2 SORTING BY RANKING

{**rank** of an item=number of items smaller}

rank each item by counting;

then place each item according to its rank.

If duplicate, then place it at the nearest slot to the right.

$W(n) = O(n(n-1)) = A(n) = B(n)$

$S(n) = O(n)$

To handle duplicates, redefine

rank(i) = number of items smaller than i or, if equal, occurring before i.

Oblivious Algorithm

3 SORTING BY SWAPPING

Bubble Sort–good when input is nearly sorted

$W(n) = O(n^2/2)$

$S(n) = O(1)$

Odd-Even Exchange Sort

a) odd-even compare & exchange

b) even-odd compare & exchange

c) repeat step (a) and (b) if exchange-count > 0

$W(n) = O(n^2)$

$S(n) = O(1)$

4 SORTING BY INSERTION

Online algorithms.

Linear Insertion Sort

Insert the next element in the ordered list prepared so far by sequential search & shifting.

$W(n) = O(n^2/2)$

$S(n) = O(1)$

$O(n)$ time on a sorted list

Binary Insertion sort

perform binary search to find location for insertion.

$W(n) = O(n \log n) + O(n^2)$.

Tree Sort

Insert into a binary search tree, then traverse tree in-order.

$W(n) = O(n^2)$

$S(n) = O(n)$

$A(n) = B(n) = O(n \log n) + O(n)$

5 SORTING BY SELECTION

Offline algorithms

Selection sort

find maximum & replace with the concurrent last

$W(n) = O(n^2)$

$S(n) = O(1)$

$O(n^2)$ even on a sorted list

Tournament Sort

$W(n) = A(n) = B(n) = O(n \log n)$

$S(n) = O(n)$ for the tournament tree

Heap Sort

- construct a max heap -$O(n)$
- delete root and update heap repeatedly -$O(n \log n)$

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

$S(n) = O(1)$

5.1 Heap Sort

- Restore-Heap(i): $O(h)$ where h is the height of the node i.

- Construct Heap:

  For $i = \left\lfloor \frac{n}{2} \right\rfloor$ down to 1 Restore-Heap(i)

  $O(n)$

- Heap Sort:

  1. Construct Heap – $O(n)$

  2. For i:=n down to 2 – $O(n \log n)$

     exchange $L[1]$ with $L[i]$

     decrement heap size

     Restore-Heap(1)

- Time Complexity of deletion phase in Heap Sort

$$\begin{aligned} W(n) &= 2\log n + W(n-1) \\ &= 2\sum_{i=1}^{n} \log i \\ &\leq 2\int_{1}^{n+1} \log x dx \end{aligned}$$

$$= [2(x \log x - x)]_1^{n+1}$$

$$= 2n \log n - 2n$$

5.1.1 Heapsort Construction

Construct Heap:

for $i = \lfloor n/2 \rfloor$ downto 1

Restore-heap(i)

Time Complexity of Iterative Algorithm for Heap Construction:

$$\sum_{h=1}^{\lfloor \log n \rfloor} \lceil n/2^{h+1} \rceil O(h)$$

$$= O\left(n \sum_{h=1}^{\log n} (h/2^h)\right)$$

$= O(n)$ (pp. 159)

Consider $\sum_{h=1}^{\log n} h/2^h$

Let

$$x = 1/2 + 2/2^2 + 3/2^3 + 4/2^4 + \cdots + y/2^y \quad (1)$$

$$2x = 1 + 2/2^1 + 3/2^2 + 4/2^3 + \cdots + y/2^{y-1} \quad (2)$$

$$x = 1 + 1/2 + 1/2^2 + \cdots + 1/2^{y-1} \quad (3)$$

$$-y/2^y \quad (4)$$

$$= \frac{(1/2)^y - 1}{1/2 - 1} - y/2^y \quad (5)$$

$$= 2(1 - (1/2)^y) - y/2^y \quad (6)$$

$$\leq 2 \quad (7)$$

5.1.2 Heapsort: Recursive Construction

Construct-Heap(n):

construct left subheap

construct right subheap

Restore-heap(1)

Time Complexity:

$$
\begin{aligned}
W(n) &= 2w(n/2) + \log n && (9)\\
&= \log n + 2w(n/2) && (10)\\
&= \log n + 2\left(\log(n/2) + 2w(\frac{n/2}{2})\right) && (11)\\
&= \log n + 2\log n - 2\log 2 + && (12)\\
&\quad 2^2 w(n/2^2) && (13)\\
&= \log n + 2\log n - 2\log 2 + && (14)\\
&\quad 2^2\left(\log(\frac{n}{2^2}) + 2w\left(\frac{n/2^2}{2}\right)\right) && (15)\\
&= \log n + 2\log n - 2\log 2 + && (16)\\
&\quad +2^2\log n - 2^2\log(2^2) + 2^3 w(n/2^3) && (17)\\
&\quad \cdots && (18)\\
&= \log n + 2\log n - 2\log 2 + && (19)\\
&\quad +2^2\log n - 2^2\log(2^2) + \cdots + && (20)\\
&\quad 2^k\log n - 2^k\log(2^k) + 2^{k+1}w(n/2^{k+1}) && (21)\\
&= \log n(1 + 2 + 2^2 + \cdots + 2^k) && (22)\\
&\quad -(2 + 2\cdot 2^2 + 3\cdot 2^3 + \cdots + k\cdot 2^k) && (23)
\end{aligned}
$$

Here, we assume that $n = 2^k$ so $k = \log n$.

Let $s = 2^0 + 2^1 + 2^2 + \cdots + 2^k$
$= \frac{2^{k+1}-1}{2-1} = 2^{k+1} - 1$

To derive this formula, one can follow these steps:

$$\begin{aligned} s &= 2^0 + 2^1 + 2^2 + \cdots + 2^k && (24)\\ 2s &= 2^1 + 2^2 + \cdots + 2^k + 2^{k+1} && (25)\\ s &= -1 + 2^{k+1} && (26) \end{aligned}$$

Thus, $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

Likewise, let

$$\begin{aligned} T &= 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + k \cdot 2^k && (27)\\ 2T &= 1 \cdot 2^2 + 2 \cdot 2^3 + \cdots + && (28)\\ &\quad (k-1) \cdot 2^k + k2^{k+1} && (29)\\ T &= -2^1 - 2^2 - 2^3 - \cdots - 2^k + && (30)\\ &\quad k2^{k+1} && (31)\\ &= k2^{k+1} - (2^1 + 2^2 + \cdots + 2^k) && (32)\\ &= k2^{k+1} - (2^{k+1} - 2) && (33)\\ &= (k-1)2^{k+1} + 2 && (34)\\ & && (35) \end{aligned}$$

$\Rightarrow W(n) = \log n(2^{K+1} - 1) - ((k-1)2^{k+1} + 2)$
$= 2n \log n - \log n - 2n \log n + 2n - 2$
$= 2n - \log n - 2$
$= O(n)$

5.1.3 Priority Queue employing Heap

Read pp. 162 (Section 6.5)

Delete Max/Min

Insert

See Exercise 6-1, pp. 166

6 SORTING BY MERGING

- **Mergesort**

  — $W(n) = O(n) + 2w(n/2)$
  $= n + 2w(n/2)$
  $= n + 2(n/2 + 2w(n/2^2)$
  $= n + n + 2^2 w(n/2^2)$
  $= 2n + 2^2 w(n/2^2)$
  $= 2n + 2^2(n/2^2 + 2w(n/2^3))$
  $= 3n + 2^3 w(n/2^3)$
  $\ldots$
  $= kn + 2^k w(n/2^k)$
  $w(n/2^k) = w(1) = 0$
  $n/2^k = 1$, $k = \log_2 n$
  $\Rightarrow W(n) = \theta(n \log n)$

  — Recursion tree for $W(n)$

  — $S(n) = O(n)$
  Can be reduced to $O(1)$ but algorithm slows down.
  A temporary array of half the size is required, however.

7 SORTING BY SPLITING

$W(n) = O(n^2)$ but $A(n) = O(n \log n)$

Quicksort (A, p, r):

if $p < r$ then $q \leftarrow$ Partition(A, p, r)

Quicksort(A, p, q)

Quicksort$(A, q + 1, r)$

7.1 Quicksort: Partitioning

Partition I.

$x := A[p]$; $i := p - 1$; $j := r + 1$

while (TRUE) **do**

  Repeat Decrement j **until** $A[j] \leq x$

  Repeat Increment i **until** $A[i] \geq x$

  if $i < j$

    then exchange $A[i]$ and $A[j]$

    else return j

endwhile

$x = 15$

15 7 23 5 20 3

Why do i and j never get out of array bounds?

$W(n) = n + 2$ comparisons

$W(n) = O(n^2) = O(n) + W(n-1)$

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

Balance Partitioning:

- Even if each split is 1% on one side and 99% on the other, recursion tree remains logarithmic.

- Alternate good and bad splits:

Quicksort Improvements

– random x

– median of first, middle and last items

– insertion sort for $n \leq 15$

$S(n) = O(n)$
(See 7-4; reduces $S(n)$ to $O(\log n)$)

7.2 Quicksort: Partitioning II

Input: A[p..r]

Invariants: {All items in $A[2..i]$ are $<$ the pivot.}

{All items in $A[(i+1)..(unknown-1)]$ are $\geq$ the pivot}

$x = A[p];\ i := p;$

for $unknown := p+1$ **to** r **do**

  if $A[unknown] < x$ **then**

    $i := i+1$; swap$(A[i], A[unknown])$

swap$(A[1], A[i])$

$W(n) = n-1$ comparisons

15 7 23 5 20 3

23 _____

7.3 Av. Case Complexity of Quicksort

Assume

- all keys distinct
- all permutations equally likely

Probability that split point is i, for $1 \leq i \leq n$, is $1/n$

$$A(n) = (n-1) + \frac{1}{n}(A(0) + A(n-1) +$$

$$A(1) + A(n-2) + \cdots A(n-1) + A(0))$$

$$= n - 1 + \frac{2}{n}\left(A(0) + A(1) + \cdots + A(n-1)\right)$$

$$A(n) = n - 1 + \frac{2}{n}\Sigma_{i=2}^{n-1} A(i), \quad A(0) = A(1) = 0$$

$$(n)A(n) = (n)(n-1) + 2\Sigma_{i=2}^{n-1} A(i)$$

$$A(n-1) = n - 2 + \frac{2}{n-1}\Sigma_{i=2}^{n-2} A(i)$$

$$(n-1)A(n-1) = (n-1)(n-2) + 2\Sigma_{i=2}^{n-2} A(i)$$

$$nA(n) - (n-1)A(n-1)$$

$$= n(n-1) - (n-2)(n-1) + 2A(n-1)$$

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

Let $B(n) = \frac{A(n)}{n+1}$ (*Changing Variable*)

Since $A(1) = 0,\ B(1) = 0$

$$\Rightarrow B(n) - B(n-1) = \frac{2(n-1)}{n(n+1)}$$
$$B(n) = \frac{2(n-1)}{n(n+1)} + B(n-1)$$
$$= \frac{2(n-1)}{n(n+1)} + \left(\frac{2(n-2)}{(n-1)(n)} + B(n-2)\right)$$
$$= B(1) + \frac{2\cdot 1}{2\cdot 3} + \frac{2\cdot 2}{3\cdot 4} + \cdots + \frac{2(n-1)}{n(n+1)}, B(1) = 0$$
$$B(n) = \sum_{i=2}^{n} \frac{2(i-1)}{i(i+1)}$$

$f(n) = \frac{2(n-1)}{n(n+1)}$ continuous decreasing function Sum_a^b f(x) <= Int_{a-1}^b f(x)dx - pp. 51

$$B(n) \le \int_2^n f(x)dx, \quad f(1) = 0$$
$$\frac{2(x-1)}{x(x+1)} = \frac{A}{x} + \frac{B}{x+1}$$
$$= \frac{(A+B)x + A}{x(x+1)}$$
$$\Rightarrow A = -2$$
$$A + B = 2$$
$$\Rightarrow B = 4$$

$$\Rightarrow f(x) = \frac{4}{x+1} - \frac{2}{x}$$

$$\int_2^n f(x)dx = \int_2^n \left(\frac{4}{x+1} - \frac{2}{x}\right) dx$$

$$= (4\ln(x+1) - 2\ln x)\,|_2^n$$

$$= 4\ln(n+1) - 2\ln n - 4\ln 3 + 2\ln 2$$

$$\approx 2\ln n$$

$$\Rightarrow B(n) \leq 2\ln n$$

$$A(n) = (n+1)B(n) \leq 2(n+1)\ln n$$

$$\Rightarrow A(n) \leq 1.4(n+1)\log_2 n$$

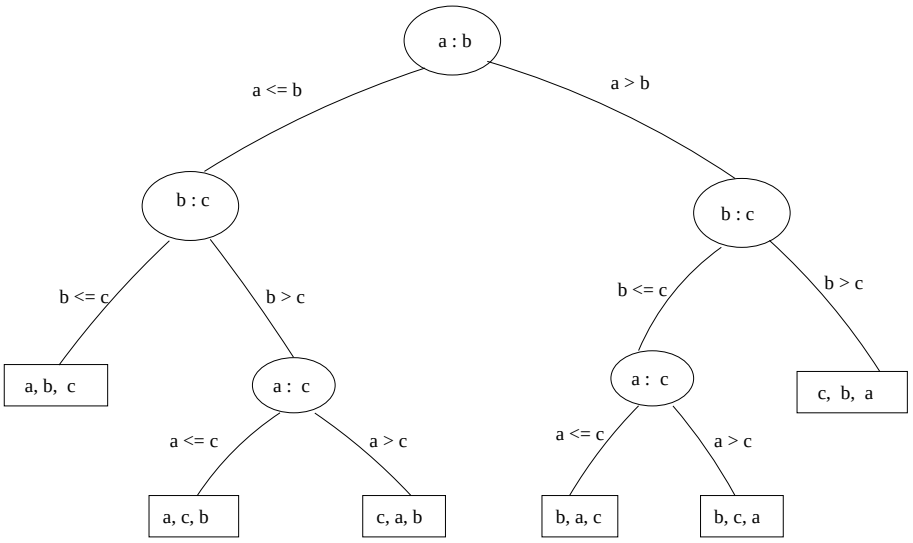
8 LOWER BOUND

(a) Local exchange only

each accomplishes in undoing one inversion
5 1 4 7 2 has inversions (5,1),(5,4),(5,2),(4,2),(7,2)
(n n-1 ... 2 1) has $n(n-1)/2$ inversions
$\Rightarrow O(n^2/2)$ lower bound

(b) lower bound on comparison based sorting
example: a b c - there are 3! outputs.
(for n numbers there are n! outputs)
Every decision tree to sort n numbers must have atleast n! leaf nodes
Every decision tree to sort n numbers must have atleast 2n! - 1 nodes
Every decision tree to sort n numbers must have depth atleast $\log_2 n!$ leaf nodes
Depth is the lower bound worst case time complexity for this class of algorithms
$\log_2 n! = log_2(n * (n-1) * (n-2) * (n-3).....1)$
$\log_2 n! = log_2(n) + log_2(n-1) + log_2(n-2) + log_2(n-3)..... + log_2(1)$
$\log_2 n! = \sum_{j=1}^{n} \log_2 j$
$\leq \int_1^n \log_2 x dx$
$= (x \log x - x)|_1^n$
$= n \log n - n$

Lower Bound on Comparison-Based Sorting Algorithm
Decision Tree for 3 numbers



Depth of binary tree with $n!$ leaves
$\geq \log_2 n!$
$\geq \int_1^n \log x dx$
$\geq n \log n - n$

9 SHELL SORT (Donald Shell)

Sort subarray comprising every h_i location, for a few selected hop sizes h_i, $k \geq i \geq 1$, and final $h_1 = 1$
$h_1 = 1$ always use insertion sort for sorting

with $h_2 = 1.72n^{1/3}$

$W(n) = (\frac{n}{1.72n^{1/3}})^2 + 1.72n^{1/3} + n^2$
$= \frac{n^2}{1.72n^{1/3}} + n^2$
$= \frac{n^{5/3}}{1.72} + n^2$
$= O(n^2)$

for $h_k = 2^k - 1$, $1 \leq k \leq \lfloor \log n \rfloor$
$W(n) = O(n$

$2^5 - 1 = 31 = (11111)_2$
$2^4 - 1 = 15 = (1111)_2$

for h_k is an integer of the form $2^i 3^j, h_k < n$
$W(n) = O(n(\log n)^2)$

$2^0 3^0, 2^1 3^0, 2^0 3^1, 2^1 3^1, 2^2 3^1$
1 2 3 6 12
too many h'_ks, hence overhead is large.

2 Chapter 9: Sorting in Linear Time

2.1 Counting Sort

(a) Rank each element

i. Count how many times i occurs

ii. Prefix sum on counter array to find how many items $\leq i$

(b) Place at its location.

Start from right, place item in the output array, decrease its corresponding count.

$W(n) = O(n)$ if range is $O(n)$.

Stable sorts – counting sort, mergesort, Insertion sort, Bucket sort.

Nonstable sorts – Quicksort, heapsort.

2.2 Bucket Sort

k Buckets.

Algorithm:

1. hash items among buckets
2. sort the buckets
3. Combine buckets

Let there be k buckets, n items

1. distribution $O(n)$
2. sort the buckets
   $w(n) = O(n \log n)$
   $A(n) = O(k\frac{n}{k} \log \frac{n}{k}) = O(n \log(n/k))$
3. combine buckets $O(n)$.

Thus, bucket sort is

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

$$A(n) = O(n \log \frac{n}{k})$$

If k is constant,

$$A(n) = O(n \log n - n \log k)$$
$$= O(n \log n)$$

$A(n) = O(n)$ if $k = n/20$, $A(n) = O(n)$

Good when item distribution is known so that bucket get equitable number of keys.

Space Usage

worst-case: each bucket should have space for n key (any allocation)

$\Rightarrow$ total $= O(nk)$

Thus, as k increases, average space increases but so does the space requirement.

If linked allocation is used

Space needed $= O(k) + O(n) = O(n + k) = O(n)$

However sorting each bucket using quicksort, mergesort, and heapsort will be difficult which require array representation.

