ECE1762 Algorithm LEC1

TOPICS:

① Background
  - Asymptotics
② SORTING
③ TREES
④ Dynamic Programming
⑤ Graphs
⑥ Parallel Algorithms
⑦ N-Completeness
⑧ Approximation Algorithms

Asymptotics:

$O(g(n)) = \{f(n) : \exists$ a constant C st. $0 \le f(n) \le Cg(n), \forall n \ge n_0\}$

$\Omega(h(n)) = \{f(n) : \exists$ a constant C st. $0 \le Ch(n) \le f(n), \forall n \ge n_0\}$

when $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n)) \Rightarrow f(n) = \Theta(g(n))$

or

$\Theta(g(n)) = \{f(n) : \exists$ a constant c_0, c_1, st $0 \le c_0 g(n) \le f(n) \le c_1 g(n), \forall n \ge n_0\}$

$o(g(n)) = \{f(n)\ \forall$ constant c, $\exists\ n_0$ st. $0 \le f(n) \le cg(n), \forall n \ge n_0\}$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \ne \emptyset$$

$\omega(g(n)) =$ similar, check text

Ex1:

$5n^3 = O(n^3) \neq o(n^3)$

$5n^3 = \omega(n^2) = \Omega(n^2)$

Ex2:

Prove: $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$

1) $O(n^{k+1})$

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

2) $\Omega(n^{k+1})$

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

$\geq \sum_{i=1}^{n} (\frac{n}{2})^k = (\frac{1}{2})^{k+1} n^{k+1}$

$= \Omega(n^{k+1})$

Ex3: Show $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

$O(n^2)$

$\frac{1}{2}n^2 - 3n \leq cn^2$

$\frac{1}{2} - \frac{3}{n} \leq c$

$c = 1/2$

$\Omega(n^2)$

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n$

$c_1 \leq \frac{1}{2} - 3/n$

Some Properties of Asymptotics

$n^a \in O(n^b)$ iff $a \le b$

$n^a \in o(n^b)$ iff $a < b$

$\log_b n \in o(n^a)\ \forall b, a$

$\log_a n \in O(\log_b n)\ \forall a, b$

$c^n \in O(d^n)$ iff $c \le d$

if $f(n) \in O(F'(n))$ & $g(n) \in O(G'(n))$ then

$f(n) g(n) \in O(f'(n) \cdot g'(n))$

$f(n) + g(n) \in O(\max(f'(n), g'(n))$

Best / Worst / Average { Probabilistic Analyses

Randomized Algorithm

Monte-Carlo (don't lie)

Las Vegas (lie)

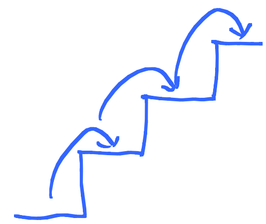
Induction :

Based on well ordering of numbers

$P(1) \wedge P(k) \rightarrow P(k+1)$

Inductive Basis
Inductive Hypothesis
Inductive Steps

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



Basis: $1 = \frac{1(1+1)}{2}$

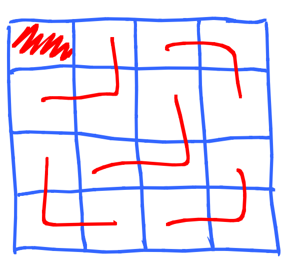
Hypothesis: $1+2+\cdots+(n-1) = \frac{n(n-1)}{2}$

Inductive: $1+2+\cdots+(n-1)+n = \frac{n(n-1)}{2} + \frac{2n}{2}$

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

Ex: Prove any $2^n \times 2^n$ grid can be tiled with L shaped (3 block) tiles leaving only one tile empty

Basis

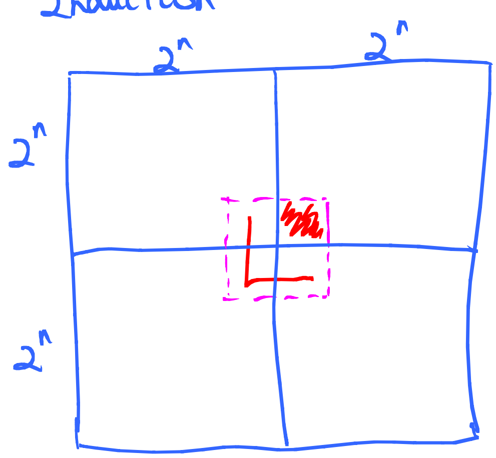


Hypothesis

$2^n \times 2^n$

Can be tiled

Induction



Recurrences:

Merge Sort

5 3 7 1   8 2   4 6

3 5 1 7   2 8   4 6

1 3 5 7   2 4   6 8

1 2 3 4 5 6 7 8

Merge Sort (A[1..n])

B = MergeSort(A[1 -- n/2])

C = MergeSort(A[n/2+1 -- n])

Merge (B, C)

$B = O(n)$ $C = O(n)$

Recurrences

$$T(n) = 2T(n/2) + O(n) = O(n \log n)$$

① Substitution (= induction)

Guess $T(n) = cn \log n$

Hypothesis: $T(n/2) \le c\frac{n}{2} \log \frac{n}{2}$ true $1 --- n/2$

Step: $T(n) \le 2c \cdot \frac{n}{2} \log(\frac{n}{2}) + O(n)$

$$= cn \log(n/2) + O(n)$$

$$= cn \log n - cn + O(n)$$

$$= cn \log n - \underbrace{n(c-c')}_{\text{c can be large enough to dominate } c'} \le cn \log n$$

$T(1) = 1 \log 1 = \phi$

Basis $n=2$ $T(2) = 4$ $T(3) = 5$

$T(2) \le c2 \log 2 = 2c$, I need $c \ge 2$

6

Errorness:

$T(n) = O(n) = c \cdot n$

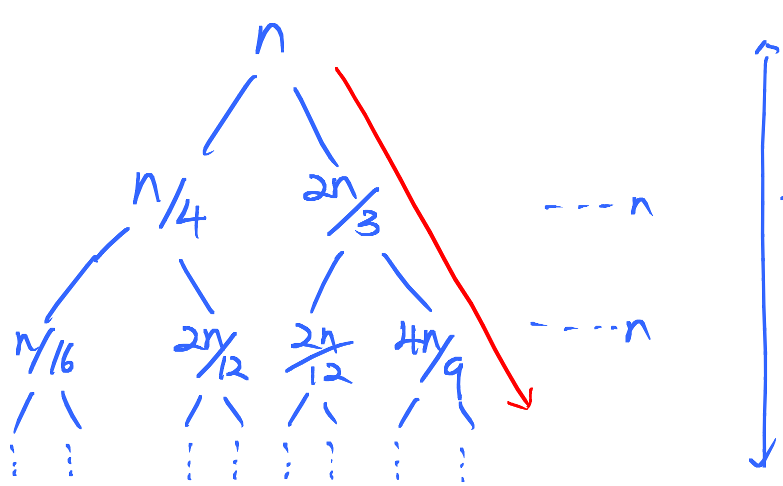
$T(n) \leq 2 \cdot c\frac{n}{2} + O(n) = cn + O(n)$

$= (c + c')n = c''n$

Induction Assumption $T(n/2) \leq c\,n/2$

② Recursion Tree

$T(n) = T(n/4) + T(2n/3) + n$



height of the tree:

$\left(\frac{2}{3}\right)^h n = 1$

$h = \log_{3/2} n$

Time $= n \cdot h$

$= n \log_{3/2} n = O(n \log n)$

How about $T(n) = T(n/10^6) + T\left(\frac{999999n}{10^6}\right) + n$

$= O(n \log n)$

ECE1762 Algorithm LEC02

Cont. Recursion

Renaming Variable

$$T(n) = 2T(\sqrt{n}) + \log n \qquad m = \log n$$

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

$$S(m) = 2S\left(\frac{m}{2}\right) + m = O(m \log m)$$

$$= O(\log n \log \log n)$$

Iteration Method

Read Text

Master Theorem:

Let $a > 1$, $b > 1$ and $f(n)$ = function

Recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + F(n) \text{ has solution}$$

- If $F(n) = O(n^{\log_b a - \epsilon})$ for $\epsilon > \emptyset$. Then $T(n) = \Theta(n^{\log_b a})$
- If $F(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$
- If $F(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon > \emptyset$, $af(n/b) \le cF(n)$ For $c < 1$ then $T(n) = \Theta(F(n))$

Ex $T(n) = T\left(\frac{2n}{3}\right) + 1 \qquad a = 1 \quad b = 3/2, \quad f(n) = 1$

$$\log_{3/2} 1 = \emptyset \qquad 1 = \Theta(n^0)$$

$$\rightarrow = \Theta(\log n)$$

$T(n) = 3T(\frac{n}{4}) + n\log n$

$f(n) = n\log n, \quad a=3, \quad b=4$

$n\log n = \Omega(n^{\log_4 3 + 0.2})$

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

$3\frac{n}{4}\log\frac{n}{4} < cn\log n$, true for $c \geq 3/4$

Some useful formula:

$$n! = \begin{cases} 1 & n=0 \\ n\cdot(n-1)! & n\geq 1 \end{cases}$$

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n+\frac{1}{12n}}$$

Fibonacci $\quad F_i = F_{i-1} + F_{i-2} \quad F_0 = 0 \quad F_1 = 1$

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$ (induction)

$\phi = \frac{1+\sqrt{5}}{2}$ $\quad \bar{\phi} = \frac{1-\sqrt{5}}{2}$ $\quad$ Golden $\quad$ Conjugate

$$\sum_{k=1}^{n} k = 1+2+\cdots+k = \frac{n(n+1)}{2} = \Theta(n^2)$$

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^k = \frac{x^{n+1}-1}{x-1}$$

$$(x+r)^n = \sum_{i=1}^{n}\binom{n}{i}x^i y^{n-i} \qquad \frac{n!}{(n-i)!\,i!}$$

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$ when $|x|<1$

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2} \qquad \sum_{k=1}^{\infty} x^k = \frac{1}{1-x} \qquad \sum k x^k = \frac{x}{(1-x)^2}$$

$$\sum_{i=1}^{n} a_i - a_{i-1} = a_n - a_0 \qquad \sum_{i=1}^{n-1} \frac{1}{k(k+1)} = 1 - 1/n$$

Logorithm

$$a = b^c \qquad \log_b a = c$$

Properties: $a = b^{\log_b a}$

$$\log(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$\log_b \frac{1}{a} = -\log_b a$$

$$\log_b \frac{a}{c} = \log_b a - \log_b c$$

$$a^{\log_b n} = n^{\log_b a}$$

$$\log^{(i)} n = \underbrace{\log \log \cdots \log}_{i \text{ times}} n$$

$$\log^* n = \min \{ i : \log^{(i)} n \le 1 \}$$

$$\log^* 2^{16} = 1 + \log^* 16 = 1 + 1 + \log 4 = 4$$

$$\log^* 2^{2^{16}} = 5$$

$f: A \to B$ (A: domain, B: range)

1-1 Function $f(a) \ne f(a')$ for $a \ne a'$

- onto $\forall$ element in range $\exists$ a map
- bijection : 1-1 & onto

Binary $R \subseteq A \times A$

Reflexive if aRa

Symmetric $aRb \Rightarrow bRa$

Transitive $aRb \wedge bRc \Rightarrow aRc$

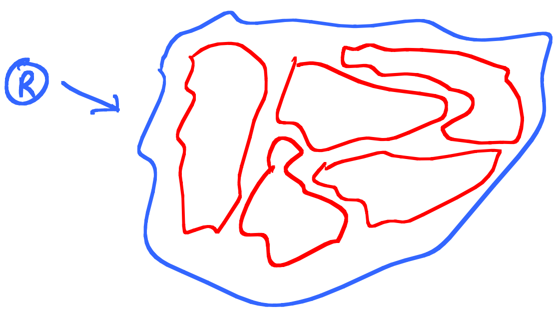
} equivalence relationship (all three)

$[a] = \{b : aRb\}$

↳ equivalence class of an element wrt R

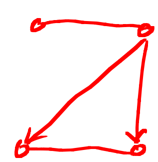
anti-symmetric : if aRb & $bRa \Rightarrow aRa$

partial order : reflexive, anti-symmetric & transitive



A partial order R is called <u>total order</u> iff $\forall a, b \in U$ we have aRb or bRa

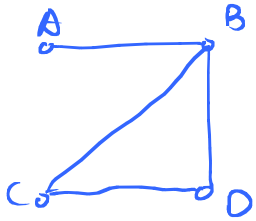
Graphs : $G = (V, E)$



- directed or undirected
- weighted G (cost, profit, ...)
- path, simple path (no edge repetition)
- connected & unconnected
- induced subgraph
- clique (complete graph) All vertices connected

Adjacency Matrix

	A	B	C	D
A		✓	×	×
B	✓		✓	✓
C	×	✓		✓
D	×	✓	✓	



Time: $O(1)$
Space: $O(V^2)$

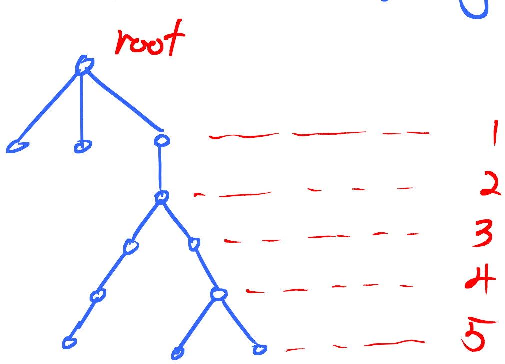
Adjacency List

C → D
D → C → E → F
E → F → D
F → D → E

Sparse graph $E \ll O(V^2)$

$O(n)$
$O(E)$

Tree = connected, undirected, acyclic graph



For a tree, equivalent statement

1) $\forall$ two vertices are connected by a simple path
2) Removing any edge disconnects the tree
3) G connected & $|E| = |V| - 1$
4) G is acyclic and $|E| = |V| - 1$
5) Acyclic and any new edge addition creates a cycle.

ECE1762 Algorithm LEC03 ①

Permutations & Combinations

Rule of Product: If we go event A that can happen in n-ways, Event B that can happen in m-ways there are $n \times m$ ways both events can happen

Rule of Sum: If we go event A that can happen in n-ways, Event B that can happen in m-ways there are $n+m$ ways A or B can happen

Permutation: $_nP_r = \frac{n!}{(n-r)!} = P(n, r)$

If n-objects are q_1 1st kind, q_2 2nd kind, $\vdots$ q_t t^{th} kind

$$\frac{n!}{q_1! q_2! \cdots q_t!}$$

Ex: Prove

$(k!)!$ is divisible by $(k!)^{(k-1)!}$

Assume you get

k objects of 1^{st} kind

2^{nd} kind

$\vdots$

k objects $(k-1)!$ kind

Sum = $k!$ obj.

$$\frac{(k!)!}{\underbrace{k!\,k!\cdots k!}_{(k-1)!\text{ times}}} = \frac{(k!)!}{(k!)^{(k-1)!}} \quad \text{QED}$$

Ex: Among 1 .. 10^{10}, how many contain digit 1 ②

→ How many does not contain 1 ?

$(9^{10} - 1)$

→ How many contain 1

$10^{10} - (10^{9} - 1)$

<u>Combinations</u>: Same definition as Permutation except order doesn't matter.

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!} = \binom{n}{r} \text{ a binomial coefficient.}$$

Ex: How many diagonals a decagon has ?

Review On Probability:

S = Sample Space

$\emptyset \leq P(A) \leq 1$

$\sum P(\text{Event}) = 1$

$Pr(\text{Event}) = \frac{1}{|S|}$ Uniform distribution

Ex. A fair Coin

$S = \{H, T\}$ if not biased $P(H) = P(T)$

$$Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B)$$

Conditional Probability

$$Pr(A/B) = \frac{Pr(A \cap B)}{Pr(B)}$$

if A&B are independent

$$Pr(A \cap B) = Pr(A)Pr(B)$$

3

$$Pr(A|B) = \frac{Pr(B|A)Pr(A)}{Pr(B)}$$

Discrete Random Variables

Maps Events to Real Numbers Allow Us to Generate Distributions.

Define Event

$X = x$ such as $\{ s \in S : X(s) = x \}$

$$Pr(X=x) = \sum_{\{s \in S : X(s)=x\}} Pr(s)$$

Expected Value

$$E[X] = \sum_x x \, Pr(X=x)$$

On Same S, r.v. X, Y

$$Pr(X=x / Y=y) = \frac{Pr(X=x \cap Y=y)}{Pr(Y=y)}$$

Properties of Expected Value

$$E(X+Y) = E(X) + E(Y)$$
$$E(aX) = aE(X)$$
$$E(XY) = E[X]E[Y] \quad x, y \text{ independent}$$

$$Var(X) = E[(X - E[X])^2]$$
$$= E(X^2) - E^2(X)$$

4

Bernoulli Trial

An experiment with two outcome $\begin{cases} S \\ F \end{cases}$

$$\Pr(S) = p \qquad \Pr(F) = 1 - p = q$$

Geometric Distribution: How many (expected) times we need to wait to get a success.

$$\Pr(X=k) = q^{k-1} \cdot p$$

$$E[X] = \sum_{k=1}^{\infty} k \cdot q^{k-1} p$$

$$= \frac{p}{q} \sum_{k=1}^{\infty} k q^k \qquad = \frac{q}{(1-q)^2}$$

$$= \frac{1}{p}$$

Binomial Distribution: How manys success over n-trials?

$$\Pr(X = k) = \binom{n}{k} p^k q^{n-k} = b(n, k, p)$$

$$E[X] = \sum_{k=0}^{n} k\, b(n, k, p)$$

$$= \sum_{k=0}^{n} k \binom{n}{k} p^k q^{n-k}$$

$$= n \cdot p \sum_{k=1}^{n} \binom{n-1}{k-1} p^{k-1} q^{n-k}$$

$$= n \cdot p \sum_{k=0}^{n-1} \binom{n-1}{k} p^k q^{(n-1)-k}$$

$$= np$$

ECE1762 Algorithm LEC04

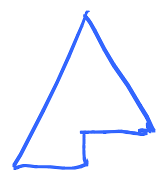
HEAPSORT:

- In place

- Tree is just indicative, not USED in practice:

HEAP: a binary tree that has two properties:

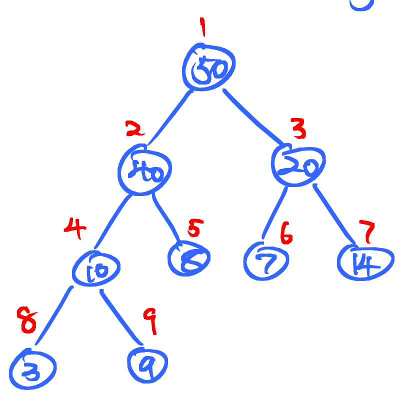
- Heap Shape Property



complete except lower level where leaves were pushed to the left

- Heap order Property

key(parent) > key(children)



1-1

1	2	3	4	5	6	7	8	9
50	40	20	10	8	7	14	3	9

parent $\lfloor \frac{i}{2} \rfloor \leftarrow$ index i $\begin{cases} \text{left child } 2i \\ \text{right child } 2i+1 \end{cases}$

Bubble-Up

$\left.\begin{array}{l} \text{Compare } (i) \text{ with } \lfloor i/2 \rfloor \\ \text{exchange if larger} \end{array}\right\} O(h)$

what is h?

a) complete tree $\Rightarrow O(\log n)$

b) $T(h) = T(\frac{2n}{3}) + 1$

$h = O(\log n)$

Build - Heap

bubble down (i)

comparing $A[i]$ w/ $A[2i]$ & $A[2i+1]$

replace w/ larger

call yourself recursively if you exchange on that element

Lemma: There are at most

$\left\lceil \frac{n}{2^{h+1}} \right\rceil$ nodes at height h in a heap with n-nodes

