarrow t$.

Example:

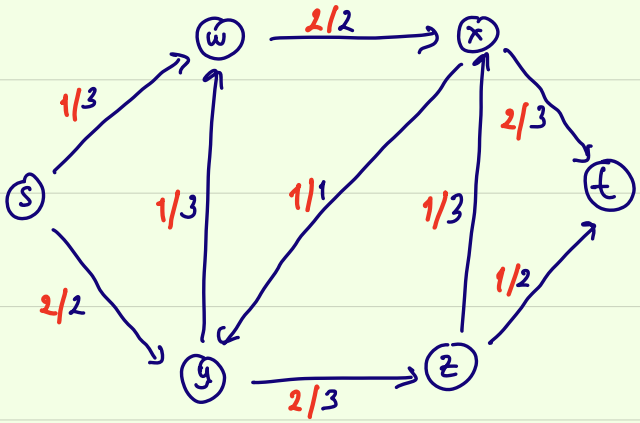


Flow: A function $f: E \to R$ with two properties:

- Capacity constraint: For any $u, v \in V$, we have $f(u,v) \leq c(u,v)$
- Flow conservation: For all $u \in V - \{s, t\}$ we have $\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$

($\sum_{v \in V} f(v,u)$: inflow of u; $\sum_{v \in V} f(u,v)$: outflow of u)

Example:



$|f| = 3$.

	inflow	outflow
w	2	2
x	3	3
y	3	3
z	2	2
s	0	3
t	3	0

Value of flow: $|f|$

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$$

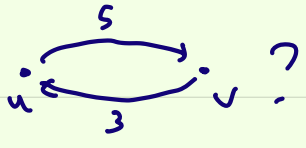
Max. flow Problem:

Input: $G = (V, E)$, $c: E \to R$, $s, t \in V$

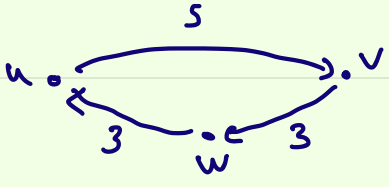
Output: A flow f s.t. $|f|$ is maximum.

Recall: We assumed

- If $(u,v) \in E$, then $(v,u) \notin E$.

What if we end up having 

?

Define an auxiliary node w and draw as follows: 

Cuts:

A cut (S,T) of a flow network $G = (V,E)$ is a partition of V into two disjoint subsets S and $T = V - S$ s.t. $s \in S$ and $t \in T$.

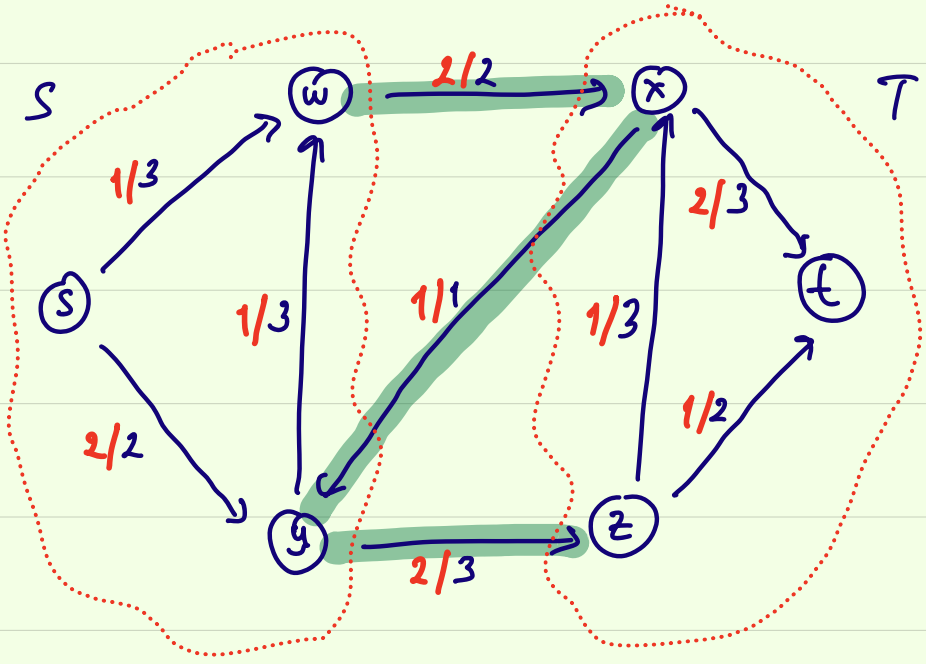
- For a flow f, the <u>net flow</u> across the cut (S, T):