If insertion sort is used to sort linked list, (buckets),

$$A(n) = O(\frac{n^2}{k^2}) * k = O(\frac{n^2}{k}) = O(n) \text{ for k} = \text{O(n)}.$$

2.3 Radix sort

for $i \leftarrow 1$ to d

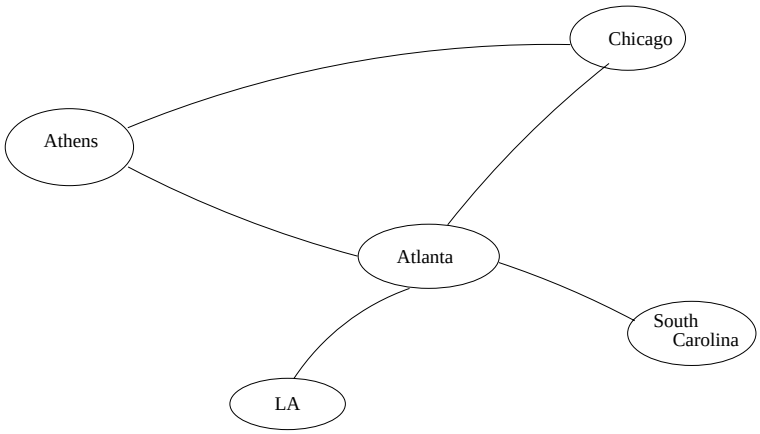
stable sort A on digit i.

(a) counting sort can be used

(b) bucket sort can also be used

2 GRAPHS

Consists of a set of nodes or point some of which are connected by edges or lines.



A (hypothetical) graph of nonstop airline flights.

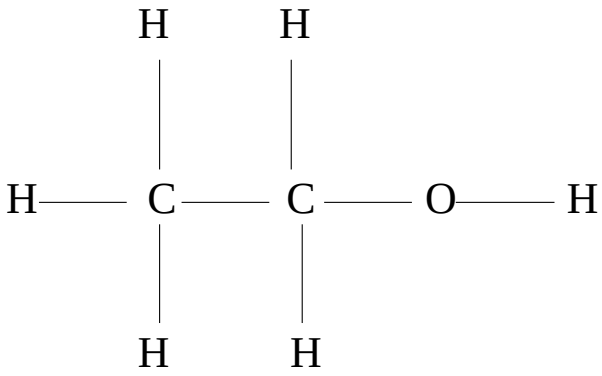
Molecule alcohol: CH_3CH_2OH

Def. A graph $G = (V, E)$ which V is the set of nodes,

$$V = \{v_1, v_2, \ldots, v_n\}$$

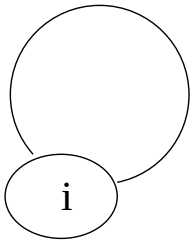
and E is the set of edges,

$$E = \{\{v_i, v_j\} | v_i \in V, v_j \in V\}$$



note:.

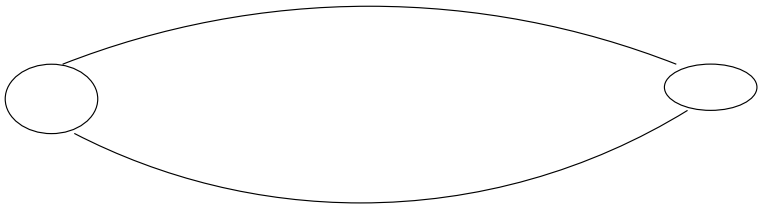
1. each edge is a set of pair of vertexes. No order implied.
2. $v\{v_i, v_j\}$, i could be equal to j.
   self-loop



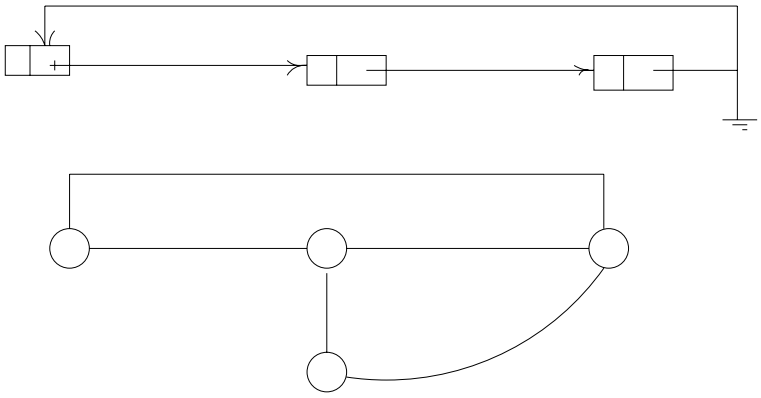
self loop

3. One also allows multiple edge between two node in a general graph

4. Cardinality of V, $|V| = n$, number of nodes, $|E| = m$, number of edges.



3 DIGRAPH



eg. A flow chart of a program

def. A graph whose edges are directed is a digraph, directed graph.
def. A digraph $G_2(V, E)$ when V is the set of vertex $V = \{v_1, v_2, \ldots, v_n\}$ and E is the set of directed edges on every $E = \{(v_i, v_j) such that v_i, v_j \in V\}$

Note

For the purpose of this chapter, we assume that V, the set of nodes, is nonempty & finite and that there no self loops or multiple edges in a graph or digraph.

Question

1. which route is cheapest?
2. which route is fastest?
3. if a node or a computer goes down in a computer network, does it get disconected?
4. is there a loop in a flowchart?

3.1 Subgraph

$V' \subseteq V, E' \subseteq E$
Induced Subgraph by V'
$G' = (V', E'), E' = \{u, v | u, v \in V'\}$

Complete graph
$E = \{\{v_i, v_j | 1 \leq j \leq n\}$ eg. if (v, w) is an edge then v & w are adjacent to each other and v and w are said to be incident with the edge (v, w).

def A path from v to w is a sequencs of vertex $v_0, V_1, v_2, \ldots,$ such that $v_0 = v, v_k = w$& $\{V_i, v_j\} \in E$.
(Books defenition is not good)
If $v_0 = v, v_k = w$& all $v_0, v_1, \ldots, v_k$ are distinct, then the length of the path is k. v_0 is a path of length 0.

Connected Graph:

Cycle is path $v_0v_1v_2 \ldots v_k$ such that $v_0 = v_k$
A graph without any cycle is called acylic.

Tree: Acyclic connected graph

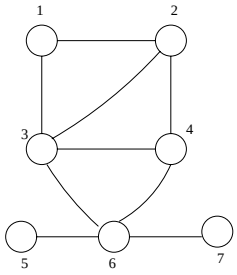
Rooted tree has a designated root vertex establishing parent & child relationship
number of edge in a tree is $n - 1$
Could be proved by induction
A connected component of a graph G is a maximal connected subgraph of G

Weighted Graph
$G = (V, E, w)$
$W : E \leftarrow 2+$
$w(e)$ weight of e (Capacity or distance)

3.2 Representation of a graph

3.2.1 Adjacency Matrix

$A = (a_{ij})_{n*n}$
$a_{ij} = 1$ if $v_i, v_j \in E$
0 else
for $1 \leq i, j \leq n$

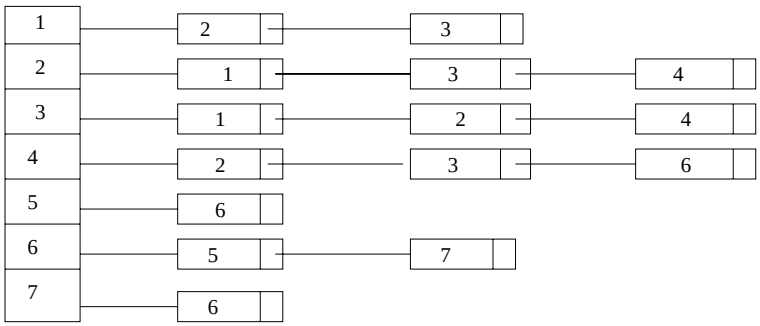


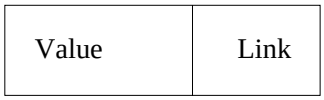
	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	1	0	1	1	0	0	0
3	1	1	0	1	0	0	0
4	0	1	1	0	0	1	0
5	0	0	0	0	0	1	0
6	0	0	1	1	0	1	1
7	0	0	0	0	0	1	0

Space usage $\Leftarrow O(n^2)$

3.2.2 Adjacency list

On any $A[1 \ldots n]$ of linked list, $A[j]$ is the linked list of all the vertices adjacent to v_j.



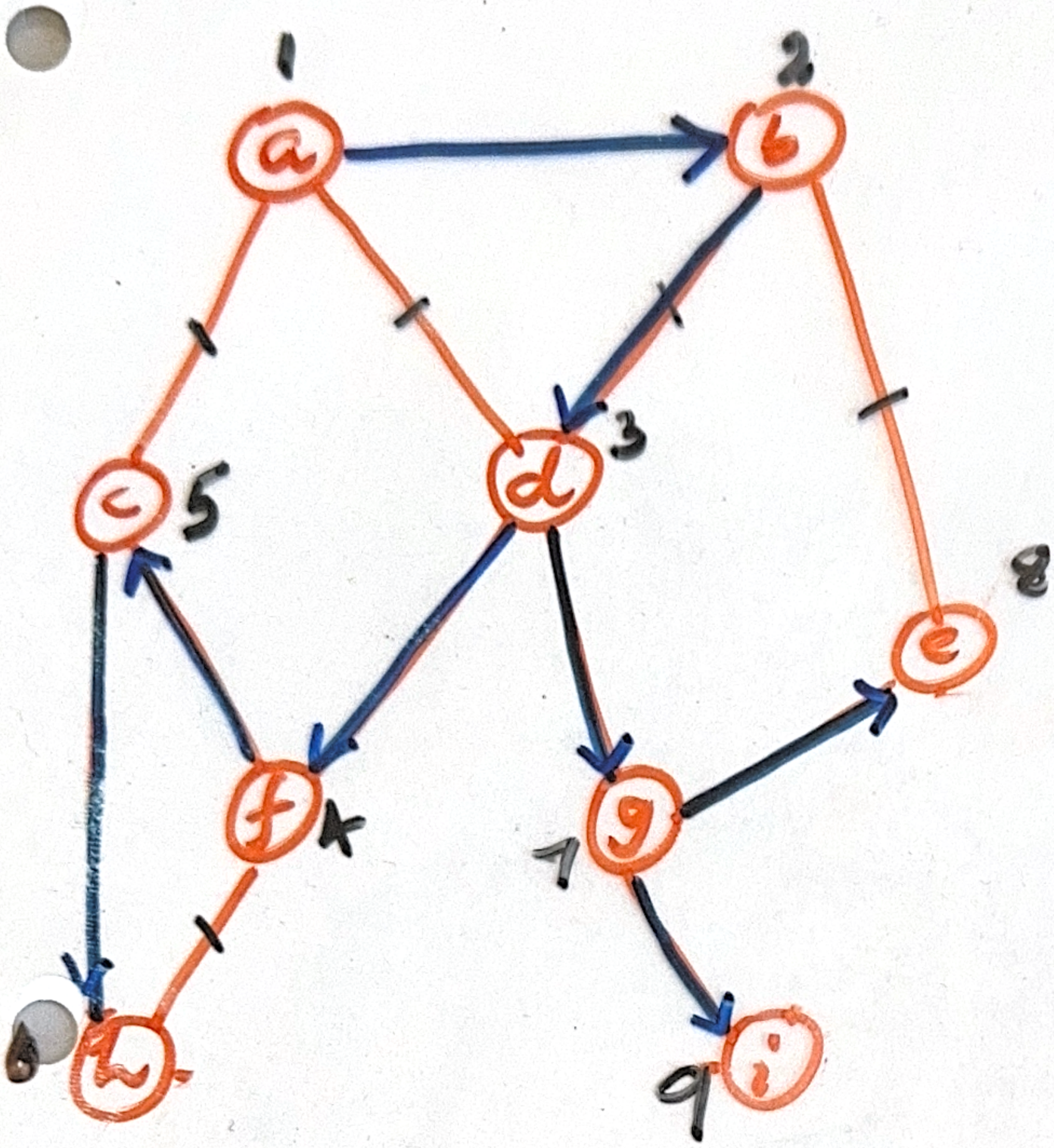


Space usage $\Leftarrow O(n) + O(m) = O(m + n)$
$= O(n^2)$ if m is $O(n^2)$

3.3 Weighted Graph Represatation

Adjacency Matrix $a_{ij} = W(v_i, v_j)$ if $\{v_i v_j\} \in E$
$= 0$ otherwise
Adjacent link

Traversing Graphs



→ Tree Edge

⊣ Back Edge

DFS (v) : Depth-First Search

mark v

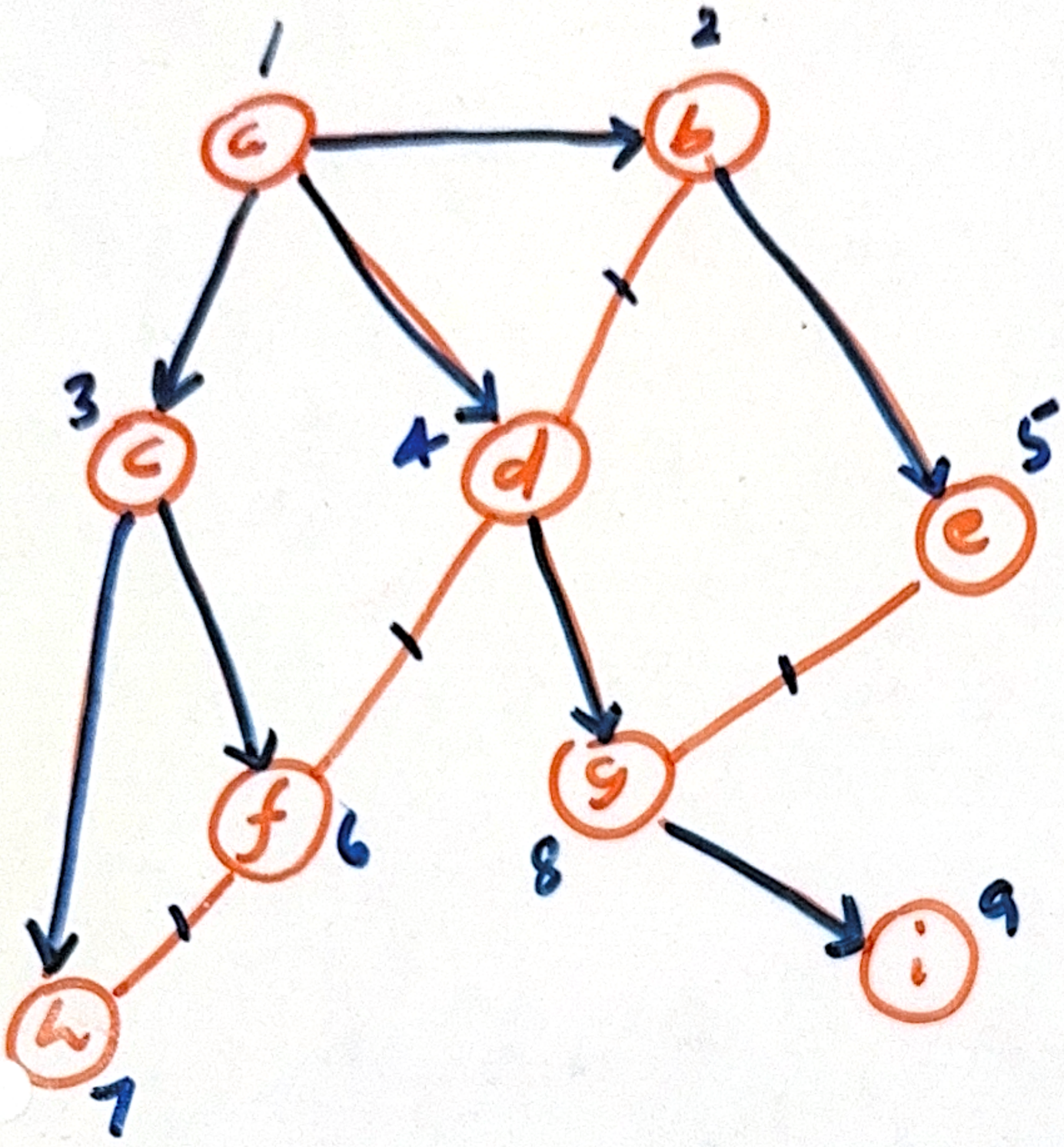
while there is an unmarked node

w adjacent to node v

DFS(w)

end

[Connected Components]



$O(m+n)$

+ cross edges

BFS (v) : Breadth-First SEARCH

mark v

Q = Qv

Q

v

while Q is nonempty do

x = deleteFront (Q)

x

for each unmarked node

w adjacent to x do

{ mark w

Q = Qw

end

4 TRAVERSING GRAPHS

DFS(v): Depth-First Search
mark v
while there is an unmarked node w adjacent to node v DFS(w)
end