Time to build a heap

$$T(\text{build\_heap}) = \sum_{h=0}^{\lfloor \log n \rfloor} h \left\lceil \frac{n}{2^{h+1}} \right\rceil = n \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{h}{2^{h+1}} \right\rceil \leq n \sum_{h=0}^{\infty} \frac{h}{2^{h+1}}$$

$$= n \, \frac{1/2}{1/4} = 2n = O(n)$$

$$* \sum_{i=0}^{\infty} i x^i = \frac{x}{(1-x)^2} \qquad |x| < 1$$

Extract Max

$O(\log n)$: replace w/ right most leaf, bubble down new root

Heap Sort

Extract - max (A)

$|A| = |A| - 1$

repeat until $|A| = 1$

$\Big\}$ $O(n \log n)$

Min/Max Heaps

Priority Queue

ECE1762 Algorithm LEC05 ①

```
QS (A, l, r)
    if l<r
        q= PARTITION(A, l, r)
            QS(A, l, q)
            QS(A, q+1, r)

PARTISON (A, l, r)
    k = l
    pivot = A[l]
    for i= l+1 ... r
        if A[i] ≤ pivot
            k= k+1
            swap (A(i), A[k])
        Swap (A(l), A(k))
    return k
```

Worst Case (informal)

$$T(\theta) = T(n-1) + \theta(n)$$

$$= T(n-2) + \theta(n-1) + \theta(n)$$

$$\vdots$$

$$= \sum_{i=1}^{n} = \theta(n^2)$$

Best Case (informal)

$$T(n) = 2T(n/2) + \theta(n) = \cdots = \theta(n \log n)$$

Average Case

$$T(n) = T\left(\frac{n}{9}\right) + T\left(\frac{8n}{9}\right) + \theta(n) = \theta(n \log n)$$

constant are different

$$\left(\frac{8}{9}\right)^h \cdot n = 1$$

Randomized Partion (A, l, r)

Randomly Swap $A(l) \longleftrightarrow A(\text{random} < r)$

Return Partition (A, l, r)

Worst Case (Formal) both deterministic & non-deterministic

$$T(n) = \max_{1 \le q \le n-1} (T(q) + T(n-q)) + \Theta(n)$$

Solve by substitution, we guess $O(n^2)$

$$T(n) \le \max_{1 \le q \le n-1} (cq^2 + c(n-q)^2) + \Theta(n)$$

$$= c \max_{1 \le q \le n-1} (q^2 + (n-q)^2) + \Theta(n)$$

$\therefore$ to max $q = 1$ or $n-1$

pick $q = 1$

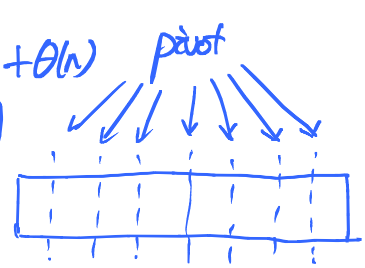
$$= c(1 + (n-1)^2) + \Theta(n)$$

$$= cn^2 - 2c(n-1) + \Theta(n) \le cn^2$$

can always choose $c >> c'$ in $\Theta(n)$

Average Case

$+\Theta(n)$



$$T(n) = \frac{1}{n}\left(T(1) + T(n-1) + \sum_{q=1}^{n-1} (T(q) - T(n-q))\right)$$

$$= \frac{1}{n}(T(1) + T(n-1)) + \frac{1}{n}\sum_{q=1}^{n-1} (T(q) - T(n-q)) + \Theta(n)$$

$$= \Theta(n) + \frac{1}{n}\sum_{q=1}^{n-1} (T(q) - T(n-q))$$

③

$$\frac{1}{n}\sum_{q=1}^{n-1} T(q)+T(n-q)+\theta(n)$$

$$= \frac{2}{n}\sum_{q=1}^{n-1} T(k)+\theta(n)$$

* Lemma:

$$\sum_{k=1}^{n-1} k\log k = \frac{1}{2}n^2\log n - \frac{1}{8}n^2$$

Assume $T(n)=an\log n + b$

$$T(n) \le \frac{2}{n}\sum_{k=1}^{n-1} ak\log k + b + \theta(n)$$

$$\frac{2a}{n}\sum_{k=1}^{n-1} k\log k + \frac{2b}{n}(n-1)+\theta(n)$$

$$= an\log n + b + \theta(n) + b - \frac{a}{4}n \le an\log n + b$$

* Difference Between

Theorem : Usually requires huge proofs.

Lemma : Small "Theorem" used to prove Theorem.

Corollary : Result of Theorem

Axioms : Can not be proved. Take it as a truth.

Proof: $\sum_{k=1}^{n-1} k\log k \le \frac{1}{2}n^2\log n - \frac{1}{8}n^2$

$$\sum_{k=1}^{\lceil n/2\rceil} k\log k + \sum_{k=\lceil \frac{n}{2}\rceil}^{n-1} k\log k$$

$$\le (\underbrace{\log\tfrac{n}{2}}_{\log n - 1})\sum_{k=1}^{\frac{n}{2}-1} k + \sum_{k=\lceil \frac{n}{2}\rceil}^{n-1} k\log k$$

$$\le \log n\sum_{n=1}^{n/2} k - \sum_{k=1}^{n/2} k + \log n \sum_{k=n/2}^{n-1}$$

4

$$= \log n \sum_{k=1}^{n-1} k - \sum_{k=1}^{n/2} k \le$$

$$\frac{1}{2} n(n-1)\log n - \frac{1}{2}\left(\frac{n}{2}-1\right)\frac{n}{2} \le \frac{1}{2} n^2 \log n - \frac{1}{8} n^2$$

Next week: Lower Bound on Comparison-based Sorting $\Omega(n\log n)$

Bubble Sort:

```
for i = 1 .. n do
    for j = 1 .. n do
        if A(i) > A(j)
            swap A(i) & A(j)
```

RADIX Sort: A stable sort algorithm

Digits sorting within a range $[0..k]$

n is the # sorting

d # of digits

RS (A, d)

LSB to MSB

stable sort A on digit i

↳ insertion sort

$[0..k] \Rightarrow [0..9]$

n = # of elements

d = # of digits

531	531	521	181
424	181	424	424
181	521	531	521
732	732	732	531
521	424	181	732

Counting Sort: Stable but not in-place

A[1..n]   B[1..n]   C[1....range]

C[i] = 0 ∀i

For j = 1 --- length(A)

  C[A[j]] = C[A[j]]+1

For i = 2..k

  C[i] = C[i] + C[i-1]

For j = length[A] down to 1 do

  B[C[A(j)]] = A[j]

  C[A(j)] = C[A(j)]-1

total time $O(n+k)$

A

4	3	5	2	1	4	3

C

1	2	3	4	5
1	1	2	2	1

loop #1

C

1	2	4	6	7

loop #2

B

1	2	3	3	4	4	5

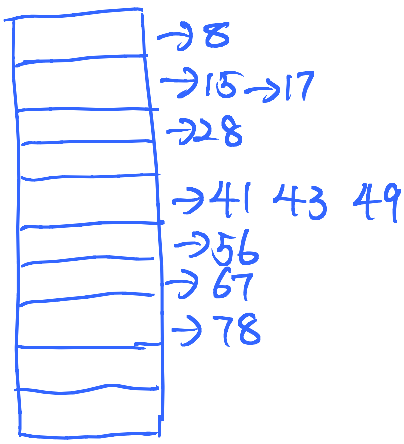
Bucket Sort:

Assume you have n #s uniformly distributed between a range [0...k]

10 #s 0--99 - Create an array B of n-slots

- $\forall i$, sort $B(i)$ w/ Bubble sort
- "Thread" slots together

17
15
28
56
8
78
67
43
41



Assume n_i elements in bucket in bucket-i

$\Rightarrow O(n_i^2)$ to sort each bucket

$$E(\text{sort one bucket}) = E(O(n_i^2))$$
$$= O(E(n_i^2))$$

$$\text{time} \sum_{i=0}^{n-1} E(O(n_i^2))$$
$$= O\left(\sum_{i=0}^{n-1} E(n_i^2)\right) = 2n - \frac{1}{n} \cdot n = O(n)$$

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

$$p = 1/n$$

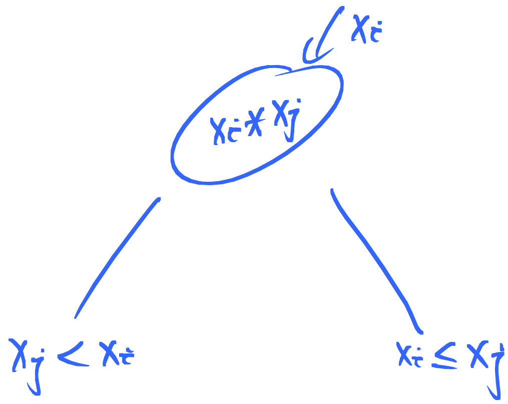
Binomial Distribution

$$E[X] = n \cdot p = 1 \qquad Var[X] = npq = n \cdot \frac{1}{n}\left(1 - \frac{1}{n}\right)$$
$$= 1 - 1/n$$

ECE1762 Algorithm LEC06 ①

Lower Bound on Sorting

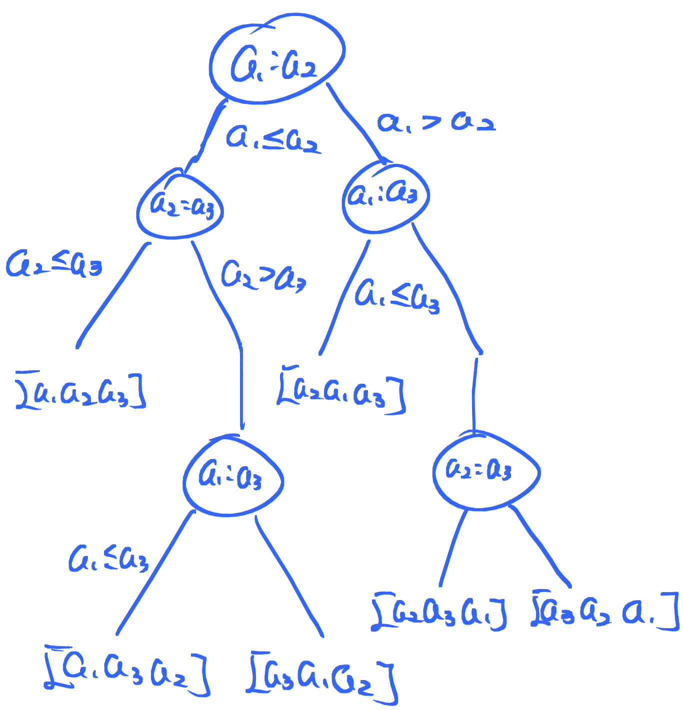
Any comparison based sorting algorithm for n-elements of unrestricted range takes $\Omega(n\log n)$ time.



Insertion Sort

5 3 1 2

i.e. Decision Tree for insertion sort of 3 elements



we have $n!$ possible outcomes of sorted sequences given n-element ②

2^h = max of leaves for binary tree

$$n! = \left(\frac{n}{e}\right)^n$$

$$2^h \geq n! \iff h \geq \log n! \geq \log\left(\frac{n}{e}\right)^n$$

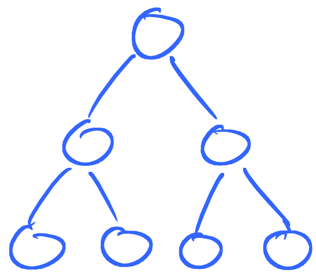
$$\Rightarrow \quad h \geq n \log \frac{n}{e} = n\log n - n\log e = \Omega(n\log n)$$

Selecting k^{th} largest

Finding largest in an array

① Compare and swap

② Comparison Tree



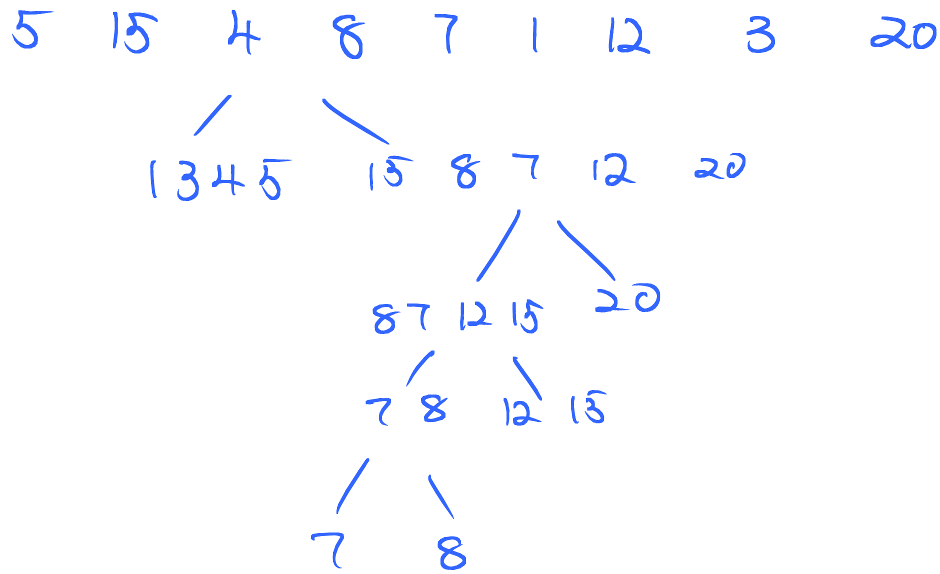
- if known max $n/2 - 1$ to find min
- 2^{nd} max $\log n$ additional comp.
- $(n-1) + k \log n$ comps for k^{th} max.

To find largest $O(n)$

Finding k^{th}-max

- Expected $O(n)$ algorithm
- Worst case $O(n)$ algorithm

```
Rand_Select(A, p, r, i)
    if p=r then return A(p)
    q = rand_partition(A, p, r)
    k = q-p+1
    if i≤k then
        Rand_Select(A, p, q, i)
    else
        Rand_Select(A, q+1, r, i-k)
```



Efficiency:

$$T(n) = \frac{1}{n}\left[T(\max\{1, n-1\}) + \sum_{k=1}^{n-1} T(\max\{k, n-k\})\right] + O(n)$$

$$\leq \frac{1}{n}\sum_{k=n/2}^{n} T(k) + O(n)$$

Assume $T(n') \leq cn'$, $n' < n$

Prove it works for n

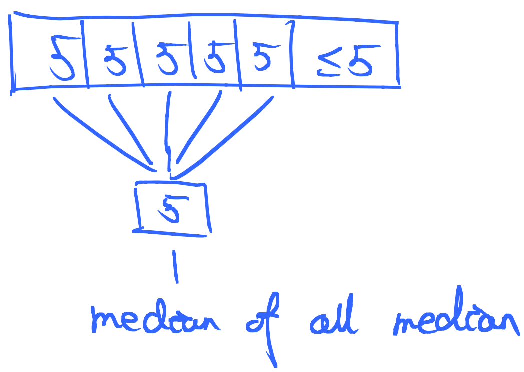
$$T(n) \leq \frac{2c}{n}\sum_{k=\lfloor n/2 \rfloor}^{n} k + O(n) = \frac{2c}{n}\left[\sum_{k=1}^{n} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k\right] + O(n)$$

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

$$\leq c \cdot n$$

Worst Case $O(n)$

1) Partition array in $\frac{n}{5}$ groups of 5 elements
2) Find the median of each group (w/ insertion sort)
3) Recursively find median of all medians, call it x
4) Partition original array around x
5) Recursively call step 1 on the right side of partition



At every iterations of the algorithm

At least half of the groups contribute 3 elements except last group & group that contains median of all medians.

$$3\left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil-2\right) \geq \frac{3n}{10}-6$$

At step 5 we will need at most $\frac{7n}{10}+6$ elements

6

$$T(n) = \begin{cases} O(1) & n < 140 \\ T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10} + 6\right) + O(n) & n \geq 140 \end{cases}$$

$$T(n) \leq c\left\lceil\frac{n}{5}\right\rceil + c\left(\frac{7n}{10} + 6\right) + an$$

$$\leq \frac{cn}{5} + c + \frac{7cn}{10} + 6c + an$$

$$= \frac{9cn}{10} + 7c + an$$

$$= cn + \underbrace{\left(-\frac{cn}{10} + 7n + an\right)}_{\text{is negative}} \leq cn$$

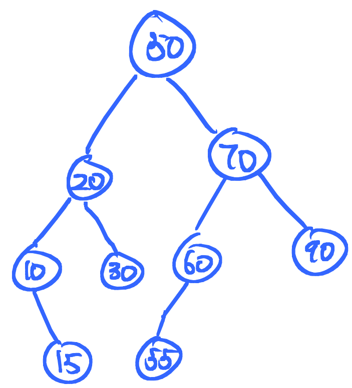
$$-\frac{cn}{10} + 7c + an \leq \emptyset \qquad c \geq 10a\left(\underbrace{\frac{n}{n-70}}_{\leq 2}\right)$$

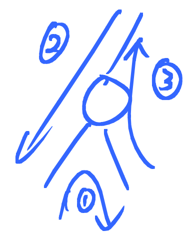
$\forall\, c \geq 20a$ makes the quantity negative

QED (Quod Erat Demonstatory)

Binary Search Trees

- Binary Tree
- left subtree key < parent key < right subtree key





Sort: In order O(n)

Search: prope the root

go left or right among comparison O(h)

Insert: Assume unique key

Search and insert as leaf O(h)

Max/Min: Most right or most left

Successor: element visited after in inorder traversal (in order). O(1)

Delete: 1) Search for it

2) a. leaf → delete

b. 1 child → replace with immediate child

c. 2 children → replace with largest on the left

or smallest on the right O(h)

$h = O(n)$ worst case

A balanced Binary Tree $h = O(\log n)$

R-B trees, AVL trees, B/B^{+} trees, 2/3 trees

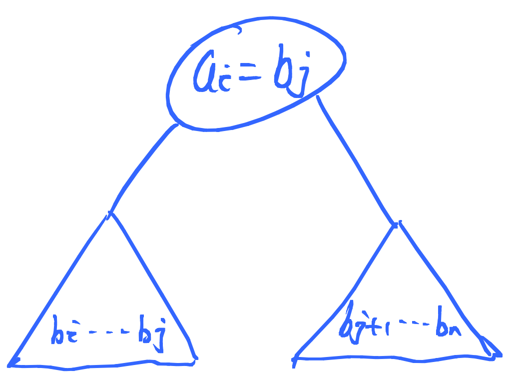
Splay Trees

How long does it take to build a BST?

$n \log n$ $\quad$ n^2 (worst)

Theorem: if keys are selected from a uniform distribution, then the expected number of comparisons is $O(n \log n)$

Let keys be $a_1, a_2, \dots, a_n$ & their sorted sequen. be $b_1, b_2, \dots b_n$.



$$T(n) = \frac{1}{n} \sum_{j=1}^{n} [(j-1) + T(j-1)] + [(n-j+1) + T(n-j+1)]$$

$$= O(n \log n)$$

ECE1762 Algorithm LEC07 ①

Most ops depend $O(h)$, for BSTs $\log n \leq h \leq n$

"Balanced Trees" = guarantee $h = O(\log n)$

Red-Black Tree Properties:

- A BST
- Every Node is red or black
- The root is always black
- A red node needs have a black child
- $\forall$ path from root to a leaf has the same # of black nodes

Def: Black height $bh(x)$ = # black nodes of any path from x to a leaf excluding x.

THM A RB-tree with n-internal nodes it has

$$h \leq 2\log(n+1) = O(\log n)$$

Lemma: A subtree rooted at x has at least $2^{bh(x)}-1$ internal nodes.