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u).$$

- Capacity of the cut (S, T):

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

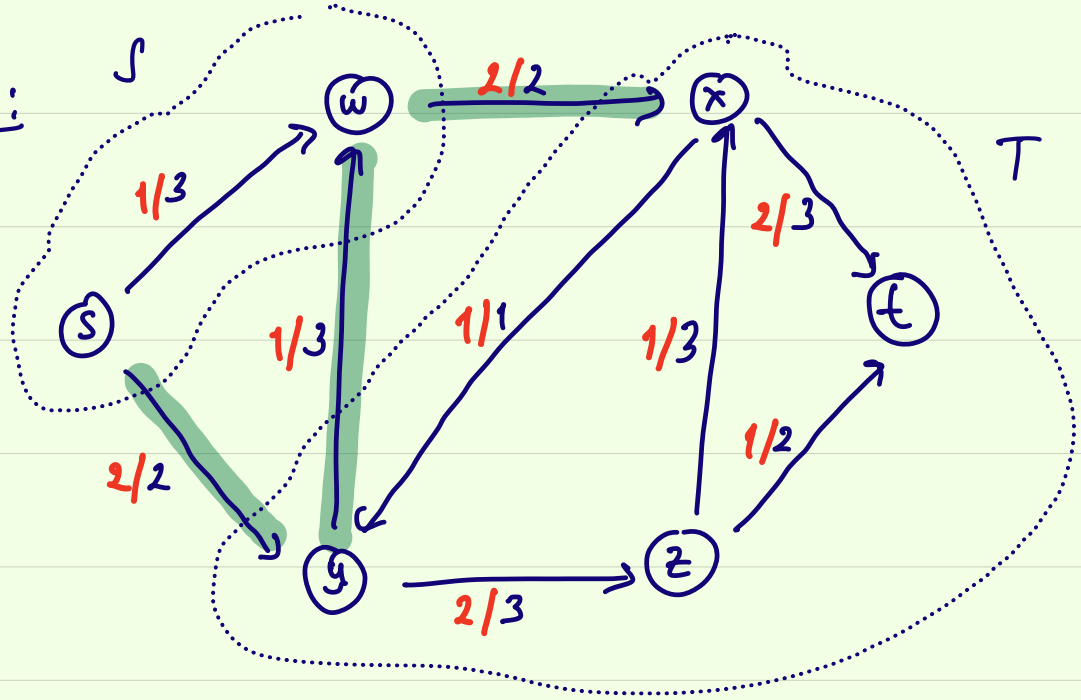
Example:



$|f| = 3.$

$$f(S,T) = 2 + 2 - 1 = 3$$
$$c(S,T) = 2 + 3 = 5$$

Example:



$|f| = 3.$

$f(S,T) = 2+2-1 = 3$

$c(S,T) = 4$

Lemma: Let f be flow. For any cut (S,T) we get $f(S,T) = |f|$.

Proof:

Recall the flow conservation: For every node $u \in V - \{s,t\}$, $\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$

$$\Rightarrow \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v) = 0$$



$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) - \sum_{u \in S-\{s\}} \left[\sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v) \right]$$

$$= \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) - \sum_{u \in S-\{s\}} \sum_{v \in V} f(v,u) + \sum_{u \in S-\{s\}} \sum_{v \in V} f(u,v)$$

$$= \left(\sum_{v \in V} f(s,v) + \sum_{u \in S-\{s\}} \sum_{v \in V} f(u,v) \right) - \left(\sum_{v \in V} f(v,s) + \sum_{u \in S-\{s\}} \sum_{v \in V} f(v,u) \right)$$

$$= \sum_{v \in V} \sum_{u \in S} f(u,v) - \sum_{v \in V} \sum_{u \in S} f(v,u)$$

(Recall: $V = S \cup T$, $S \cap T = \emptyset$)

$$= \left(\sum_{v \in S} \sum_{u \in S} f(u,v) + \sum_{v \in T} \sum_{u \in S} f(u,v) \right) - \left(\sum_{v \in S} \sum_{u \in S} f(v,u) + \sum_{v \in T} \sum_{u \in S} f(v,u) \right)$$

$$= \cancel{\sum_{v \in S} \sum_{u \in S} f(u,v)} + \sum_{v \in T} \sum_{u \in S} f(u,v) - \cancel{\sum_{v \in S} \sum_{u \in S} f(v,u)} - \sum_{v \in T} \sum_{u \in S} f(v,u)$$

$$= \sum_{v \in T} \sum_{u \in S} f(u,v) - \sum_{v \in T} \sum_{u \in S} f(v,u)$$

$$= f(S,T)$$

Corollary: The value of any flow is less than or equal to capacity of any cut.