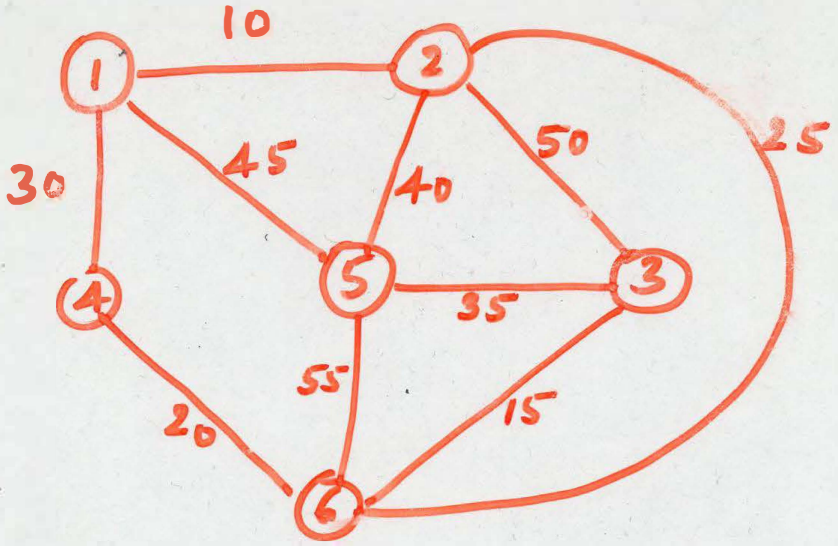
connected components
$O(m+n)$

BFS(v): Breadth-First Search
mark v
Q=Qv
while Q is nonempty do

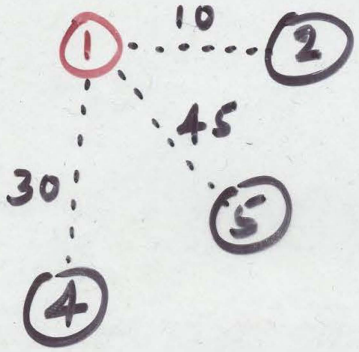
x = delete Front(Q)

for each unmarked node w adjacent to x do mark w
Q=Qw

PRIM-DIJKSTRA'S MST ALGORITHM



PARTIAL MST

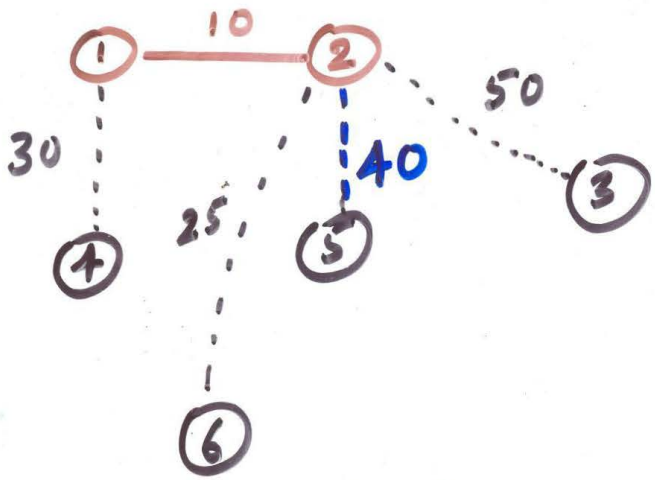


Fringe Vertices

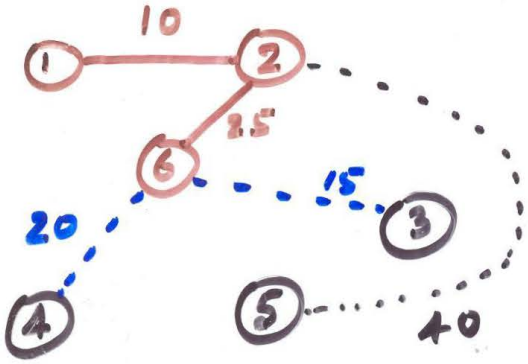
3 unseen vertices

6

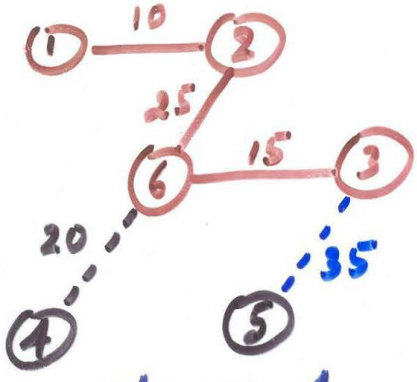
Choose node 2



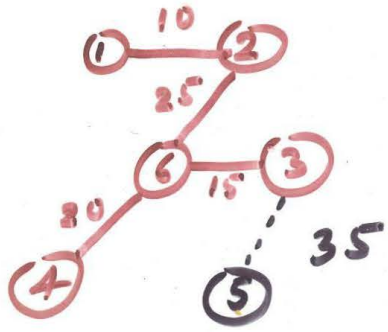
choose node 6



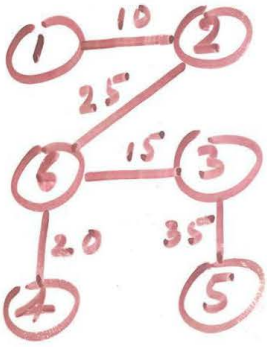
choose node 3



choose 4



choose 5



MST
W = 105

PRIM-DIJKSTRA MST ALGORITHM

Input: $G = (V, E, w)$

Output: $T = (V_T, E_T)$

$E_T = \emptyset$

Select a vertex $v \in V$ and move v to V_T: $V_T = \{v\}$, $V = V - \{v\}$

For $i := 1$ to $n-1$ do

$\Theta(n^2)$ Let $\{v, w\}$ be an edge such that $v \in V_T$, $w \in V$, and for all such edges, $\{v, w\}$ has the minimum weight.

$V_T = V_T \cup \{w\}$

$E_T = E_T \cup \{\{v, w\}\}$

$V = V - \{w\}$

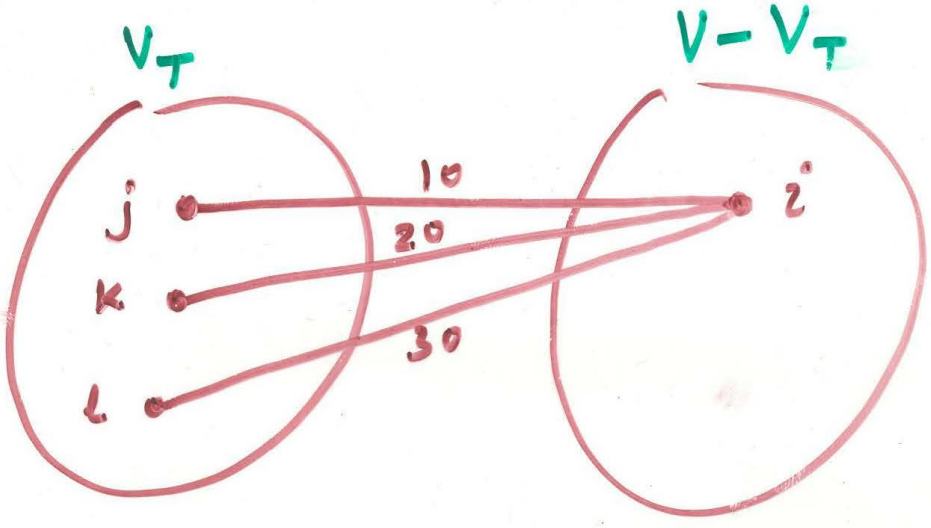
endfor

$W(n) = O(n^3)$

DATA STRUCTURE FOR PRIM'S ALGORITHM

NEAR[1..n]

$$NEAR[i] = \begin{cases} 0 & \text{if } i \in V_T \\ j & \text{if } w(i,j) \text{ is minimum among all } j \in V_T \end{cases}$$



NEAR[i] = j

NEAR[j] = 0

NEAR[l] = 0

MODIFIED PRIM'S ALGORITHM

Input $G = (V, E, w)$, a connected weighted graph

Output $T = (V_T, E_T)$, a MST.

1. $E_T = \emptyset$
2. NEAR[1] = 0 /* $V_T = \{1\}$ */
   NEAR[2..n] = 1 /* $V = V - \{1\}$ */
3. FOR i = 1 to n-1 do

   $O(n)$ $O(\log n)$ FIND j such that NEAR[j] ≠ 0 AND w(j, NEAR[j]) is min.

   NEAR[j] = 0 /* $V_T = V_T \cup \{j\}$ */

   $E_T = E_T \cup \{\{j, NEAR[j]\}\}$.

   /* update NEAR[1..n] */

   for k = 1 to n do

   $O(n \log n)$ DECREASE-KEY if NEAR[k] ≠ 0 AND w(k, NEAR(k)) > w(k, j) then NEAR[k] = j ←

   endfor

endfor.

$w(n) = O(n^2)$

Thm Let $G = (V, E, w)$ be a weighted connected graph and $T = (V, E_T)$ be a MST of G. Let $T' = (V', E')$ is a subtree of T. If $\{x, y\}$ is the minimum weight edge such that

$x \in V'$ and $y \in V - V'$

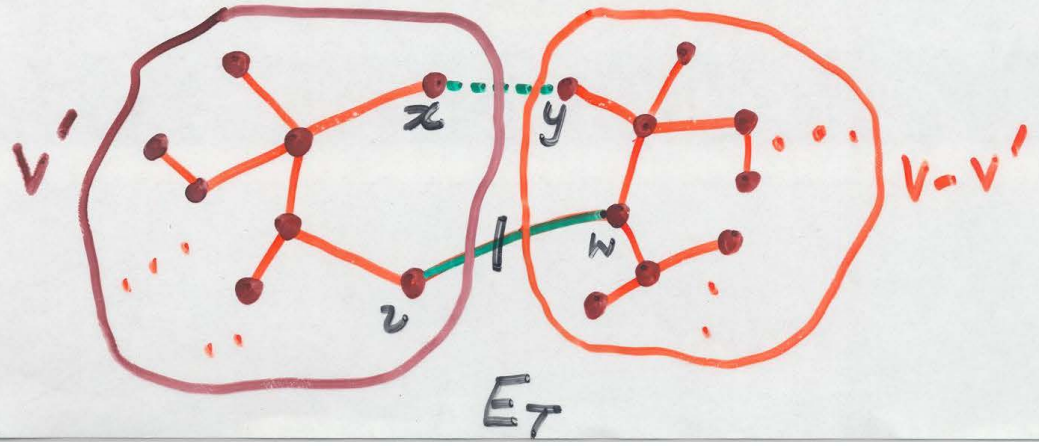
then $T'' = (V' \cup \{y\}, E' \cup \{\{x, y\}\})$

is a subtree of a MST of G.

Proof

1. If $\{x, y\} \in E_T$, done.
2. Let $\{x, y\} \notin E_T$.

   Then $E_T \cup \{\{x, y\}\}$ has a cycle:



By the choice of $\{x, y\}$

$$w(\{x, y\}) \leq w(\{v, w\})$$

Consider $E_T \cup \{\{x, y\}\} - \{\{v, w\}\}$

Its weight is no more than the weight of T and it is a spanning tree.

$\Rightarrow E' \cup \{\{x, y\}\}$ is a subtree of a MST of G.

□

Kruskal's Algorithm

input: $G = (V, E, W)$, a connected graph

output: $T = (V, E_T)$, a MST of G.

$T \leftarrow \emptyset$

while $|T| < n-1$ do

Let $\{v, w\}$ be the least cost edge in E.

$E \leftarrow E - \{\{v, w\}\}$

if $\{v, w\}$ does not create a cycle in T

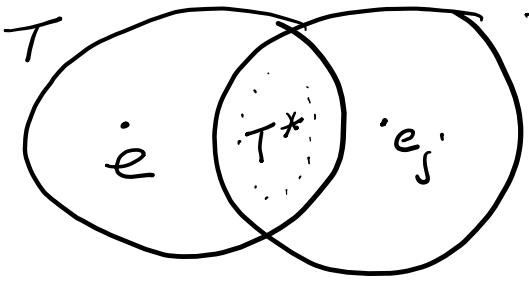
then add $\{v, w\}$ to T

end while

Kruskal's MST Algorithm
Proof of Correctness

Th: Let T be the spanning tree for G generated by kruskal's algorithm. Let T' be a minimum cost spanning tree for G. Show that $W(T) \leq W(T')$. and T is a MST.

Proof:



Let $e \in T - T'$ of min weight.

$\{e\} \cup T'$ has a cycle.

Let $e_j \in T' - T$ in this cycle.

$W(e_j) \geq W(e)$ else $e_j \in T$ by kruskal's

{Why? e_j would have been considered before e for inclusion into a subset $T^* \subseteq T \cap T'$

$\Rightarrow \{e_j\} \cup T^*$ would not form cycle because $T^* \subset T'$.

$\Rightarrow W(T') \geq W(\underbrace{T' - \{e_j\} \cup \{e\}}_{T''}) \geq W(T)$

$\Rightarrow T$ is a MST. $\square$

Repeat for each $e \in T - T'$ $\Rightarrow$ T' becomes T

Kruskal's Algorithm (Refined)

OPERATIONS

1. UNION(i,j), of sets i & j, contains elements of sets i and j.
2. FIND(v) = i iff v ∈ set i.

a) Construct a min-heap E of edges in E.

b) Each $v \in V$ forms singleton set by itself, such that FIND(v) = v.

c) $E_T = \phi$ {Tree is empty}

d) while $|E_T| < n-1$ do

log m Delete the root edge {v,w} from min-heap E and restore heap E.

O(1) O(log n) if FIND(v) ≠ FIND(w)
{$E_T \cup \{\{v,w\}\}$ has no cycle}

then

O(1) $E_T = E_T \cup \{\{v,w\}\}$

/* now, combine the components of E_T joined by {v,w} into one */

O(1) UNION (FIND(v), FIND(w))

end if

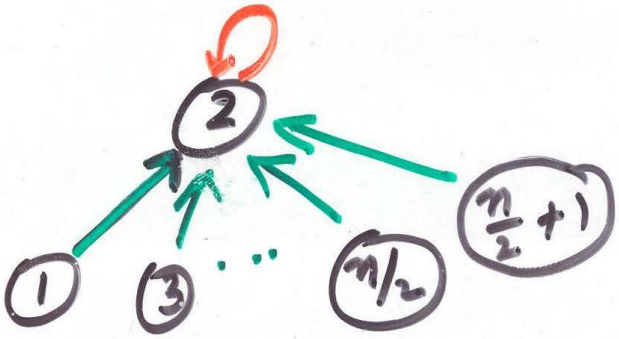
end while O(m log m)

1 2 ... n

1. UNION (1, 2)
2. UNION (2, 3)
⋮
n/2. UNION (n/2, n/2+1)

n/2+1 FIND (1)
FIND (1)
⋮
n FIND (1)

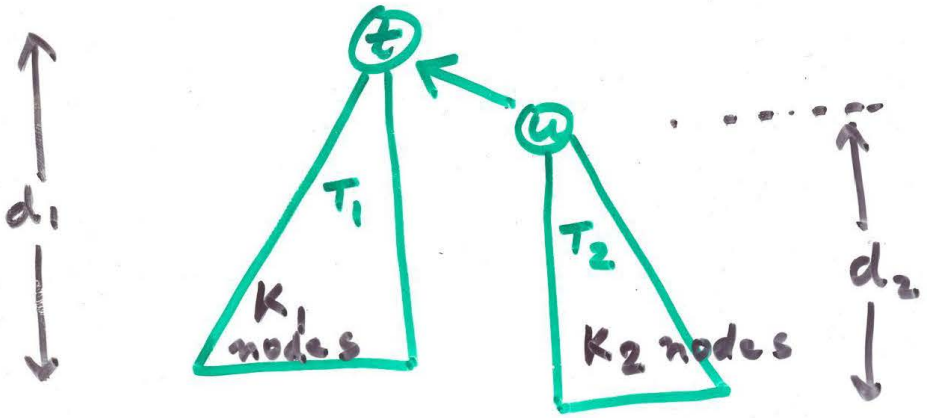
TREE WITH UNWEIGTED UNION



TREE WITH WEIGTED UNION

Lemma With W-UNION, ANY TREE THAT HAS K NODES HAS DEPTH AT MOST $\lfloor \lg K \rfloor$.

W-UNION (t, u)



Let $k_1 \geq k_2$, $K = k_1 + k_2$

$d = d_1$ if $d_1 > d_2$

$= d_2 + 1$ if $d_1 \leq d_2$

BASIS: $K = 1$ $\lfloor \lg 1 \rfloor = 0$

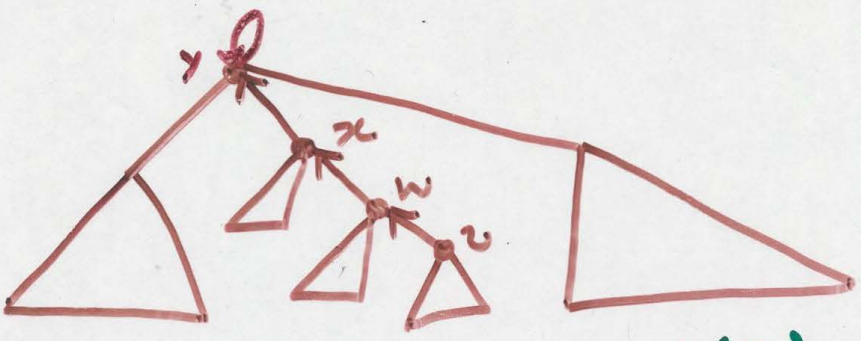
HYPOTHESIS: for $k' < k$, depth $\leq \lfloor \lg k' \rfloor$

INDUCTION

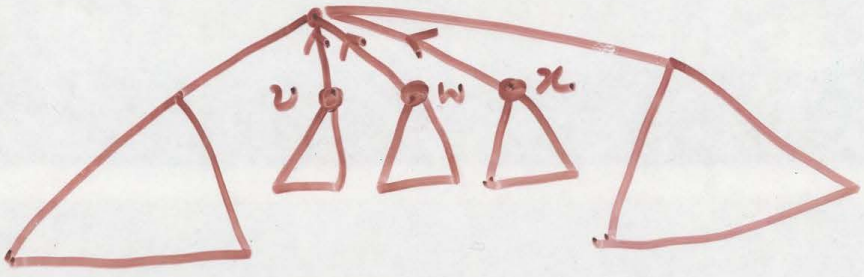
I. $d = d_1 \leq \lfloor \lg k_1 \rfloor \leq \lfloor \lg (k_1 + k_2) \rfloor = \lfloor \lg K \rfloor$

II. $d = d_2 + 1 \leq \lfloor \lg k_2 \rfloor + 1 = \lfloor \lg 2k_2 \rfloor \leq \lfloor \lg (k_1 + k_2) \rfloor$

Lemma A UNION-FIND PROGRAM OF SIZE n DOES $\Theta(n \lg n)$ link OPERATIONS IN THE WORST CASE IF THE WEIGTED UNION IS USED.



BEFORE C-FIND(v)
COMPRESSING-FIND(v)



AFTER C-FIND(v)

Lemma THE NUMBER OF LINK OPERATIONS DONE BY A UNION-FIND PROGRAM OF LENGTH n IMPLEMENTED WITH W-UNION AND C-FIND IS $O(n G(n))$.

Tarjan & Hopcroft

$G(n) = \log^* n$

$= \text{SMALLEST } i \text{ SUCH THAT}$

$\log^{(i)} n \leq 1$

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

and $\log^{(0)} n = n$

$G(65536) = \log^*(2^{16}) = 4$

$\log 2^{16} = 16, \quad \log 16 = 4, \quad \log 4 = 2, \quad \log 2 = 1$

$G(2^{65536}) = 5. \Rightarrow G(n) \leq 5$ for all reasonable n.