Proof: Induction on h of tree

$h=0 \Rightarrow bh(x)=0 \Rightarrow 2^0-1 = \emptyset$ (internal nodes)

H depends b/r color of x and it is $2^{bh(x)}-1$ or $2^{bh(x)}-1$

nodes subtree of $= (2^{bh(x)-1}-1)+(2^{bh(x)-1}-1)+2 = 2^{bh(x)} > 2^{bh(x)-1}$

THM A RB-tree with n-internal nodes it has

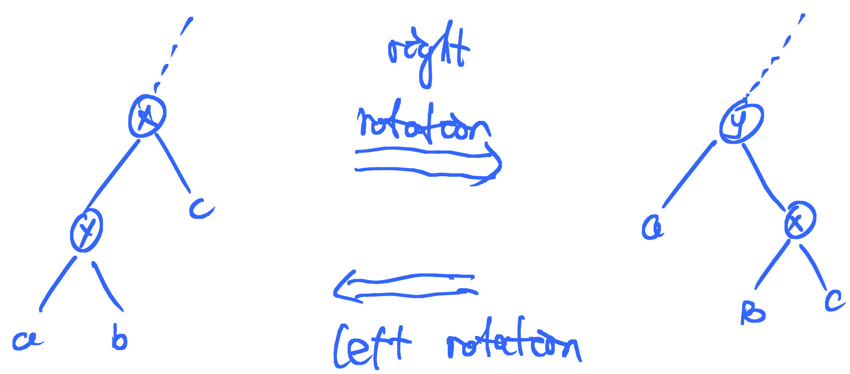
$$h \le 2\log(n+1) = O(\log n)$$

Proof

Lemma $\Rightarrow$ $n \ge 2^{bh(root)} - 1 \ge 2^{h/2} - 1$

$$h \le 2\log(n+1) = O(\log n)$$

Rotations in BSTs



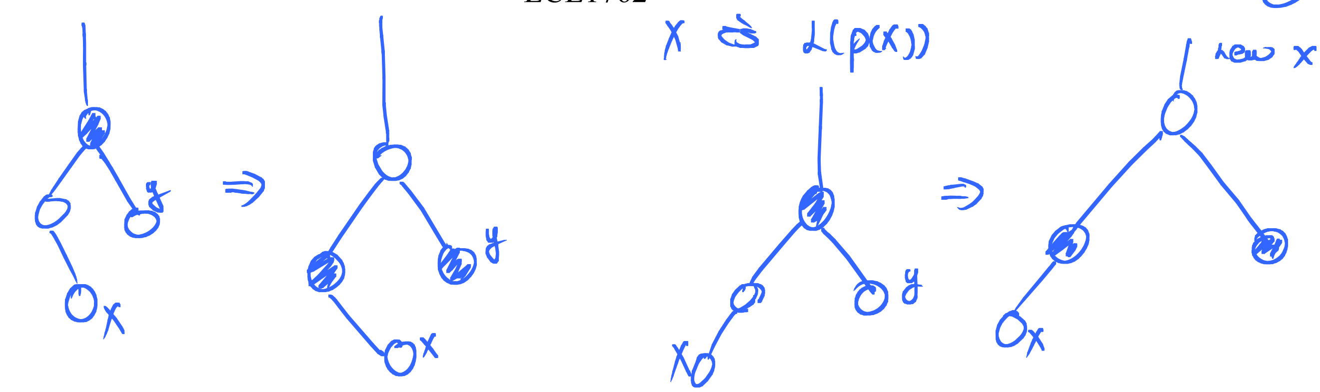
rotations preserve BST-order property

Insert = Search and Insert : rotate to fix color

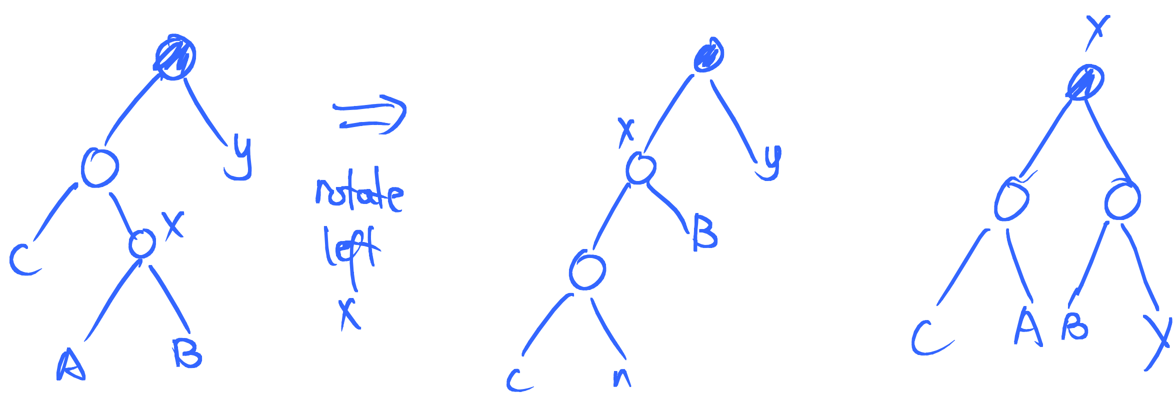
Case I:

y sibling p(x) is red

a) x is R (p(x))   b) x is L (p(x))



Case 2 Sibling y of p(x) is black x is LC to p(x) X is RC of p(x)



Hashing: Dictionary Operations Insert/Delete/Search $\simeq O(1)$

$U = SIN$

One way: Place them in array $O(1)$ times

Bad News: XXX XXX XXX

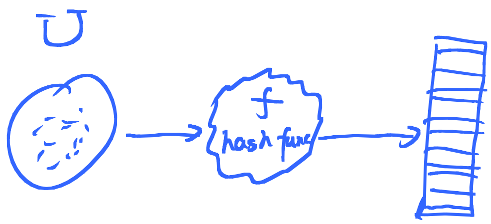
$10^9 \rightarrow$ storage

means 96% of space wasted

Motivation: Want A D.T. that achieves $\simeq O(1)$ time but uses $\simeq O(n)$ space as well

n-elements to store
m-entries long

Usually $m \geq n$ $\qquad \frac{n}{m} = \alpha =$ load factor



When $h(x) = h(y)$ for $A \neq B$ then we have a collision

1) resolution by chaining
2) open addressing

Simple Uniform Hashing

If h hashes a key / probability $1/m$ to any slot, then the expected # of probes to search is $O(1+\alpha) = O(1+\frac{n}{m})$

$$\# \text{probes} = \begin{pmatrix}\text{time to} \\ \text{calculate } h\end{pmatrix} + \begin{pmatrix}\text{time to} \\ \text{search end} \\ \text{of list}\end{pmatrix} = O(1) + \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{i-1}{m}$$

$$= O(1) + \frac{n}{n} + \frac{1}{nm}\sum_{i=1}^{n} i-1 = O(1) + \frac{(n-1)}{2} \cdot \frac{1}{nm}$$

$$= O(1) + \frac{\alpha}{2} - \frac{1}{2m} = O(1+\alpha)$$

Different Types of Hash Functions

5

Division Method

$h(k) = k \bmod m$

bad value $m = 2^p$ $p \in \mathbb{Z}$

Multiplication Method

$h(k) = \lfloor m \cdot (k \cdot A \bmod 1) \rfloor$ $\emptyset < A < 1$

when $m = 2^p$

Good Value A

$-\hat{\phi} = \frac{\sqrt{5}-1}{2}$ (inverse golden conjugate)

Universal Hashing

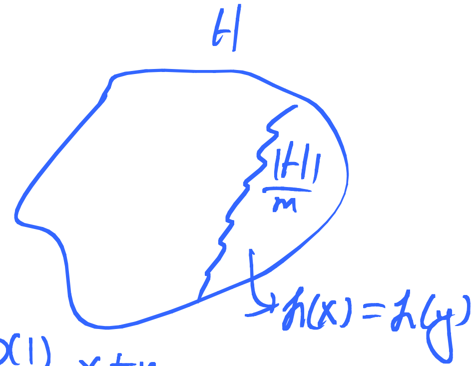
There are m^n ways to create a hash function

Which one is good? It is the one that hashes two distinct keys x & y to same bucket w/ probability $1/m$

We call H a family of universal hash functions iff the # of hash functions from H that wrap two distinct keys into the same bucket is $|H|/m$

$$Pr(h(x) = h(y)) = \frac{|H|/m}{|H|} = 1/m$$

Performance:



$$C_{xy} = \text{r.v} = \begin{cases} 1 & \text{if } x \,\&\, y \text{ collide} \\ 0 & \text{otherwise} \end{cases}$$

$$E[C_{xy}] = 1/m$$

$$E[\#\text{ collisions for } x] = \sum_{y \neq x} E(C_{xy}) = \frac{n-1}{m} = \alpha = O(1) \quad x \neq y$$

Lemma $\forall\, n > 1$ if $\gcd(a, n) = 1$ then

$$ax = b \pmod{n} \iff (ax) \bmod n = b \bmod n$$

has a unique solution for x

Ex:

$$8 \cdot x = 13 \pmod{15} \qquad x = 1$$

$$7 \cdot x = 11 \pmod{5} \qquad x = 3$$

How to create H? Pick m = prime

Decompose every key x into r-pieces

$$x = \langle x_0 \; x_1 \; \cdots \; x_{r-1} \rangle$$

$$\forall\, x_i < m$$

Let $a = \langle a_0 \; a_1 \; \cdots \; a_r \rangle$

where

$\forall\, a_i$ is selected randomly from $\langle 0, 1, 2, \cdots, n-1 \rangle$

7

Define $h_a(X) = \left(\sum_{i=0}^{r} a_i x_i\right) \bmod m$

$H = h_a \quad \forall$ choice of a, $|H| = m^{r+1}$

This is a universal hash function set. Why? Pick distinct key X & y and m log are different because $x_0 \neq y_0$. Let

$h(a) = h_a(y)$

$$\sum_{i=0}^{r} a_i x_i = \sum_{i=0}^{r} a_i y_i \pmod m \Leftrightarrow a_0(x_0 - y_0) = -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod m$$

$\frac{m^r}{m^{r+1}} = 1/m$ hash function collide x & y

QED

ECE1762 Algorithm LEC08

Amortized Analysis

Bounds defined over a sequence of n-ops

- → Aggregade
- → Accounting
- → Potential

STACK

push, pop, multipop (=empty) n ops in total

pop = push = $O(1)$

multi-pop = $O(k) = O(n)$

max: $O(n^2)$ cycles

Bit - Counter

A

0	1	0	1	1

n bits

Increment

$i = \emptyset$

while $i <$ length(A)

and $A[i] = 1$

$A[i] = \emptyset$

$i{+}{+}$

if $i \leq$ length(A) $A(i) = 1$

One Op = $O(n)$

$O(n^2)$ = n-ops

Aggregate / Stack

pop = $O(1)$
push = $O(1)$ $\Big\}$ $O(n)$

total

$O(n)$ or $\frac{O(n)}{n}$ / op $= O(1)$

on average

$n_1 + n_2 + n_3 + \cdots + n_i = O(n)$

Aggregate / Counter

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	1	0
⋮	⋮	⋮	⋮

$$\sum_{i=0}^{n} \frac{n}{2^i} \le \sum_{i=0}^{\infty} \frac{n}{2^i} = n \sum_{i=0}^{\infty} \frac{1}{2^i} = n \frac{1}{2-1} = O(n)$$

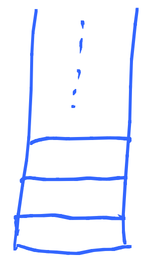
Accounting Method

Every op charges a fixed $

- Some pay for op
- Some are left as credit on ADT to pay for future ops
- You can never run on negative credit

Accounting Method / Stack

	Actual Cost	Amort Cost
push	1	\$2
pop	1	\$$\phi$
m pop	k	\$$\phi$



$O(1)$ / op

Every time we flip $\phi \to 1$

\$1 for $\phi \to 1$ and \$1 for $1 \Rightarrow \phi$

n-increament cost $2n$

or

$O(1)$ / per increament

Potential ADT has "potential" ϕ and associate op-i with actual cost c_i & amortized cost $\hat{c}_i$

$$\hat{c}_i = c_i + [\phi_i - \phi_{i-1}] \Leftrightarrow \hat{c}_i = c_i + \Delta\phi$$

$$\text{Total Cost} = \sum_i^n \hat{c}_i = \sum c_i + \Delta\phi$$

if $\Delta\phi = \phi_n - \phi_0 \geq \phi$

Then $\hat{c}_i$ is an upper bound to actual cost

Define potential ϕ_i = # elements in stack after op-i ⊕

Potential / Counter ϕ_i = #1 bits in counter.

Assume i^{th} increment resets t_i bits. Its actual cost is t_i+1

How many bits you got after i^{th} increment $= b_i$

$$b_i \leq b_{i-1} - t_i + 1$$

$$\Delta\phi = \phi_i - \phi_{i-1} \leq b_{i-1} - t_i + 1 = -t_i + 1$$

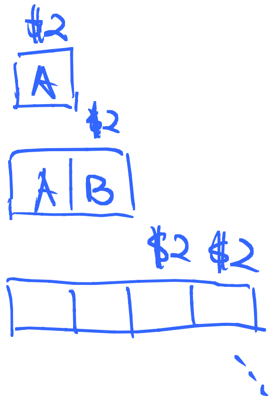
$$\hat{c} = t_i + 1 + (-t_i + 1) = 2$$

A very interesting topic on Hashing (Dynamic hashing table)

You don't know how many elements you hash a priori

$n \leq m$ load factor $\alpha = \frac{n}{m} \leq 1$

Insert: Start w/ table of size 1, when add 2^{nd} elem. make its size 2, 4, 8, 16



contract in 1/2 when 1/4 of table is filled

ECE1762 Algorithm LEC09 (1)

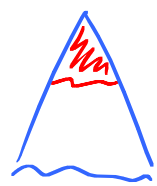
SPLAY TREES (Sleator & Tarjan, 1983)

insert
search
delete
sort

$k_1, k_2, \ldots, k_n$
$w_1, w_2, \ldots, w_n$

If know frequencies off-line then can use Dynamic Programming a statically a BST

$\frac{w_i}{w}$



more frequently

Theoretical Optimal

SPLAY TREES

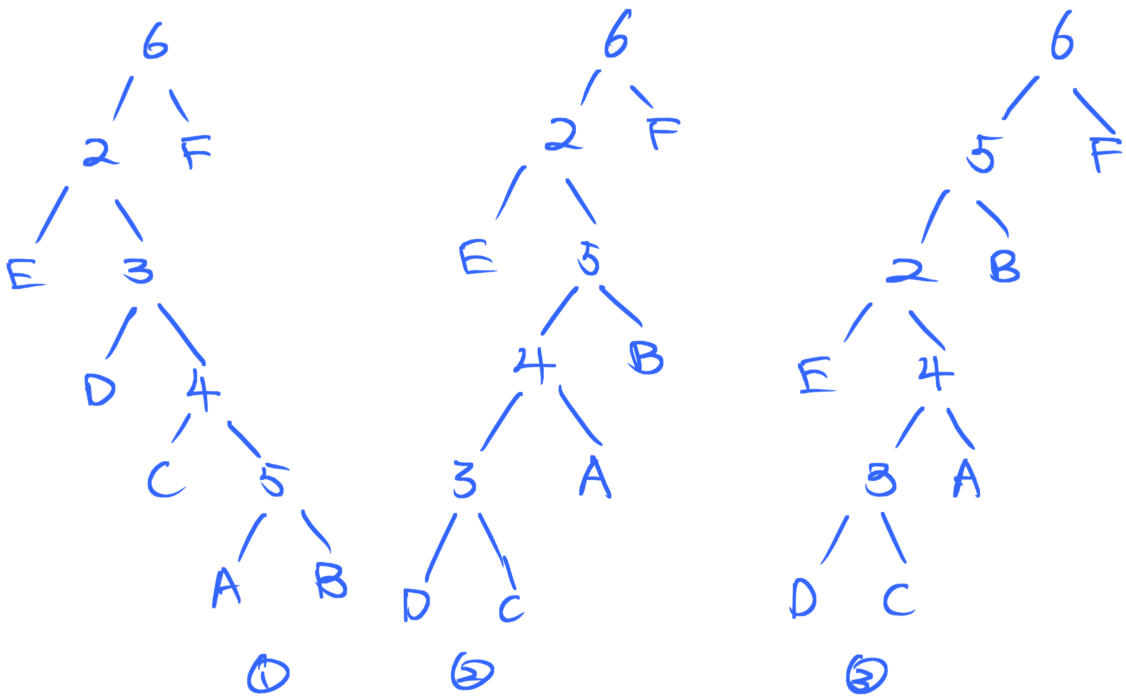
A BST tree

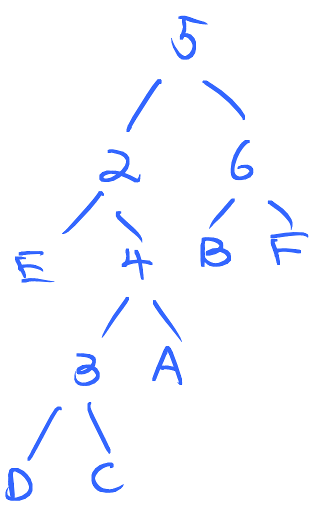
No balancing condition

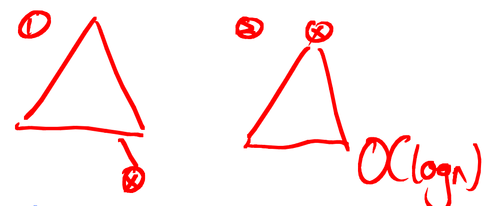
access $\sum_i \frac{w_i}{w} \log \frac{w_i}{w}$

SPLAY(x) /* rotates till node becomes tree root */

while (x) not root
- if $p(x)$ = root
  - rotate $p(x)$ } zig
- if both x & $p(x)$ are left or right children
  - rotate $p(p(x))$
  - rotate $p(x)$ } zig-zig
- else
  - rotate $p(x)$
  - rotate new $p(x)$ } zig - zag





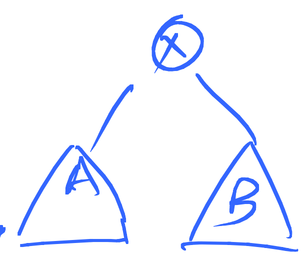


Insert — Like BST, splay (new leaf)

Search — =//=, splay (reached element) $O(\log n)$

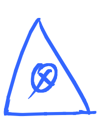
Delete — =//=, splay $p(x)$ $O(\log n)$

Join — Splay tree (A) ≤ X < splay tree (B) $O(\log n)$



Split — Splay on X & split as

$O(\log n)$



Define weight (x) = # internal nodes under x & x

$\text{rank}(x) = \lceil \log wt(x) \rceil$

Credit Invariant: Every node will rank(x) \$s on it.