Proof: let (S,T) be a cut and f be a flow.

$|f| = f(S,T)$ by the previous lemma.

(by def.) $$= \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u)$$

$$\leq \sum_{u \in S} \sum_{v \in T} f(u,v)$$

$$\leq \sum_{u \in S} \sum_{v \in T} c(u,v) \quad \text{(by capacity const.)}$$

$$= c(S,T) \blacksquare$$

Corollary: Max flow value $\leq$ Min-cut capacity.

Lecture 26

Max. Flow Problem cont...

Ford-Fulkerson Algorithm.

Residual Networks

Given $G = (V, E)$ and a flow f, consider an edge $(u, v) \in E$.

i.e. $u \bullet \xrightarrow[f(u,v)]{c(u,v)} \bullet v$

Q: How much more material/flow can be sent from u to v?

...residual capacity of (u, v)

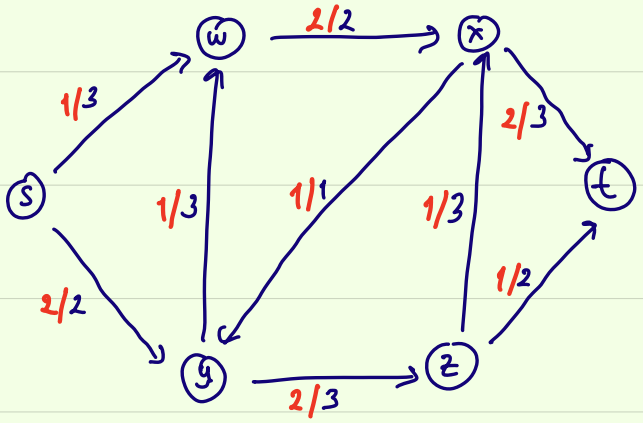
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(u,v) & \text{if } (v,u) \in E \\ 0 & \text{if } (u,v) \notin E \\ & \quad (v,u) \notin E \end{cases}$$

The residual network for G: G_f

$G_f = (V, E_f)$ where $E_f = \{(u,v) \mid c_f(u,v) > 0\}$

Example:

G:



$|f| = 3.$

$$c_f(s,w) = c(s,w) - f(s,w) = 3 - 1 = 2$$

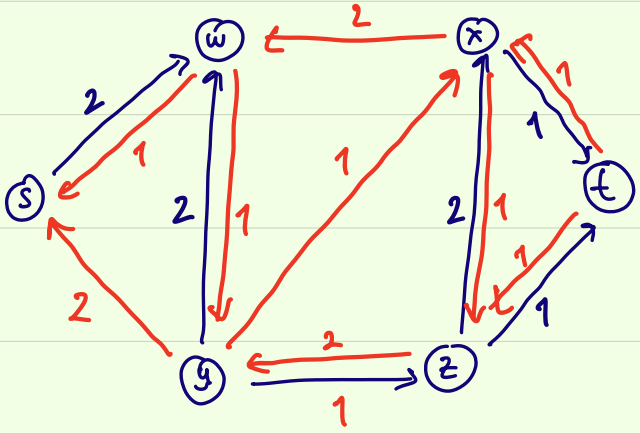
$$c_f(w,s) = f(s,w) = 1$$

$$c_f(y,w) = c(y,w) - f(y,w) = 2$$

$$c_f(w,x) = c(w,x) - f(w,x) = 2 - 2 = 0$$

$$c_f(x,w) = f(w,x) = 2$$

G_f:



Definition: Given a flow f and a graph $G = (V, E)$ and a flow f' in $G_f = (V, E_f)$, an "augmentation" of f by f', denoted $f \uparrow f'$, is a function from E to $\mathbb{R}$ defined as:

$$f \uparrow f'(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E. \\ 0 & \text{owise.} \end{cases}$$

Augmentation Path:

A simple path from source s to sink t that allows extra flow.