[pp. 498-508 ch 22]

COMPARISON

m	$O(n)$	$O(\frac{n^2}{\log n})$	$O(n^2$
Kruskal's	$O(n \log n)$	$O(n^2)$	$O(n^2 \lg n)$
Prim's	$O(n^2)$	$O(n^2)$	$O(n^2$

Kruskal's
$O(m \log m)$
$= O(m \log n)$

PRIM's
$O(n^2)$

Time Complexity of Prim's

- with heap $O(n \log n + m \log n)$
- with Fibonacci Heap
  $O(n \log n + m)$

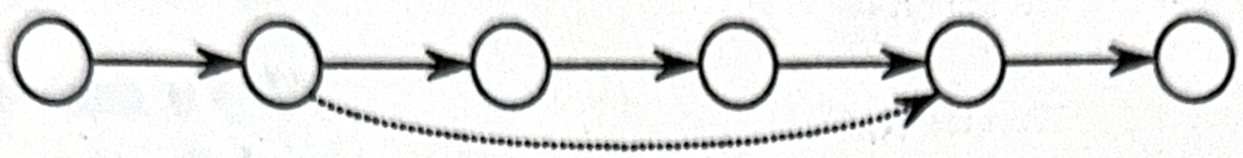
Shortest paths — from Chapter 25 introduction and Section 25.1

- *Generalization of breadth-first-search (from last lecture) to handle weighted graphs.*
- Directed graph $G = (V, E)$, weight function $w : E \to \mathbf{R}$
  (BFS — $w(e) = 1 \ \forall e$)
- e.g. street map (with distances on roads between intersections)
- Weight of path $p = v_0 \to v_1 \to \cdots \to v_k$ is

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \ .$$

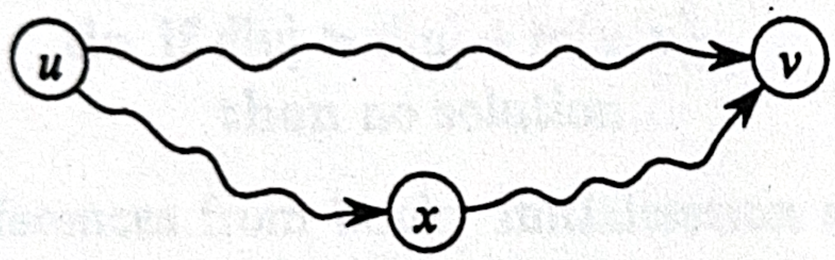
- "Shortest" path = path of minimum weight

- **Optimal substructure**
  (Thus, will see greedy and dynamic-programming algorithms.)
  Theorem: Subpaths of shortest paths are shortest paths.
  ⟨⟨Lemma 25.1, with both statement and proof in English rather than in mathematics.⟩⟩
  Proof: By "cut and paste"



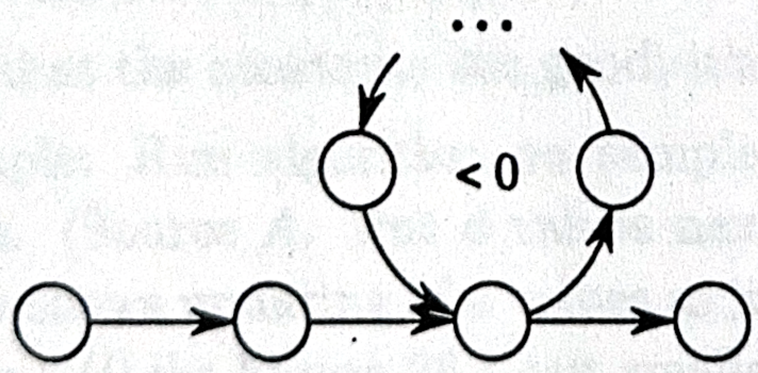
  If some subpath were not a shortest path, could substitute the shorter subpath and create a shorter total path.

- Def: $\delta(u, v) =$ weight of a shortest path from u to v
- **Triangle inequality**
  Theorem: $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$
  ⟨⟨Generalization of Lemma 25.3⟩⟩
  Proof:



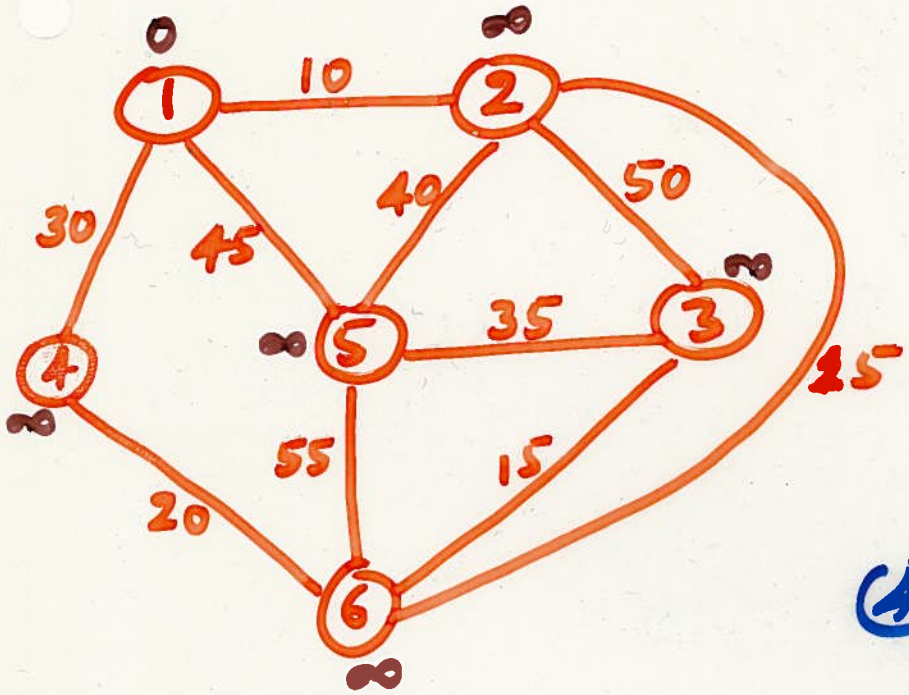
  Shortest path $u \rightsquigarrow v$ is no longer than any other path — in particular, the path that takes the shortest path $u \rightsquigarrow x$, then the shortest path $x \rightsquigarrow v$.

- **Well-definedness**
  Negative-weight cycle in graph ⇒ some shortest paths may not exist.
  Argument: Can always get a shorter path by going around the cycle again.

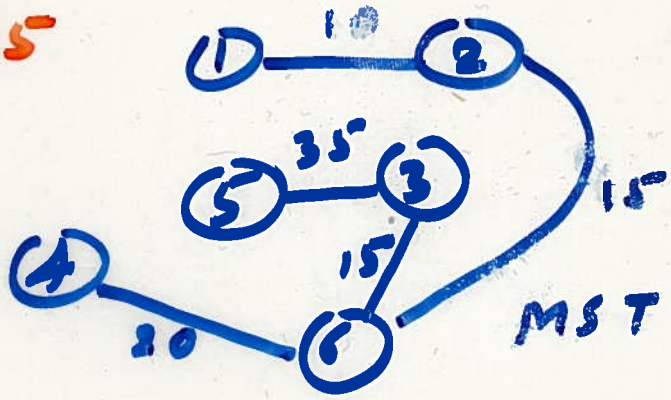


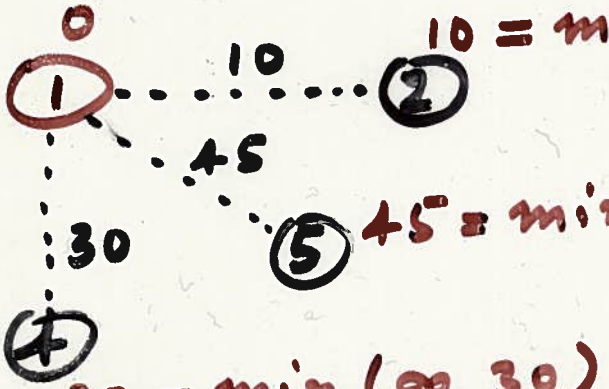
- (Above were high points of theory — rest of theory is in book.)