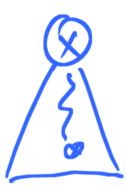
CLAIM: Every zig, zig-zag, or zig-zig operation needs at most $3(nr(x) - or(x))$ \$s except zig that may need \$1 additional

new rank

old rank

THM: if claim is true, then $O(\log n)$ \$s is enough for a splay

Let the change of ranks be rank_k (when root), rank_{k-1}, ..., ..., rank_0 (start)

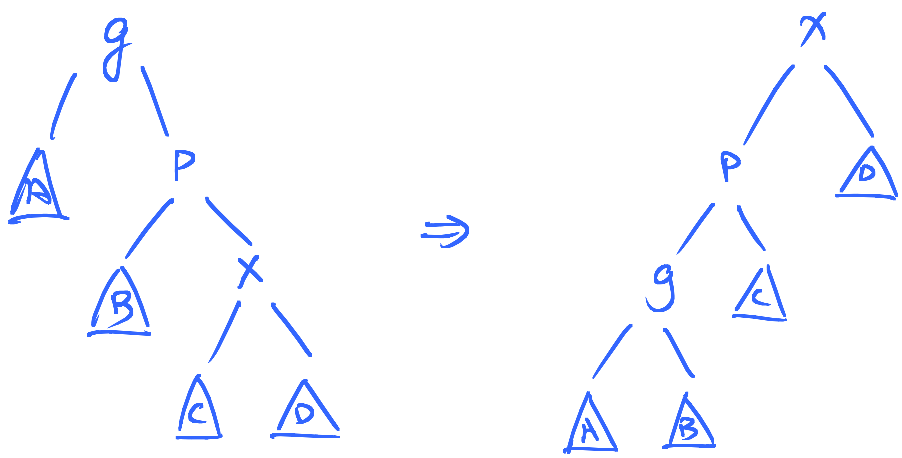


Because of claim:

$$\begin{aligned} & 1 + 3(\text{rank}_k - \text{rank}_{k-1}) + \\ & 3(\text{rank}_{k-1} - \text{rank}_{k-2}) + \\ & \cdots + \\ & 3(\text{rank}_1 - \text{rank}_0) \\ = \; & 1 + 3(\text{rank}_k - \text{rank}_0) \\ \le \; & 1 + 3(\log n - 0) \\ = \; & O(\log n) \end{aligned}$$

Excercise: SPLAY a linked list and observe how the tree balance

Need to prove the claim



Only g/p/x change ranks. To restore credit invariant we need

$nr(g) + nr(p) + nr(x) - or(x) - or(g) - or(p)$

$nr(g) \leq nr(x)$

$nr(p) \leq nr(x)$

$or(p) \geq or(x)$

$nr(x) = or(g)$

we need $2(nr(x) - or(x))$ how much \$ I need

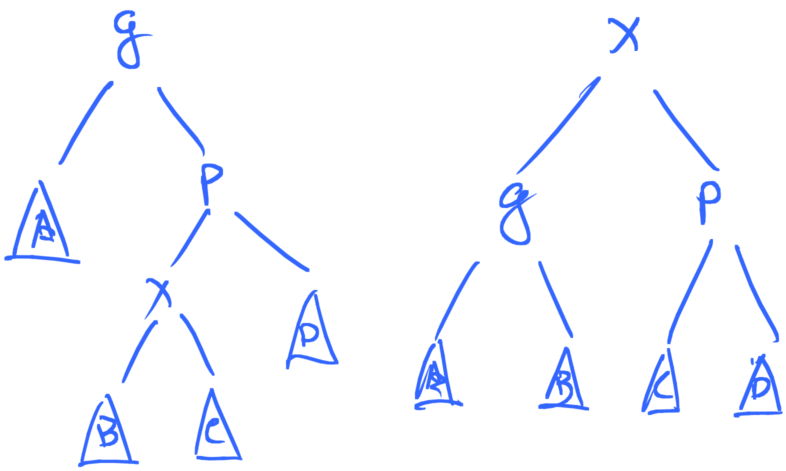
If $nr(x) = or(x) \Rightarrow$ no new \$ allocated to pay for rotation

$\Rightarrow$ > 1/2 tree nodes under x (C&D)

$\Rightarrow$ < 1/2 are on A&B

$\Rightarrow$ g drops in rank to pay for rotation

ZIG-ZAG ⑤



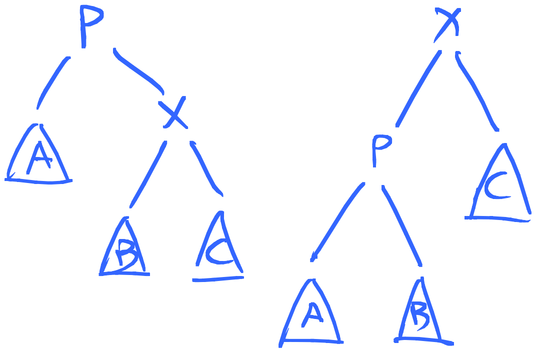
Analysis same as before except case
$nr(x) = or(x)$

More than 1/2 children under B & C

Case 1: if (i.e.) $B \gg C$
then p decreases in rank

Case 2: $B \approx C$
then p and g decrease in rank to pay for rotations

ZIG



$OR(x) \le NR(x)$
$Ol(p) \ge nr(p)$
we can deposit $nr(x) - or(x)$ on x to restore the invariant

6

The remaining two parts pay for rotation

If $nr(x) = or(x)$ then use the extra \$1 to pay for rotation (single case)

ECE1762 Algorithm LEC10

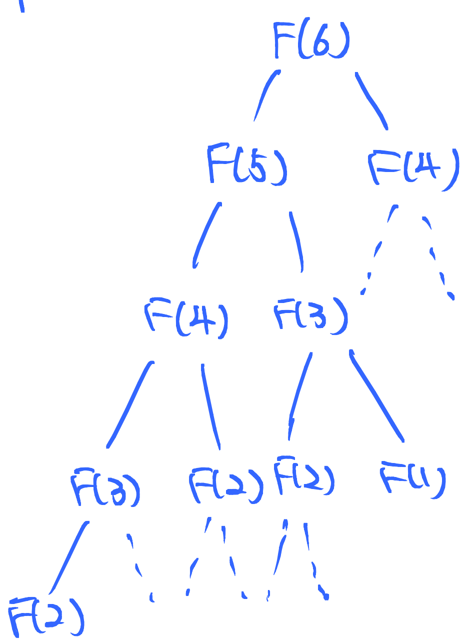
Dynamic Programming

"Divide & Conquer"

Characteristics

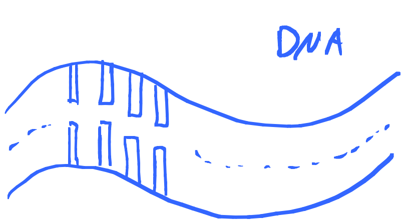
- Overlapping Sub-Problems
  (Similar to Greedy)
- Memorization

Example : find F(6)



use an array to store intermediate result

Example :



pattern matching

Matrix Multiplication (parentesization)

$$A_1 \quad A_2 \quad A_3$$
$$10\times100 \quad 100\times5 \quad 5\times50$$

$(A_1 \; A_2)\; A_3 = (10\times100\times5) + 10\times5\times50$
$= 7500$

$n\times m \quad m\times k$

need $n\times m\times k$ scalar muls

$A_1\,(A_2 \; A_3) = 10\times5\times50 + 10\times100\times50$
$= 75000$

Given a set of matrices $A_1 \; A_2 \; \cdots \; A_n$ to be multiplied in that order. Find the parenthesization that minimizes the # of scalar muls

Naive Approach

$\Omega(4^n/n^{3/2})$

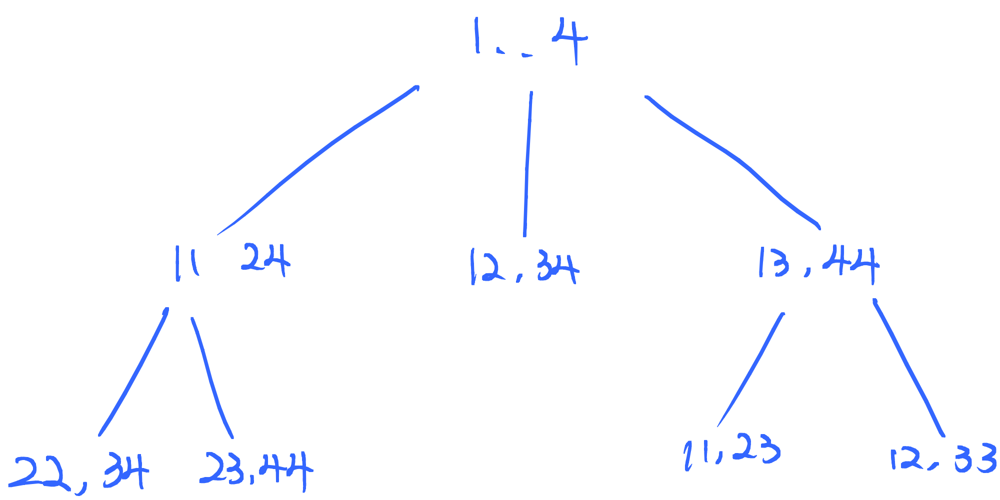
$$P(n) = \sum_{k=1}^{n-1} P(k)\,P(n-k) = \Omega(4^n/n^{3/2})$$

the min # of muls to

Find $A_i \; A_{i+1} \cdots A_j$

$$m[i,j] = \begin{cases} \phi & i=j \\ \min\limits_{i\le k<j} \{m[i,k] + m[k+1,j] + P_{i-1}P_kP_j\} & \text{if } i<j \end{cases}$$

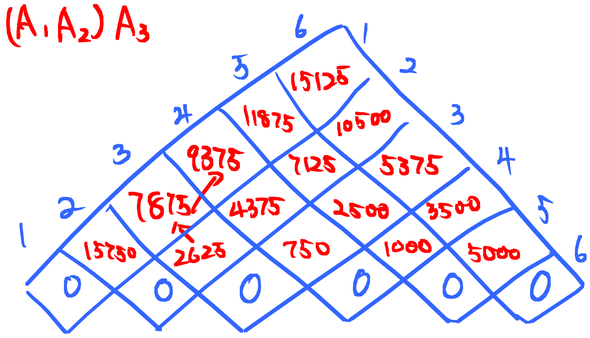
3



A_1	30	35
A_2	35	15
A_3	15	5
A_4	5	10
A_5	10	20
A_6	20	25

m[1,3]

$A_1(A_2A_3)$ $(A_1A_2)A_3$



Dynamic

$O(n^2) * O(n) = O(n^3)$

square per square

4

LONGEST COMMON SUBSEQUENCE

Given two strings

$$X = x_1 x_2 \cdots x_m \quad \& \quad Y = y_1 y_2 \cdots y_n$$

and you want to find longest sequence of not necessarily consecutive characters but in that order

s p r i n g t i m p

p i o n e e r

If $m \le n$

$n \cdot 2^m$ $\quad O(n \cdot 2^m)$ Very Bad.

THM: Let $Z = z_1 z_2 z_3 \cdots z_k$ be a LCS of X & Y

A) if $x_m = y_n$ then Z_{k-1} is LCS of X_{m-1} & Y_{n-1}

B) if $x_m \neq y_n$ and $z_k \neq x_m$ then

Z is a LCS of X_{m-1} & Y_n

C) if $y_n \neq x_m$ and $z_k \neq y_m$ then

Z is LCS of X_m & Y_{n-1}

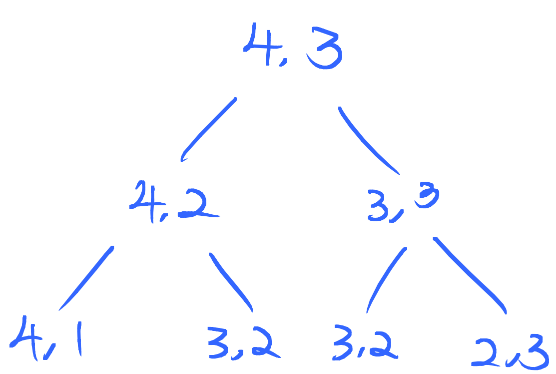
Proof for A) ③

Assume A) is not true

$$\underline{|'Z_1| \cup z_k} > |z_{k-1}| \cup z_k$$

LCS for $\bar{X}$ & $\bar{Y}$ longer than Z a contradiction!

$$C(i,j) = \begin{cases} \emptyset & \text{if } i=\emptyset \text{ or } j=\emptyset \\ C[i-1, j-1]+1 & \text{if } i,j>\emptyset,\ x_i = y_i \\ \max\{C[i-1,j], C[i,j-1]\} & \text{if } x_i \neq y_j \end{cases}$$



Example

		a	m	p	u	t	a	t	i	o	n
		0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	1	1	1	1	1	1	1	1
a	0	1	1	1	1	1	2	2	2	2	2
n	0	1	1	1	1	1	2	2	2	2	3
k	0	1	1	1	1	1	2	2	2	2	3
i	0	1	1	1	1	1	2	2	3	3	3
n	0	1	1	1	1	1	2	2	3	3	4
g	0	1	1	1	1	1	2	2	3	3	4

Binomial Coefficient $\quad C(n,k) = \binom{n}{k}$

$C(n,0) = 1$

$C(n,n) = 1$ $\quad$ & $C(n,k) = C(n-1,k-1) + C(n-1,k)$

Use a table to memorize.

Optimal Polygon Triangulation

You are given a convex polygon with vertices $v_0, v_1, \ldots, v_n$ where $v_0 \equiv v_n$ and you want to divide into triangles using chords minimizing the Euclidean Distance

$$\text{MIN} \left[w(\triangle v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i| \right]$$

of all triangles

Matrix Multiplication = POLYGON TRIANGULATION

Table $O(n^2)$ $\quad$ Time $O(n^3)$

$$t[i,j] = \text{weight of optimal triangulation for vertice} = \begin{cases} \emptyset & \text{if } i = j \\ \min\limits_{i \le k \le j-1} \left\{ t[i,k] + t[k+1,j] + w(\triangle v_{i-1} v_k v_j) \right\} & \text{if } i < j \end{cases}$$

GREEDY ALGORITHM

- Overlapping Optimal Sub Problems

- Greedy Principle: at every step, do greedy selection to move to the next step

Notes: Good for minimization and maximization problems
Based on theory of Matroids
Good Approximate solutions to NP Complete Problems

Optimal Class Scheduling Problem

1. A single room and a set of classes to schedule
Maximize the # of classes to be scheduled.
Algorithm: Sort by finish time, select the one finish the earliest. $O(n\log n)$

Proof for optimality: Assume a contradiction that greedy $c_1, c_2, \dots, c_n$ is not optimal but $g_1, g_2, \dots, g_k$ is optimal

$$c_1, \quad c_2, \quad \dots, \quad c_n$$
$$\downarrow \quad \downarrow$$
$$g_1, \quad g_2, \quad \dots, \quad g_n, \underbrace{\dots, g_k}$$

Example: A thief that enters a store with n items (8)

Item i cost v_i and weigh w_i. The thief can take a maximum of w.

① 0-1 version, take or not take [dynamic]

② functional, can take a portion [greedy]

Problem: maximize the profit

$$C[i,w] = \begin{cases} \phi & \\ C[i-1,w] & \text{if } w_i > w \\ \max(C[i-1,w],\ C[i, w-w_i]+v_i\} & \text{if } w_i \le w \end{cases}$$

Assume the item is sorted in values

$O(n)$

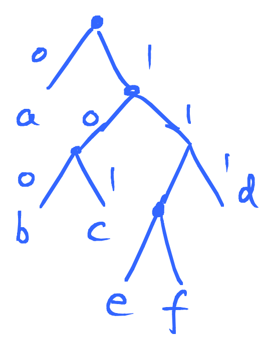
ECE1762 Algorithm LEC11 ①

Huffman encoding for data compression

A file with frequency of characters: $T(B) = f(c_i)\, k_i$

	a	b	c	d	e	f
$f(c)$	45%	13%	12%	16%	9%	5%

Running Time



Greedy Principle: Assume optimal tree where the two lowest frequency keys x & y are the leaves in the deepest of the tree.

Optimal Substructure: Suppose you get an optimal tree, building on top of it gives another optimal tree.

2

Lemma 2.1 *Suppose C is the optimal code for S, p_1, p_2 and l_1, l_2 are the probabilities and code lengths of messages s_1 and s_2, respectively. Then $p_1 > p_2 \Rightarrow l_1 \leq l_2$.*

Proof. Suppose $p_1 > p_2$ and $l_1 > l_2$, we swap the code words for s_1 and s_2, and get a new code C'. The length of C' is $l_a(C') = l_a(C)+p_1(l_2-l_1)+p_2(l_1-l_2) = l_a(C)+(p_1-p_2)(l_2-l_1) < l_a(C)$. This contradicts the optimality of code C. ■

Lemma 2.2 *Without loss of generality, the two messages of smallest probability occur as siblings in the code tree for an optimal code.*

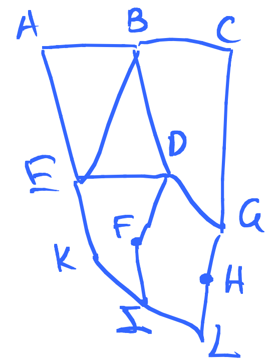
Proof. Given the code tree for an optimal code, we will show that it can always be modified without increasing the average code length so that the two smallest probability nodes are siblings. From Lemma 2.1, the smallest probability node must occur at the largest depth in the code tree. Note that the sibling of this node is also at the same depth. Now the sibling can be swapped with the second smallest probability node to obtain a code tree of the desired structure. This transformation does not increase the average code length. ■

Graph Algorithms

$G = (V, E)$ V: vectors e: edges

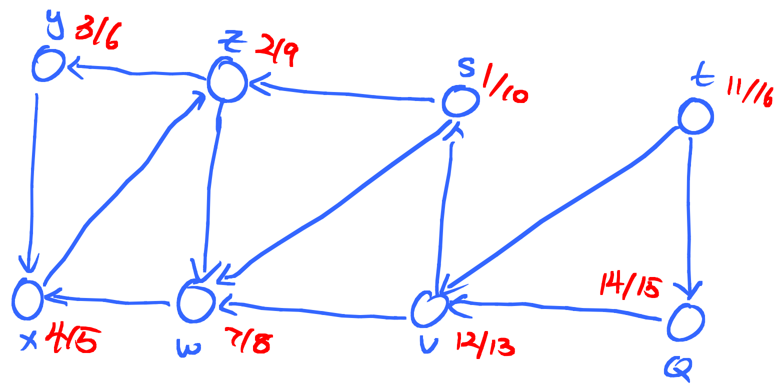
$E = O(V^2)$ worst case clique

Breath First Search



Time $O(V+E)$

Depth First Search

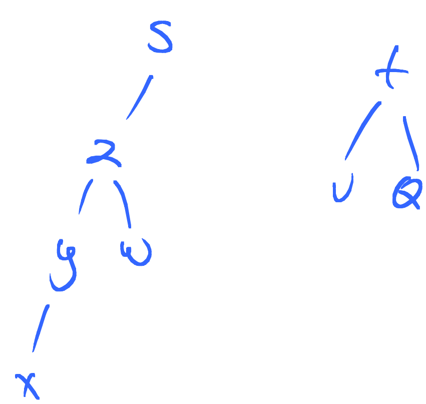


Discovery / Finish

Use a stack

① Create Depth First Forest

Time $O(V+E)$



Edge classifications in a DFS

Tree edges

Back edges: to an ancestor

Fwd edges: to a descendant

Cross edges: between different trees

DFS Algorithm

```
preorder(node v)
{
visit(v);
for each child w of v
    preorder(w);
}
```

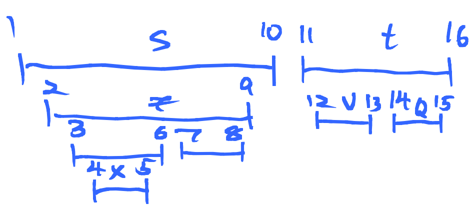
```
dfs(vertex v)
{
visit(v);
for each neighbor w of v
    if w is unvisited
    {
    dfs(w);
    add edge vw to tree T
    }
}
```

Theorem: In the DFS Forest of an undirected graph G every edge is a tree edge or a back edge

Fwd → Back edges

Cross → Tree edges

DFS Parentheses THM (Well Parenthesized)



two edges are either disjoint or fully contained.

every two vertices

u & v you have

[d(u), F(u)] & [d(v), f(v)]

are either completely disjoint or one contained within the other.