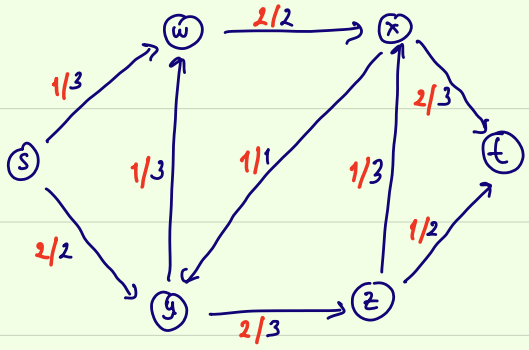
e.g. $s \xrightarrow{3} \cdot \xrightarrow{2} \cdot \xrightarrow{1} \cdot \xrightarrow{4} t$

$$c_f(p) = \min \{ c_f(u,v) \mid (u,v) \in p \}$$

additional amount that can be sent thru the path p.

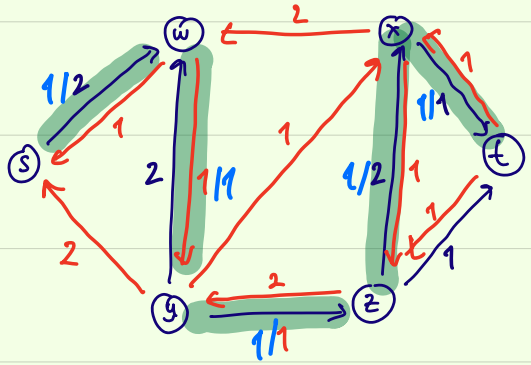
Example:

G:



$|f| = 3.$

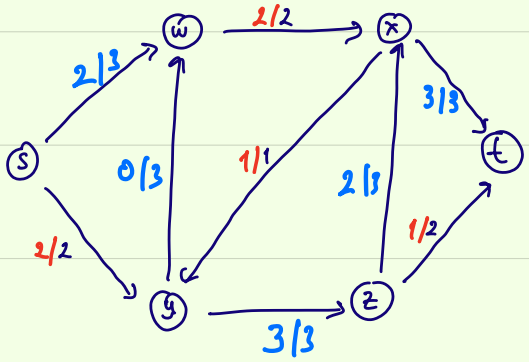
G_f:



$|f'| = 1$

p: $\langle s, w, y, z, x, t \rangle$

G_2:

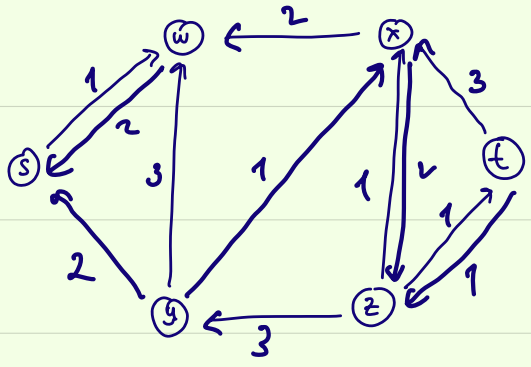


$|f \uparrow f'| = 4$

$f \uparrow f'(s,w) = f(s,w) + f'(s,w) - f'(w,s)$
$= 1 + 1 - 0 = 2$

$f \uparrow f'(y,w) = f(y,w) + f'(y,w) - f'(w,y)$
$= 1 + 0 - 1 = 0$

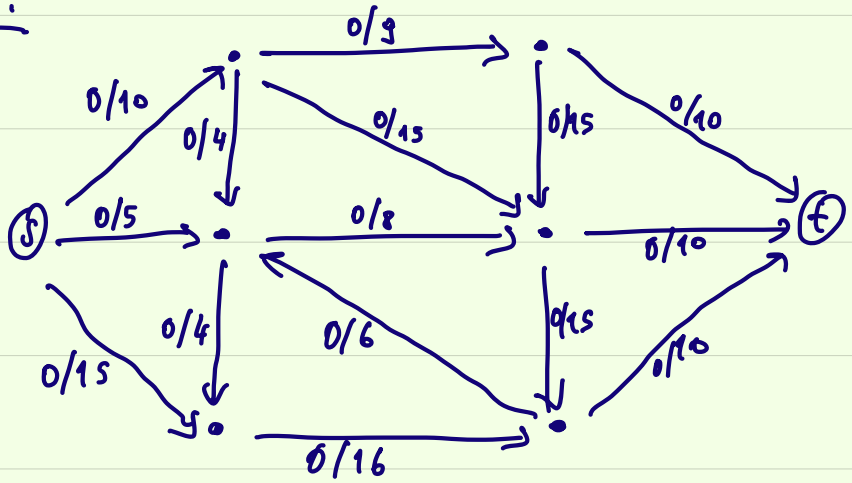
G_{2f}:



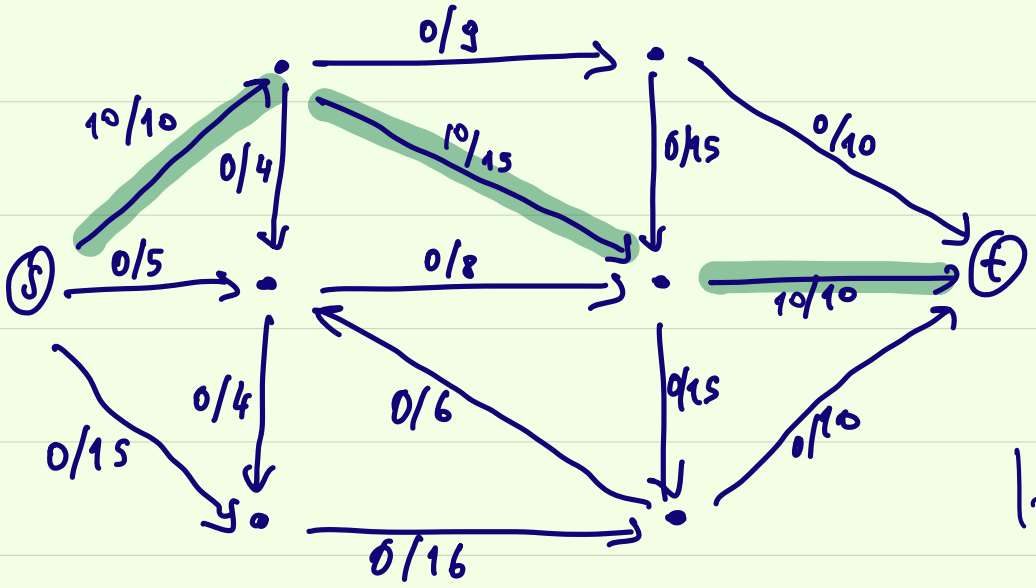
no path from s to t.

$\Rightarrow$ Max. flow thro G : 4.

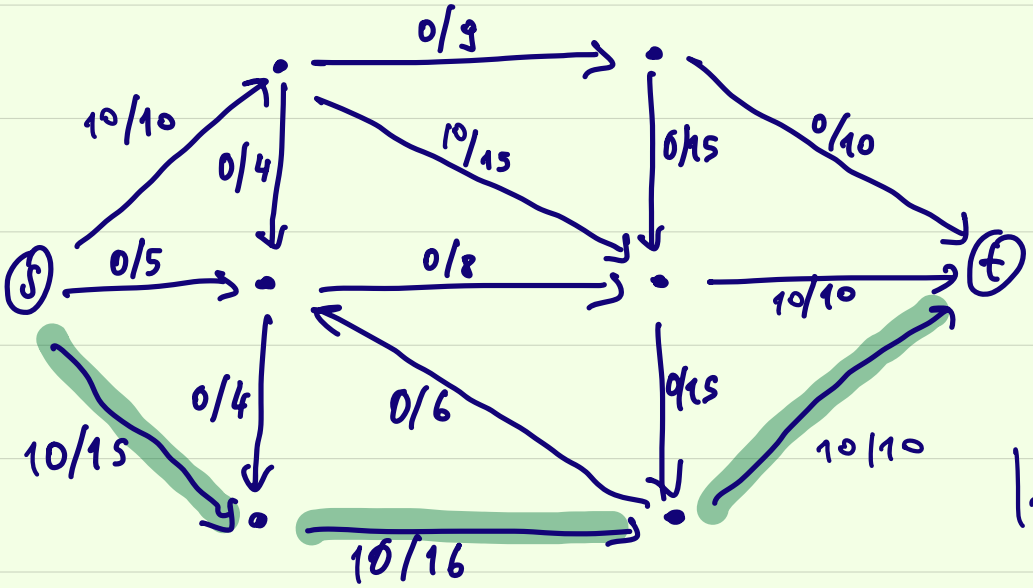
Example:



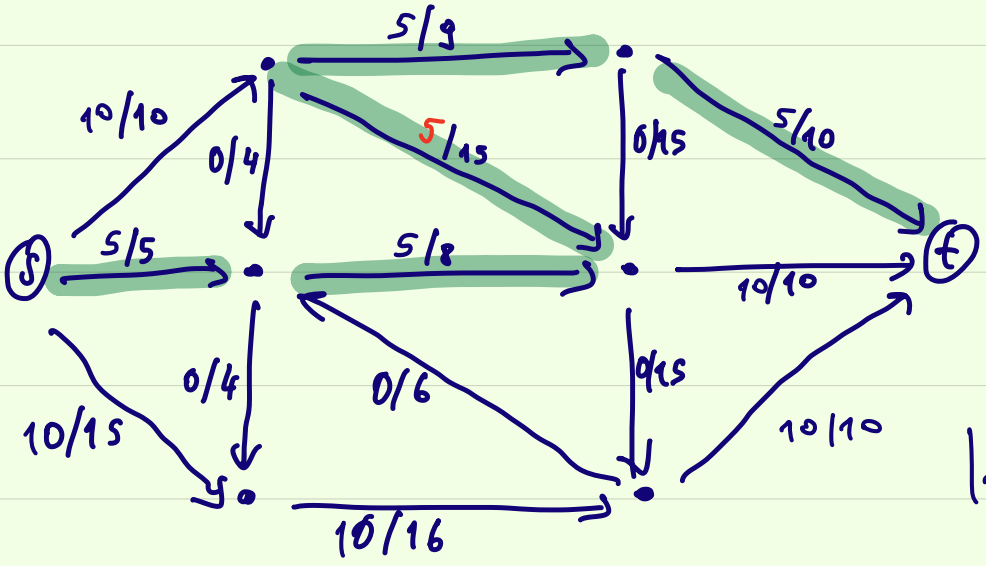
start with initial flow: 0


0/9
10/10
0/4
10/15
0/5
0/8
0/5
0/10
0/4
0/6
0/15
0/10
0/16

$|f| = 10$

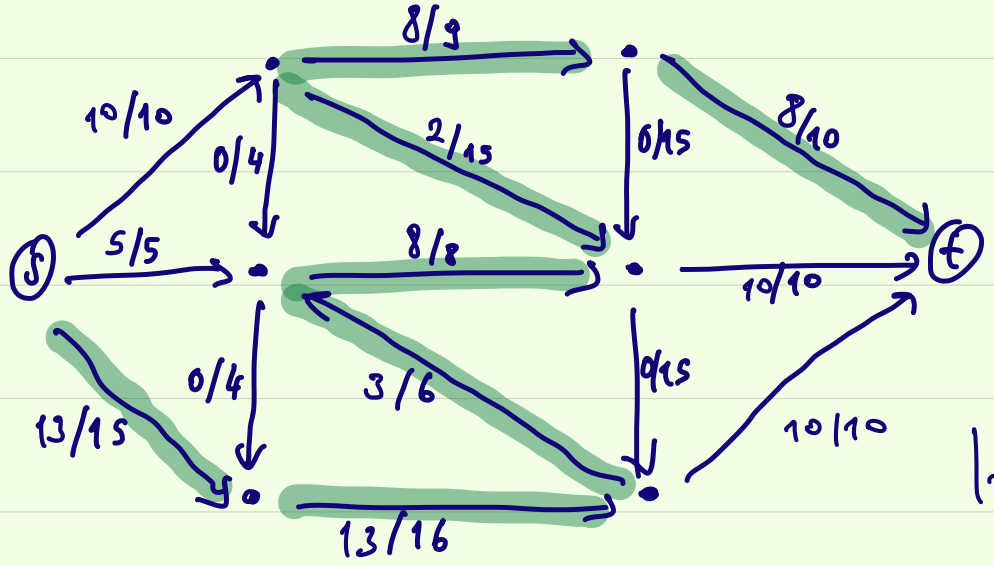

0/9
10/10
0/4
10/15
0/5
0/8
0/5
0/10
10/10
0/4
0/6
0/15
10/15
10/10
10/16

$|f| = 20$


5/9
10/10
0/4
5/15
0/5
5/10
5/5
5/8
0/5
10/10
0/4
0/6
0/15
10/15
10/10
10/16

$|f| = 25$

G':



$|f| = 28$

No more paths w/out full forward edges and nonempty backward edges.

Exercise: Draw the residual network of G' and show that there is no augmenting path from s to t in it.

In summary, F-F a/g:

- Start with initial flow = 0
- Increase the flow by going thru all the augmentation paths.

Given G, f and G_f, let p be an augmenting path in G_f. Let $f_p : E \to \mathbb{R}$ be defined as:

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{owise.} \end{cases}$$

(extra amount thru path p.)

<u>Lemma:</u> f_p is a flow in G_f with value $|f_p| = c_f(p)$.

<u>Corollary:</u> $f \uparrow f_p$ is a flow in G with value $|f \uparrow f_p| = |f| + |f_p|$.