SHORTEST PATH



Q: Find a shortest path from 1 to 3



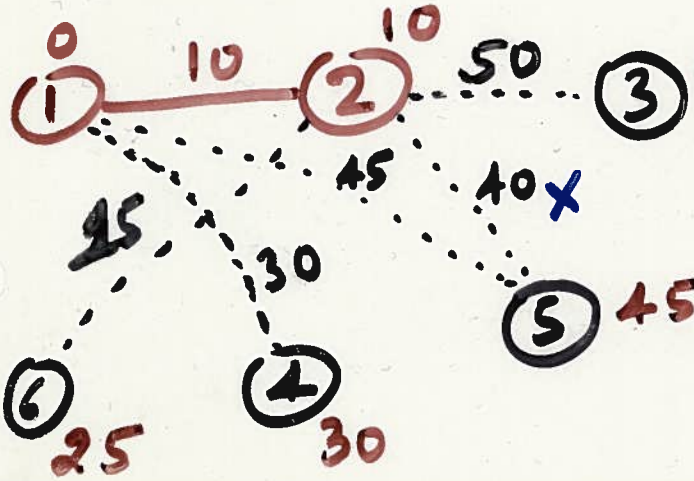


$10 = \min(\infty, 10)$

$45 = \min(\infty, 45)$

$30 = \min(\infty, 30)$

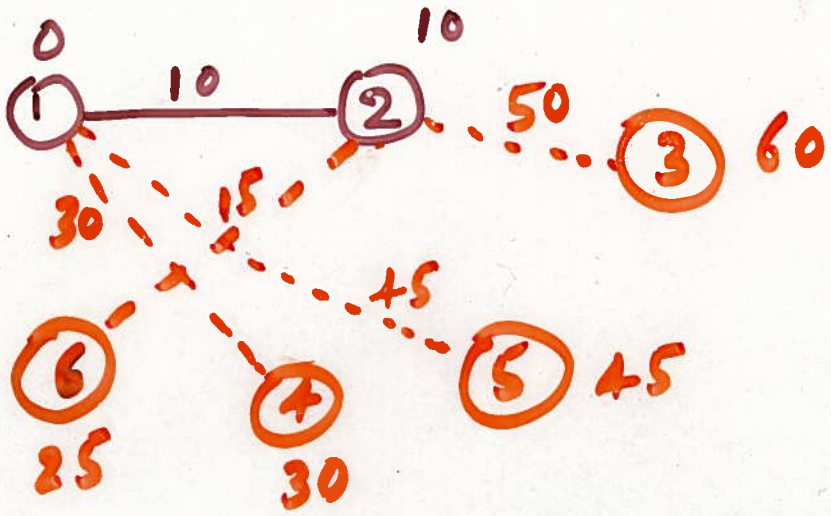
choose 2



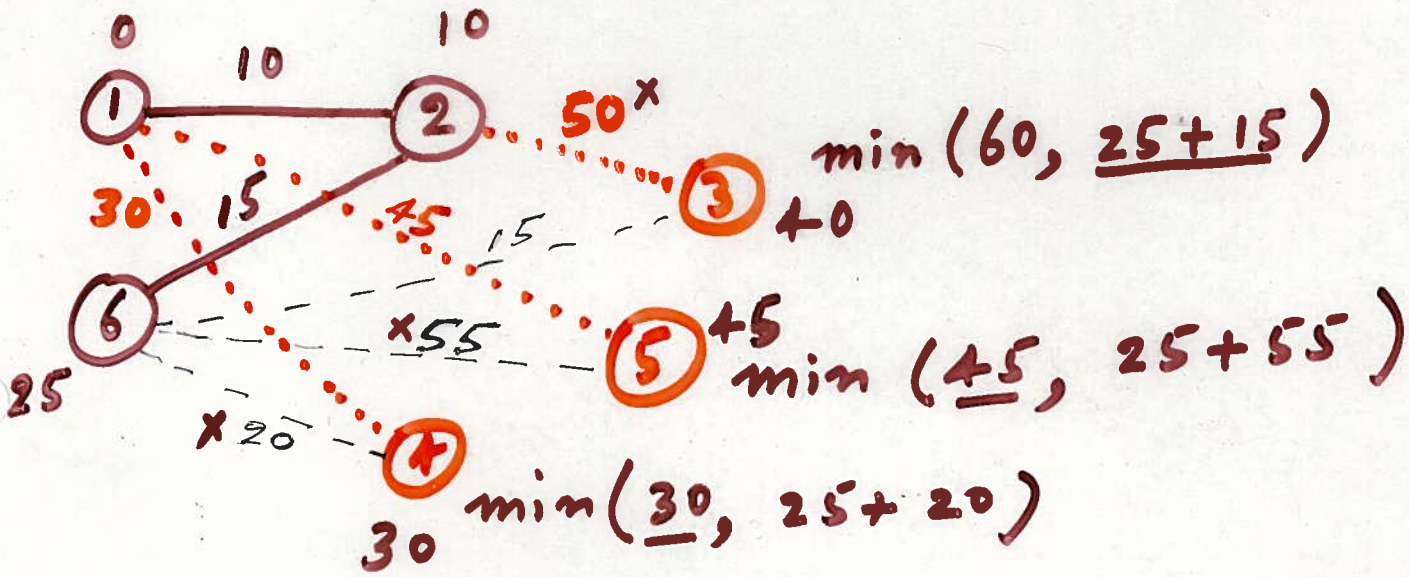
$60 = \min(\infty, \underline{10+50})$

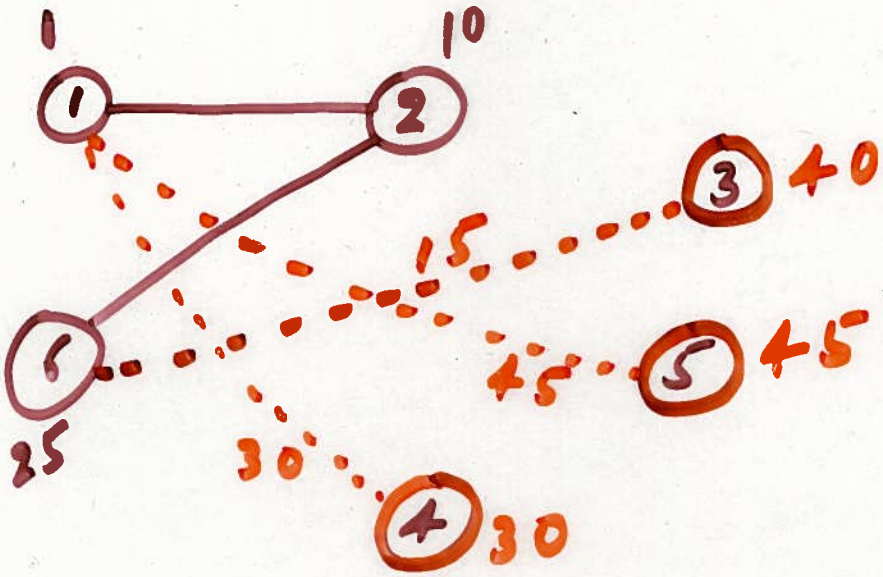
$45 = \min(\underline{45}, 10+40)$

choose 6

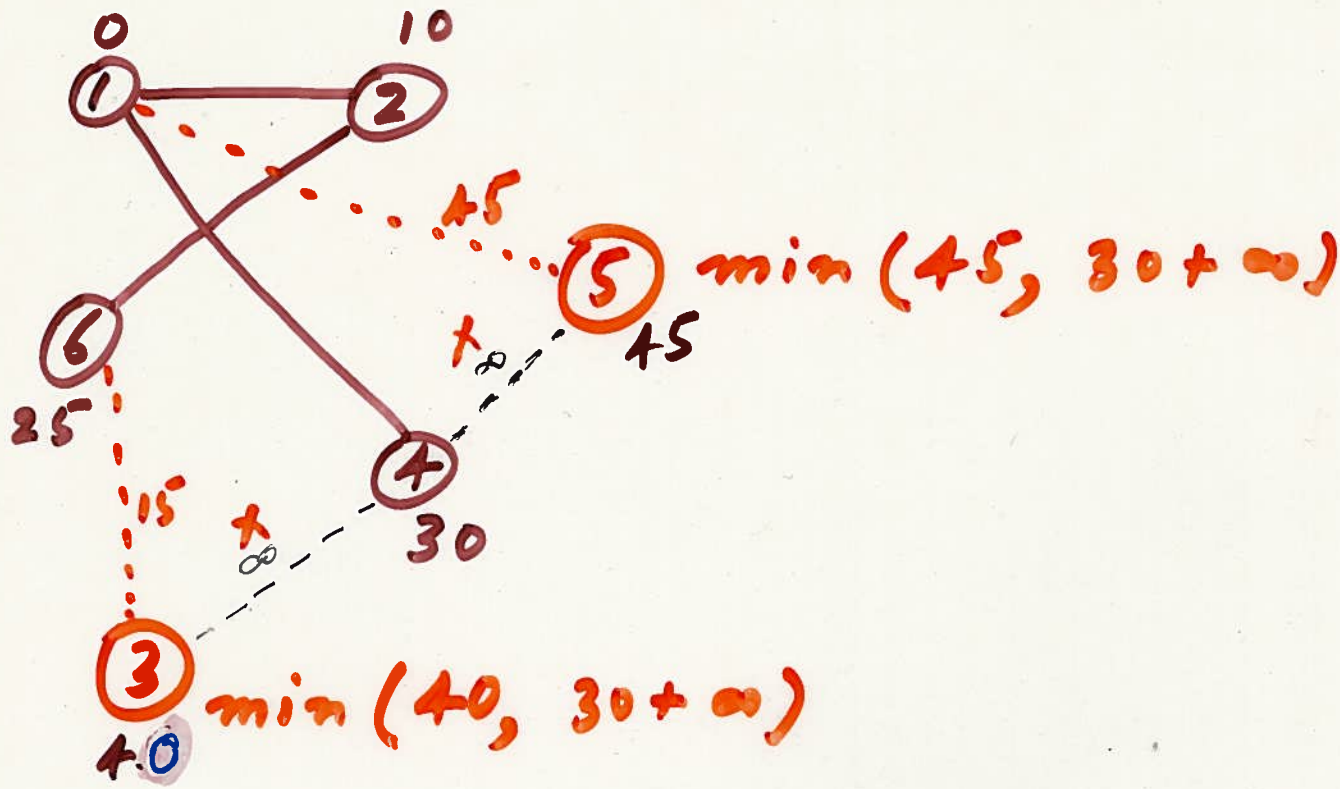


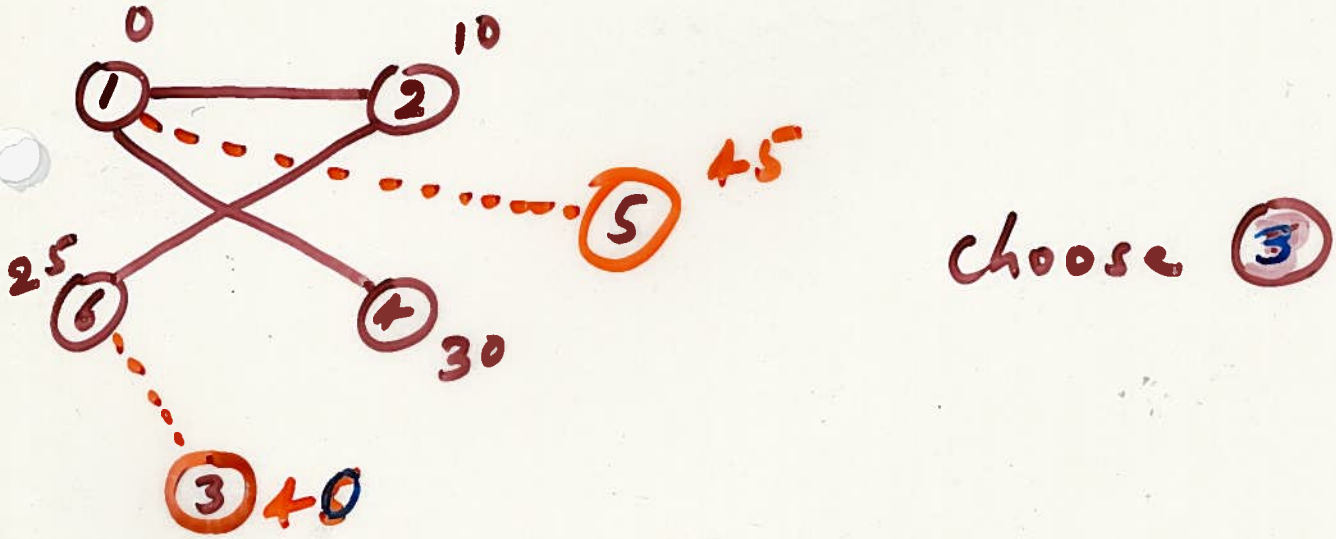
Choose 6



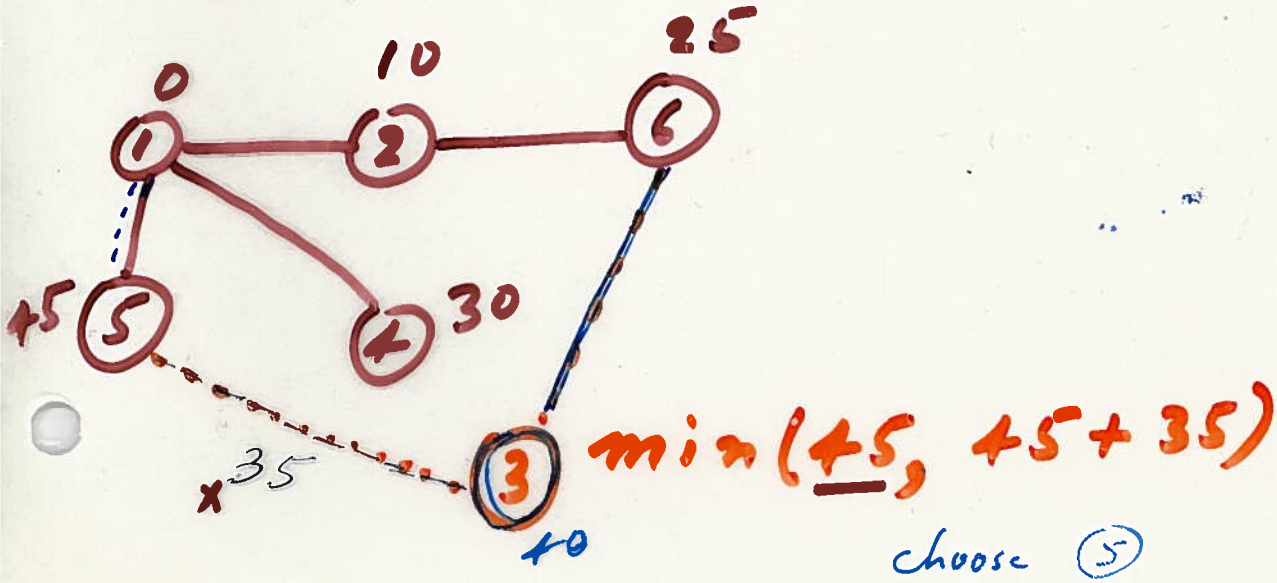


Choose 4

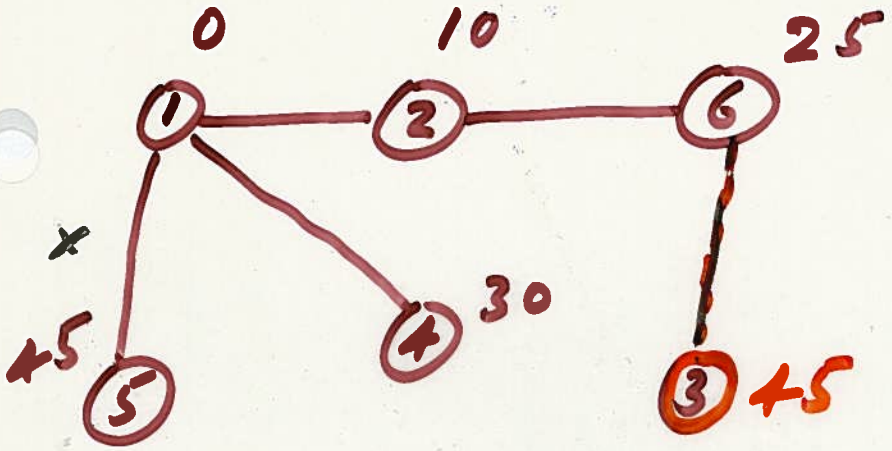




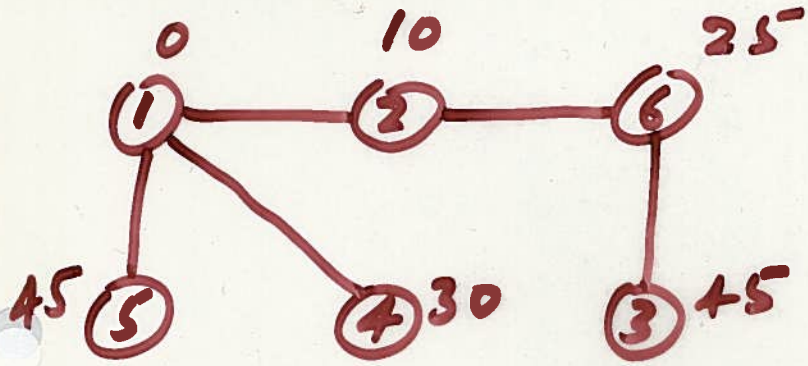
choose (3)



choose (5)



Choose 3



Shortest Path Tree with root node 1

Assumption: No negative weight.

```
DIJKSTRA(G)
    for each v ∈ V
        do d[v] ← ∞
    d[s] ← 0
    S ← ∅
    Q ← V
    while Q ≠ ∅
        do u ← EXTRACT-MIN(Q)
           S ← S ∪ {u}
           for each v ∈ Adj[u]
               do if d[v] > d[u] + w(u, v)
                     then d[v] ← d[u] + w(u, v)
```

Q : Priority Queue

* DECREASE KEY (v)

Q	n* Extract-Min + m* Decrease-key	
array	$n * O(n) + m * O(1)$	$= O(n^2)$
heap	$n * \log(n) + m * \log n$	$= O(m \log n)$
fibonacci	$n * \log(n) + m * O(1)$	$= O(n \log n + m)$

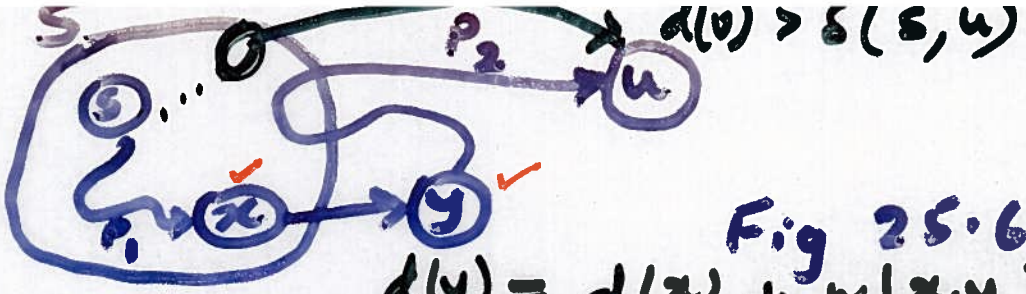


Fig 25.6

Correctness: Prove that whenever u is added to S, $d[u] = \delta(s,u)$
⟨⟨Theorem 25.10⟩⟩

Proof: ⟨⟨Basically same as in book, but derives a different contradiction.⟩⟩

- Figure 25.6
- Note that $\forall v\ d[v] \geq \delta(s,v)$
  (because lemma proved for Bellman-Ford above was just about relaxation, didn't depend on order of relaxing edges)
- Let u be first vertex picked such that $\exists$ shorter path than $d[u]$
  $\Rightarrow d[u] > \delta(s,u)$ — (i)
- Let y be first vertex $\in V - S$ on actual shortest path from s to u
  $\Rightarrow d[y] = \delta(s,y)$ — (ii)
  Because:
  - $d[x]$ is set correctly for y's predecessor $x \in S$ on the shortest path (by choice of u as first choice for which that's not true)
  - when put x into S, relaxed (x,y), giving $d[y]$ correct value,

if $y = u$, we are done.

$$\begin{aligned} d[u] &> \delta(s,u) && \text{— (i)} \\ &= \delta(s,y) + \delta(y,u) && \text{(optimal substructure)} \\ &= d[y] + \delta(y,u) && \text{(ii)} \\ &\geq d[y] && \text{(no negative weights)} \end{aligned}$$

- But $d[u] > d[y]$ $\Rightarrow$ algorithm would have chosen y to process next, not u.
  Contradiction.

Bellman. Ford Algorithm

1. **for** each $v \in V$
   do $d[v] \leftarrow \infty$
   $d[s] \leftarrow 0$

$O(nm)$

2. **for** $i \leftarrow 1$ **to** $|V| - 1$
   do for each edge $(u, v) \in E$
   do if $d[v] > d[u] + w(u, v)$
   then $d[v] \leftarrow d[u] + w(u, v)$

3. **for** each edge $(u, v) \in E$
   do if $d[v] > d[u] + w(u, v)$
   then no solution

Relaxation

Lemma: $d[v] \geq \delta(s, v)$ always.

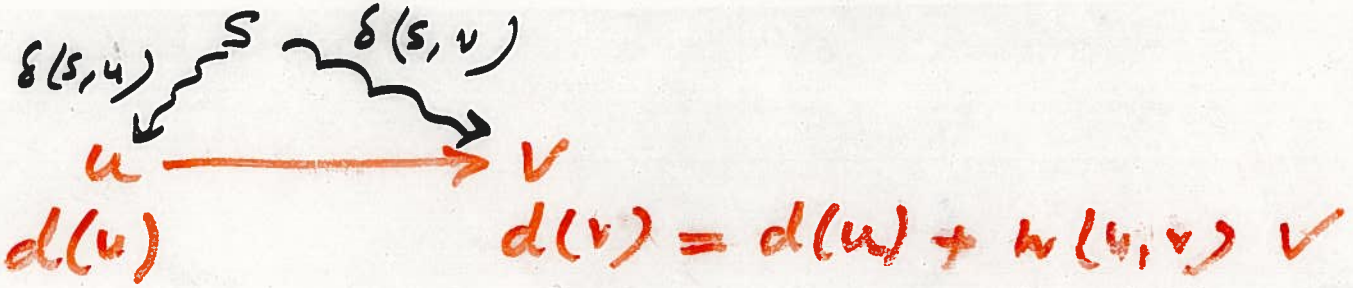
⟨⟨First part of Lemma 25.5, Section 25.1⟩⟩

- Initially true
- Let v be first vertex for which $d[v] < \delta(s, v)$, and let u be vertex that caused $d[v]$ to change:
- $d[v] = d[u] + w(u, v)$ — (i)
- Then

$$\begin{aligned} d[v] &< \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) && \text{(Triangle inequality)} \\ &\leq d[u] + w(u, v) && (v \text{ is first violation}) \end{aligned}$$

$\Rightarrow d[v] < d(u) + w(u,v)$ — (ii)

contradicts $d[v] = d[u] + w(u, v)$ (above).



- Bellman-Ford is correct (after $|V|-1$ passes, all the d values are correct) ⟨⟨Lemma 25.12⟩⟩

  Proof: ⟨⟨Informal version of book's proof.⟩⟩

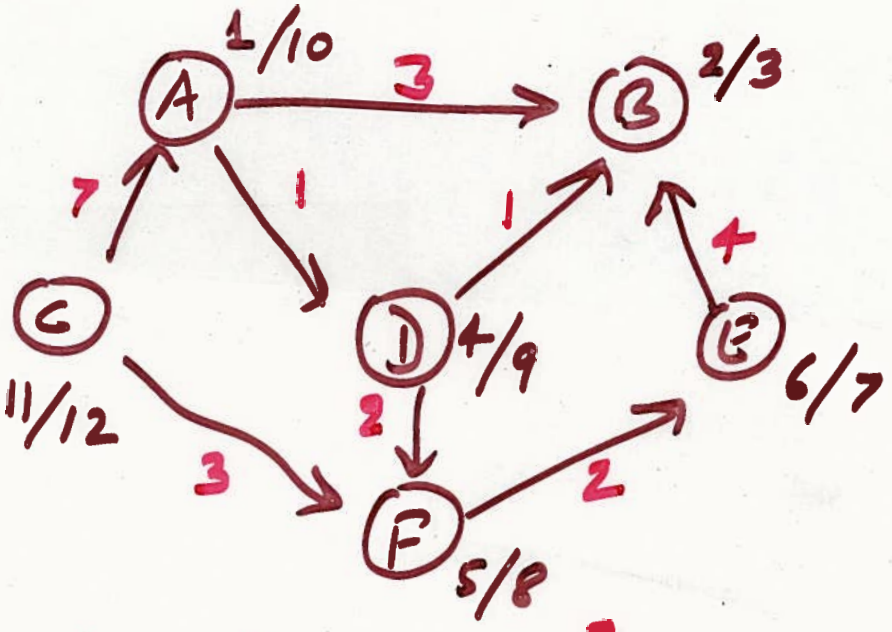
  Let v be a vertex, and consider shortest path from s to v (assuming no neg-weight cycles):

  $s \to v_1 \to v_2 \to v_3 \to v_4 \to v$

  - Initially, $d[s] = 0$ is correct (and doesn't change thereafter — by previous lemma and fact that code never increases d)
  - After 1 pass through edges, $d[v_1]$ is correct (and doesn't change...) ($d[s]$ is correct, and by optimal substructure, shortest distance is $w(s, v_1)$. 1st pass sets $d[v_1] = d[s] + w(s, v_1)$, which is right answer.)
  - After 2 passes through edges, $d[v_2]$ is correct (and doesn't change...)

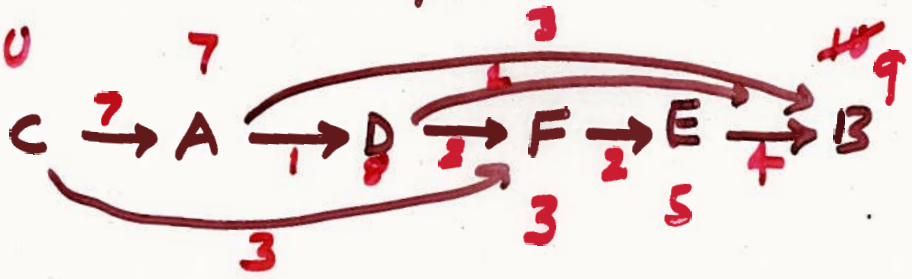
    ⋮

TOPOLOGICAL SORTING



DAG & DFS

(u)
visiting time
finishing time

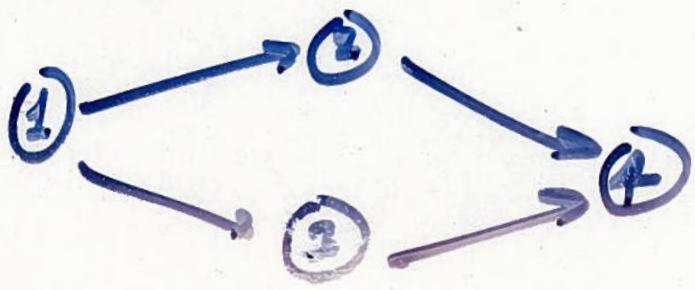


SHORTEST PATH IN DAGs

1. TOPOLOGICAL SORT USING DFS
2. ONE ITERATION OF BELLMAN-FORD ALGORITHM

$O(n+m)$

1 2 3 4
1 3 2 4



All Pair Shortest Path

1. Bellman-Ford n times $= O(n \cdot nm) = O(n^2 m) = O(n^4)$ for dense.
   (↳ adj. list representation)
2. Dijkstra $O(n \cdot n^2) = O(n^3)$ for dense
   (↳ priority queue of edges)
   $O(n(n+m)\log n) = O(n^2 + mn)\log n)$
   $= O(n^2 \log n)$ for sparse

- We will do 2 dynamic prog. algorithms w. tight codes (small coefficients) on $O(n^3 \log n)$ and another $O(n^3)$

goal create $n \times n$ matrix of shortest-path distances $\delta(u,v)$

graph Representation (adj matrix)

$W_{n \times n} = (w_{ij})$ of edge weights.