5

Assume $d[Q] < d[U]$ compare

$d[U]$ & $F(Q)$

$\geq$
$\leq$

TOPOLOGICAL SORT

Given a DAG Find a total order of the vertices

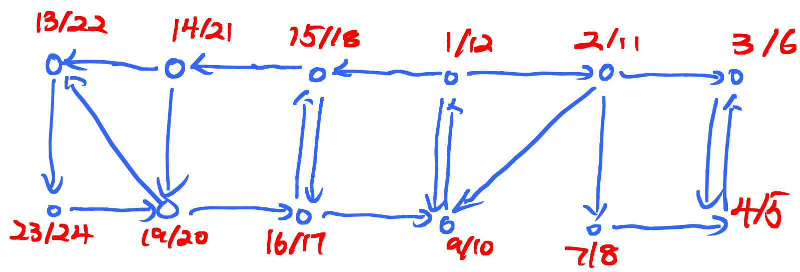
Directed Acyclic Graph

Note: topological sort is not unique

Strongly Connected Components

6

In a directed graph, a SCC is a maximal subset $V' \subseteq V$ st $\forall\, Q, v \in V'$ $\exists$ path $Q \rightsquigarrow v$ & $v \rightsquigarrow Q$



Algorithm:

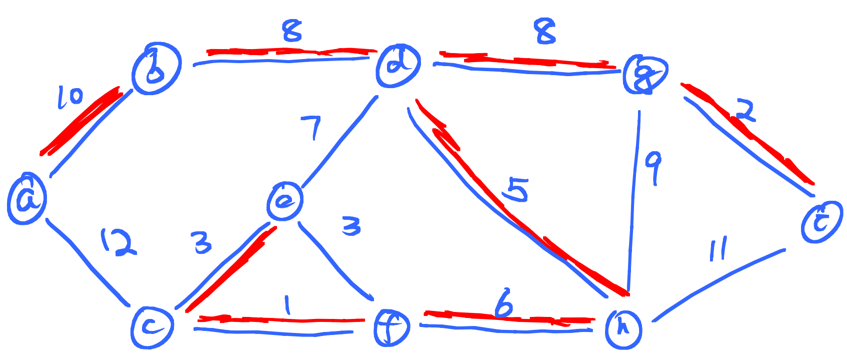
- Run DFS
- Transpose $G \Rightarrow G^T$
- Run DFS (G^T) but pick vertices in decreasing finish time
- Each Forest Tree, is a SCC

Running Time $O(V+E)$ w/ adjacency matrix

ECE1762 Algorithm LEC12 ①

Minimum Spanning Tree

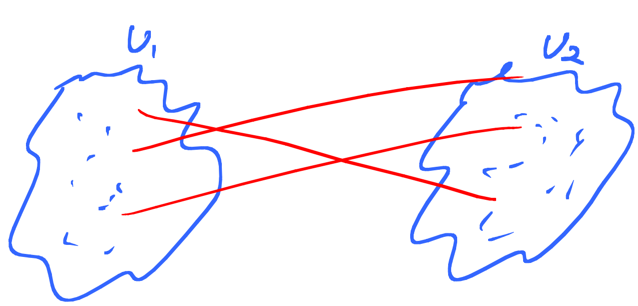
Given a weighted connected, undirected $G(V,E)$ with $w(u,v)$ weights on edges.
Find a tree that minimizes $w(T) = \sum_{(u,v)\in T} w(u,v)$



MST is not unique — weights can be gain, costs, distance etc.

Given a partial MST T_p, a safe edge is an edge that can be added to T_p and give a "bigger" partial MST

We define a cut (V_1,V_2) on $G=(V,E)$ to be partition of the graph s.t. $V_1 \cup V_2 = V$ & $V_1 \cap V_2 \neq \emptyset$



Given a cut, an edge (Q, v) crosses the cut if ②

$v \in V_1$ & $v \in V_2$ (or vice versa)

An edge crossing a cut is called light if it has the minimum weight along all edges crossing the cut

THM: Given a cut, a light edge is a safe edge

Let A be part of MST and consider the cut $V_1 = A$ &

$V_2 = V - A$

Let (u, v) be a light edge crossing the cut. Assume towards a contradiction it is not a safe edge but (x, y) is, where weight $w(u, v) < w(x, y)$

Add to Final MST (u, v) creating a cycle, and break cycle by removing (x, y). You get a new tree with lower weight, contradicting that the original was part of MST.

Prims (Greedy)

- Insert every key in queue Q with ∞ weight
- Decrease(Q, root, $\emptyset$)

- while $(Q \neq \emptyset)$

  Extract Min $(Q) \to Q$

  $\forall$ v adjacent

  if $w(Q, v) < key(v)$

  decrease $(Q, v, w(u,v))$

Can use a heap to implement, running time

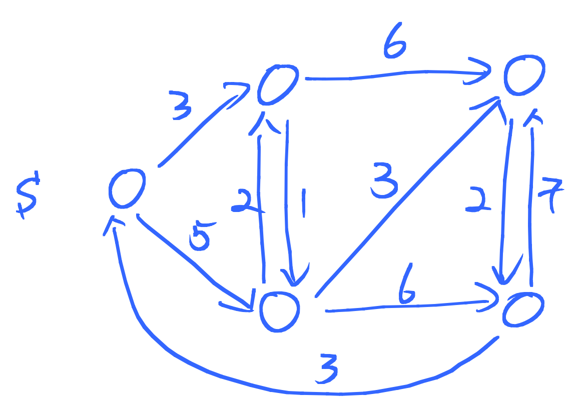
$$E\log v + V\log v$$
$$= E\log v$$

with Fibonacci Heap $O(v\log v + E)$

Single Source Shortest Path

Given a directed $G=(V,E)$ with weight function

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\} & \text{if } \exists p : u \xrightarrow{p} v \\ \\ \infty & \text{otherwise} \end{cases}$$



Notes:

- Numeric value: weights, cost, profit

- can have negative weights but not cycle

4

Dijkstra: only positive weights (Greedy)

Bellman-Ford: works for negative weights & detects negative cycles

SSSP: Single Source Single Destination

Single destination SPs, all pair SPs (Dynamic Programming)

$\delta(u, v) = SP \quad u \overset{SP}{\rightarrow} v$

The Algorithms will always keep track an estimate $d[v]$ of SP. Always $d[v] > \delta(v, v)$ and they will keep improving the estimate.

RELAX $(u, v, \omega(v, v))$

If $d[v] > d[u] + \omega(v, v)$

$d[v] = d[u] + \omega(u, v)$

$\pi(v) = u$

Initially

$d[v] = \infty$

(optimal substructure) 5

Every Sub-Path of a shortest Path (SP) is also a SP

(Triangle Inequality) For Relaxing Edge (u, v)

$\delta(s, v) \leq \delta(s, u) + \omega(u, v)$ & $d[v] \leq d[u] + \omega(u, v)$

(Upper Bound) As you relax edges always $d[v] \geq \delta(s, v)$ and once $d[v] = \delta(s, v)$ it changes

(No Path) if $\delta[s, v] = \infty$ then $d[v] = \infty$

(Convergence) if $s \rightsquigarrow Q \to v$ is a SP and $d[Q] = \delta(s, u)$ after relax (u, v) you get $d[v] = \delta(s, v)$

Path Relaxation:

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a SP. If we relax $\langle v_0, v_1 \rangle$, $\langle v_1, v_2 \rangle \cdots \langle v_{k-1}, v_k \rangle$ to that order, even mixed with other relaxations in between, at the end we get

$$d[v_k] = \delta(s, v_k)$$

"Cut & Paste" 6

$$\underbrace{u \rightsquigarrow x \rightsquigarrow v}_{SP}$$

Suppose $\exists$ $u \to x$ not SP, but there is another one $\to$ cut & paste, you get a shorter SP $u \to v$ a contradiction

Let v be the first vertex where relaxing (u, v) makes $d[v] < \delta(s, v)$

$$d[v] < \delta(s, v) \leq \delta(s, u) + w(u, v) \leq d[u] + w[u, v]$$
$$= d[v]$$

A contradiction

No path : $d[v] \geq \delta(s, v)$

Convergence: $d[v] \leq d[u] + w(u, v)$

$= \delta(s, v)$ by optimal substructure

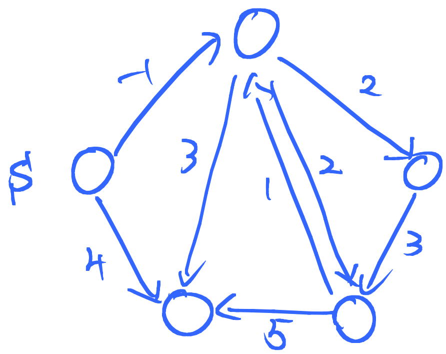
$d[v] \geq \delta(s, v)$ $\not>$ $d[v] = \delta(s, v)$

Run induction over the index can prove

$$v_0 \to v_1 \to v_2 \searrow v_3 \nearrow v_4 \cdots v_k$$

7

Bellman-Ford Algorithm works on negative weights and negative cycles



for $1 \ldots |V|-1$

$\forall$ edge $(u, v) \rightarrow E$

relax(u, v)

for $\forall$ edge $(u, v) \in E$

if $d[v] > d[u] + w(u, v)$

return false/negative cycle

Proof for correctness: if no negative cycle

- Find SPs due to path relaxation
- $d[v] = \delta(s, v) \leq \delta(s, u) + w(u, v) = d[u] + w(u, v)$

if negative cycle $v_0 v_1 \cdots v_k$ then exit

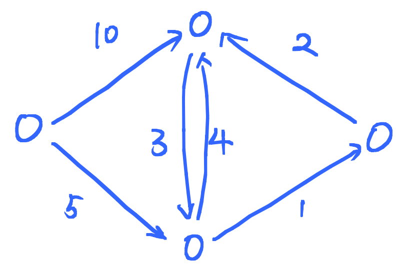
Contradiction

$$d[v_1] \leq d[v_0] + w(v_0, v_1)$$
$$d[v_2] \leq d[v_1] + w(v_1, v_2)$$
$$\vdots$$
$$d[v_k = v_0] \leq d[v_{k-1}] + w(v_{k-1}, v_k)$$

$$0 \leq \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$ no negative cycle

Dijkstra / non-negative weights / generalization of BFS to weights ⑧

$S = \emptyset$; $Q = V$ (priority queue)



while $Q \neq \emptyset$

$u = \text{extract\_min}(Q)$

$S = S \cup \{Q\}$

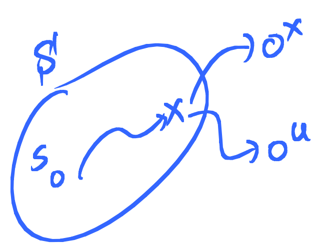
$\forall$ vertex adj to a

Relax(u, v)

$O(E \log V)$ w/heaps

Correctness: Need to show that when u is added in S, $d[u] = \delta(s,u)$

By contradiction, let u be the first vertex where $d[u] \neq \delta(s,u)$ and also assume $\delta(s,Q) \neq \infty \Rightarrow \exists$ sp from $s \overset{sp}{\rightsquigarrow} u$



Let y be the first on this sp to u which is not in S

Let x be the vertex in S where $x \to y$

Decompose $s \overset{sp}{\rightsquigarrow} u \Rightarrow s \overset{sp}{\rightsquigarrow} p \to y \overset{sp}{\rightsquigarrow} x$

Claim $d[y] = \delta(s,y)$ when u added into S since u is the first added in S where $d[u] \neq \delta(s,u)$

For x it holds, that is $d[x] = \delta(s,x)$ and by convergence, it holds for y, that is $d[y] = \delta(s,y)$

we got $d[y] = \delta(s,y) \leq \underbrace{\delta(s,u) \leq d[u]}_{\text{upper bound}}$

9

But y & u are outside S and u is selected

$$\Rightarrow d[u] \leq d[y]$$

$$\Rightarrow d[u] = d[y]$$

$$\Rightarrow \delta(s,u) = d[u]$$

Single Source DAGs

- Topologics Sort Vertices $O(V+E)$
- Relax Edges in that topological sort order
- Correctness due to path relaxation

ECE1762 Algorithm LEC13 ①

Difference Constraints

$A_{m \times n}$ $B_{n \times 1}$ $C_{n \times 1}$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

want a solution for $\bar{X}$ that minimizes or maximizes objectives function

unknowns

$$[c_1 \cdots c_n] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- Ellipsoid
  Poly But runs slow
- Simplex
  Exp But Fast

has one 1
one -1

- $\forall$ unknown introduce a vertex
- for $x - y \leq b$ introduce $y \xrightarrow{b} x$
- Introduce a pseudo source with edges pointing to all vertices w/ weight zero
- Run Bellman-Ford
  - SPs are the solutions

- If Neg Cycle $\Rightarrow \nexists$ no feasible solution

Example

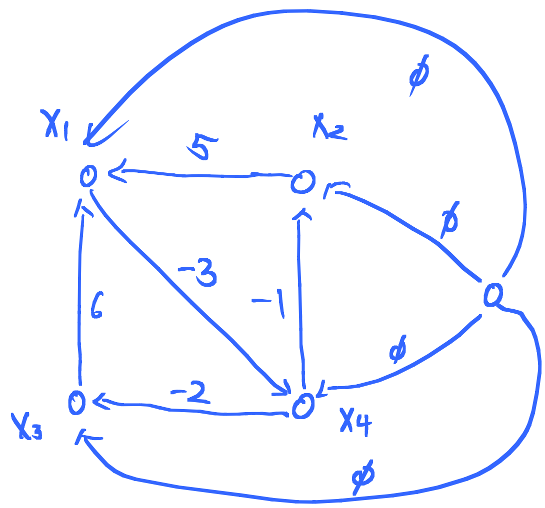
$x_1 - x_2 \le 5$

$x_1 - x_3 \le 6$

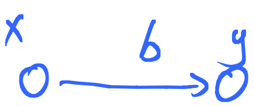
$x_2 - x_4 \le -1$

$x_3 - x_4 \le -2$

$x_4 - x_1 \le -3$



- $\nexists$ Neg Cycle $\Rightarrow$ gives correct answer



$$\delta(s, y) \le \delta(s, x) + w(x, y)$$

$$y \le x + b$$

$$y - x \le b$$

- $\exists$ neg cycle $\Rightarrow \nexists$ no feasible solution

$$x_0 - x_1 \le b_1$$
$$x_1 - x_2 \le b_2$$
$$\vdots$$
$$x_{k-1} - x_0 \le b_k$$

$$\overline{\phi \le \Sigma \text{ cycle}}$$

A contradiction

All Pairs Shortest Paths

③

Suppose weights are stored in a matrix $D^{(0)}$

$D^{(k)}$ = distance from every vertex by following at most k-edges

We would like to compute $D^{(n-1)}$

$D^{(m)}$

$d_{i,j}^{(m)} \rightsquigarrow$ the distance = min $\left\{ d_{i,j}^{(m-1)}, \min_{1 < k \leq n} \left\{ d_{i,k}^{(m-1)} + w(k,j) \right\} \right\}$

the shortest distance between vertices (i,j) w/ at most m edges

$m \leq n-1$

$= \min_{1 < k \leq n} \left\{ d_{ik}^{(m-1)} + w(k,j) \right\}$ because $w_{jj} \neq 0$

4

For $i=1..n$
  For $j=1..n$
    $d_{ij}=\infty$
$O(V^3)$ For $k=1..n$
      $d_{ij}=\min\{d_{ik}+w_{kj}\}$

Need run this $(n-1)$ times to get $D^{(1)}\cdots D^{(n-1)}$

Total $O(V^4)$

Matrix Mult $(A,B)\to C$

For $i=1..n$
  For $j=1..n$
    $C_{ij}=\phi$
    For $k=1..n$
      $C_{ij}=C_{ij}+a_{ik}b_{kj}$

Strassen

$O(n^{2.81\cdots})$

The best performance so far

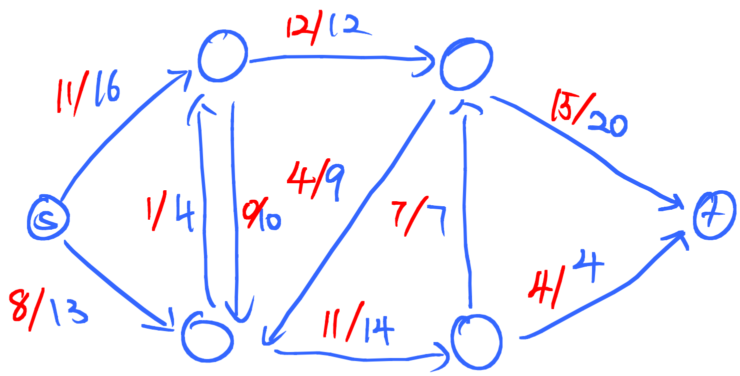
$D^{(1)}\to D^{(2)}\to\cdots\to D^{(8)}$

Need run this $(n-1)$ times to get

$D^{(1)}\cdots\cdots D^{(n-1)}$

Total $O(V\cdot V^{2.81})\to O(V^{2.81}\cdot\log V)$

Maximum Flow



Observation

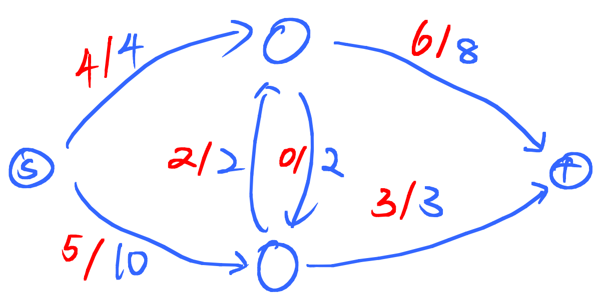
$\forall$ vertex $- \{s, t\}$

Σ Flow Outgoing

$\|$

Σ Flow Incoming

Max Flow / Min Cut



Defn : Directed weighted Graph with two special vertices, a source s and a sink t. Weights are edges capacities

Imagine : edges are pipes & tunnel through them you can transfer Fluid from source

Some Definitions

Directed, Connected, Positively weighted graph G where weights denote capacities (u, v)

If $(u,v) \notin G$ then assume it does exist with capacity zero we have source/sink Flow network 6

Capacity constraint: $\forall (u,v) \in E \; f(u,v) \leq C(u,v)$

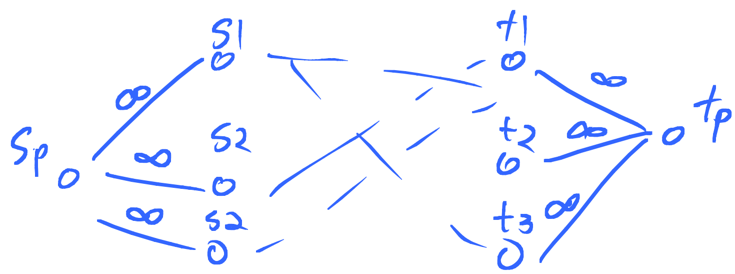
Skew symmetry: $\forall (u,v) \in E$ we have $F(u,v) = -F(v,u)$

Flow Conservation: $\forall u - \{s,t\}$ we get $\sum_{v \in V} f(u,v) = \emptyset$

no leakage. Net flow $(u,v) = F(u,v)$

Problem: Maximize $|f| = \sum_{v = V} f(s,v)$

Observation: Multi-Source and Multi-Sink



add pesudo s and t with ∞ capacity

Def: define flow between sets of vertices as we did between vertices

$$F(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$$

Lemma

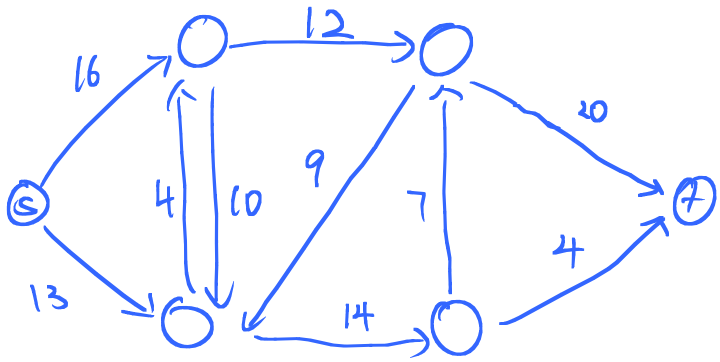
- $F(\bar{X}, \bar{X}) = \emptyset$
- $F(\bar{X}, \bar{Y}) = -F(\bar{Y}, \bar{X})$
- $F(\bar{X} \cup \bar{Y}, \bar{W}) = f(\bar{X}, W) + F(\bar{Y}, \bar{W})$
- $F(\bar{W}, \bar{X} \cup \bar{Y}) = f(\bar{W}, \bar{X}) + F(\bar{W}, \bar{Y})$