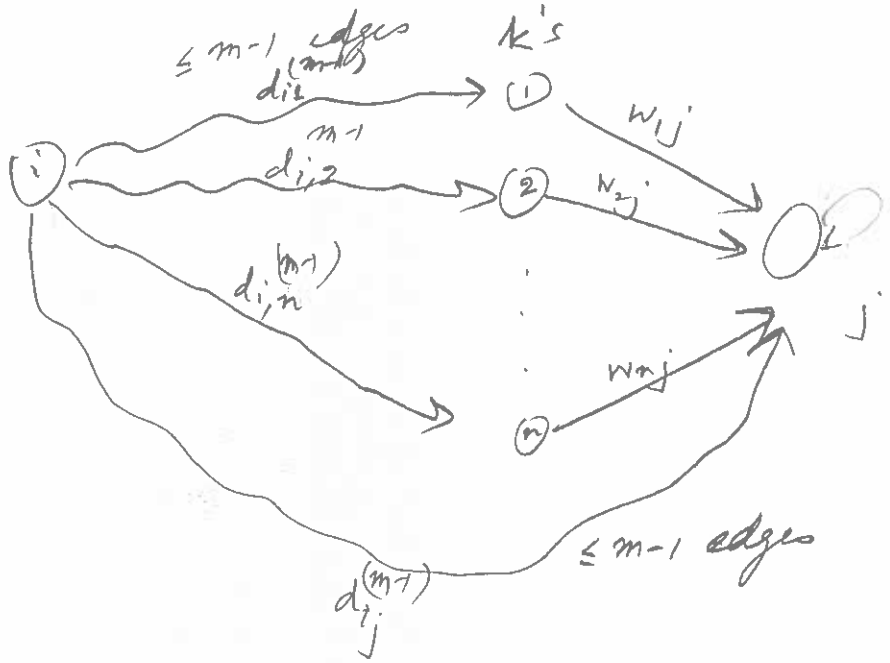
$w_{ii} = 0$

#IET

Let $d_{ij}^{(m)}$ = weight of shortest-path from i to j that uses at most m edges

$$d_{ij}^{(0)} = \begin{cases} 0 & i = j \\ \infty & i \neq j \end{cases}$$

$d_{ij}^{(m)} =$



$$d_{ij}^{(m)} = \min\left\{ d_{ij}^{(m-1)} , \min_{k}\left\{ d_{ik}^{(m-1)} + w_{kj} \right\} \right\}$$

$$= \min\left\{ d_{ij}^{(m-1)} , \left\{ d_{ij}^{(m-1)} + w_{jj} \right\} , \min_{k \neq j}\left\{ d_{ik}^{(m-1)} + w_{kj} \right\} \right\}$$

($k = j$: $w_{jj} = 0$)

$$d_{ij}^{(m)} = \min_{1 \le k \le n}\left\{ d_{ik}^{(m-1)} + w_{kj} \right\} \quad \text{—— (1)}$$

Answer: $\delta(i,j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \cdots$

Pseudocode for 'relaxation step'

for $k \leftarrow 1$ to n — $O(n)$

  if $d_{ij} > d_{ik} + w_{kj}$

    then $d_{ij} \leftarrow d_{ik} + w_{kj}$

$d_{ij}^{(n-1)}$ — $O(n^2 \cdot n)$

Time: $O(n \cdot n^3) = O(n^4)$

Similarity to Matrix Multiplication

$C_{n\times n} = A_{n\times n} \cdot B_{n\times n}$ $n \times n$ matrices

$$c_{ij} = \sum_{1\le k\le n} a_{ik} \cdot b_{kj} \qquad -(2)$$

Compare with

$$d_{ij}^{(m)} = \min_{1\le k\le n} \{ d_{ik}^{(m-1)} + w_{kj} \} \qquad -(1)$$

Replace $c_{ij} \to d_{ij}^{(m)}$
$a_{ik} \to d_{ik}^{(m-1)}$
$b_{kj} \to w_{kj}$
$+ \to \min$
$\cdot \to +$

$$\Rightarrow D^{(m)} = D^{(m-1)} \text{ "}\times\text{" } W \qquad ---- \qquad (3)$$

---

identity for "$\times$"

mat mult $\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & & & \\ \vdots & & \ddots & & \\ 0 & & & & 1 \end{bmatrix} \to \begin{bmatrix} 0 & \infty & \infty & \cdots & \infty \\ & 0 & & & \\ & & \ddots & & \\ & & & & 0 \end{bmatrix}$

$= D^{(0)}$

min	+
∞	0

+	*
0	1

$D^{(1)} = D^0 W = W$

$D^2 = D^1 W = W^2$

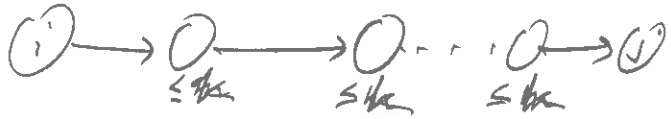
$D^{(n-1)} = W^{n-1}$ $\qquad n\left(\Theta(n^3)\right) = \Theta(n^4)$

But $W \to W^2 \to W^4 \to \cdots \to W^{2^{\lceil \log_2 n-1 \rceil}}$ Overshoot, no problem $\Theta(n^3 \log n)$

Floyd-Warshall Algorithm

Dynamic Prog but faster : $\Theta(n^3)$

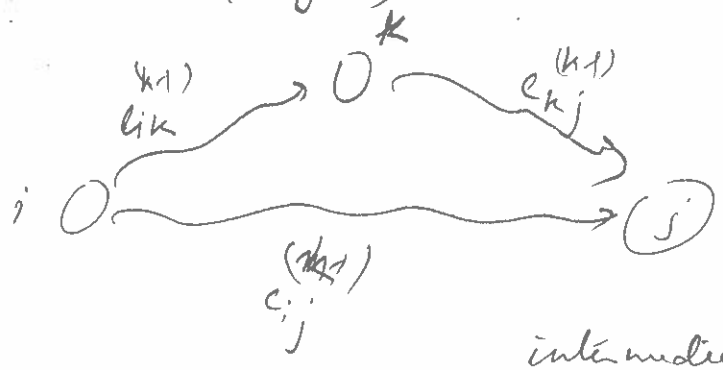
$c_{ij}^{(k)}$ = weight of a shortest path from i to j with intermediate vertices in $\{1, 2, \ldots, k\}$



Then $\delta(i,j) = c_{ij}^{n}$

$c_{ij}^{0} = w_{ij}$

$c_{ij}^{(k)} = \min\left\{ c_{ij}^{(k-1)}, \; c_{ik}^{(k-1)} + c_{kj}^{(k-1)} \right\}$



intermediates on $\{1, 2, \ldots, k-1\}$

pseudocode

```
for k ← 1 to n
    for i ← 1 to n
        for j ← 1 to n
            c_ij^(k) = min( c_ij^(k-1), c_ik^(k-1) + c_kj^(k-1) )
```

$\Theta(n^3)$.

space $O(n^2)$

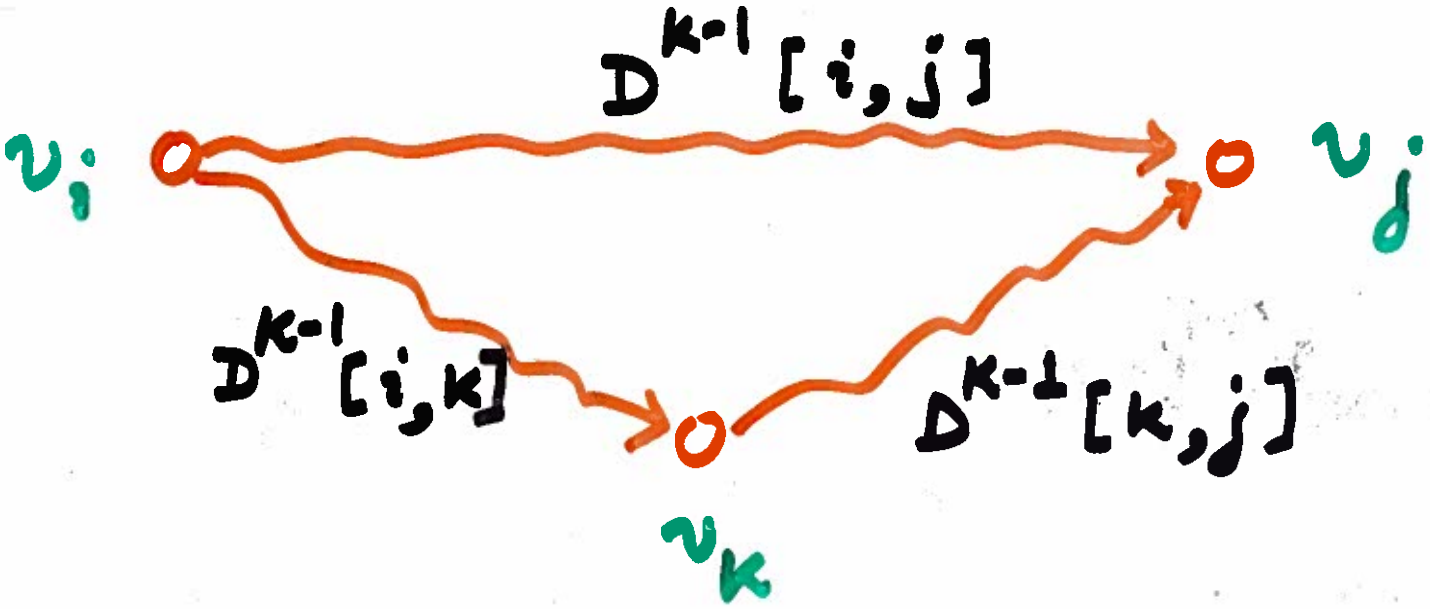
$O(n^2)$

ALL-TO-ALL SHORTEST PATHS

$D^k[i,j]$ = the weight of a shortest path for v_i to v_j using nodes from $\{v_1, v_2, \ldots v_k\}$ as intermediate vertices in the path.

$$D^{(0)} = (w_{ij})_{n \times n} = \begin{cases} 0 & i = j \\ \infty & \{i,j\} \notin E \\ \text{weight of } \{i,j\} \end{cases}$$

$D^{(n)}$ = output

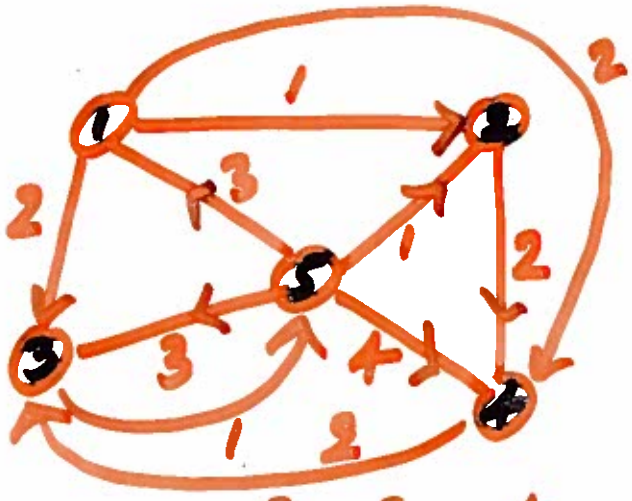


$$D^{(k)}[i,j] =$$

$$\min\left[D^{(k-1)}[i,j],\ D^{(k-1)}[i,k] + D^{(k-1)}[k,j]\right]$$

ALL-TO-ALL

```
D = Weight Matrix
for k ← 1 to n do
    for i ← 1 to n do
    for j ← 1 to n do
        D[i,j] = min [ D[i,j] ,
                       D[i,k] + D[k,j] ]
    endfor ; endfor
endfor.
```



$D^{(0)} =$

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

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

$D^{(1)}$

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

$D^{(2)}$

$D^{(3)}$

0	1	2	2	3
∞	0	∞	2	∞
∞	∞	0	∞	1
∞	∞	2	0	3
3	1	3	3	0

$D^{(4)}$

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

$D^{(5)}$

0	1	2	2	3
8	0	4	2	5
4	2	0	4	1
6	4	2	0	3
3	1	3	3	0

Transitive Closure

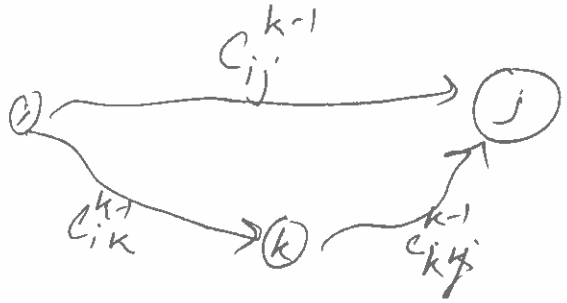
$G^* = (V, E^*)$

$(i,j) \in E^*$ iff $\exists$ path from i to j in G.

Solution

Adj Mat $(0,1)$ matrix

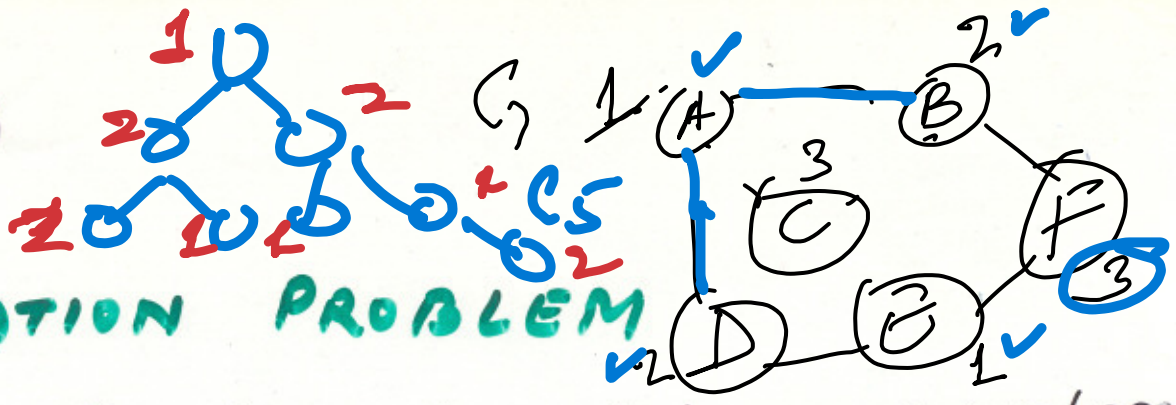
In Floyd Warshal



$$c_{ij}^{k} = c_{ij}^{k-1} \text{ OR } \left(c_{ik}^{k-1} \text{ AND } c_{kj}^{k-1} \right)$$

$O(n^3)$

COLORING



OPTIMIZATION PROBLEM

GIVEN G, find the minimum number of colors to color G. ($\chi(G)$?)

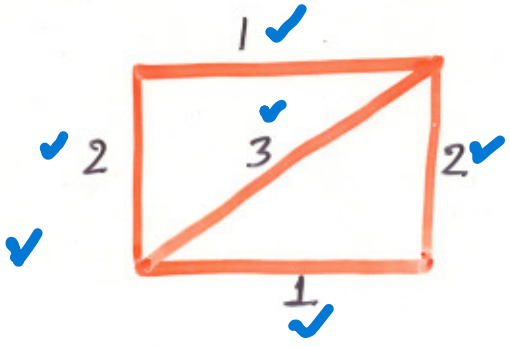
DECISION PROBLEM

GIVEN graph G and positive integer k, is $\chi(G) \leq k$?

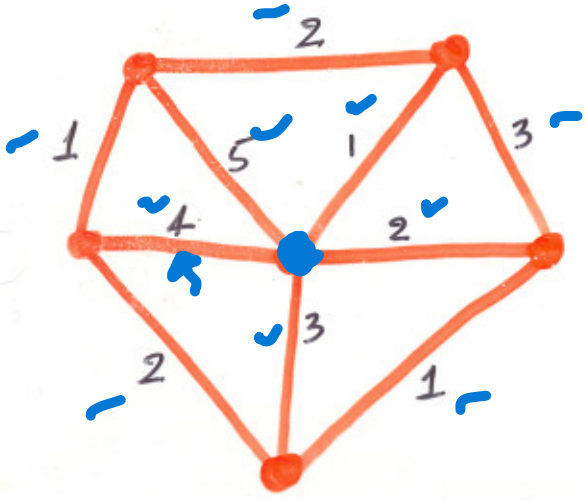
EQUIVALENCE OF OPTMIZATION AND DECISION PROBLEMS

$\chi(G)$ = CHROMATIC NUMBER OF G

EDGE COLORING



CHROMATIC INDEX = 3



Chromatic Index = 5
= highest degree $\Delta(G)$

VIZING'S THEOREM

CHROMATIC INDEX OF A GRAPH G is EITHER $\Delta(G)$ or $\Delta(G)+1$.

CLIQUE

CLIQUE$_{OPTIMIZATION}$ $\propto$ CLIQUE$_D$

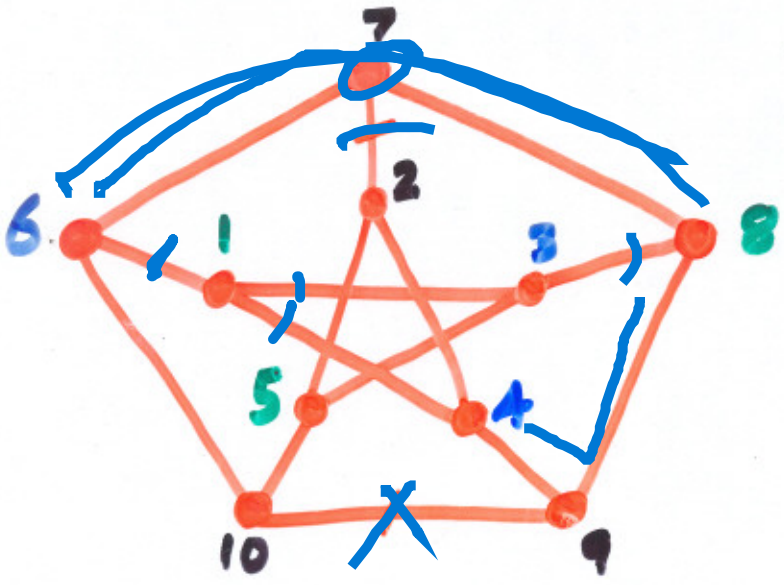
Given a graph G, find the size of the maximum clique in G.

CLIQUE$_{DECISION}$

Given G and integer k, does G have a k-clique?

CLIQUE$_{SEARCH}$ $\propto$ CLIQUE$_{OPT}$

Given G, find a maximum clique of G.



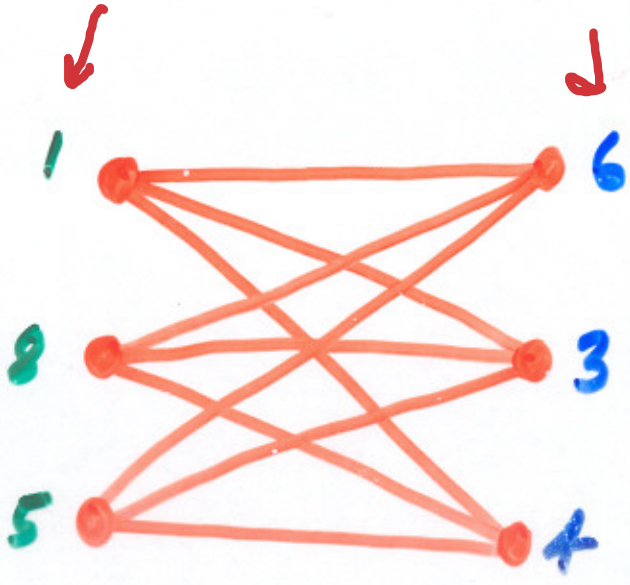
Peterson's Graph

KURATOWSKI'S THEOREM

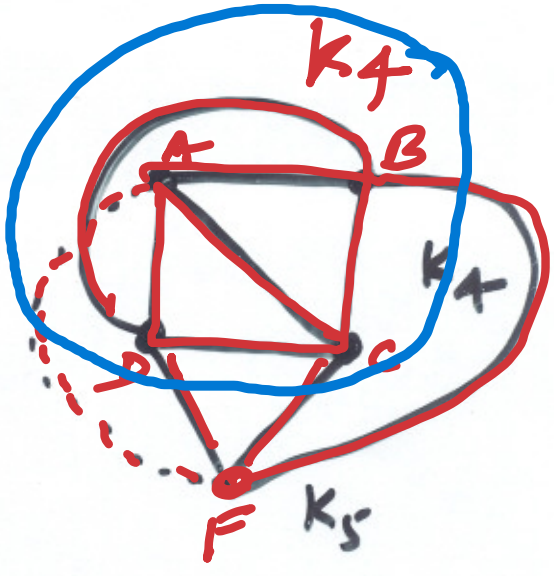
A graph is planar iff it is not homeomorphically reducible to either K_5 or $K_{3\times3}$.