Ex: $|F| = F(s, V) = F(V, t)$

"All unit leaving source they do go to sink"

$$|F| = F(s, V) = F(V, V) - F(V-s, V)$$
$$= \cancel{F(V, V)}^{0} + F(V, V-s)$$
$$= F(V, t) - F(V, V-s-t)$$
$$= F(V, t) + \cancel{F(V-s-t, V)}^{0}$$
$$= F(V, t)$$

8



Residual Capacity of edge $c_F(u, v)$

$$c_F(u, v) = c(u, v) - F(u, v)$$

Residual Network $G_F(V, E_F)$

A Graph on some set of vertices, with edge-set

$E_F = \{(u, v) \in V^2 : c_F(u, v) > \emptyset\}$

Augmenting Path in G_F: a simple path $s \overset{P}{\rightsquigarrow} t$ in G_F

Residual Capacity of an augmenting path.

$$c_F(P) = \min\{c_F(u, v) \text{ for } (u, v) \in P\}$$

Lemma:

If F' is a flow in G_F then

$(F+F')(u, v) = F(u, v) + F'(u, v)$ is a Flow in G

with value $|F+F'| = |F| + |F'|$

9

Skew Symmetry: $(F+F')(u,v) = F(u,v) + F'(u,v)$

$$= -F(v,u) - F'(v,u) = -(F+F')(v,u)$$

Capacity Constraints $F'(u,v) \leq C_F(u,v)$

$$(F+F')(u,v) = F(u,v) + F'(u,v) < F(u,v) + C_F(u,v)$$

$$= F(u,v) + C(u,v) - F(u,v) = C(u,v)$$

Flow Conservation

$$\sum_{v \in V} (F+F')(u,v) = \sum_{v \in V} F\cancelto{\emptyset}{(u,v)} + \sum_{v \in V} \cancelto{\emptyset}{F'(u,v)}$$

$$= \emptyset$$

Size of New Flow: $\sum F(s,v) + \sum F'(s,v) = |F| + |F'|$

Ford - Fulkerson

Initialize $F(u,v) = \emptyset$
repeat

- Compute G_F
- Find an augmenting Graph P_A in G_F
- Add the residual capacity of P_A in initial Flow
- Until $\nexists$ augmenting path in G_F

ECE1762 Algorithm LEC14

Max Flow: Directed Weighted $C(u,v)$ = capacity graph of

no $(u,v) \in E \Rightarrow C(u,v) = \emptyset$

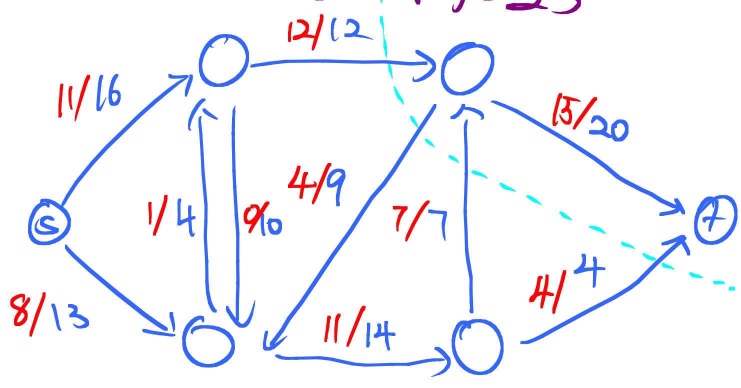
Capacity Constraint $f(u,v) \le C(u,v)$

SKEW SYMMETRY $f(u,v) = -f(v,u)$

FLOW CONSERVATION $\sum_{v \in V} f(u,v) = \emptyset \quad u \in \{s,t\}$

Maximize $|F| = \sum_{v \in V} f(s,v)$

$(S, T) = 23$

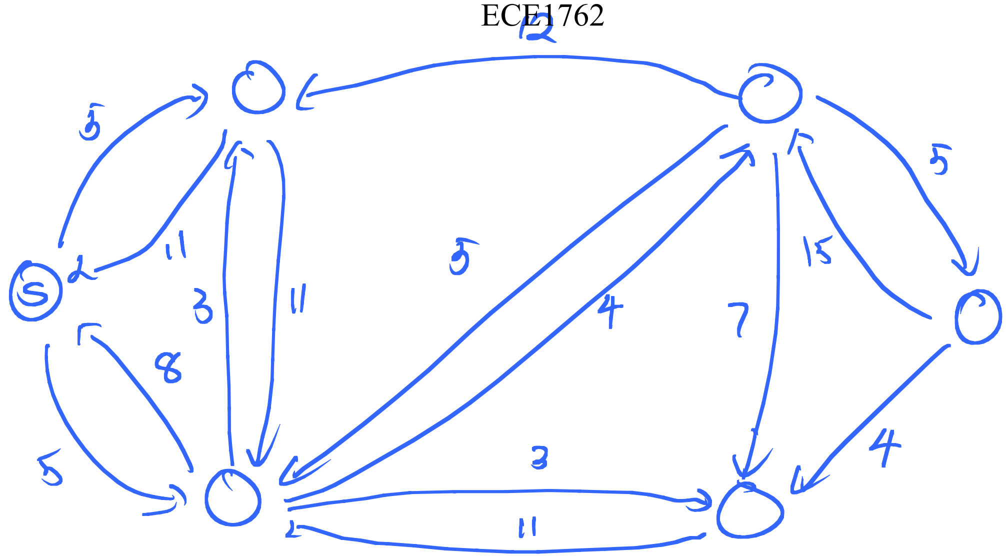


Capacity of Residual Edge

$$C_f(u,v) = C(u,v) - f(u,v)$$

Residual Network

$$G_F(V, E_F) \qquad C_F(u,v) > \emptyset$$



Augmenting Path P: A simple Path $s \overset{p}{\rightsquigarrow} t$ in G_F

Residual Capacity P: $C_F(P) = \min\{C_F(u,v) : \text{edge } (u,v) \text{ on } P\}$

LEMMA: Let p path in G_F. Then

$$F_p(u,v) = \begin{cases} C_F(P) & \text{if } (u,v) \in P \\ -C_F(P) & \text{if } (v,u) \in P \\ \emptyset & \text{otherwise} \end{cases}$$

is a Flow where $|F| = |C_F(p)| > \emptyset$

To Prove: Prove the three properties one by one

Corollary: $|F| + |F_p| = |F_{new}| > |F|$

③

Ford Fulkerson

1) $\forall$ edge (u,v) set $f(u,v) = f(v,u) = \emptyset$

2) while $\exists$ augmenting path in G_F: $s \overset{p}{\rightsquigarrow} t$ → $O(E)$ with BFS

3) $C_F(p) = \min\{C_F(u,v) : (u,v) \in p\}$

4) For Every $(u,v) \in p$ do ← $O(E|F_{max}|)$

$$F(u,v) = F(u,v) + C_F(p)$$
$$F(v,u) = -F(u,v)$$

Proof of Correctness

Lemma: Let f be a Flow and (S,T) a cut. The $F(S,T) = |F|$

$$F(S,T) = F(S,V) - \overset{0}{\cancel{F(S,S)}} = F(S,V) - F(\overset{\emptyset}{\cancel{S-s}}, V)$$
$$= F(s,V) = |F|$$

Corollary: $\forall$ Flow f: $|F| \le c(S,T)$ For any cut (S,T)

$$|F| = F(S,T) = \sum_{u \in S}\sum_{v \in T} f(u,v) \le \sum_{u \in S}\sum_{v \in T} c(u,v) = c(S,T)$$

MAX FLOW MIN CUT THEOREM

$\forall$ flow on G, Following statements are equivalent

4

1) F is a max-flow in G

2) G_F has no augmenting path

3) $|F| = c(S,T)$ for some cut (S,T)

1) ⇒ 2) because you can add it & increase value of Flow

2) ⇒ 3) Since G_F has no AP, Let $S = \{$ vertices reachable in G_F from $s \}$

Clearly $(S, V-S-T)$ is a cut.

Now Every (u,v) st. $u \in S$ & $v \in T$

we got $F(u,v) = c(u,v) \Leftrightarrow \sum f(u,v) = \sum c(u,v)$

$\Leftrightarrow |F| = F(S,T) = C(S,T)$

3) ⇒ 1) By definition $|F| \le c(S,T)$ by def, because its equal it has to be maximum.

Lemma: Value MAX_FLOW = CAPACITY of MIN_CUT

$|f| \le C_1 \le C_2 \le C_3 \cdots \le C_k$

Edmonds - Karp

In line 2 select the shortest AP $O(\underbrace{VE}_{\text{\# of iterations}}\text{-}E)$ of minimum Length

Lemma: If you run E+K algorithm, the SP $\delta_F(s,v)$ to every vertex $v \in G_F$ monotonically increases (5)

Proof:

Let v be the first that between flow augmentations, F & F' it decreases, that is, $\delta_{F'}(s,v) < \delta_F(s,v)$, and wlog assume this is the one closest to s that the decrease happens

Consider SP in $G_{F'}$ $s \to u \to v$

We know $\delta_{F'}(s,u) \geq \delta_F(s,u)$

CASE: $F(u,v) < C(u,v)$ in $G_F \Rightarrow G_F$ contains $u \to v$ edge $\in E_F$ $\delta(s,v) \leq \delta_F(s,u)+1 \leq \delta_{F'}(s,u)+1$

$= \delta_{F'}(s,v)$

$\Rightarrow$ Contradiction !

CASE: $F(u,v) = C(u,v) \Rightarrow u \to v \notin G_F$ but $u \to v \in G_{F'}$

$\Rightarrow$ AP_F traveres

$\Rightarrow$ $\delta_F(s,v) = \delta_F(s,u) - 1 \leq \delta_{F'}(s,u) - 1$

$= \delta_{F'}(s,v) - 2$ a contradiction

THM (Correctness & Run Time)

Total # of Flow Augmentations is $O(UE)$

Proof: Given Augmenting Path, call the edge on path that determines capacity as critical edge. Suppose $u \to v$ is critical edge. We show that for two consecutive time u becomes critical the SP to u increases by 2. If this is true, every left side on edge can become critical $V/2 = O(V)$ times & you got $O(E)$ edges $\Rightarrow O(VE)$ iterations.

when $u \to v$ becomes critical you got

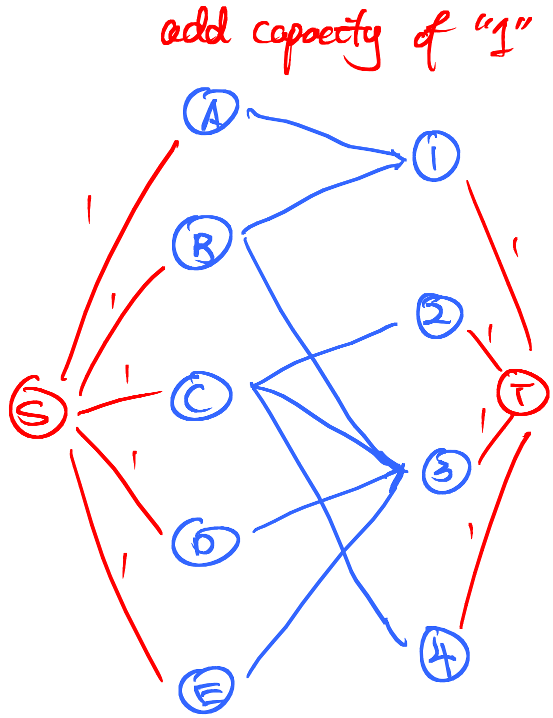
$$\delta_F(s, v) = \delta_F(s, u) + 1$$

& reverse direction. For $u \to v$ become critical again, some Flow F' must be pushed through $v \to u$ later on & before.

$$\delta_{F'}(s, u) = \delta_{F'}(s, v) + 1 \quad \& \quad \delta_F(s, v) \leq \delta_{F'}(s, v)$$

$$\Rightarrow \quad \delta_{F'}(s, u) \geq \delta_F(s, v) + 1 = \delta_F(s, u) + 2$$

BIPARTITE MATCHING MAXIMUM



Matching $\subseteq E$

St $\forall v \in V$ at most one edge point from M exists

Maximal A-1 C-3

Maximum B-1
C-2
E-3

How could we solve the problem using Max Flow?

Lemma:

a) M = matching $\Rightarrow \exists$ Flow w/ $|F| = |M|$

b) F in $G_{NEW} \Rightarrow \exists$ matching M w/ $|F| = |M|$

a) $\Rightarrow$ trivial to prove

b) define

$M = \{(u, v) : u \in L \;\&\; v \in R\}$

$$|M| = F(L,R) = F(\cancel{L},V)^{\to 0} - F(\cancel{L},L)^{\to 0} - F(\cancel{L},t)^{\to 0} - F(L,S)$$ 8

$$= -F(L,S)$$

$$= F(S,L) = |F|$$

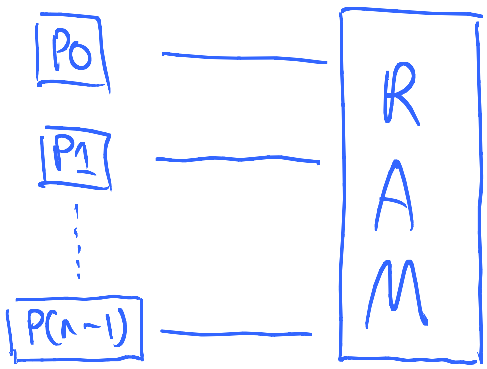
Corollary

Maximum Bipartite Matching $\Longleftrightarrow$ Max Flow in G_{NEW}

ECE1762 Algorithm LEC15

PARALLEL ALGORITHMS

PRAM Model



EREW -
ERCW
CREW
CRCW -

Total Time $T(n)$
processors $P(n)$
$Cost(n) = T(n) * P(n)$
Work $\leq Cost(n)$

The following statements are equivalent

1) The algorithm requires $T(n)$ time w/ $P(n)$ procs
2) Has cost $C(n)$ and time $T(n)$
3) time $O(\frac{Cost(n)}{p})$ for any # of procs $p \leq P(n)$
4) Use time $O(\frac{Cost(n)}{p} + T(n))$ $\forall$ # procs p

1) $\Rightarrow$ 2) by definition

2) $\Rightarrow$ 3) Every step $\frac{P(n)}{p} \cdot T(n) = \frac{P(n)}{p} \frac{Cost(n)}{P(n)} = \frac{Cost(n)}{p}$

3) $\Rightarrow$ 4) $p \leq P(n)$ then 3) Applies
$p > P(n)$ time $T(n)$

$\Big\} O(\frac{Cost(n)}{p} + T(n))$

②

4) ⇒ 1) $P(n) = P$

Optimal:

$$P(n) \cdot T(n) = cost(n) = \Theta(T_{SEQ}(n))$$

Optimal Speed Up:

$$P(n) = \frac{T_{SEQ}(n)}{T(n)}$$

Lemma: Suppose Optimal algorithm w/ $P(n)$ procs, we can create a new algorithm with $p < p(n)$ procs $T_p(n) = \frac{Cost(n)}{p}$ which is also optimal.

The new algorithm $p \frac{cost(n)}{p} = cost(n) = \Theta(T_{SEQ}(n))$

Strongly Optimal For Problem: The less time among all optimal algorithms.

NC - Class "efficiently parallelizable problems"

$O(\log^{O(1)} n)$ time w/ $O(n^{O(1)})$ procs

Brent's Thm: Assume PRAM algorithm w/ work(n) & time (n)

If at each step i

- Constant time to identify the # of ops executed.

— =//= to allocate the procs for those ops ③

Then we can run the algorithm in $O(T(n) + \frac{work(n)}{p})$ time using p procs

Work(n)

$$p = \frac{work(n)}{T(n)} \qquad T(n) + \frac{work(n)}{p = \frac{work(n)}{T(n)}} = T(n)$$

$p < P(n)$

Example: $\log n$ time with $O(n)$ work $O(n)$ procs

$$T(seq) = O(n)$$

$$\frac{n}{\log n} \text{ procs} \times \log n = O(n)$$

Proof: Let $work_i$ = amount of work at step i

$$\sum_{i}^{T(n)} \left(\frac{work_i}{p} + 1\right) = O\left(T(n) + \frac{work(n)}{p}\right)$$

Prefix Sums

④

5	3	2	4	1	9	8	5
1	2	3	4	5	6	7	8

① $T_{SEQ} = O(n)$

② Sub-Optimal:

$O(n)$ procs

$O(\log n)$

Cost $= O(n \log n)$ Work $\sum_{h=1}^{\log n} \frac{n}{2^h} \leq 2n = O(n)$

Can still do it in $O(\log n)$ time with

$\frac{\text{Work}(n)}{T(n)} = \frac{n}{\log n}$ procs.

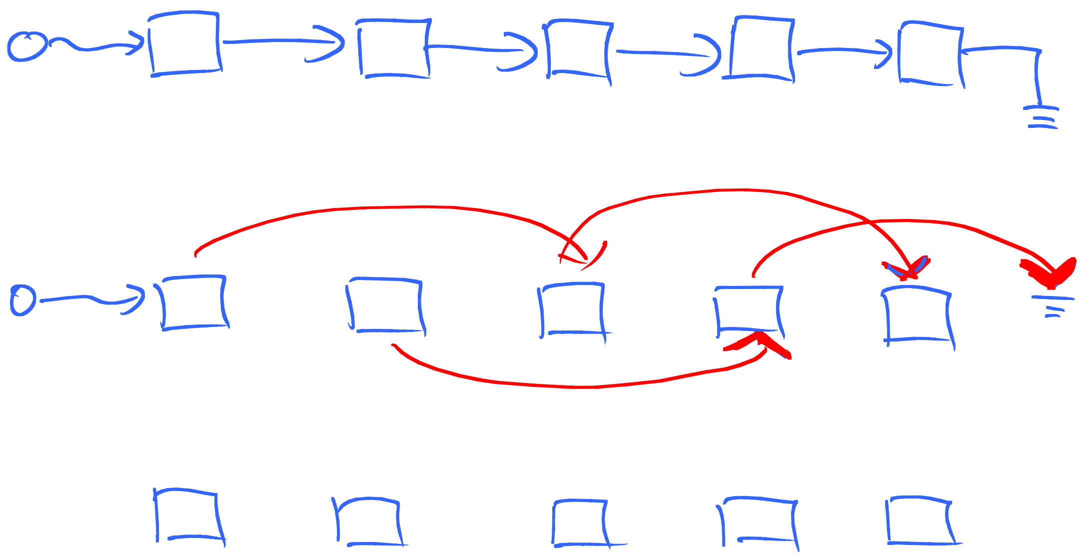
$\log n$ time w/ $n/\log n$ procs

Break original array in $n/\log n$ pieces of $\log n$ elements each. Allocate 1 proc/piece to find sum in $\log n$ time. Use sub-optimal algorithm in $n/\log n$ partial sums.

$\text{time}(\log(n/\log n)) = O(\log n - \log\log n) = O(\log n)$

Pointer Jumping (List Ranking) 5

(Prefix Sums in Lists) $T_{SEQ} = O(n)$



Find Maximum in Array CRCW

4.2 Finding the Max in $O(1)$ time

Recall that the common CRCW PRAM allows several processors to write to the same memory location, as long as they all write the same value.

Theorem 1 *We can find the max of n elements in $O(1)$ time on a common CRCW PRAM with n^2 processors.*

To show this, assume that we have $n^2/2$ processors, and let m be an auxiliary boolean array of size n.

Step 1. For $i = 1, \ldots, m$, set $m[i]$:= true.

Step 2. For all i and j in parallel, where $1 \leq i < j \leq n$, if $A[i] < A[j]$, set $m[i]$:= false.

Now $m[i]$ is true exactly when $A[i] \leq A[j]$ for all j, *i.e.* when $A[i]$ holds a copy of the max value.

Step 3. For $i = 1, \ldots, n$, if $m[i]$ = true then set $max = A[i]$.

Even if several locations hold a copy of the same value, all values written to max will be the same.

This algorithm runs on a CRCW PRAM with $n^2/2$ processors in $O(1)$ time. However, this algorithm is not optimal (since the problem can be solved sequentially in $O(n)$ time). Valiant proved that any n-processor CRCW PRAM algorithm requires $\Omega(\log \log n)$ time to find the max of n elements. In fact, as we will see, $O(\log \log n)$ time is sufficient as well.

Notice that this discussion also shows that the CRCW model is strictly more powerful than the EREW and CREW models.

6

4.5 Accelerating cascades

We conclude our presentation with the presentation of a general technique that can be often used to improve the performance of a parallel algorithm. In our case, accelerating cascades produces a strongly optimal algorithm for the problem of finding the maximum.

The idea behind *acclerating cascades* is as follows: you have several algorithms to solve a problem, each with a different time/processor requirement. Typically, you have slower algorithms that are optimal, and faster algorithms that are non-optimal because they require too many processors. Each of these algorithms uses a sequence of steps to reduce the problem to a similar problem of smaller size. Start with the optimal but slower algorithm, and reduce the size of the problem. Then use a faster algorithm to reduce the size even more, and continue like this. In many cases, this will allow you to derive an algorithm which is both fast and closer to optimal.

For example, to find the max of elements we already described a couple of algorithms with different time/processor requirements:

1. Solve the problem optimally (sequentially) in $O(n)$ time with 1 processor.
2. Solve the problem optimally in $O(\log n)$ time with $O(n/\log n)$ processors using the balanced binary tree scheme (pairwise comparisons). Each step of this algorithm reduces the size of the problem by $\frac{1}{2}$.
3. Solve the problem in $O(\log\log n)$ time with $O(n)$ processors (Valiant's algorithm).

We will now combine these algorithms to get an optimal $O(\log\log n)$ time one that has only $O(n)$ cost (*i.e.* uses $O(n/\log\log n)$ processors).

Here are two ways that you can do it:

1. Use (2) and (3). First we apply $\log\log\log n$ steps of the binary tree algorithm (2). There is $O(n)$ work here and, since each step reduces the problem size by $\frac{1}{2}$, it reduces the problem to one of size $N = n/2^{\log\log\log n} = n/\log\log n$. Now apply Valiant's algorithm (3) to these elements: this requires $O(\log\log N) = O(\log\log n)$ time and work
$$O(N\log\log N) = O\left(\frac{n}{\log\log n}\cdot\log\log\left(\frac{n}{\log n}\right)\right) = O(n)\ .$$
So the total time is $O(\log\log n)$ and work is $O(n)$. By Brent's Theorem, this can be done with $O(n/\log\log n)$ processors.

2. Similarly, we can use (1) and (3) and $n/\log\log n$ processors. Partition the array into $n/\log\log n$ blocks of size $\log\log n$ each. Assign one processor to each block and (using (1)) each processor computes the max of its block of elements. This reduces the problem to one of size $n/\log\log n$ in $\log\log n$ steps. Now apply Valiant's algorithm to these elements and find the max in $O(\log\log n)$ steps using only $n/\log\log n$ processors.

It can be shown [1] that any n-processor CRCW PRAM algorithm that uses only comparisons on elements of the array requires $\Omega(\log\log n)$ time to find the max of n elements. Therefore, both algorithms described above are strongly optimal comparison based parallel algorithms for this problem.

ECE1762 Algorithm LEC16

Intro. Theory of Computation

Computability & Complexity

$f(x) = -4x^3 + x^2 + 4x + 7$ is there a solution?

Machine $\Longleftrightarrow$ Algorithm

Deal with Languages from now on.

Problem $\equiv$ Sequence of 0's & 1's

Finite Automata

Regular Expression over a Finite Alphabet Σ is defined recursively as follows:

- The ϵ is a regular expression (re)

base def

- $\forall$ $a \in \Sigma$ is a re of r is re (r) is re
- If r & s are re's then $r \cdot s$ (concatenation) is re
- if r & s are re's then $r|s$ ("r" or "s") is are
- If r is re, then $r^* = r \cdot r \cdot \dots \cdot r$ is re (kleene star)

$r^0 = \epsilon$ ≥ 0 times

②

$L(r) = \{ w : \text{strings produced by } r \}$

Example:

$L\{(0|1), (0|1)\} = \{00, 01, 11, 10\}$

$L\{(0|1)^*\} = \{\varepsilon, 0, 1, 01, 10, 11, \ldots\} = \{$any binary # including empty string$\}$

$L\{(0|1)^* 1\} =$ any binary # that ends with 1

digit → 0~9

letter → a~z, A~Z

Symbol → _, -, ^

(letter | digit | symbol | letter)* ≡ typical programming variables

Given a string w we want to build a machine that recognizes whether $w \in L$?

Deterministic Finite Automata

③

Σ: alphabet K: Finite set of states

s: start state F: set of Final states

δ: transition Function $\delta: \underset{\text{present states}}{K} \times \Sigma \rightarrow K^{\text{next states}}$

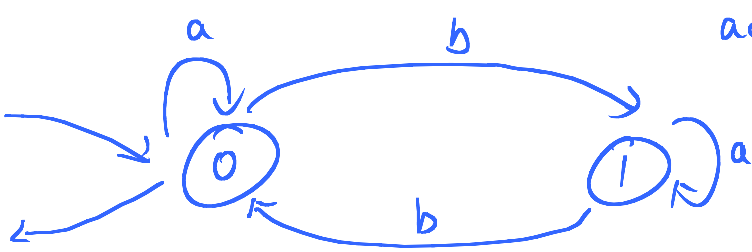
Example:

$\Sigma = \{a, b\}$ $K = \{0, 1\}$ $S = \{0\}$ $F = \{0\}$

present state	next state a	next state b
0	0	1
1	1	0

Transition Table

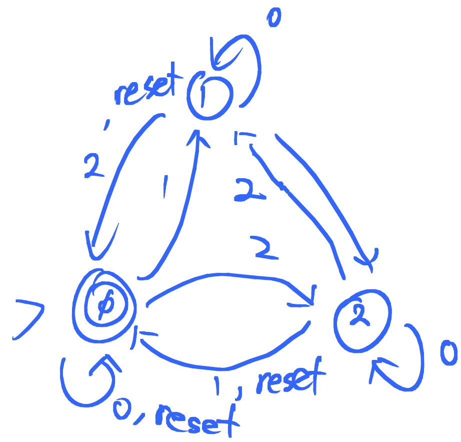
Transition Diagram



aaababa

$L(DFA) = \{w :$ any string $\{a,b\}^*$ with even # of b's

$a^*(a^* b\ a^* b)^* a^*$



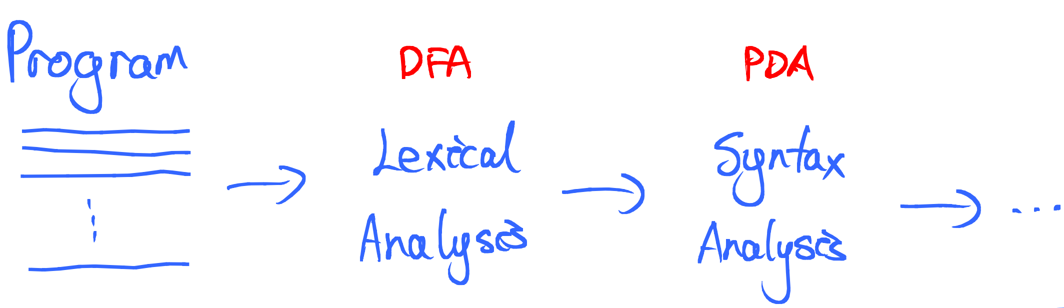
$L = \{w$: have at least one 1 and there is an even # of 0's following the last 1$\}$

$L = \{w$: sum of digits mod 3 $= \emptyset\}$

THM: The set of regular languages are those recognized by a DFA

$$\text{D.F.As} \equiv \text{re}$$

Example: Compiler



Non-deterministic Machine

Σ: alphabet K: Finite set of states

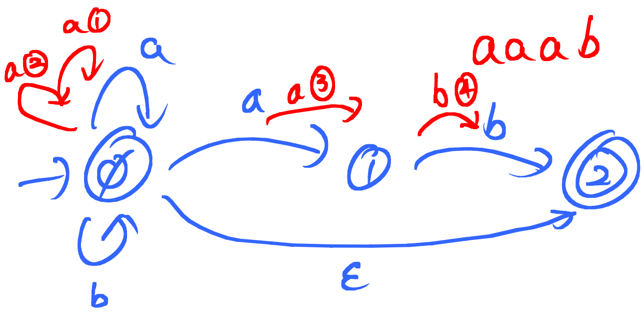
s: start state F: set of Final states

Δ: transition relation $\Delta \subseteq K \times \{\Sigma \cup \epsilon\} \times K$

Non-determinism

$\Sigma\{a,b\}$ $k=\{0,1,2\}$

$S=\{0\}$ $F=\{2\}$



PS	NS a	NS b	NS ∈
0	0, 1	0	2
1	–	2	–
2	–	–	–

$(a|b)^*ab \,|\, \epsilon$

Defference: - No transestion from every symbol

- ϵ moves

- randomness

NFA are used to understand a problem. They are not real machine.

THM: The set of regular languages are those recognized by a DFA. NFAs $\equiv$ DFAs $\equiv$ r.e.

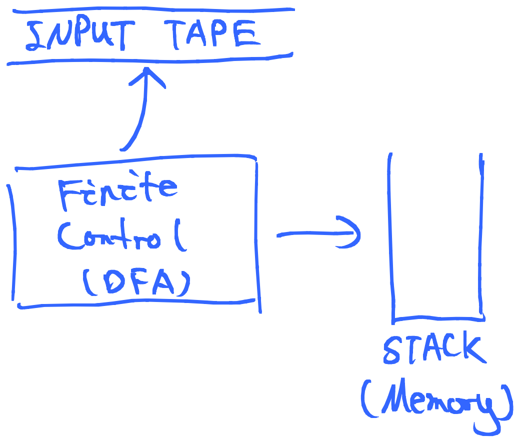
$L=\{0^n1^n : n \geq \emptyset\}$ $n \leq k$ fixed $O(2^k)$ states

$L=\{ww : w \in \{0,1\}^*\}$

$L=\{w : w$ has equal # of 0s & 1s$\}$

$L=\{ww^R : ww^R : w \in \{0,1\}^*\}$

Pushdown Automata (PDA)



Context Free Grammars

U: set of terminal & non-terminal symbols

T: set of terminal symbols

P: Productions $\subseteq (U-T) \times U^*$ written as $U-T \to U^*$

S: start symbol $\subseteq U-T$

Ex: $U = \{E, num, +, -, (,), 0, 1, \cdots, 9\}$ $S = \{E\}$

P: $E \to E+E$ $E \to E-E$ $E \to (E)$

$E \to num$ $E \to -E$

$num \to 0$ ..... $num \to 9$

$E \to E+E \to -E+E \to -E+E-E \to -E+num-E$

$\cdots \to -num+num-E \to \cdots \to -5+3-2$

$L = \{$a simple calculator$\}$

7

① $S \to 0S1 \mid \varepsilon \quad \to L = \{0^n 1^n : n \geq 0\}$

② $S \to aSa \mid bSb \mid \varepsilon$

$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow \cdots$

THM: CFL $\Leftrightarrow$ PDA

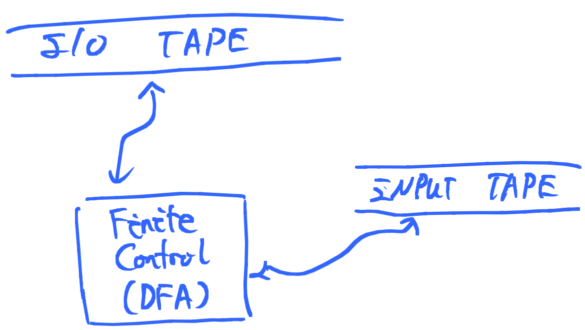
What about ?

$L = \{ww : w \in \{0,1\}^*\}$

$L = \{0^i 1^j 2^k : 0 \leq i \leq j \leq k\}$

Cannot be recognized by PDA.

TURING Machine



is equivalent to modern computer

- Read a character
- May write on R/W Tape & move the heads left / right on the R/W and/or Input Tape

Halting Problem:

Given a machine and an input, will the machine will ever terminate on this input?

Example of a TM that recognizes palindromes ⑧

i.e. Rise to vote sir

Algorithm:

1) Copy input to r/w tape
2) Move input head to beginning
3) Move read head to end
4) Starting moving heads step-by-step declaring accept or reject

States = $\{q_{start}, q_{copy}, q_{left}, q_{test}, q_{halt}\}$

q_{start} = move head to left, switch to q_{copy}

q_{copy} = copy symbol if non-blank from input to r/w tape q_{left} at end

q_{left} = move head to left

q_{test} = it will move heads, opposite directions to check similar chars, declare accept/reject and move to q_{halt}

q_{left} = end

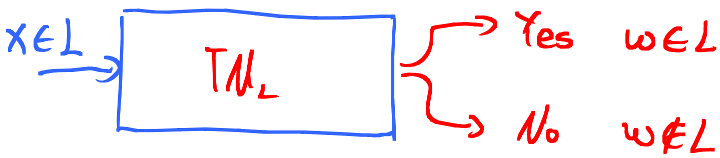
ECE1762 Algorithm LEC17

Computability & Complexity

- TM ≡ Algorithm ≡ Program
- Problem $\longrightarrow$ 01001....

Translation

(Poly. Time)



- Optimization vs Decision Problem

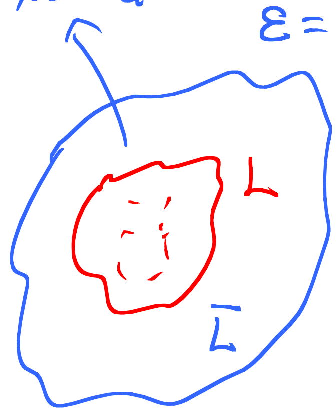
Example: Max Flow_o = what is Max Flow in a graph? / Optimization

V/

Max Flow_D = Does G have a Flow of size k? / decision

Not Flow k and Not G

$\Sigma = \{0, 1\}^*$



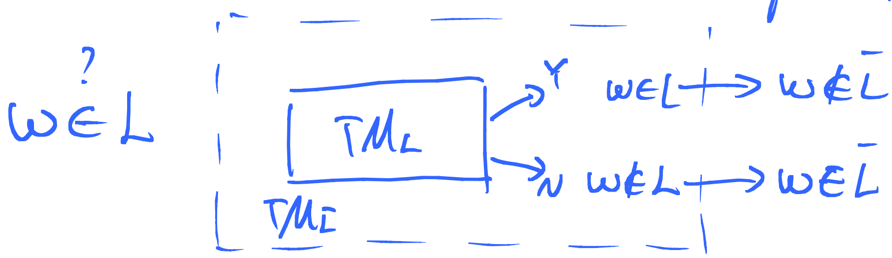
A TM decides a problem if after a finite of # of steps returns w/ answer yes or no

A TM accepts a problem if when it returns w/ yes or no the answer is correct, but as long as machine runs you don't know whether it will terminate or not.

Some things to prove

① if L is decides, then $\bar{L}$ is also decided

$P \in co\text{-}NP \equiv P$ is closed under complementation



② ∃ problem that are undecidable

Halting: Is there a TM that can decide whether given any machine M & input x (M, x), M halts on x?

Proof: Diagonalization

Complexity Class P

$$P \equiv \{L \in \{0,1\}^* : \exists \text{ TM that decides them in poly-time}\}$$

THM: $P = \{L \in \{0,1\}^* : \exists$ TM to accept them in poly $O(n^c)$ time$\}$

Verification: Given an instance to a problem, and a candidate solution to problem, how "easy" is to verify it is indeed a solution?

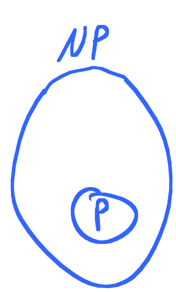
Algorithm A verifies language L iff given instance L there exists a certificate (witness) y st. $A(x,y) = 1$ or 0
(candidate solution)

Complexity Class NP

suspect soln is poly to problem size

$$NP \equiv \{L \in \{0,1\}^* : \exists \text{ certificate } y = O(|x|^c) \text{ and poly-time algorithm } A \text{ st } A(x,y) = 1\}$$

THM: $P \subseteq NP$



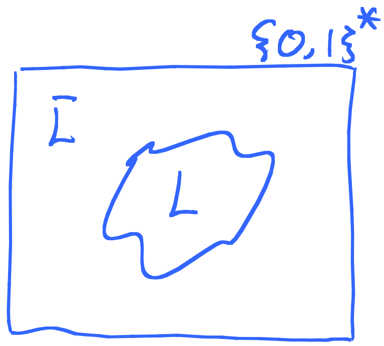
④

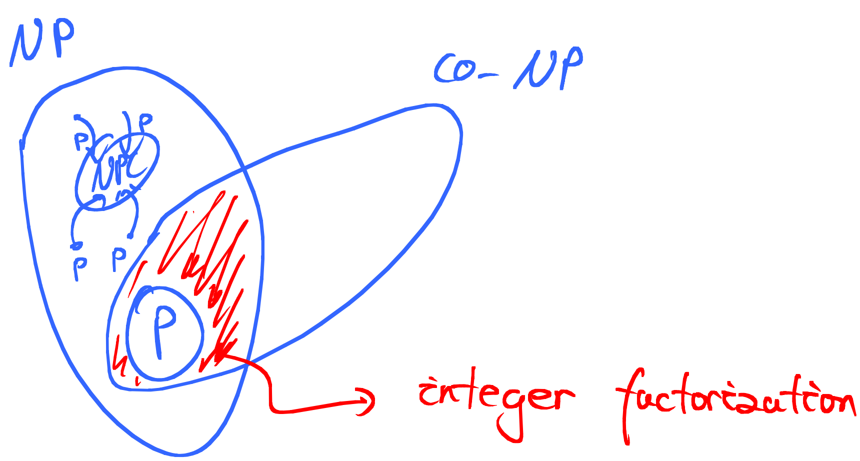
co-NP $\equiv$ if $L \in NP$ then $\bar{L} \in$ co-NP

$$= \{ L \in \{0,1\}^* : \forall y = O(|x|^c)\ \exists \text{ poly } A(x,y) = \emptyset \}$$

HAM-CYCLE (NP Class)

Given a connected, undirected graph a simple cycle that traverses all vertices?



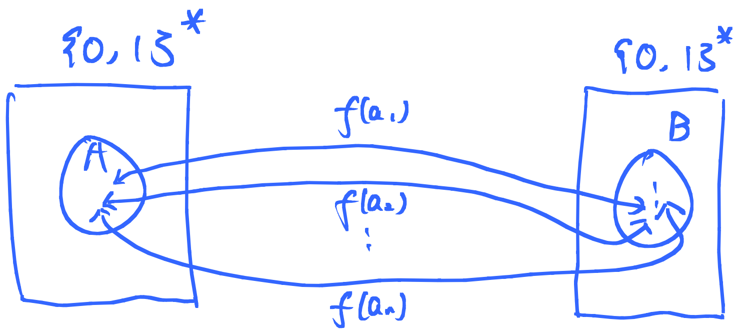


$NP \stackrel{?}{=} P$ $\qquad$ $NP \cap \text{co-}NP \stackrel{?}{=} P$

$NP \stackrel{?}{=} \text{co-}NP$

Reducibility (informal)

Problem A can be reduced to problem iff $\exists$ translation algorithm that maps every instance of A into an instance of B, and those mapped instances back to instances of A



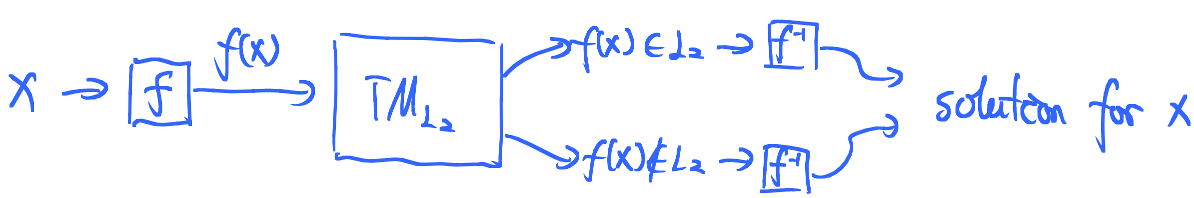
Polynomial Time Reducibility

we say that L is poly-reducible to L' iff $\exists$ poly function $f(\,)$ s.t.

$$x \in L \quad \text{iff} \quad f(x) \in L'$$

denoted as $L \le_p L'$

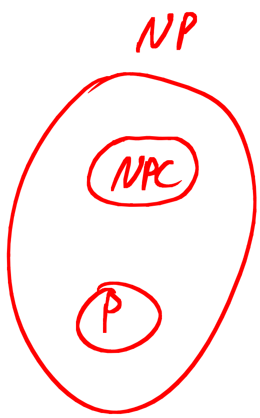
THM: if $L_1 \le_p L_2$ and $L_2 \in P$ then $L_1 \in P$



6

Language L is NP-Complete (NPC) iff

- $L \in NP$ (takes poly-time to verify)
- $\forall L' \in NP, \quad L' \leq_p L$ (NP-hard)



THM: if $\exists L \in NPC$ & $L \in P \Rightarrow P = NP$

THM: $NP = co\text{-}NP$ iff $\exists L \in NPC$ st $\bar{L} \in NP$

$\Rightarrow$ Easy

$\Leftarrow$ Let $L \in NPC$ st $\bar{L} \in NP$

Pick any $L' \in NP$ $\quad L' \leq_p L \Leftrightarrow \bar{L}' \leq_p \bar{L}$

$\Rightarrow \bar{L}' \in NP$ because $\bar{L} \in NP$

$\in co\text{-}NP$

NP-Complete Methodology

To prove that L is NPC, do the following

- Show that $L \in NP$ (usually easy to show)
- Pick any known $L' \in NPC$ and show $L' \leq_p L$

anybody $\leq_p L' \leq_p L$

Cook's Thm (1971) Circuit-SAT is NPC

Given circuit w/ $O(n)$ AND/OR/NOT gates & single output. Find an input vector that makes output "1"

(A) Circuit-SAT $\in NP$

(B) $\forall$ NP problem $\le_p$ Circuit-SAT

Outline of Proof:

$$
\begin{tikzcd}
 & \text{Circuit-SAT} \arrow[d] & \\
 & \text{Formula-SAT} \arrow[d] & \\
 & \text{3-SAT} \arrow[dl] \arrow[dr] & \\
\text{CLIQUE} \arrow[d] & & \text{HAM-CYCLE} \arrow[d] \\
\text{vertex cover} & & \text{travelling salesman}
\end{tikzcd}
$$

Formula - SAT: Given a Formula ϕ with Boolean Variables & connectives $\neg$, $\wedge$, $\vee$, $\leftrightarrow$, (,), $\rightarrow$ and $O(n)$ symbols all together, is there an assignment to the variables that make $\phi = 1$? 8

$$\phi = ((x \vee y) \rightarrow z) \leftrightarrow (a \wedge b \vee c \rightarrow d)$$

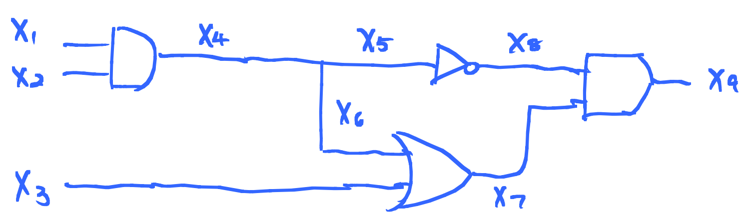
A) Show Formula SAT $\in NP$ YES!

verification in poly time verified

B) Circuit - SAT can reduce to Formula-SAT

Circuit - SAT $\leq_p$ Formula SAT

Good Enough to show an example



$(a \wedge b) \leftrightarrow c$ characteristic function

$\Updownarrow$

$$\phi = x_9 \wedge ((x_1 \wedge x_2) \leftrightarrow x_4) \wedge (x_5 \leftrightarrow x_4) \wedge (x_6 \leftrightarrow x_4) \wedge$$
$$((x_6 \vee x_3) \leftrightarrow x_7) \wedge (x_8 \leftrightarrow \neg x_5) \wedge (x_9 \leftrightarrow (x_8 \wedge x_7))$$

ECE1762 Algorithm LEC18

To prove language L is NPC

- Show that $L \in NP$ (verified in poly-time)
- Pick known NPC language L' and show that $L' \leq_p L$ (NP-hard)

3-SAT

Given a set of clauses with three variables which is a conjunction of disjunctions. Find a satisfying assignment

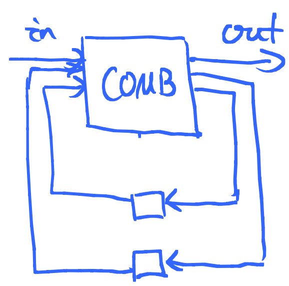
$$\phi = \underbrace{(x_1 \vee x_2 \vee \bar{x}_3)}_{\text{clause}} \wedge (x_4 \vee \bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_5) \wedge \dots$$

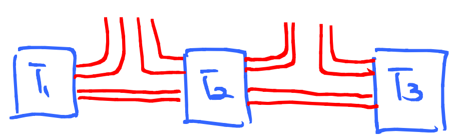
2-SAT = poly time

CNF = Conjunctive Normal Form

Verification

QBF: PSPAPE $\supseteq$ NP





Proof: ②

A) $3-SAT \in NP$ easy

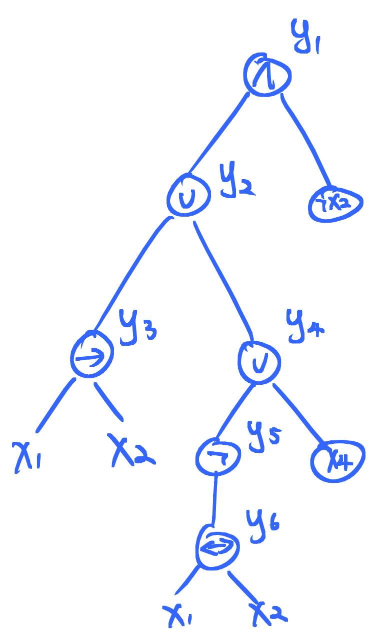
B) $\text{Formula-SAT} \leq_p 3\text{-SAT}$

Given a Formula ϕ I will create a 3-SAT instance s.t.

Formula is SAT iff 3-SAT is SAT

$$\phi = ((x_1 \rightarrow x_2) \vee \neg(\neg x_1 \leftrightarrow x_3) \vee x_4) \wedge \neg x_2$$

1) Build a parse tree for formula



$$\begin{aligned}\phi = y_1 \wedge\ & (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \wedge \\ & (y_2 \leftrightarrow (y_3 \vee y_4)) \wedge \\ & (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \wedge \\ & (y_4 \leftrightarrow (y_5 \vee x_4)) \wedge \\ & (y_5 \leftrightarrow \neg y_6) \wedge \\ & (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\end{aligned}$$

replace with clauses

2) Build Characteristic Function $\forall y_i$ & Find Maxterms 3

y_1	y_2	x_2	$y_1 \leftrightarrow (y_2 \wedge \neg x_2)$
0	0	0	1
0	0	1	1
0	1	0	$\emptyset$
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$$\bar{\phi}_{PART} = (\bar{y}_1 \wedge y_2 \wedge \bar{x}_2) \vee (y_1 \wedge \bar{y}_2 \wedge \bar{x}_2) \vee (y_1 \wedge \bar{y}_2 \wedge x_2) \wedge (y_1 \wedge y_2 \wedge x_2)$$

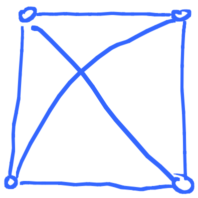
$$\bar{\bar{\phi}}_{PART} = (y_1 \wedge \bar{y}_2 \wedge x_2) \wedge (\bar{y}_1 \wedge y_2 \wedge x_2) \wedge \ldots$$

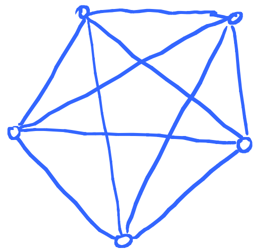
$$(x \quad y) = (x \vee y \vee p) \wedge (x \vee y \vee \bar{p})$$

- You have to show reduction
- You have to show solution to $L' \Leftrightarrow$ solution to L
- You have to show reduction takes Polynomial time

CLIQUE (Complete Graph) 4

K4 K5





Decision Problem

Given a graph G does it have a clique of size k?

A) CLIQUE $\in NP$ (Easy)

B) 3-SAT $\leq_p$ CLIQUE

$$\phi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\hat{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

ϕ has a solution $\Longleftrightarrow$ G has clique of size (# clauses)

1) Introduce a vertex $\forall$ literal & "group" according to clauses

2) Connect variables between different groups if not complementing each other

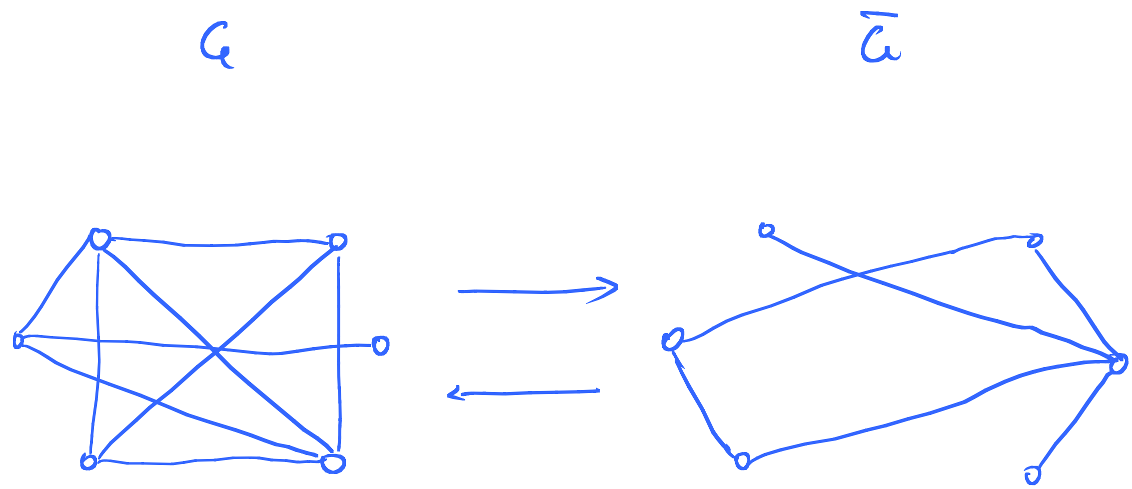
5

VERTEX COVER: Given a graph, a VC is a set of vertices s.t. every graph edge is adjacent to at least one vertex from the cover

Decision Problem does G have VC of size k?

A) $VC \in NP$ (Easy)

B) $CLIQUE \leq_p VC$



CLAIM: G has a clique of size k iff $\bar{G}$ a VC of size $V-k$

The reduction is poly-time

Assume Ham Cycle for the following Problem ⑥
is NP Complete

Travelling Salesman Problem

Given a complete graph undirected w/ weights
what's the lowest cost simple cycle?

Decision: Does G have a TSP of weight w?

A) TSP $\in$ NP (early)

B) HAM CYCLE $\leq_p$ TSP

1) Given G to find a Ham Cycle, introduce the remaining edges to make G_{NEW} a complete graph

In E_{new} = { weight ϕ of edge existed in G
weight 1 of edge new introduced

Does G_{new} have TSP of weight ϕ?

Remind : To prove $L \in NPC$

A) Show $L \in NP$

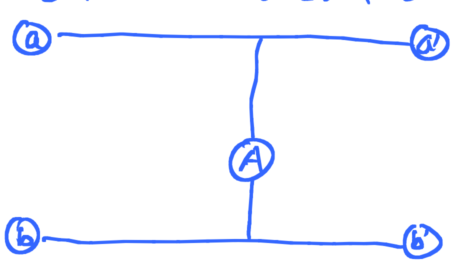
B) Pick know $L' \in NPC$ & show $L' \leq_p L$

HAM-CYCLE: Does there exist a simple cycle to traverse all vertices?

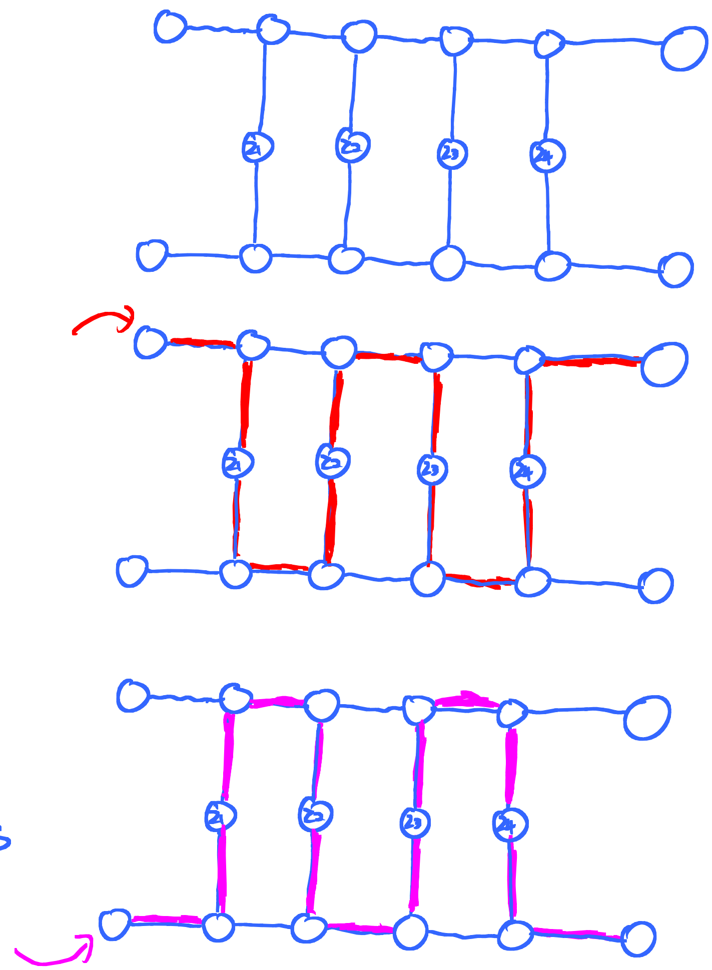
A) Easy

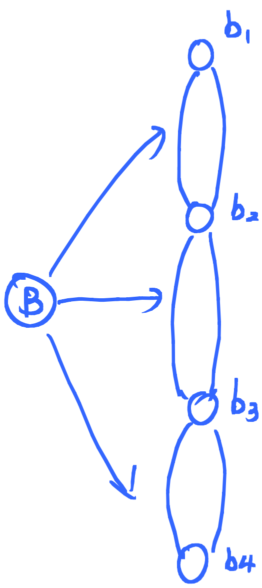
B) 3-SAT $\leq_p$ HAM CYCLE

enters a $\longrightarrow$ exit a'



enter b $\longrightarrow$ exit b'





it can never traverse all three outer b_i edges & traverse all inners ones

ECE1762 Algorithm LEC19 ①

APPROXIMATION ALGORITHMS

Approximate Solution w/ error bound $\rho(n)$

$$\max\left\{\frac{C}{C_{opt}}, \frac{C_{opt}}{C}\right\} \leq \rho(n)$$

minimization   maximization

error bound

$$\left|\frac{C - C_{opt}}{C_{opt}}\right| \leq \varepsilon(n)$$

$$\varepsilon(n) \leq \rho(n) - 1$$

Different Approximation

- Fixed Error Bound
- depends on n
- trade off

  time vs error

(1992)

Approximating CLIQUE is NPC

Approximation Vertex Cover ②

Subset $V' \subseteq V$ st. ∀edge has an adjacency in V'

1 $C = \emptyset$, $E' = E$

2 while $E' \neq \emptyset$ do

randomly pick (u, v) edge

$C = C \cup \{u, v\}$

$E' = E' - \{$edges adjacent to vertices $u, v\}$

$|C| \leq 2|C_{opt}|$

the VC discovered is never twice bigger than the optimal (minimal) one

<u>Proof</u> Let A be set of edges picked at line 3

we have $C = 2 \cdot A$ by construction.

Also $A \leq C_{opt}$

$\therefore C \leq 2C_{opt}$

Run-time $O(E)$

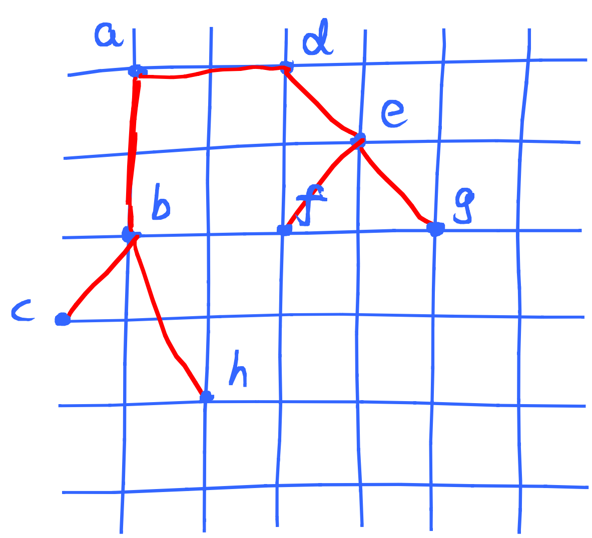
Approximating TSP

③

- Assume complete graph w/ no negative weights
- Assume weights on graph obey triangle inequality

$$w(A,B) \le w(A,C) + w(C,B)$$

TSP on Euclidean Space



Preorder:

a b c h d e f g

Let $C(H) = \sum_{(u,v) \text{ on Ham}} C(u,v)$ Then

$C(H) \le 2C(H_{opt})$

<u>Proof</u> $C(MST) \le C(H_{opt})$

Let w be preorder walk in step 3

$C(w) \le 2c(MST)$

cont. next pg

1. Select Arbitary Root
2. Grow an MST
3. Let L preorder of MST
4. Return a ham-cycle that resembles preorder traversal

$C(H) \le C(w)$ because of triangle inequality (4)

$\therefore \; C(H) \le 2\, C(H_{opt})$

THM: If we drop triangle inequality from requirement $\nexists$ poly algorithm to approximate TSP, unless P = NP

Proof: By contradiction, if $\exists$ exists algorithm to approx TSP we solve HAM-CYCLE in poly time! Let poly approx. with ratio bound ρ to approx. TSP.

Given G to find HAM CYCLE introduce remaining edges with weights as follows

$$(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ \rho|V|+1 & \text{if } (u,v) \notin E \end{cases}$$