✓ $G \rightarrow G - \{\{u,v\}\}$

✓ 



$K_{3,3}$



planarity testing $O(n)$

SAT (SATISFIABILITY)

Boolean Expressions

$$c_1 = x_1 + x_2 + x_3$$

$$c_2 = \bar{x}_1 + \bar{x}_2$$

$$c_3 = \bar{x}_3 + x_2 + x_4$$

$$c_4 = x_2 + \bar{x}_4$$

Satisfying Truth Assignment

$x_1 = 1$

$x_2 = 0$

$x_3 = 0$

$x_4 = 0$

$$c_1 \wedge c_2 \wedge c_3 \wedge c_4$$

$c_1 = x_1 + x_2$

$c_2 = \bar{x}_1 + \bar{x}_3$

$c_3 = x_2 + x_3$

$c_4 = \bar{x}_1 + \bar{x}_2$

$c_5 = \bar{x}_2 + \bar{x}_3$

$c_6 = x_1 + x_3$

x_1	x_2	x_3	
1	1	0	X
0	1	0	X

no satisfying truth assignment

variables: x_1, x_2, x_3

literals: $x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3$

clauses: $c_1, c_2, \ldots, c_6$

SAT: Given n variables and m clauses over these variables, is there a satisfying truth assignment?

3SAT: All clauses have ~~at most~~ exactly 3 literals.

2SAT can be solved in polynomial time.

P and NP

P: Class of decision problems which have polynomially bounded algorithms

Polynomially Bounded

An algorithm whose worst-case time complexity $W(n) \leq p(n)$

n: input size

p: a polynomial such as $n^2 + 2n + 5$

Non Deterministic Algorithms

Graph Coloring

1. Guess a coloring $O(n)$

$$C : V \to \{1, 2, 3, \ldots, k\}$$

$C(w)$ is the color of node w

2. Verify that the guess is correct

for all $\{u, v\} \in E$ $O(m)$

$$c(u) \neq c(v).$$

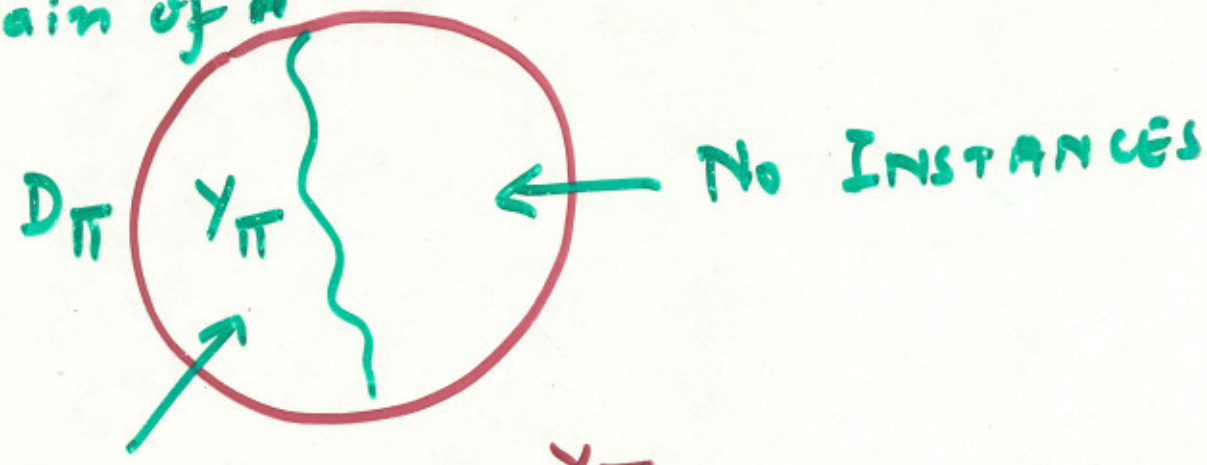
$O(n+m)$

What if one allows ability to guess correctly, if there exists a correct guess?

⇒ Non Deterministic Algo. will be

Definitions

- Problem π : Coloring
- INSTANCE I : A graph G and number of colors k.
- Domain of π



- YES INSTANCES Y_π

  $I \in Y_\pi$: if G is k-colorable
- No INSTANCES $D_\pi - Y_\pi$

  $I \notin Y_\pi$

- Polynomially Bounded Non-Deterministic algorithm is whose $w(n)$ is polynomial for all yes-instances.

NP: A class of decision problems which have polynomially bounded non-deterministic algorithms.

- NP contains those problems whose guesses are polynomial time verifiable.

- Coloring $\in$ NP
- clique $\in$ NP
- SAT $\in$ NP

guess: a truth assignment

$O(n)$ $T : \{x_1, x_2, \ldots, x_n\} \rightarrow \{0, 1\}$

check: verify that each clause is satisfied

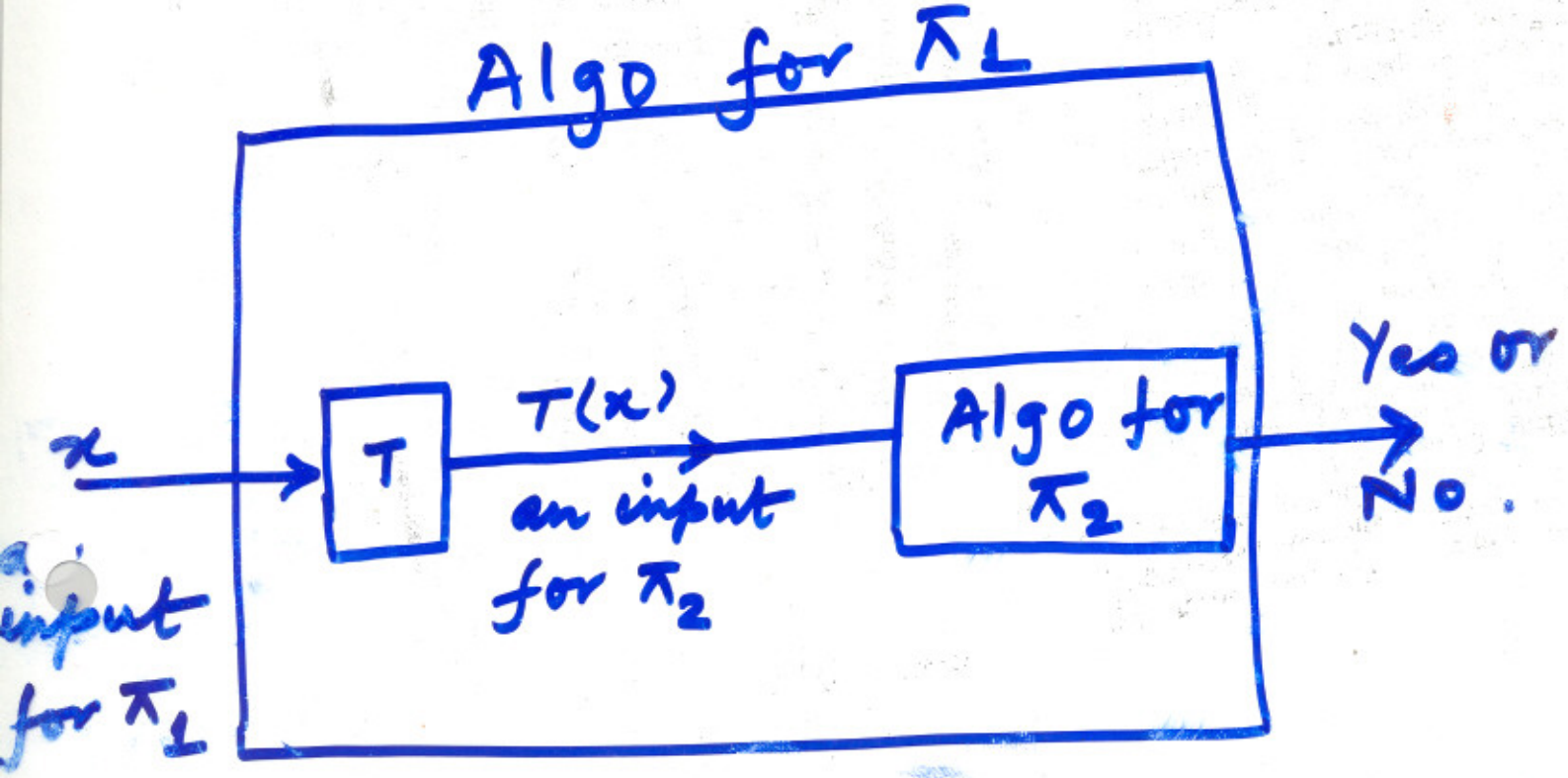
for all $c \in \{c_1, c_2, \ldots, c_m\}$

c has atleast one true literal.

$O(mn)$

$O(n+m)$

Polynomial Reduction



MST
COLOR

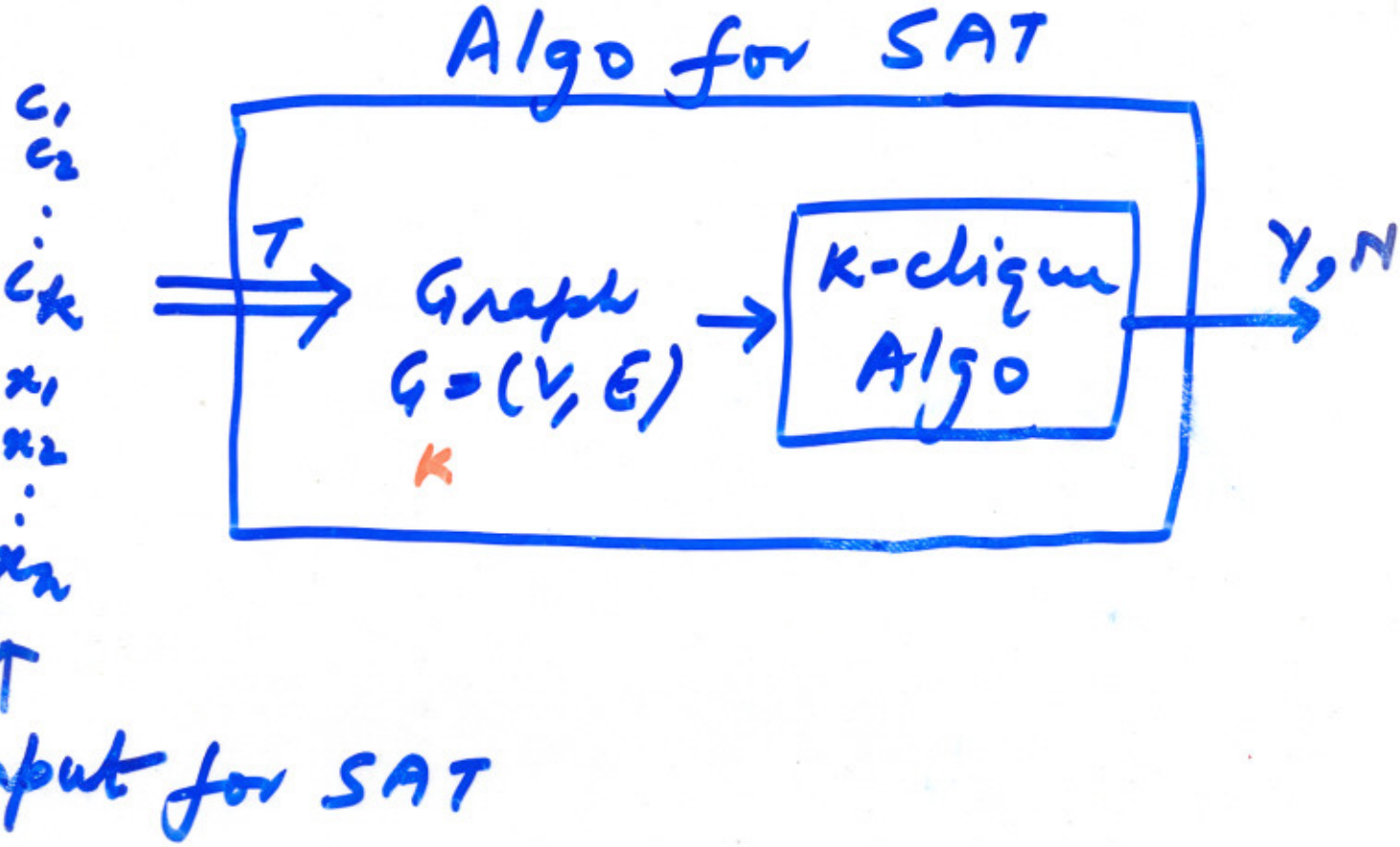
$\pi_1 \propto \pi_2$

sorting
SAT

T :1. polynomial time transformation

2. Answer Preserving

g. SAT ∝ K-Clique



↑
input for SAT

$$T:\begin{cases} V=\{\langle \sigma, i\rangle \mid \sigma \text{ is a literal in } c_i\} \\ E=\{\{\langle \sigma, i\rangle, \langle \delta, j\rangle\} \mid i\neq j \ \&\ \sigma\neq\bar{\delta}\} \end{cases}$$

clique size $K = k$, no. of clauses

T: Polyno in k & n.

$P_1 \Leftrightarrow P_3 \Rightarrow \neg P_1 \Rightarrow \neg P_2$

$C_1 = x_1 + x_2 + x_3$

$C_2 = \bar{x}_1 + \bar{x}_2$

$C_3 = \bar{x}_3 + \bar{x}_2 + x_4$

$C_4 = x_2 + \bar{x}_4$

Satisfying Truth assignment

$x_2 = 1$

$x_1 = 0$

$x_3 = \phi$

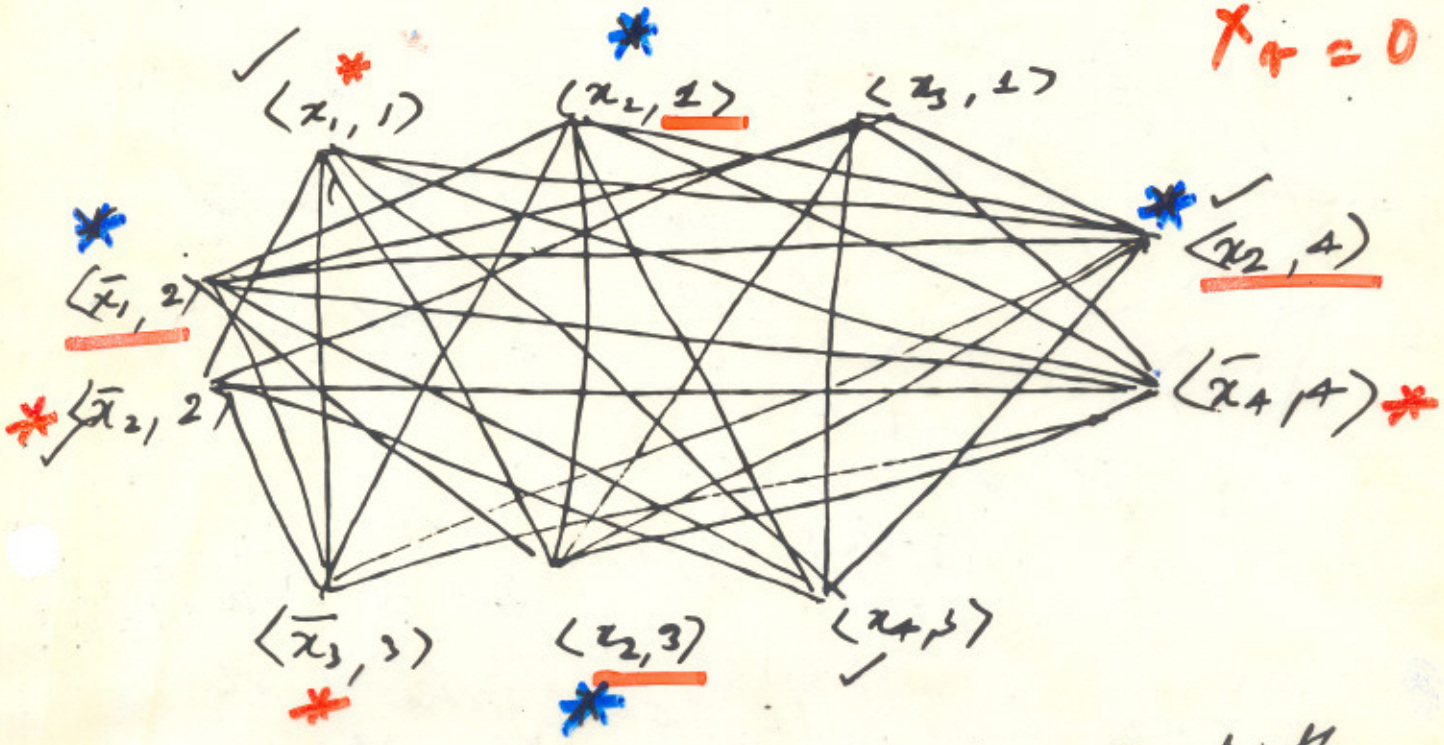
$x_4 = \phi$

$x_1 = 1$

$x_2 = 0$

$x_3 = 0$

$x_4 = 0$



4-clique? iff satisfiable truth assignment.

SAT length $O(kn \log n)$

Time for T: kn nodes

$\frac{k^2n^2}{2}$ edges

$O(k^2n^2 \log n)$

- If $\pi_1 \propto \pi_2$ and $\pi_2 \in P$ then $\pi_1 \in P$

- NP-complete (NPC)

A problem π is NP-complete

if $\pi \in NP$ ——————— (i)

and

for every other problem $\pi' \in NP$

$\pi' \propto \pi$. ——————— (ii)

- If $\pi \in NPC$, and $\pi \in P$ then $P = NP$.

Cook's theorem

$$SAT \in P \iff P = NP$$

or

If $SAT \in P \Rightarrow P = NP$

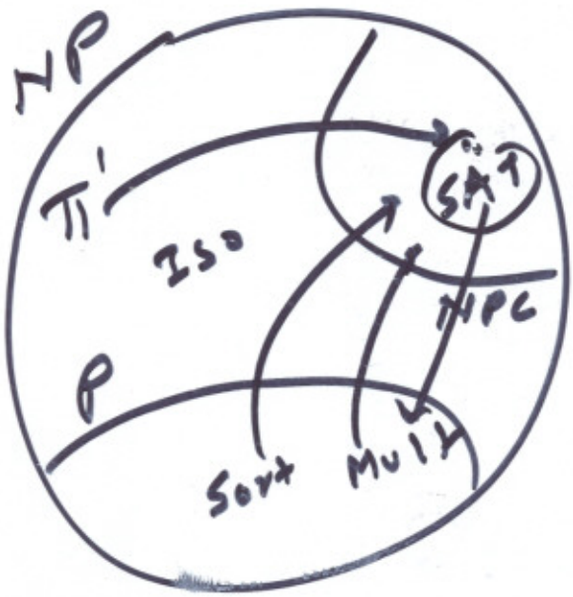
or for every problem $\pi \in NP$,
$\pi \propto SAT$.

or

$$SAT \in NPC$$

Cook's Theorem 1972

SAT ∈ NPC



$\pi' \propto$ SAT

Known Facts

1. $\pi' \in NP$.
2. π' has a poly time ND algo.
3. $W(n) \leq p(n)$

If SAT ∈ P, then P = NP

Karp
- Color — $\pi' \propto$ color; SAT $\propto$ color; color ∈ NP
- Clique
- Vertex Cover — Edge Cover
- Independent Set
- Subset Sum — $O(nC)$
- Dominating Set

Facts & Condition

To show: $\Pi' \propto SAT$

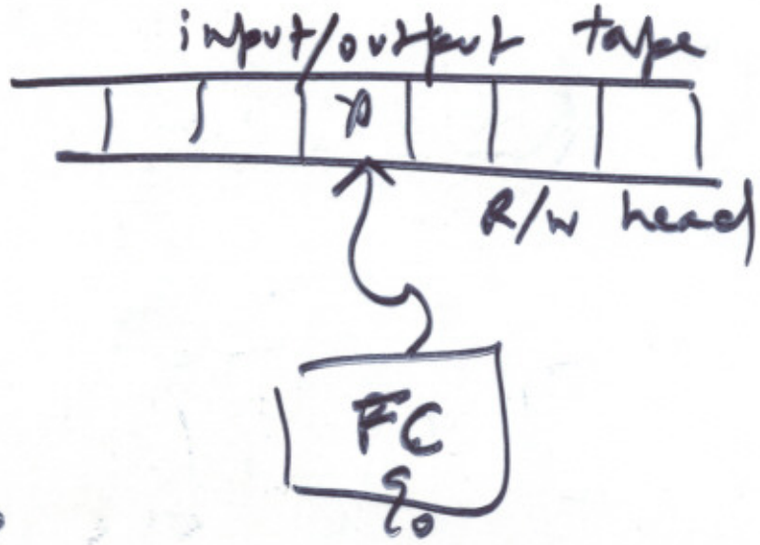
$I_{\Pi'} \xrightarrow{T} I_{SAT}$

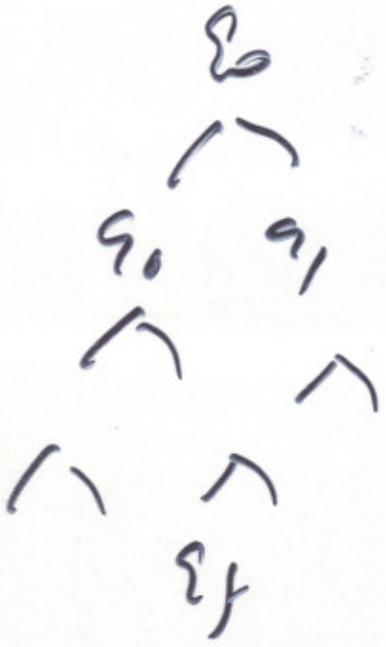
cli~~ck~~ $\Pi' \propto SAT$

1. $\Pi' \in NP$

$\Rightarrow \Pi'$ has NDTM algorithm whose time complexity $W(n) \leq p(n)$.

(5)





1. Read input char
2. Replace with another char
3. Go to another state
4. Move left/right

6. $q_0 \rightsquigarrow q_f$

Accepting Computation $\Longleftrightarrow$ Satisfying Truth Assignment

NP

SAT → COLOR

It can be shown that

(i) graph coloring $\in$ NP

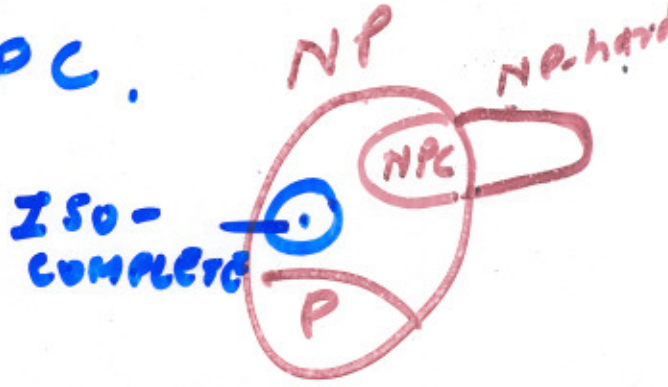
(ii) SAT $\propto$ coloring

$\Rightarrow$ graph coloring $\in$ NPC

Similarly

k-Clique, HC, TSP etc.

are also NPC.



NP-hard

π is NP-hard if for all problem $\pi' \in$ NP, $\pi' \propto \pi$.

Restriction of NPC problems

g. coloring.

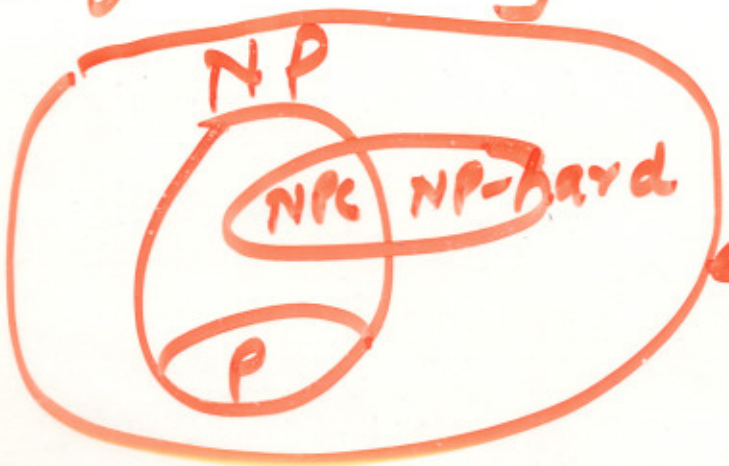
2-coloring $\in P$

3-coloring $\in$ NPC

Coloring of planar graph using 4 colors takes linear time.

3-coloring of planar graph is NPC.

Edge Coloring Δ ? $\Delta+1$



← decidable

APPROXIMATION ALGORITHMS

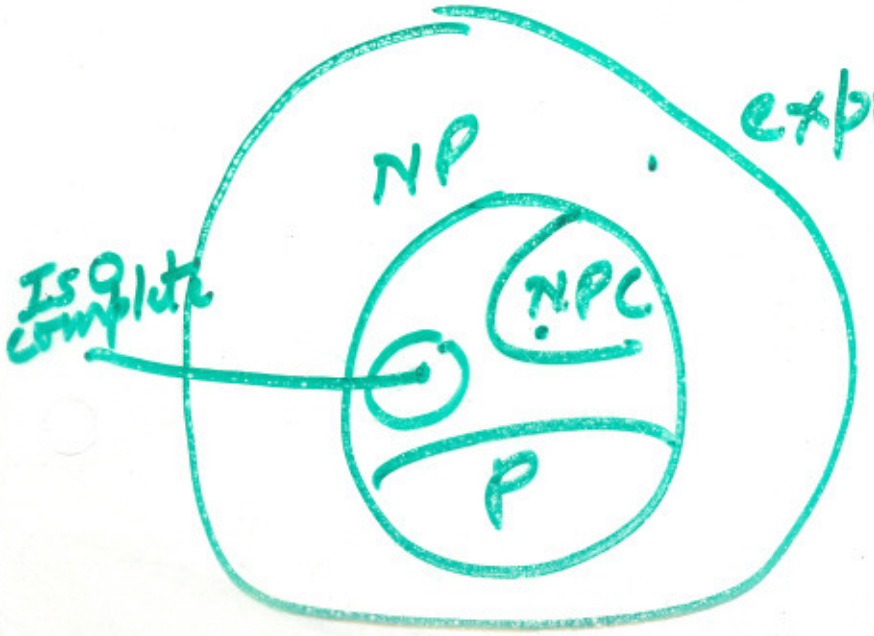
Polynomial-time algorithm for an NP-Complete (or NP-hard) problem which do not guarantee the optimal solution but would generally give one that is close to optimal.

BIN PACKING PROBLEM

Given: n objects to be placed in bins of capacity L each.

Object i requires l_i units of bin capacity.

Objective: determine the minimum number of bins needed to accomodate all n objects.



exponential

$P \stackrel{?}{=} NP$

eg. Let $L = 10$, $l_1 = 5$, $l_2 = 6$, $l_3 = 3$, $l_4 = 7$, $l_5 = 5$, $l_6 = 4$

3 bins needed.

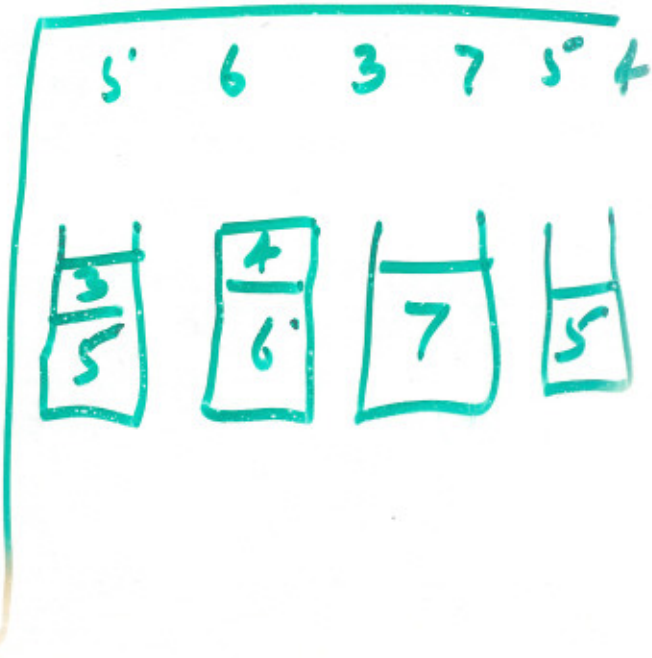
Th Bin packing problem is NP complete when formulated as a decision problem.

As an optimization problem, bin packing is NP-hard.

Approximation Algorithms for Bin Packing

1. FIRST FIT (FF)

- Label bins as 1, 2, 3, ...
- Objects are considered for packing in the order 1, 2, 3, ...
- Pack object i in bin j where j is the least index such that bin j can contain object i.



Complexity $O(n^2)$.

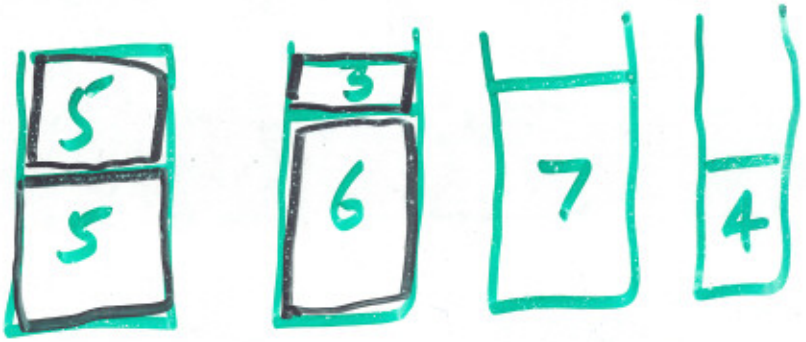
II BEST FIT (BF)

Same as FF, except that when object i is to be packed, find out that bin which after accomodating object i will have the least amount of space left.

5 6 3 7 5 4

$O(n^2)$

eg.



III First Fit Decreasing (FFD)

reorder objects so that

$l_i \geq l_{i+1}$, $1 \leq i \leq n$.

then use FF. $O(n^2)$

IV Best Fit Decreasing (BFD)

Reorder objects as above & then use BF. $O(n^2)$

Th. Packing generated by either FF or BF uses no more than $\frac{17}{10}$ OPT + 2 bins.

70% of OPT

That by either FFD or BFD uses no more than

$\frac{11}{9}$ OPT + 4 bins.

22% of OPT

$\frac{11}{9} = \frac{9}{7} + \frac{2}{7}$

1.7 OPT

Johnson 1.07 OPT

FSP $\propto$ BP

References

NP-Complete Theory, applications, examples, etc.

COMPUTER and INTRACTABILITY: A Guide to the Theory of NP-completeness

by Michael R GAREY & David S JOHNSON

Publisher: W. H. Freeman 1979

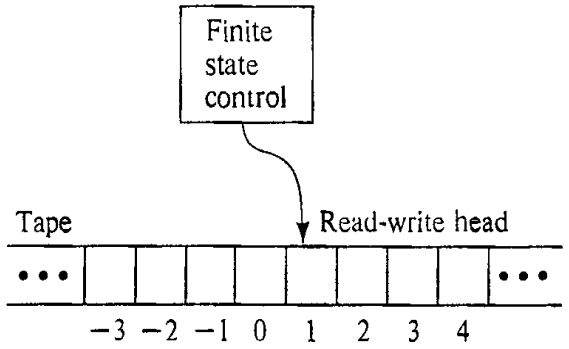


Figure 2.1 Schematic representation of a deterministic one-tape Turing machine (DTM).

A *program* for a DTM specifies the following information:

(1) A finite set Γ of tape *symbols*, including a subset $\Sigma \subset \Gamma$ of *input symbols* and a distinguished *blank symbol* $b \in \Gamma - \Sigma$;

(2) a finite set Q of *states*, including a distinguished *start-state* q_0 and two distinguished *halt-states* q_Y and q_N;

(3) a *transition function* $\delta: (Q - \{q_Y, q_N\}) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$.

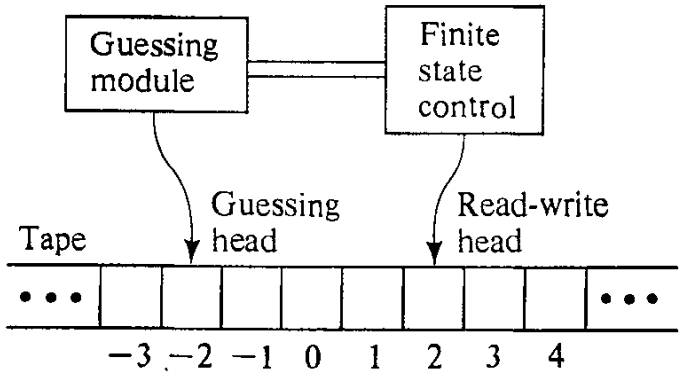


Figure 2.4 Schematic representation of a nondeterministic one-tape Turing machine (NDTM).

*Michael R. Garey and David S. Johnson. 1972/1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA.

Proof of Cook's Theorem (Garey & Johnson, 1972*)

Variable	Range	Intended meaning
$Q[i,k]$	$0 \leqslant i \leqslant p(n)$ $0 \leqslant k \leqslant r$	At time i, M is in state q_k.
$H[i,j]$	$0 \leqslant i \leqslant p(n)$ $-p(n) \leqslant j \leqslant p(n)+1$	At time i, the read-write head is scanning tape square j.
$S[i,j,k]$	$0 \leqslant i \leqslant p(n)$ $-p(n) \leqslant j \leqslant p(n)+1$ $0 \leqslant k \leqslant v$	At time i, the contents of tape square j is symbol s_k.

Figure 2.7 Variables in $f_L(x)$ and their intended meanings.

*Michael R. Garey and David S. Johnson. 1972/1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA.

Proof of Cook's Theorem (Garey & Johnson, 1972)

$x \in L$ $\Leftrightarrow$ there is an accepting computation of M on x

$\Leftrightarrow$ there is an accepting computation of M on x with $p(n)$ or fewer steps in its checking stage and with a guessed string w of length exactly $p(n)$

$\Leftrightarrow$ there is a satisfying truth assignment for the collection of clauses in $f_L(x)$.

*Michael R. Garey and David S. Johnson. 1972/1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA.

Proof of Cook's Theorem

Clause group	Restriction imposed
G_1	At each time i, M is in exactly one state.
G_2	At each time i, the read-write head is scanning exactly one tape square.
G_3	At each time i, each tape square contains exactly one symbol from Γ.
G_4	At time 0, the computation is in the initial configuration of its checking stage for input x.
G_5	By time $p(n)$, M has entered state q_Y and hence has accepted x.
G_6	For each time i, $0 \leqslant i < p(n)$, the configuration of M at time $i+1$ follows by a single application of the transition function δ from the configuration at time i.

Figure 2.8 Clause groups in $f_L(x)$ and the restrictions they impose on satisf truth assignments.

Clause group	Clauses in group
G_1	$\{Q[i,0], Q[i,1], \ldots, Q[i,r]\},\ 0 \leqslant i \leqslant p(n)$ $\{\overline{Q[i,j]}, \overline{Q[i,j']}\},\ 0 \leqslant i \leqslant p(n), 0 \leqslant j < j' \leqslant r$
G_2	$\{H[i,-p(n)], H[i,-p(n)+1], \ldots, H[i,p(n)+1]\},\ 0 \leqslant i \leqslant p(n)$ $\{\overline{H[i,j]}, \overline{H[i,j']}\},\ 0 \leqslant i \leqslant p(n), -p(n) \leqslant j < j' \leqslant p(n)+1$
G_3	$\{S[i,j,0], S[i,j,1], \ldots, S[i,j,v]\},\ 0 \leqslant i \leqslant p(n), -p(n) \leqslant j \leqslant p(n)+1$ $\{\overline{S[i,j,k]}, \overline{S[i,j,k']}\}, 0 \leqslant i \leqslant p(n), -p(n) \leqslant j \leqslant p(n)+1, 0 \leqslant k < k' \leqslant v$
G_4	$\{Q[0,0]\}, \{H[0,1]\}, \{S[0,0,0]\},$ $\{S[0,1,k_1]\}, \{S[0,2,k_2]\}, \cdots, \{S[0,n,k_n]\},$ $\{S[0,n+1,0]\}, \{S[0,n+2,0]\}, \ldots, \{S[0,p(n)+1,0]\},$ where $x = s_{k_1} s_{k_2} \cdots s_{k_n}$
G_5	$\{Q[p(n),1]\}$

Figure 2.9 The first five clause groups in $f_L(x)$.

*Michael R. Garey and David S. Johnson. 1972/1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA.

Proof of Cook's Theorem

The remaining subgroup of G_6 guarantees that the *changes* from one configuration to the next are in accord with the transition function δ for M. For each quadruple (i,j,k,l), $0 \leqslant i < p(n)$, $-p(n) \leqslant j \leqslant p(n)+1$, $0 \leqslant k \leqslant r$, and $0 \leqslant l \leqslant v$, this subgroup contains the following three clauses:

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, H[i+1,j+\Delta]\}$$
$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, Q[i+1,k']\}$$
$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, S[i+1,j,l']\}$$

where if $q_k \in Q-\{q_Y,q_N\}$, then the values of Δ, k', and l' are such that $\delta(q_k,s_l) = (q_{k'},s_{l'},\Delta)$, and if $q_k \in \{q_Y,q_N\}$, then $\Delta = 0$, $k' = k$, and $l' = l$.

Length $[f_L(x)] = |U| \cdot |C| = O(p(n)^4)$, and is bounded by a polynomial function of n as desired.

*Michael R. Garey and David S. Johnson. 1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA.