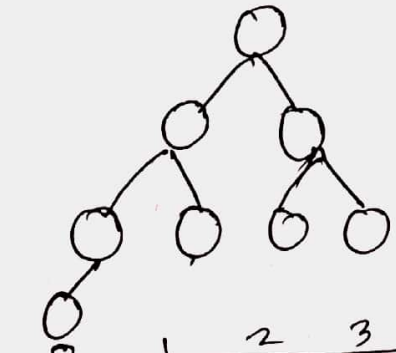
CSE-215 20.08.2023

DSA-II

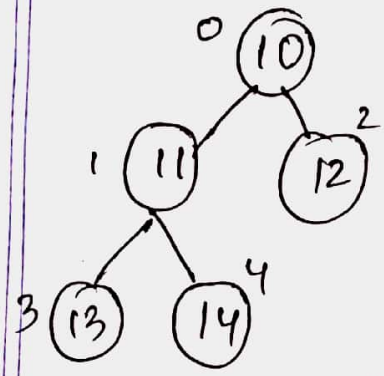
## 04d3Khk

Heap → dynamic memory allocation



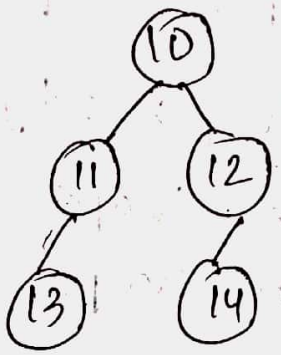
→ complete binary tree.

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 10 | 11 | 12 | 13 | 14 |

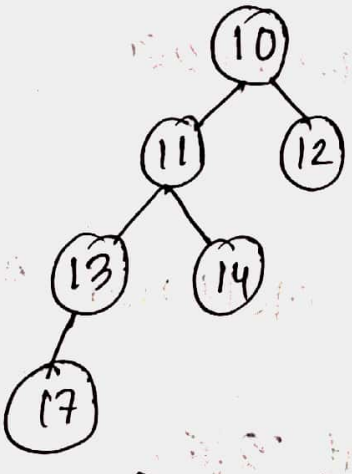



→ complete

height = 2

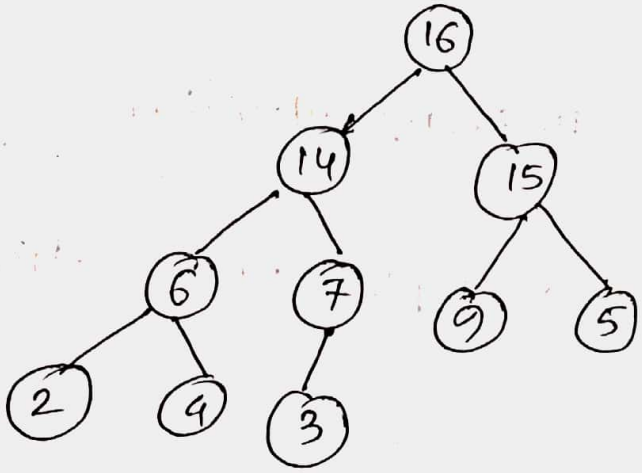



একটা child থাকা তাই এটা complete না।




h(13) = 1.

নিচ থেকে count করতে হবে। h(n) → এরকম আকলে।




Level order Traversal.

| 16 | 14 | 15 | 6 | 7 | 9 | 5 | 2 | 4 | 3 |
|---|---|---|---|---|---|---|---|---|---|

Binary heaps 2 types:

(1) max - heaps.

(2) min - heaps

## max-heap:

(1) root has to be largest element

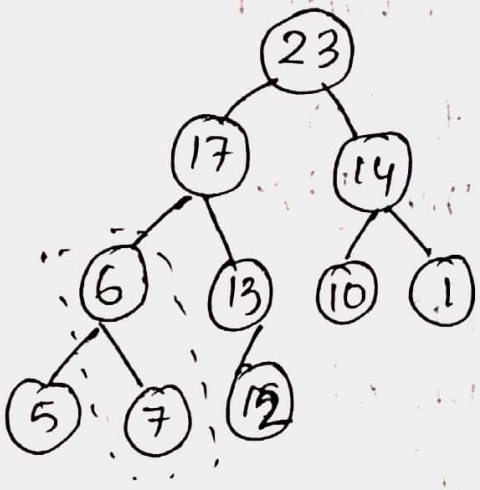
(2) Parents node, always child node এর থেকে বড় হবে,

## min heap:

(1) Parents node alway ছোট হবে child node এর থেকে।

(2) root node smallest element হবে।

| 23 | 17 | 14 | 6 | 13 | 10 | 1 | 5 | 7 | 12 |
|---|---|---|---|---|---|---|---|---|---|




→ complete binary tree.

// level বজায় থাকে partial order maintain করে।

Parent node < child node

$O(\ )$ → upper bound

$\Omega(\ )$ → lower "

$\Theta$ (Theta) → Tight bound (exact value পাওয়া)

upper bound = lower bound হলে tight bound

height → $\Theta(\log n)$

time complexity → $O(\log n)$

# MAX-HEAPIFY → maintaining max-heap property.

MIN-HEAPIFY → " min- " "

23.08.23

→ 0 based

for leave node: $\left[\frac{n-1}{2}+1 \ldots n\right]$

1 based index: $\left[\frac{n}{2}\right]+1, \left[n/2\right]+2, \ldots n$

Time complexity = $O(\log n)$

⇒ height এর উপর depend করে তাই $\log n$

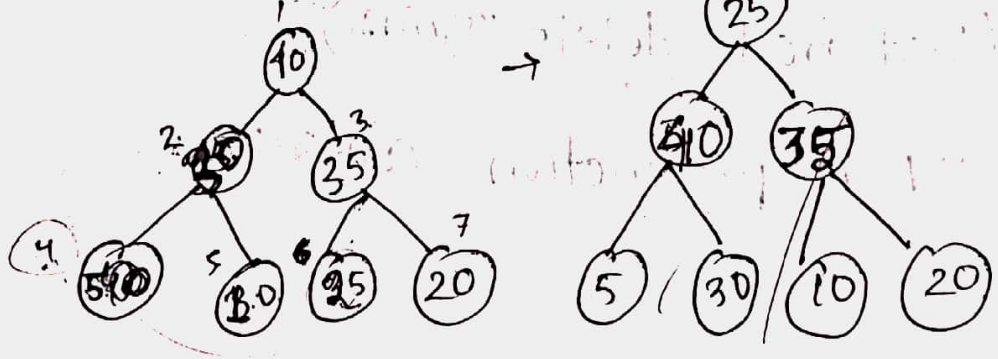
Heapify → শুধু child এর মধ্যে Property maintain করে,
→ subtree " " " " করে না।

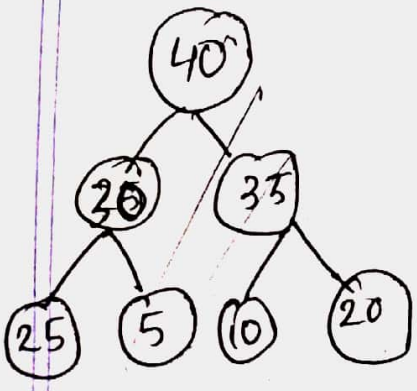
* why not top-down approach?

1 based → non-l
1 — $n/2$
1 → 3
bottom up
L → 3 — 1
3 →
2 →

max value এটাকে shift করে

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 10 | 5 | 25 | 40 | 30 | 35 | 20 |



Time complexity of build - max-heap?

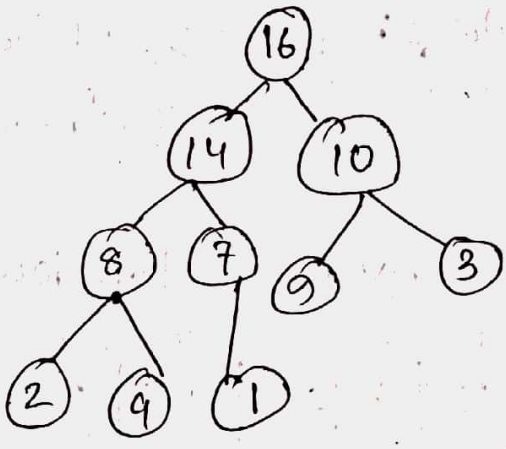
1 → O(1)

2 → $\frac{n}{2}$ → $O\left(\frac{n}{2} \log n\right) \rightarrow O(n \log n)$

3 → log n

Tighter upper bond → O(n)

## Deletion:




step → (1) 16 এর সাথে exchange (root node) এর last leaf node.

(2) h.size--;
(last node delete করলাম)

(3) Build heap function call করব।

Q:- A = 4
n = 3

sorting ची array साठी heap size diff घ्या

## Heap Sort :

① Build - max-Heap (Array ला Heap ला convert)

Time complexity: $O(n \log n - \log n)$

$= O(n \log n)$

(2) for (i = A length down to 2)

(3) exchange A[1] with A[i];

(4) A.heapsize = A.heapsize - 1;

(5) Max - heapify (A, 1)

Time comp. $= O(n-1)(\log n)$

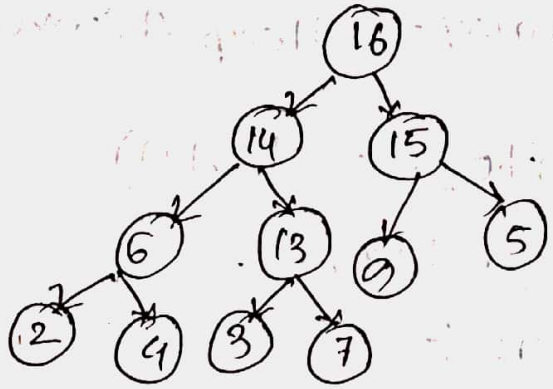
$= O(n \log n - \log n)$

$= \boxed{O(n \log n)}$

* Heap चे basic operation -

27.08.23

Parent node এর আগে exchange করলে → O(log n)

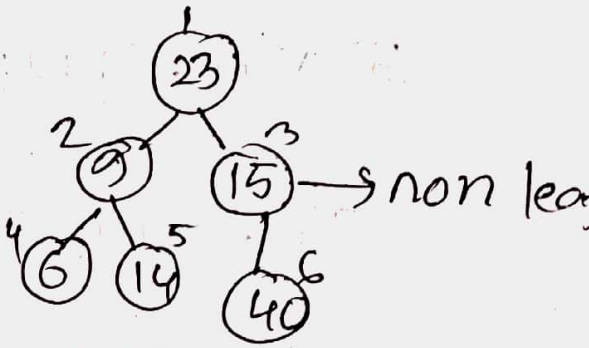



*max heap, min heap
Algorithm
*basic op of Heap - O(log n)

Build Max-Heap:

1. A.heapsize = A.length
2. for i = [A.length/2] down to 1
3. max heapify (A, i)

Heapsort → Heapify method
→ 4 → non leaf থেকে
→ $\frac{n}{2}+1$ to n




এখানে 3 থেকে শুরু করতে হবে। কারণ $\frac{6}{2}+1=4$
bt 4 হচ্ছে leaf node.

root delet হবে → শেষের element root হবে

→ time complexity O(n). Heapify method
→ faster

# Priority Queue

element [Priority এর উপর depend]

value — key (Priority key)

2type → max - Priority Queue
→ min → " "

⇒ যার Priority value বেশি – max
" " " কম → min

// Heap max – O(1). [root return]
" Extract-max → O(log n) [root delete]

*max heapify করলে log n.

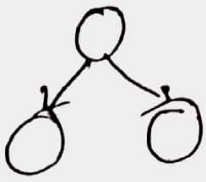
*Heap Increase-key → O(log n) [পুরো height পর্যন্ত Compare করে]
↳ key এর উপর base করে

* nodes of height = $\left\lceil \frac{n}{2^{h+1}} \right\rceil$

Per node এ → O(h)

height $h = \left\lceil \frac{n}{2^h+1} \right\rceil$

$$\sum_{h=0}^{\lceil \log n \rceil} \left\lceil \frac{n}{2^h+1} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lceil \log n \rceil} \frac{h}{2^h}\right)$$



→ n height time complexity $O(h)$

delete - log n
insert → log n
Extract max → root থেকে বড় value delete হতে থাকবে
→ একদম শেষ Node এর value টা root হবে।
→ max-heapify function call হবে।

```
int extractmax (int heap[], int heap_size)
{ int max = heap[1];
  heap[1] = heap[heap_size] → শেষ element root এ আসবে
  heap_size -= 1; [size 1 কমবে]
  max_heapify (heap, heap_size, 1);
  return max;
}
```

insert node

```
int insert_node (int heap[], int heap_size, int node)
{
 int i, p, temp;
 heap_size += 1;
 heap[heap_size] = node;
 i = heap_size;
 while (i > 1 && heap[i] > heap[Parent(i)])
 {
   p = Parent(i);
   temp = heap[p];
   heap[p] = heap[i];
   heap[i] = temp;
   i = p;
 }
 return heap_size;
}
```

03/09/2023

# TRIE

→ Tree base data structure

K=26 → TRIE

k-ary tree → একটা Node এর k টা children থাকবে।

binary এর ক্ষেত্রে k=2

→ string based

→ প্রত্যেক Node এর 26টা children থাকতে পারে।

→ TRIE comes from → Retrieval থেকে।

→ huffman

m → no of string

m → length of string

→ abc
  abd
  acd

m
m
m

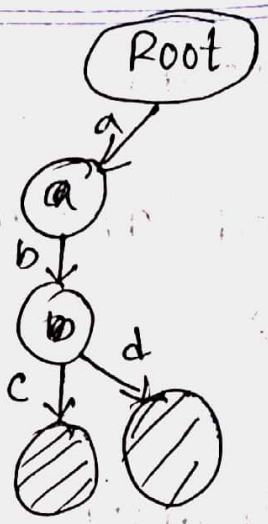
Time complexity = nxm

Trie use করলে Time complexity = O(len(acd))

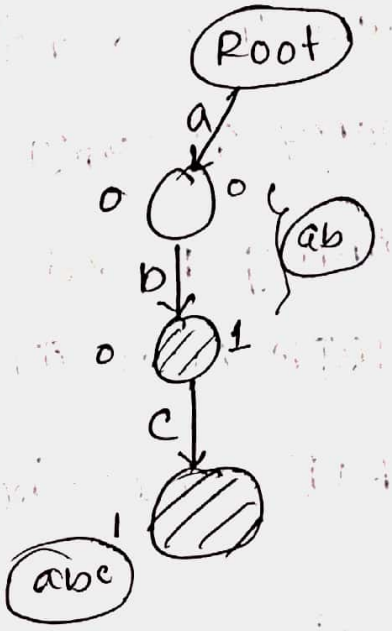
## insert (abc)

1. Root নিতে হবে।
2. a এর জন্য দেখতে হবে কোন child আছে কি না, root থেকে child create
3. b এর " a থেকে child create
4. c " " b " " "
5. c থেকে একটা ending node create.

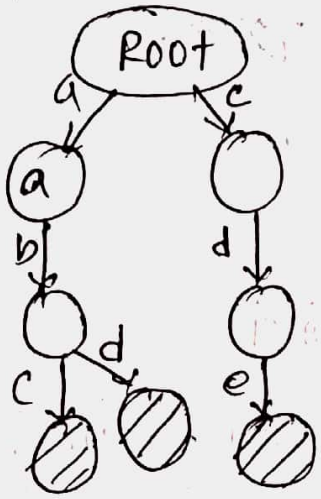
abc
abd



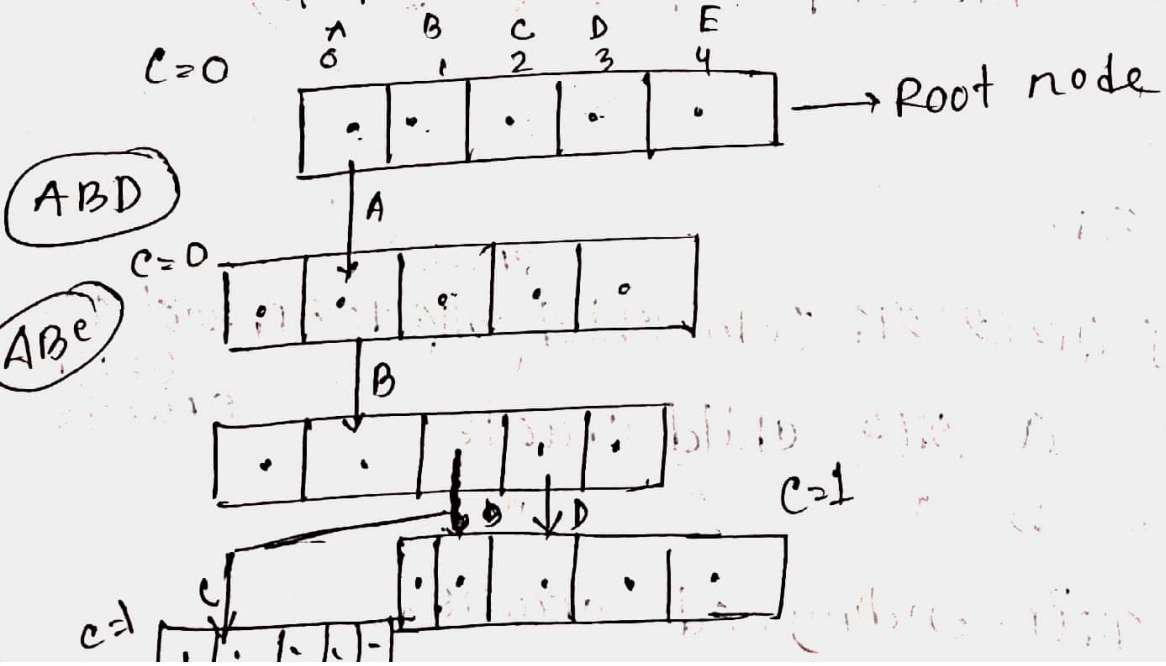

Counter
abc
ab




cde




*প্রত্যেকটা Pointer initially null থাকে।
আর Pointer এর data type Node



04.09.23

```
struct Node{
int EoW; ——→ counter
Node *children [26]
```

datatype

(CT) TRIE এর insertion নিয়ে

→ Pseudo code - step, Algo
→ simulation
→ Explain

insert (x)

```
Node pointer u←root
for (k←0 to size(x) -1
   r←x(k) - 65
  if  u→ children [r] is null;
      u→ children (r)←new Node( )
      u← u→ children (r)
      u→ EoW ← u→ EoW +1;
```

Search:

find (x, Node pointer cur←root, k←0)

05/09/2023

## lexicographical Order:

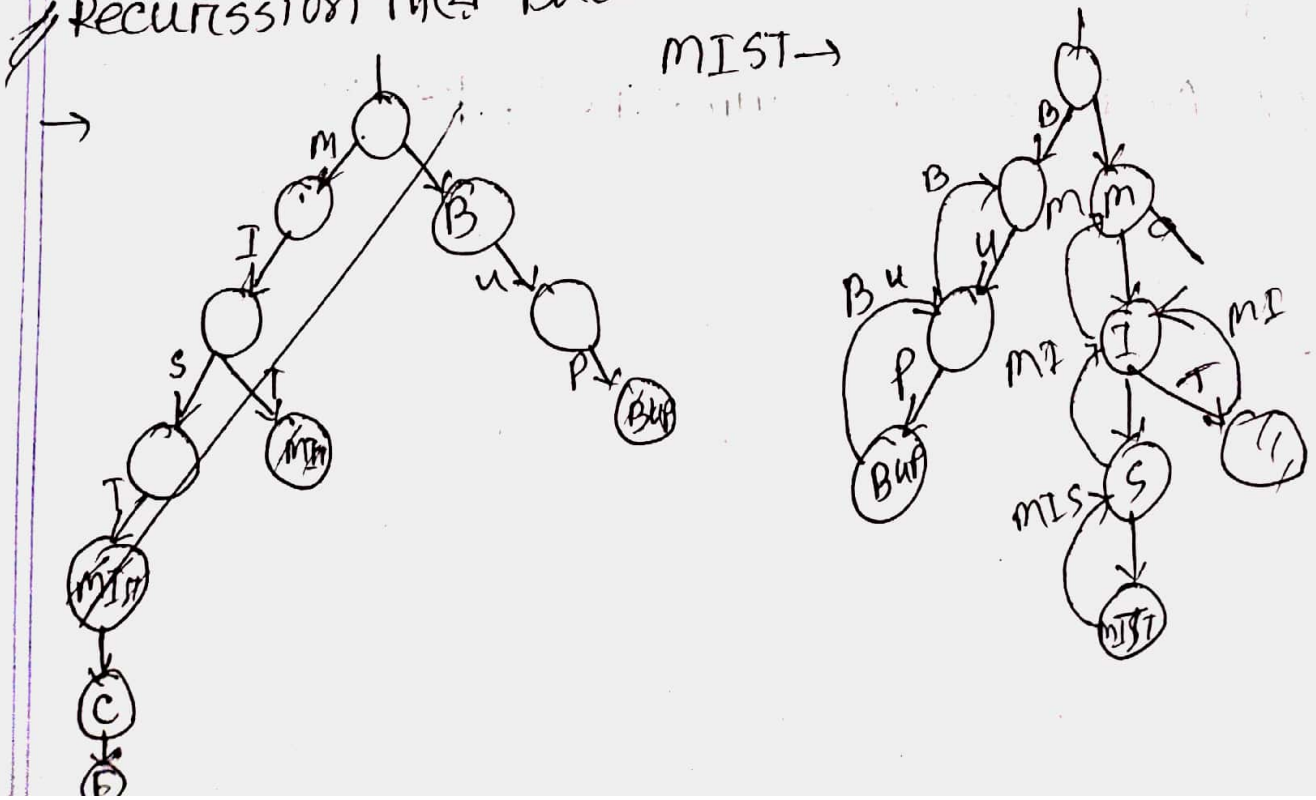
```
void printTRIE (Node*cur = root, string s = "")
{ if (cur→EoW >=1)
  { cout << s << endl;
  }
 for (int i=0; i<26; i++)
 { if (cur→children [i] != NULL)
    { char c = char (i+65);
      PrintTRIE (cur→children, s+c);
    }
  }
}
```

count 1 হলে Print

// Recurssion first BackTrack

→

MIST→



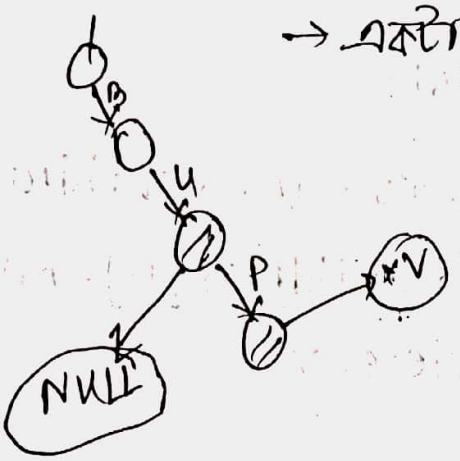
প্রত্যেকটা node এ Pointer দিয়ে point করা.

(1) MIST, MIT, MI

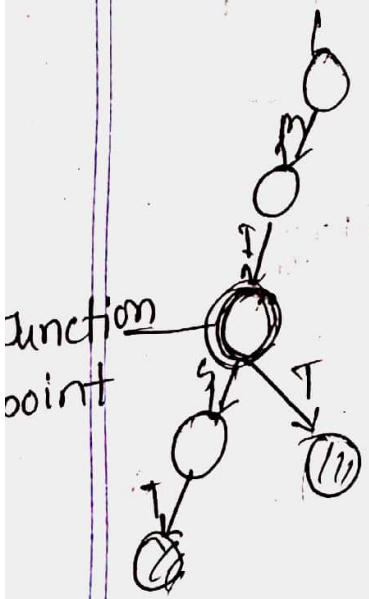
যদি MI dulete করতে চাই, তবে MI যদি Prefix হয় তাই MI এ word count = 0 করতে হবে।

(2) BU, BUP যদি BUP delete করতে চাই, যাকে delete করতে চাই তার নামের আগের string আছে

→ একটা node pointer নিব




সেখানে কর delete করে দিব।




MIT delete করতে হবে,

→ first এ t delete

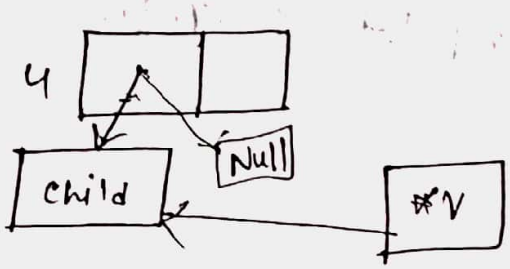
→

07.09.23

```
bool isLeaf(Node *u)
{
 for(int i=0; i<26; i++)
```

Junction point এর উপরে return 0 হবে
" নিচে " 1 হবে, হলে

Node গুলো delete হয়ে যাবে।




```
Node *v = u -> children
u -> children[n] = NULL
delete v
```

↓

3 step এ হলো? 1step এ কেন delete এর কাজ টা হলো না?

```
delete edge(Node *u, char c, int d)  ~> flag
if dis 0
 return without doing anything
 n = c - 65
 Node *v = u -> children[n]
 u -> children[n] = NULL
 delete v
```

```
delele (string x, Node *u = root, k=0)
    if u is NULL
        return 0
    if k is equals size(x)
        if u→EoW is 0          } No word count = 0
            return 0
        if isleaf(u) is false
            u→EoW = 0          } non-leaf node
            return 0
    n = x[k] - 65
    d = delete(x, u→children[n], k+1)
    j = isJunction(u)   [leaf node ও parent junction
                         বিনা check করা]
    removeEdge(u, x[k], d)
    if j is 1
        d = 0
    return d
```

junction node remove function এর যে use করা হল কী হবে?

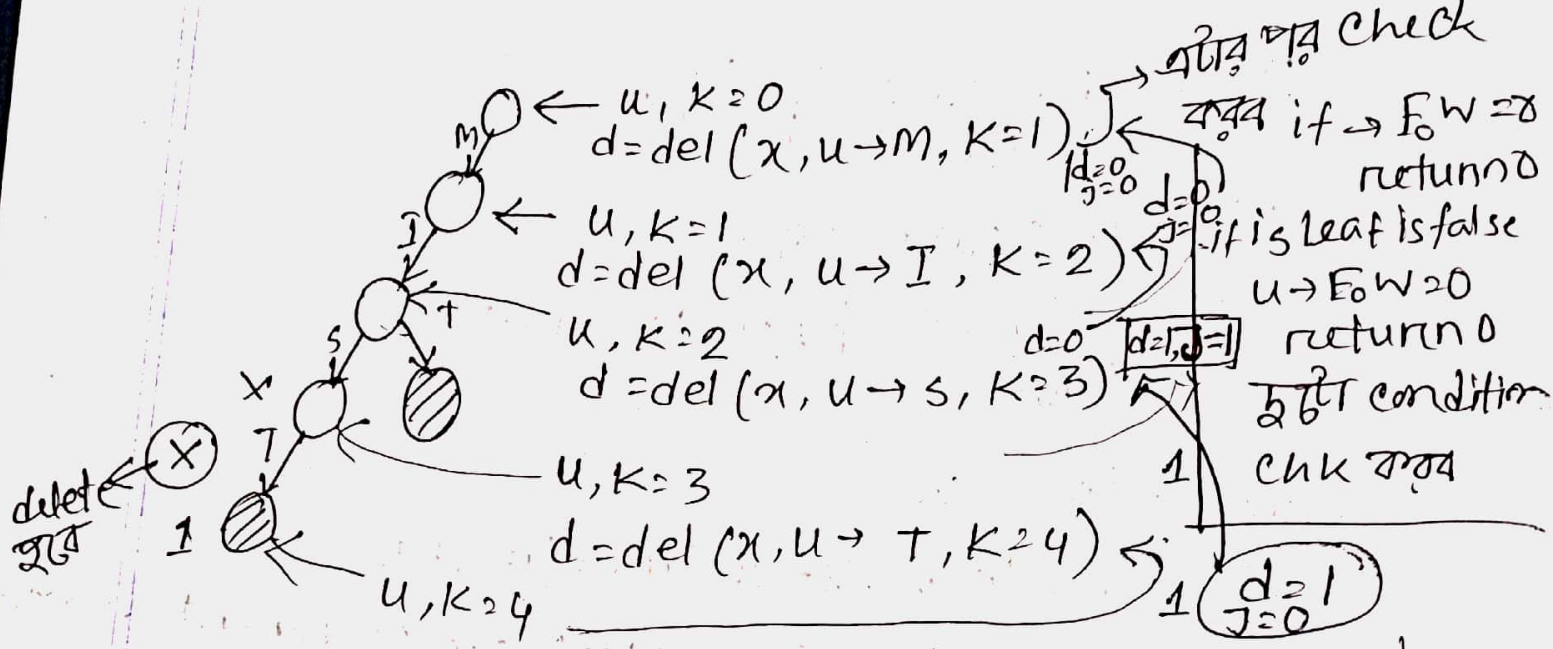
10.09.2023

*** deletion

(Xm) → recursive call কিভাবে হয় simulation সহ দেখাও

x = "MIST"
k=0

* recursion প্রথম call → root

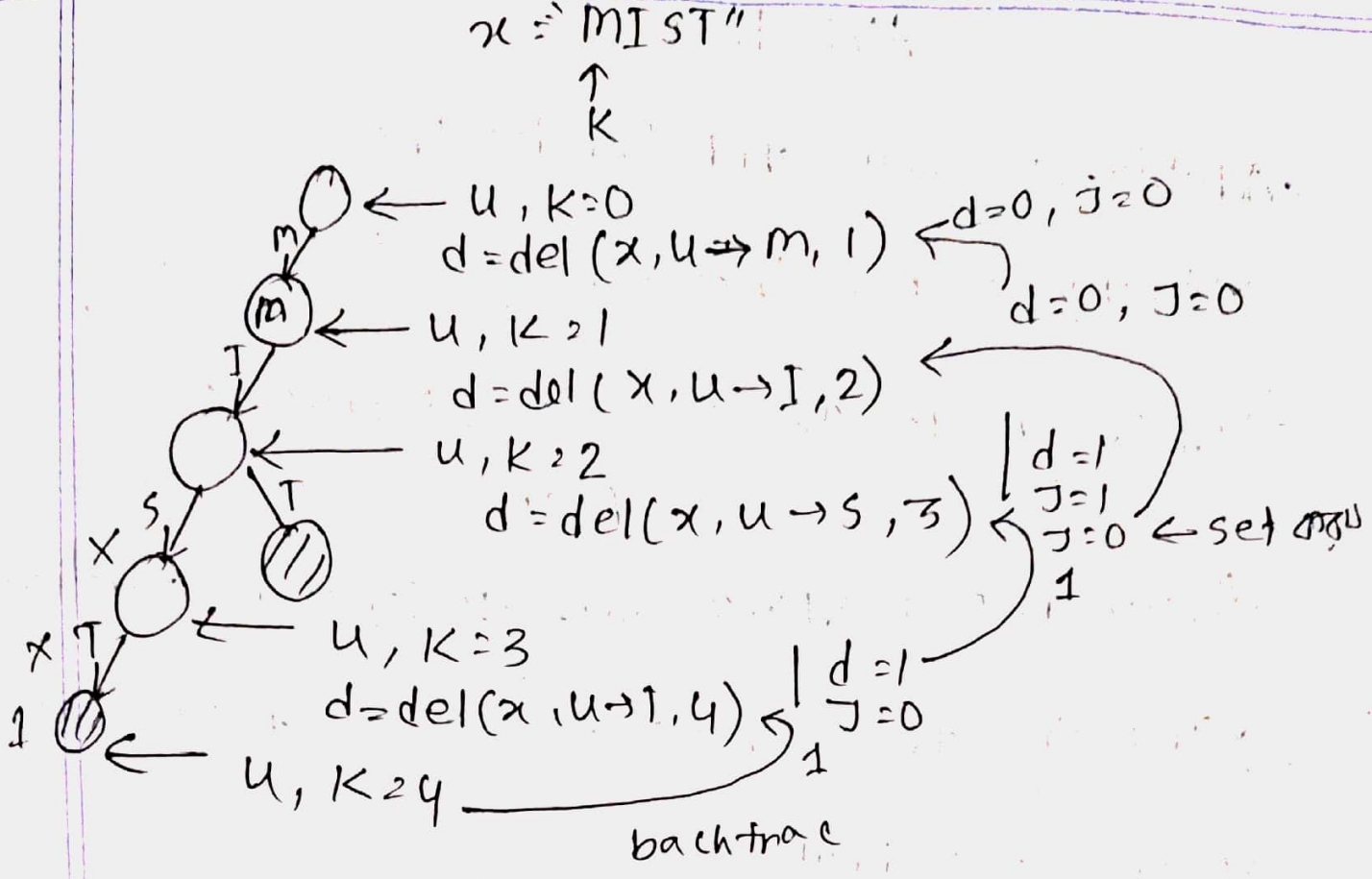



back Track → কারে recursive call করেছে

J=0 কারণ S node junction না

d=1 হলে delete

o(length) → Time




Junction Point:

(1) A node containing an EoW = 1

(2) multiple children

*** Exam delete এর পরে junction এর code টা লিখো

Error দিবে না কেন?

⇒ Leaf node → return 1

যদি junction point এর code - error → leafnode chk করে নাও

# AVL Tree

Adelson-Velskii and Landis Tree

→ binary search Tree — height balanced

→ প্রত্যেক Node এর জন্য Left subtree এবং Right " " এর difference 1 থাকতে হবে। → Balanced condition

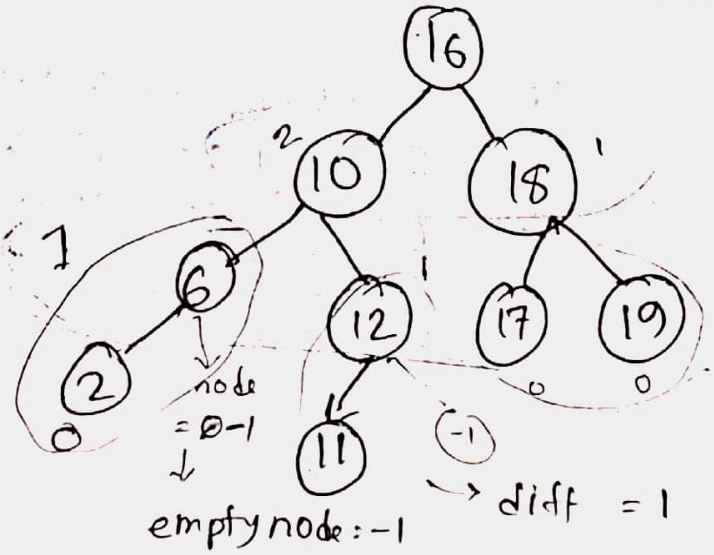
⇒ Indexing এর ক্ষেত্রে use → searching time কম

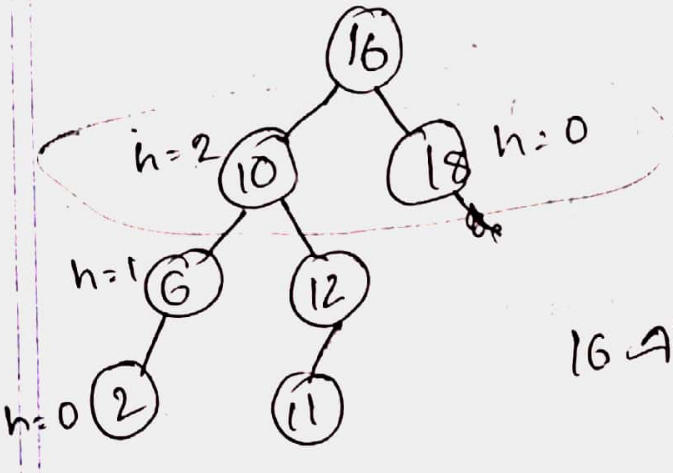
প্রথমে → bst কিনা chk

→ পরে height

empty tree কোনো node নেই যার তার height = -1

height → leaf node থেকে।


16
10
18
6
12
17
19
2
11
node = 0-1
empty node : -1
-1
diff = 1
2-1
diff = 1
1-0
diff = 1
diff (0-(-1)) = 1


16
10
18
6
12
2
11
h=2
h=0
h=1
h=0
diff (2-0) = 2
AVL না

11/09/23

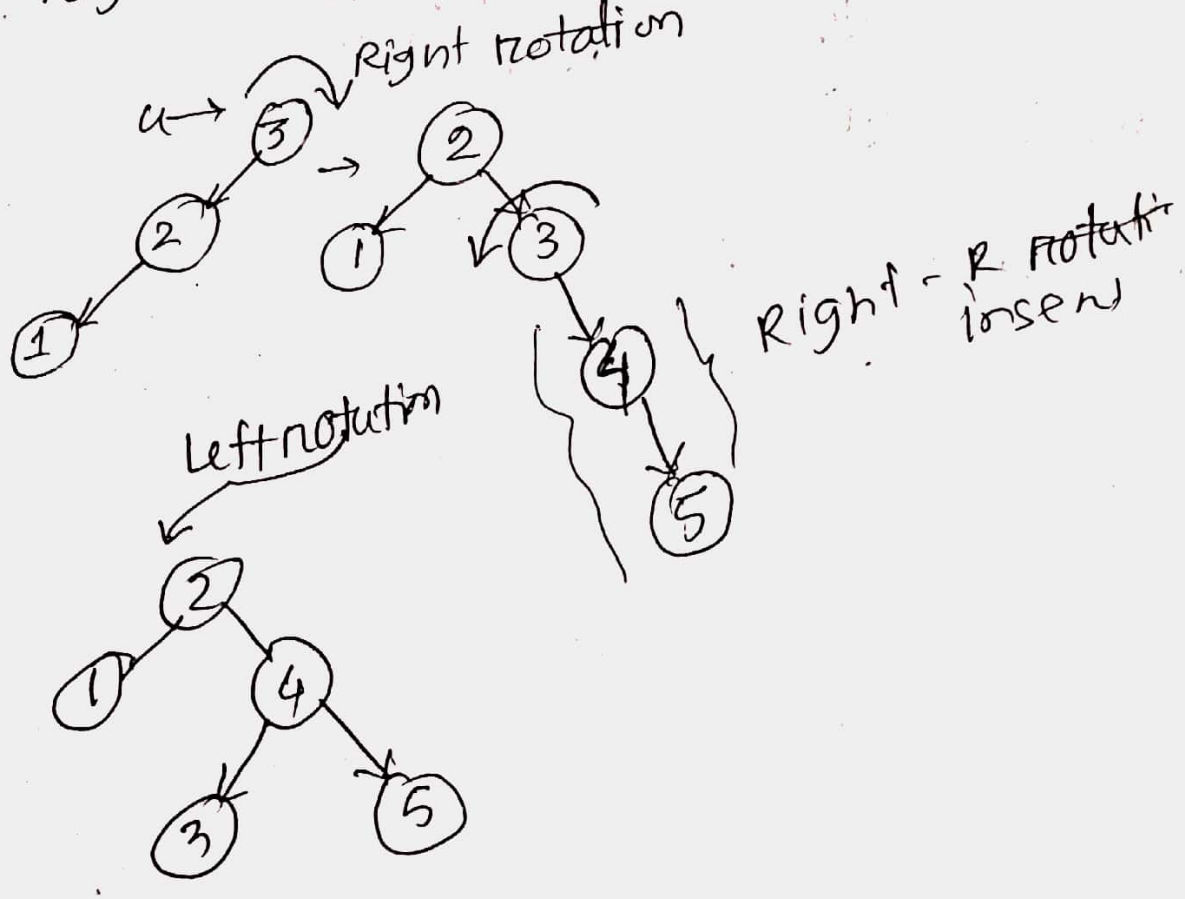
```
struct Node
{
    Node *left;
    Node *Right;
    int value;
    int height;
};
```

u → L-L, R-R { single rotation — AVL property maintain করার জন্য
R-L, L-R { double rotation

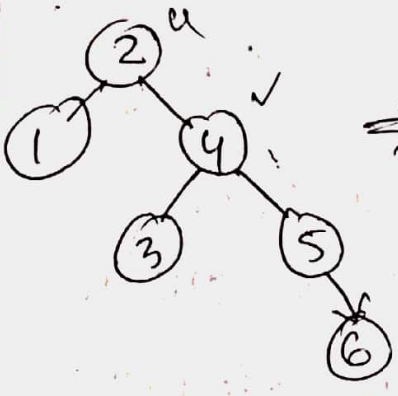
```
int h (Node *u)
{return (u==NULL) ? -1 : u->height}
```

(? → if, : → else)

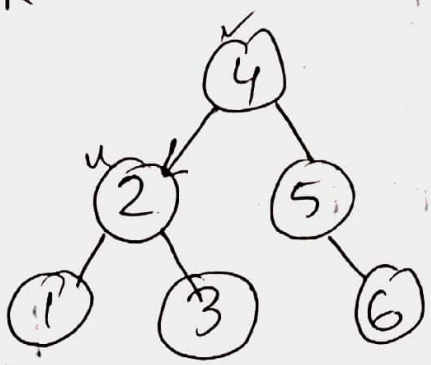
Left - left insertion → Right rotation
right-right " → Left "



After inserting 6




R-R insert — left rotation

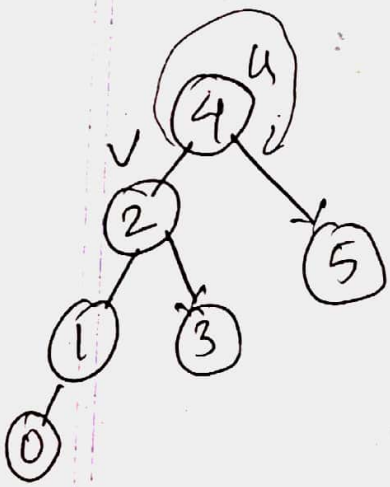


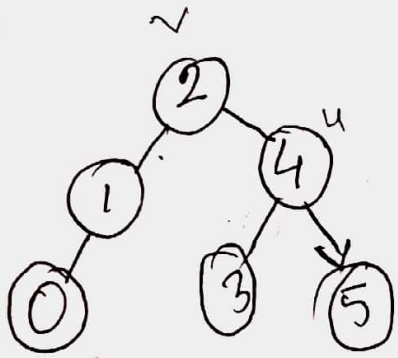

v का left child
u का right u
बन गया।

v = u → right ;

u → right = v → ~~right~~ left

v → left = u




```
void Right (Node * u)
{
  Node *v = u → ~~Right~~ ;
  v → left = v → Right
  v → Right = u;
  u → height = max (h (u → left),
```

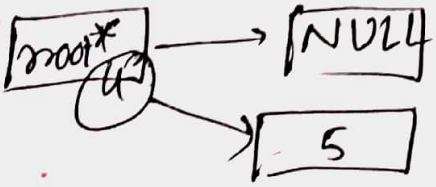
17. Sept. 23

```
void (val, *&u)
*root=NULL
insert (5, root);
u=new Node(5)
```

root* ⟶ *NULL

u* ⟶ [5]

*&u দিলে root সেই u




- u-এর address এই root এর u same

***&u = *root***

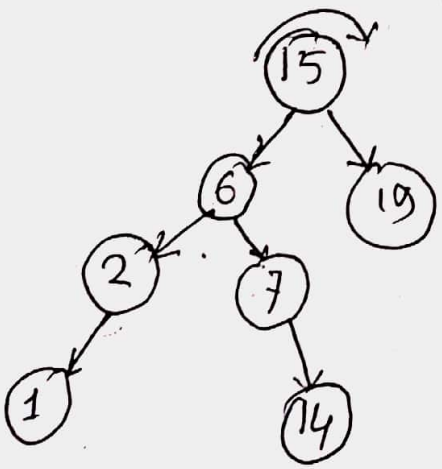
void balance (Node *&u)

insertion, time complexity O(log n)

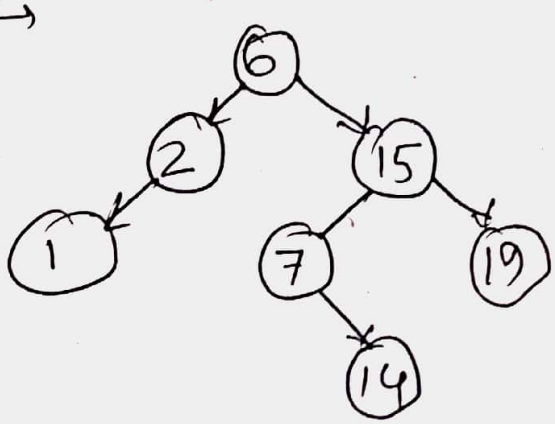
delete, " O(log n)

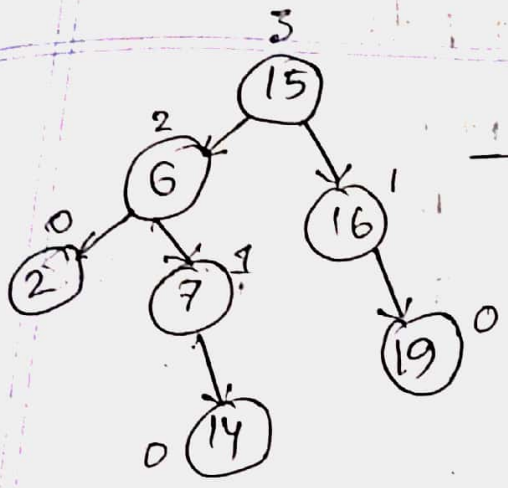
bottom→top approach — balance call

right → min




right rotation






child

Parent

left right rotation

AVL - ds left + slide

Trie → 4

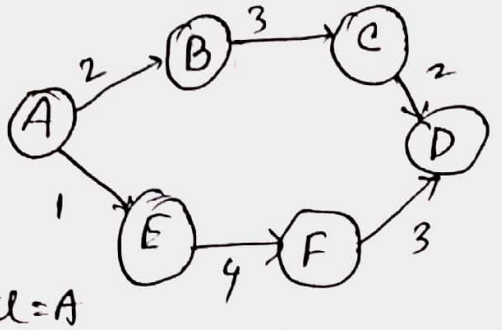
20/09/23

# Single source shortest Path

A-D Path

A, B, C, D

Path - source → destination




Source = A

All possible Path

$A \rightarrow B$

$A \rightarrow B \rightarrow C$

$A \rightarrow B \rightarrow C \rightarrow D$ — $(2+3+2) = 7$ ✓ shortest path.

$A \rightarrow E$

$A \rightarrow E \rightarrow F$

$A \rightarrow E \rightarrow F \rightarrow D$ $(4+3+1) = 8$

$$w(P) = \sum_{i=1}^{k} w(v_{i-1}, v)$$

Shortest path, $\delta(u, v)$ node

$$\delta(u, v) = \begin{cases} \min\{w(P) : u \overset{P}{\rightsquigarrow} v & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

[u → v তে যাওয়ার কোনো way নাই]

variants:

Single destination shortest Paths Problem

" pair " " "

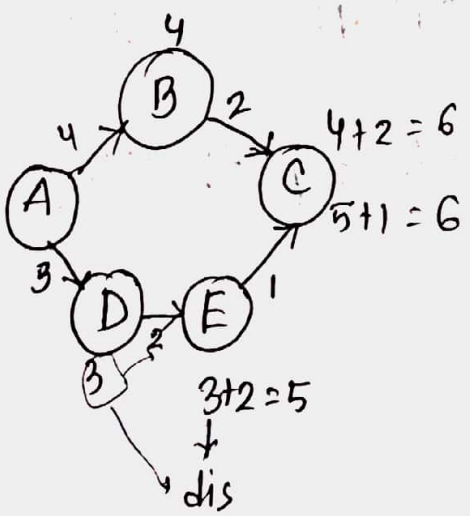
All pairs " " "

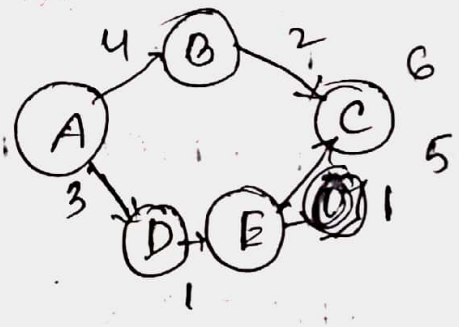
↳ প্রত্যেকটা vertex এর জন্য single source shortest path বানাতে হবে।

* কোনো Prob এ overlapping subproblem Property থাকলে — greedy হবে না; dynamic Pry → Bellman

optimal substructure → greedy, dynamic, dijkstra

weight → Penalty, loss, time, distance বুঝায়




node:
→ Parents
→ distance fro source

A → B

A → D

A → D → E

A → D → E → C

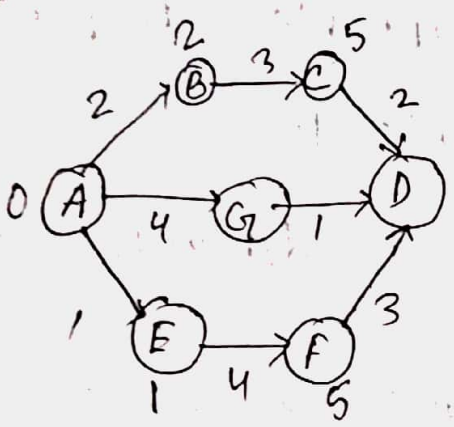
Optimal substructure property , Overlapping

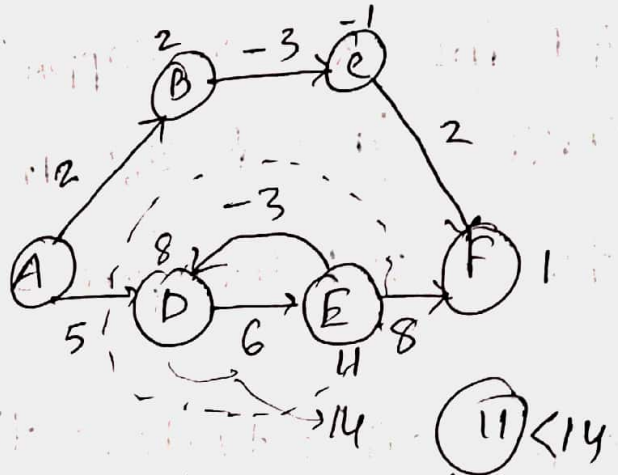
Subpaths of shortest . . . .

Proof

$d[v] >= \delta(s,v)$

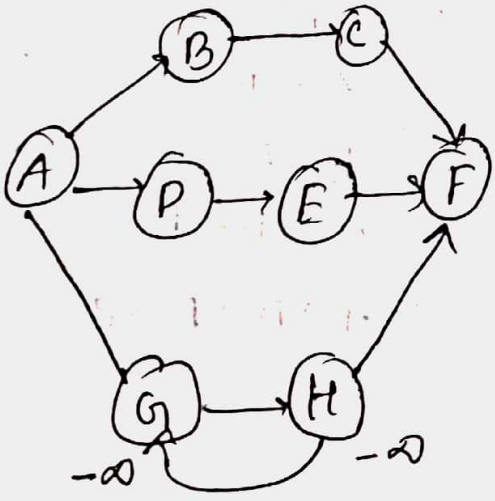
shortest




11<14

এবার vertex থেকে আগ vertex এ যাওয়ার সময় Neg weighted cycle থাকলে, তখন 'A' vertex টার shortest path হবে $-\infty$

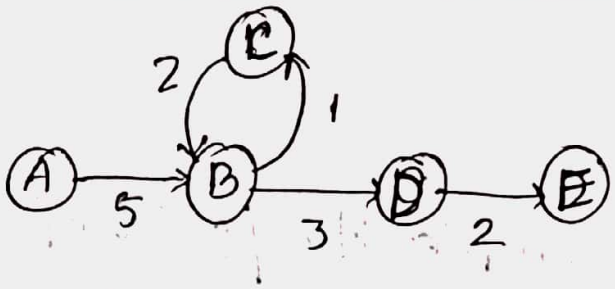


increase / decrease ∆ time complexity → O(logn)

Dijkstra Algorith → neg edge এর জন্য কাজ করবে না,

Bellman ford Algorithm → neg " " " " bt neg -

## Cycle in shortest Path:




## Dijkstra Algorithm: (-) edge নিয়ে কেন কাজ করবে না?

Dijkstra (G, w, s)

1. Initalize - single-source (G, s) → Time complex O(V)
2. S = ∅
3. Q = G.V → O(V)
4. while Q ≠ ∅
5. u = Extract min (Q) → O(V.logV) → O(logn) → O(logn)
6. S = S ∪ {u}
7. for each vertex v ∈ G.adj[u] → O(E) → so (E.logV)
8. Relax (u, v, w)

26/09/2023

Relax → O(1)

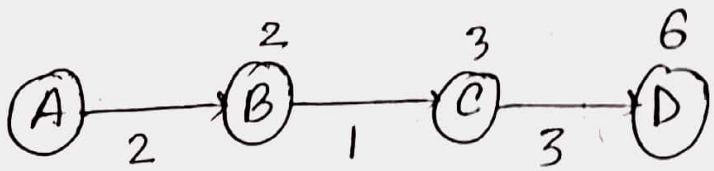
Dijkstra time complexity :

heap দিয়ে করলে " "

→ $O[(E+V)\log V]$

Bellman Ford : time complexity $O(V \times E)$

কেন?




order maintain করলে প্রত্যেকটা edge কে ১বার Relax করবে। Shortest path, shortest distance পাবো।

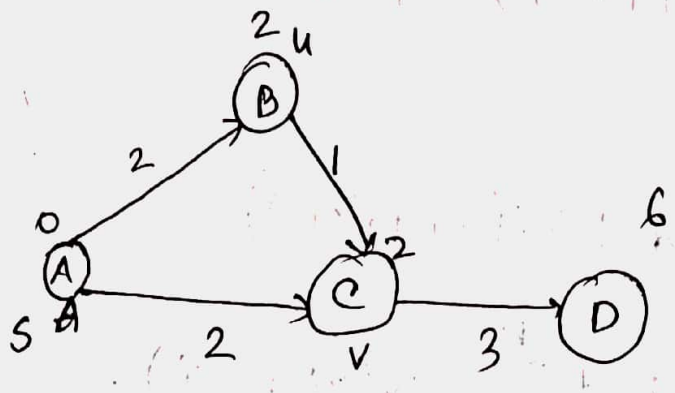
// Line 2-4 এ কেন 3বার relax করা হচ্ছে?

## Tringle Inequality Property:

For any edge $(u,v) \in E$, we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$

$2 \leq 2+1$

$2 \leq 3$




$v.d = \delta(s,v)$

$\leq \delta(s,u) + w(u,v)$ (by the tringle inequality)

$= u.d + w(u,v)$

return true

## Bellman Algorithm:

if $v.d > u.d + w(u,v)$

return false

** negative weighted cycle না থাকলে True Return

proof

Proof

কখন false return করবে?

→ negative weighted cycle থাকলে

$v_i.d \le v_{i-1}.d + w(v_{i-1}, v_i)$ for $i = 1, 2 \ldots k$

Summing,

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

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

$$0 \le \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$k = 2$, হলে,

$v_1.d + v_2.d$

$v_0 = v_k$ সমান কেন?

Theorem (24.6)

20221411

27.09.2023

Bellman ford – simulation

## Dynamic Programming

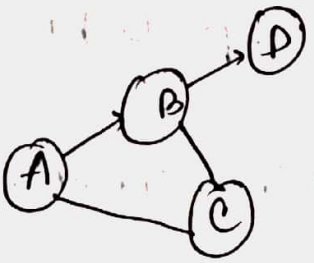
LCS → Longest Common Subsequence

optimal → greedy + dynamic

overlapping → ✗ ✓

optimal + overlapping → dynamic

Optimal substructure property:
↳ Subpath of shortest paths are shortest path.




B → AB path common } overlap

D → " " "

Optimization prob ও যেভাবে solve

1. Greedy
2. Dynamic Programmic
3. branch and bound

Shortest path prob → minimization problem

* যখনই sub prob কে ২বার solve করব।

dynamic Program – ৪টা step needed

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal soln, typically in a bottom-up fashion.
4. Construct an optimal soln from computed info

LCS: ৪টা step

→ subsequence need not be consecutive

X = A B C D  Y = J B A G H C E D

X = A B C D  Y = J B A G H C E D

LCS - optimization Problem (maximize करना)

X = A B C B D A B.  LCS of X and Y = B C B A

Y = B D C A B A

## optimal substructure property:

Let $X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2 \ldots y_n)$ be sequences and let $Z = (z_1, z_2 \ldots, z_k)$ be any LCS of X and Y

(1) If $x_m = y_n$, then $z_k = x_m = y_n$ and $z_{k-1}$ is an LCS of $x_{m-1}$ and $y_{n-1}$.

(2) If $x_m \neq y_n$ the $z_k \neq x_m$ implies that Z is an LCS of $x_{m-1}$ and Y

(3) If $x_m \neq y_n$ Then $z_k \neq y_n$ " " Z is an LCS of X and $y_{n-1}$.

1. $X = ABCD$ ($X_m$) $\quad Z = ACD$ ($Z_k$)

$Y = BAGHCED$ ($Y_n$) $\quad AC \rightarrow$ LCS of $X_{m-1}$ and $Y_{n-1}$

2.

2. $X = ABCDI$ ($X_m$)

$Y = BAGHCEDG$ ($Y_n$) $\rangle$ $Z = ACD$ ($Z_k$)

// $c[i,j] = 0$ if $i = 0$ or $j = 0$

$c[i,j] = 1 + c[i-1, j-1]$ <if $x_i = y_j$

if $x_i \neq y_j$

$c[i,j] = \max(c[i-1,j], c[i, j-1])$

Step-3: Length

$X = ABCB$

$Y = BDC$

$i = 2$ বলতে B/AB কে বুঝায়

| | | B | D | C |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 0 | 0 | 0 | 0 |
| A 1 | 0 | 0 | 0 | 0 |
| B 2 | 0 | 1 | 1 | 1 |
| C 3 | 0 | 1 | 1 | 2 |
| D 4 | 0 | 1 | 1 | 2 |

→ table construct

step-3 LCS → $O(m \times n)$ – time complexity

Diagonal হলে print করব

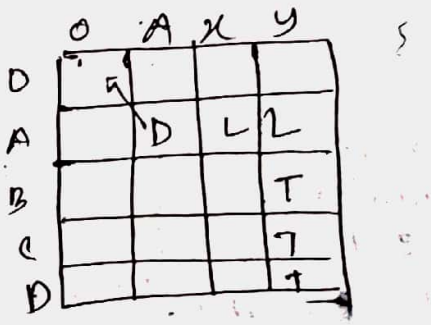
string এর Loop 0 – (m-1) পর্যন্ত

01/10/2023

LCS construct এর step - 3 পর্যন্ত

কোন। আইলে length পর্যন্ত।

step-4 বলালে full code




* **step-4**: constructing an LCS

time complexity : O(m+n)

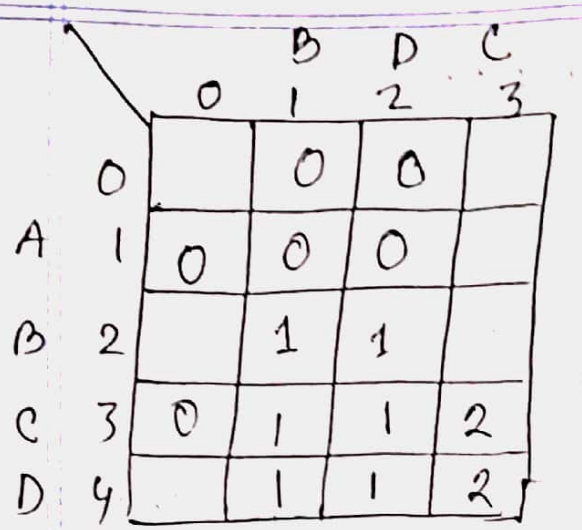
**step-3**: Computing length of an LCS (TOP-DOWN : Memoi tio

* (-1) দিয়ে initialize করতে হবে।

* 0 " " বসা যাবে না।

# memoization Top-down

LCS (ABCB, BDC, 4, 3)







সবগুলো Prob slv করবার নাই। যেগুলো ঘরকার সুধু তাদের " করতে হবে।

কিছু Tabulation এ Ao Prob slv করতে হয়।

Memoization time complexity O(mxn) Topdown

Tabulation " " O(mxn) bottom up

4 steps:

① $i=0$ or $j=0$ → return $c[i][j] = 0$;

② $c[i][j] != -1$ → return $c[i][j]$; (change 없이)

if ③ $X[i-1] == Y[j-1]$ → return $c[i][j] = 1 + LCS[X, Y, i-1, j-1]$

else ④ $X[i-1] != Y[j-1]$ → max (LCS(x, y, i-1, j), LCS(X, Y, i, j-1));

Brute Force :→ efficient 하지

04.10.23

কয়েকটা item available

0-1 knapsack — optimal-substructure property

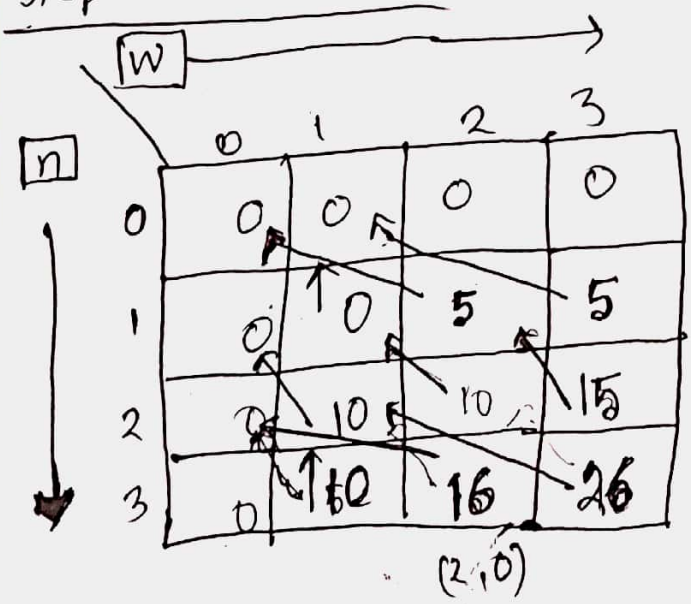
Step-2: Recursive solution:

$$c[i,w] = \begin{cases} 0 & \text{if } i=0 \text{ or } w=0 \\ c[i-1,w] & \text{if } w_i > w \\ \max(v_i + c[i-1, w-w_i], c[i-1,w]) & \text{if } i>0 \text{ and } w \geq w_i \end{cases}$$

item : 1 2

| v | 5tk | 10tk | 16tk |
|---|---|---|---|
| w | 2kg | 1kg | 2kg |

Step-3: Bottom up




↑ উপর থেকে আসছে
n (nibo na)

↖ diagonally
t (talun)

## Step-4 : Constructing an optimal solution:

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | N | T | T |
| 0 | T | T | T |
| 0 | N | T | T |

memoization → simulation

knapsack bottomup → time $O(n * W)$

top down → " " "

Naive Recursive → $O(2^n)$

## Fractional knapsack :

0-1 knapsack → greedy fails solve ✗

fractional → " " " ✓

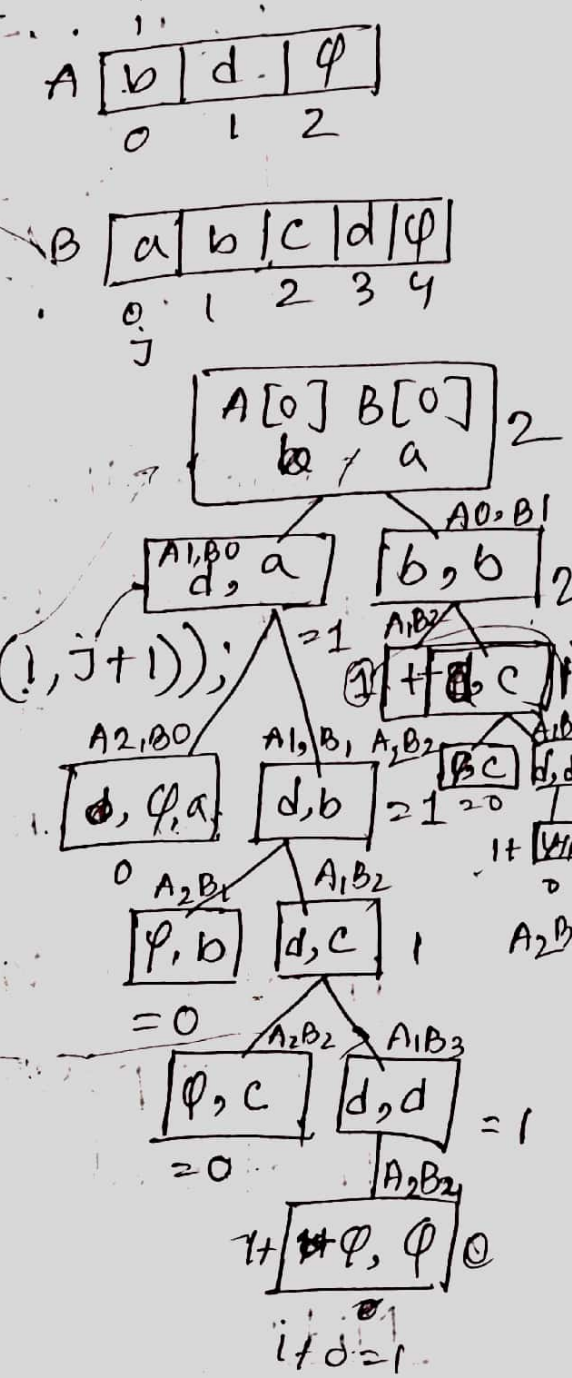
## Mid :

# LCS

## using recursion + memoization

Algo:

```
int LCS(i, j)
{
  if(A[i] == 0 || B[j] == 0)
      return 0;
  else if (A[i] == B[j])
      return 1 + LCS(i+1, j+1);
  else
      return max(LCS(i+1, j), LCS(i, j+1));
}
```




Top-down approach

→ overlapping problem

By solving this problem we can use memoization.

Recursion: $2^n$ → time complexity

→ 1st এ recursive এর চিন্তা করতে হবে কোন Box এ first এ পৌঁছেছি।

A = | b | d | ϕ |
0 1 2

B = | a | b | c | d | ϕ |
0 1 2 3 4

| | a 0 | b 1 | c 2 | d 3 | e 4 |
|---|---|---|---|---|---|
| b 0 | 2 | 2 | | | |
| d 1 | 1 | 1 | 1 | 1 | |
| ϕ 2 | 0 | 0 | 0 | | 0 |

memoization: O(mxn).

dynamic programming (bottom-up approach)

A | b | d |
1 2 i

B | a | b | c | d |
1 2 3 4

| A \ B | 0 | a 1 | b 2 | c 3 | d 4 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| b 1 | 0 | 0 | 1 | 1 | 1 |
| d 2 | 0 | 0 | 1 | 1 | 2 |

j

b d

O(mxn)

```
if (A[i] = B[j])
    LCS[i,j] = 1 + LCS(i-1, j-1)
else
    LCS[i,j] = max | LCS(i-1, j)
                   | LCS(i, j-1)
```

Str 1: stone

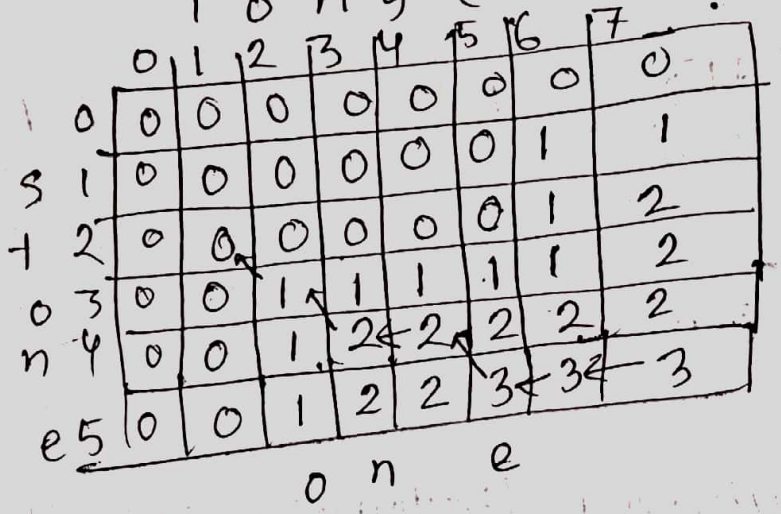
2: longest

if (A[i] = B[j])

LCS(i,j) = 1 + LCS(i-1, j-1)

else

LCS(i,j) = max(LCS(i-1,j), LCS(i,j-1))

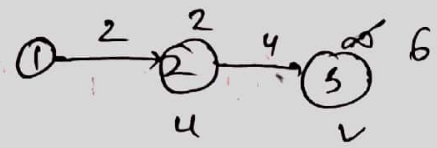


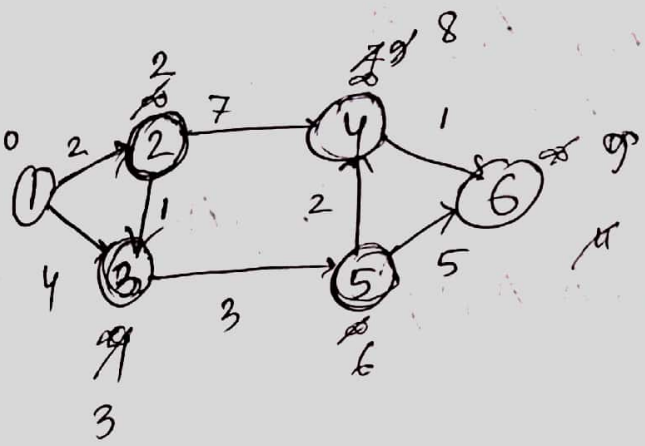
shortest path – minimization prob → optimization → greedy method

Relaxation:

if (d[u] + c[u,v] < d[v])

d[v] = d[u] + c(u,v)

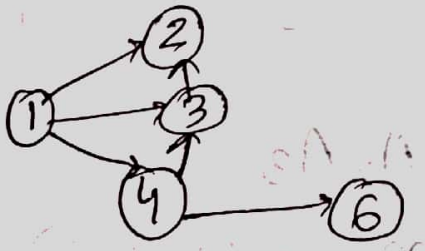



| vertex | distance |
|---|---|
| 2 | 2 |
| 3 | 4 |
| 4 | 8 |
| 5 | 6 |
| 6 | 9 |

→ (-)ve হলে না dijkstra তে

Bellman → dynamic



Rubayet Mam

18.10.23

# MCM

→ square matrix এর জন্য → mcm algo apply করতে পারব না,

$A_1 \rightarrow p \times q$

$A_2 \rightarrow q \times r$

$A_1\ A_2\ A_3\ A_4$

$((A_1A_2A_3)A_4)$ $((A_1A_2)(A_3A_4))$ $(A_1(A_2A_3A_4))$

$A_1\ A_2\ A_3$

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

i j i j

$A_1\ A_2\ A_3$

mcm of (1, 2, 3)

i=1 j=3

$A_i, A_{i+1}, A_{i+2} \cdots A_j$

$A_i$ ——— $A_j$

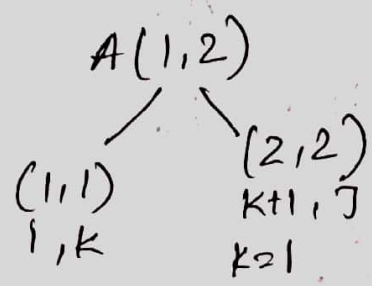
$A_k$

$(A_i \ldots A_{i+1} \cdots A_k)$

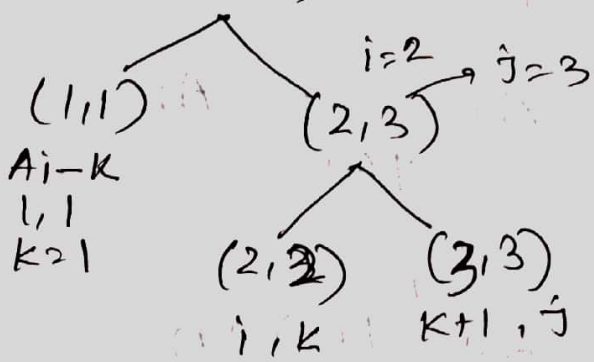
$(A_{k+1} \ldots A_{k+2} \cdots A_j)$

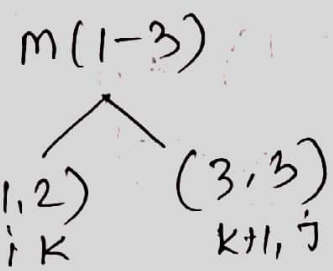
30

A(1,2) mcm(1,2)




m(1-3)




A = 4×5
B = 5×6
C = 6×7

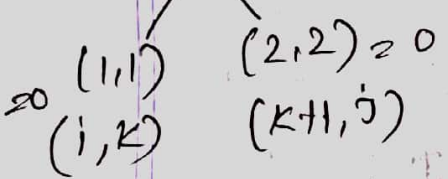
~~Pi-1 k~~

k=1

$P_{i-1}$ $P_k$ $P_j$

$P_0$ $P_1$ $P_3$

= 4 × 5 × 7




k=2

$P_0$ $P_2$ $P_3$

4 × 6 × 7

mcm → optimal + overlap

↓

সহজে যা Property maintain করে তাহলে dynamic Programming.

23.10.23

exponential time of n

$A_i \cdots A_k$ $\quad$ $A_{k+1} \cdots A_j$

$T(K)$ $\quad$ $T(n-k)+1$

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

$P_{i-1}$ $P_k$ $P_j$

$T(1) \geq 1 = 2^0$

mcm $2^{n-1}$

$$1 = 2^0$$
$$2\sum_{i=1}^{n-1} 2^i + n$$
$$= 2(2^{n-1}) + n$$
$$= n2^n + n \geq 2^{n-1}$$

$k = i$ to $j-1$

cost = mcm $(i,k)$ + mcm $(k+1, j)$ + $P_{i-1} P_k P_j$

if cost $< m[i,j]$

$m[i,j]$ = cost

memoization এ value store করে রাখা তাই recursive call হবে কিন্তু time complexity কম হবে।

Time complexity : memoization

$O(n^2) \cdot n = O(n^3)$

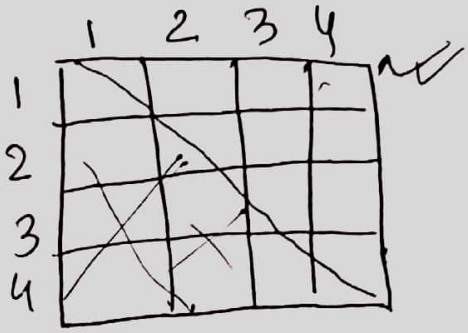
recursive, $T = O(2^n)$

408 Pg → Fig – 15.7

recursive algo use করে time complexity – কমানো

memoization " " " → কম

Illustration




378 Pg 15.2-2, (3,4) → recursive

15.2-5 → বাম

i j

MCM (1,3) অথবা (1,3)

(1×2)3

(1,3)

(1,1) (2,3) (s[i,j], +i,j)

A1

(i, s[i,j]) (2,2) (3,3)

A2 A3

(1,2) (3,3)

i, j সমান না হওয়া পর্যন্ত recursive call

$(A_1 (A_2 A_3))$

15.3-2 →

merge sort → sub problem property নাই

381 → Page — slide – 402

diff → greedy, dp and divide and conquer

কেন আমরা bottom up use করব কেন করব top down ?

25.10.23

# Branch N Bound

State space tree

Backtracking — more efficient

Backtracking
→ DFS
→ Top most

Branch N Bound → less

↳ backtracking হবে না।

→ bound এর উপর depend করে Branch হয়।

optimal substructure + overlapping ⇒ Branch n Bound

** descending order করে নিতে হবে আগে।

| Item | weight | value | value/weight | |
|---|---|---|---|---|
| | 4 | 40 | 10 | always descending order |
| | 7 | 42 | 6 | |
| | 5 | 25 | 5 | |
| | 3 | 12 | 4 | |

$ub = v + (w - w_i)\ (v_{i+1} / w_{i+1})$

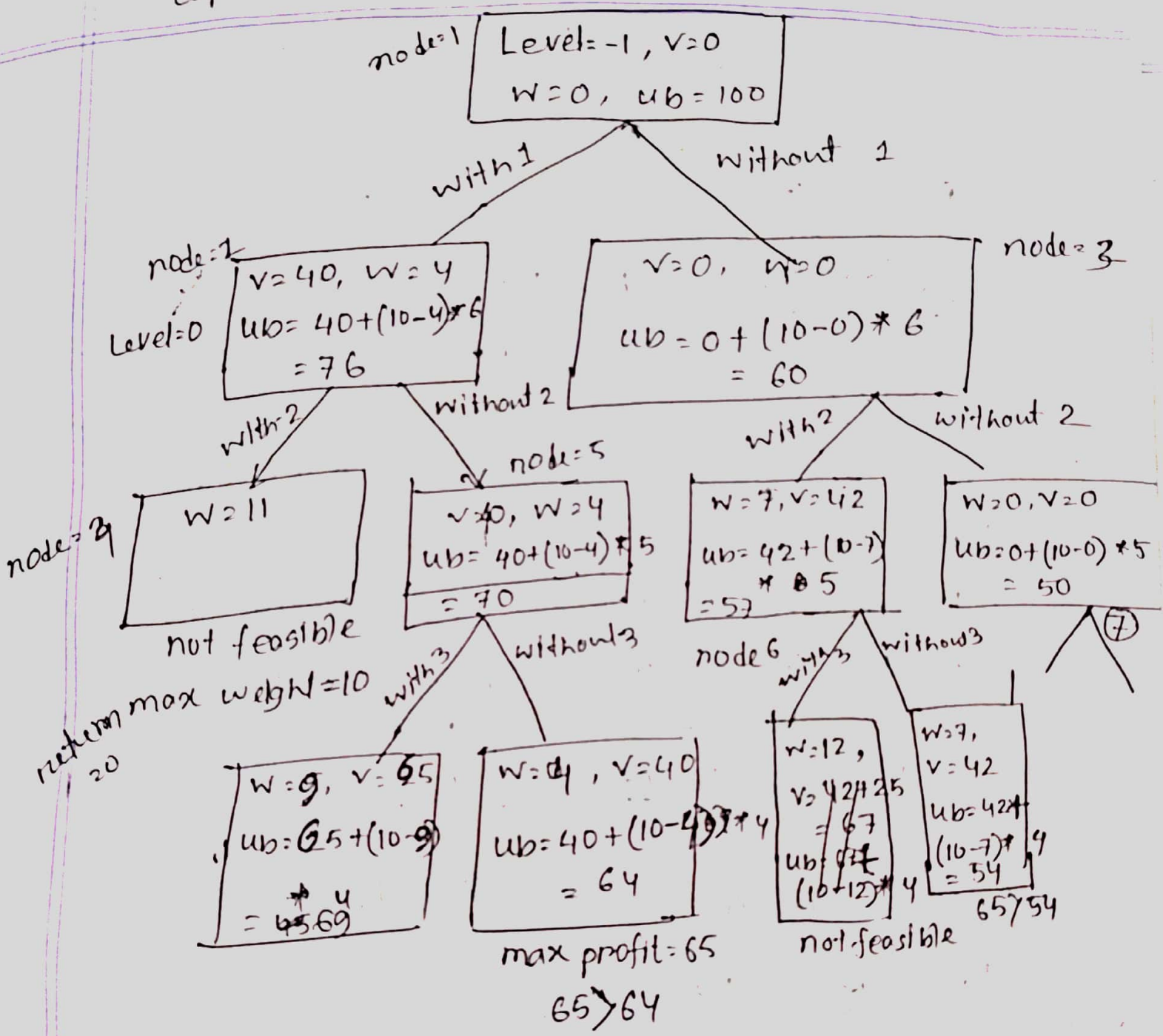
| Item | W | value | V/W |
|---|---|---|---|
| 1 | 4 | 40 | 10 |
| 2 | 7 | 42 | 6 |
| 3 | 5 | 25 | 5 |
| 4 | 3 | 12 | 4 |

⇒ Colour available থাকতে হবে।

| |
|---|
| W=0, V=0 |
| ub = 100 |

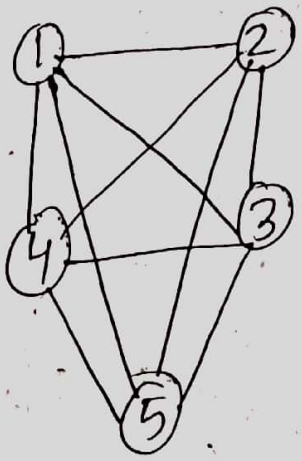
$$ub = V_i + (W - W_i) * \frac{V_{i+1}}{W_{i+1}}$$

Capacity, W = 10



# Travelling Salesman Problem

• Change




Row=1 : reduce by 10

প্রত্যেক row এর যে value টা সবচেয়ে কম এইটা সব value থেকে বিয়োগ করতে হবে।

$$\begin{bmatrix} inf & 20 & 30 & 10 & 11 \\ 15 & inf & 16 & 4 & 2 \\ 3 & 5 & inf & 2 & 4 \\ 19 & 6 & 18 & inf & 3 \\ 16 & 4 & 7 & 16 & inf \end{bmatrix} \quad \begin{bmatrix} inf & 10 & 20 & 0 & 1 \\ 13 & inf & 14 & 2 & 0 \\ 1 & 3 & inf & 0 & 2 \\ 16 & 3 & 15 & inf & 0 \\ 12 & 0 & 3 & 12 & inf \end{bmatrix}$$

কলাম এ সবচেয়ে কম value দিয়ে বিয়োগ করুন:

Reduce matrix

$$\begin{bmatrix} inf & 10 & 17 & 0 & 1 \\ 12 & inf & 11 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & 10 & inf \end{bmatrix}$$

Reduce matrix

from cost of [1][2] = 10

$$\begin{bmatrix} \infty & 10 & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 10 & \infty \end{bmatrix}$$

Reduce করতে চাইলেও হবে না এখানে কোনো change হবে না

তাই RCL = 0

cost 1 + RCL [1,2) + cost of (1,2)

= 25 + 0 + 10 = 35

Cost 2

Cost of 3 (1,3) = ~~17~~ 25+11+17 = 53

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 10 & \infty \end{bmatrix}$$

Reduce করা

RCL = 0

Cost (4) = Cost (1) + RCL + Cost (1,4)

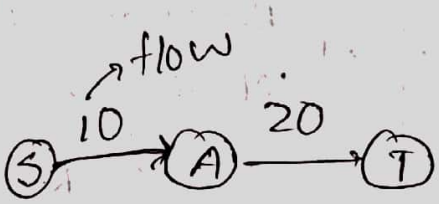
= 25 + 0 + 0 = 25

(25+5+1)

Cost 5 = 31

30/10/23

## Flow Network




S থেকে A তে যাবে

i/p = o/p

flow in = flow out

S-A → 10 = 10

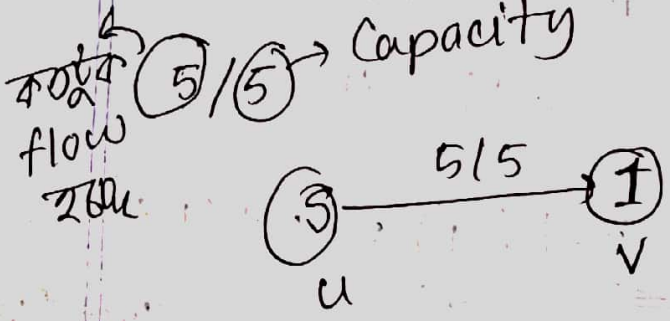
S A

T তে 10 unit যেতে পারবে

যদিও capacity A-T = 20

* ford fulkerson method

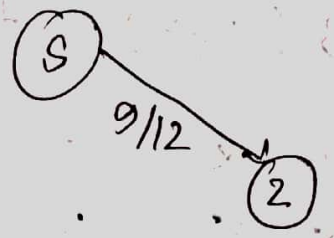
* Edmond karp




f(uv) = 5

c(uv) = 5

| f(uv) | c(uv) |
|---|---|
| 5 | 5 |




c = 12

f = 9

residual capacity = 3

$c_f(u,v) = c(u,v) - f(u,v)$

$= 12 - 9$

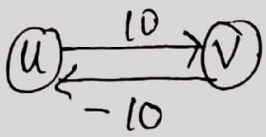
$= 3$

u, v ∈ V - {s, t} → vertex

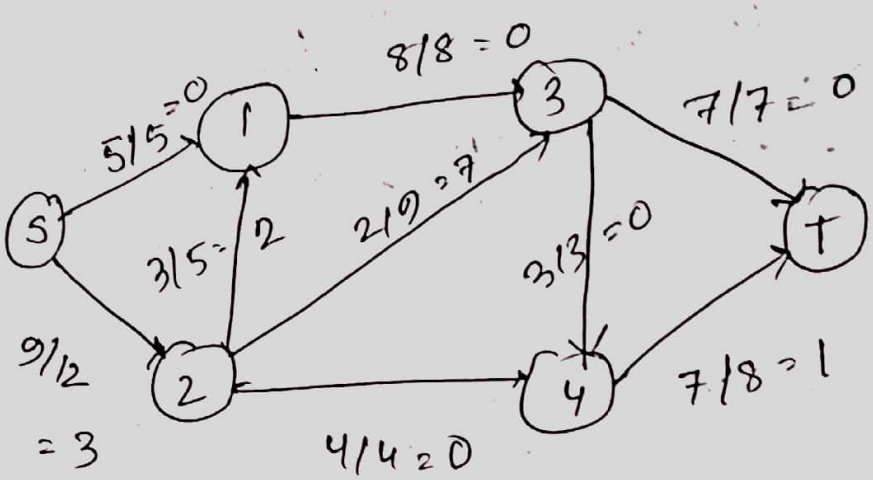
$\sum_{u \in V} f(u,v) = \sum_{v \in V} f(u,v)$

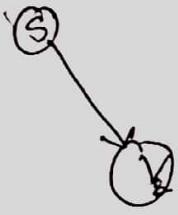
* flow in equals to flow out

Forman book



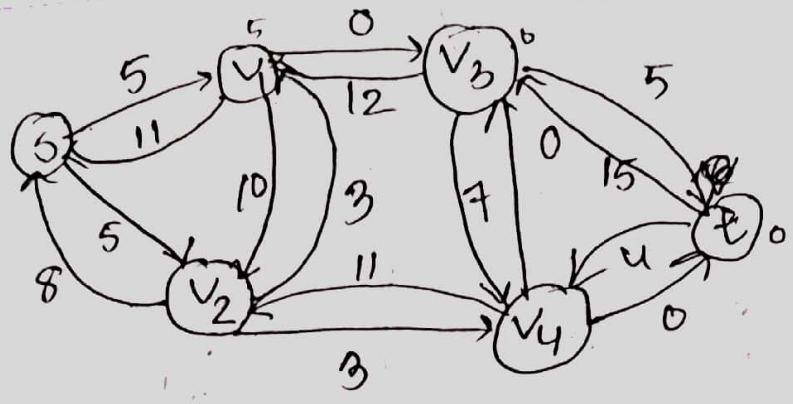

$u, v \in V$



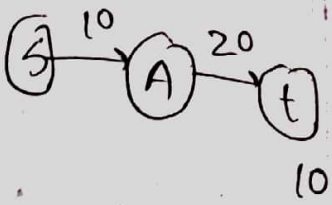



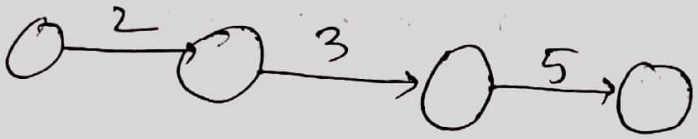

residual network =?

1/11/23




Flow/capacity

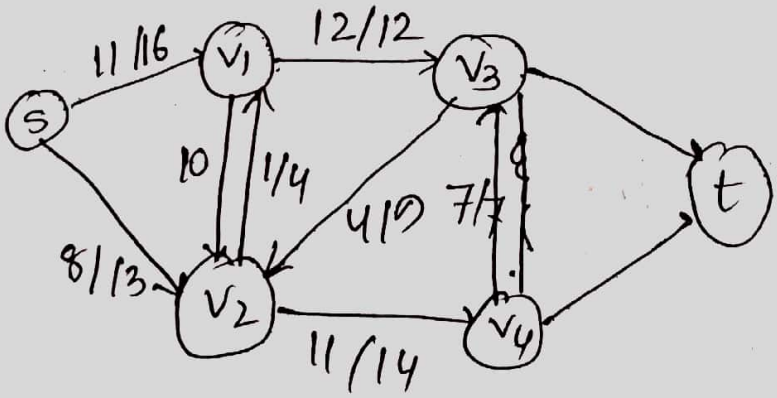



residual capacity = 2

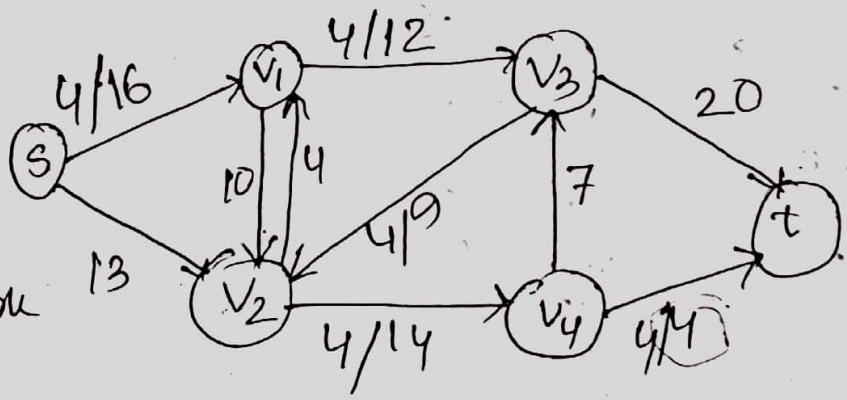
$C_f(P) = \min\{c_f(u,v) : (u,v) \text{ is on } P\}$

residual

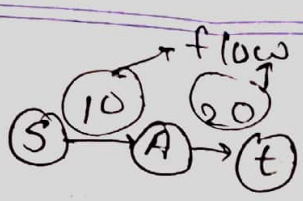



Capacity দেওয়া থাকলে বের করতে হবে network max flow কত হবে using ford-fulkerson

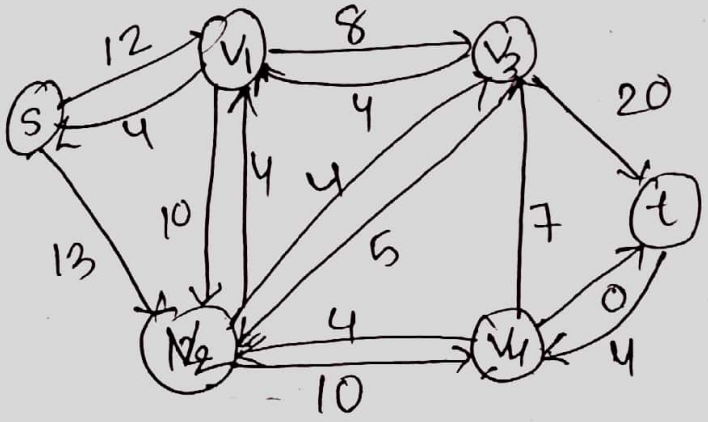
Dfs




Flow network



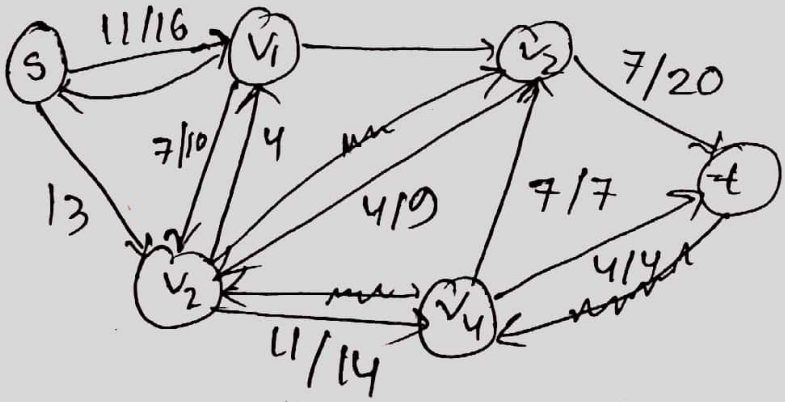

$S \rightarrow V_1 \rightarrow V_3 \rightarrow V_2 \rightarrow V_4 \rightarrow t$ [max 4 unit flow পাচ্ছি]
$= 4$



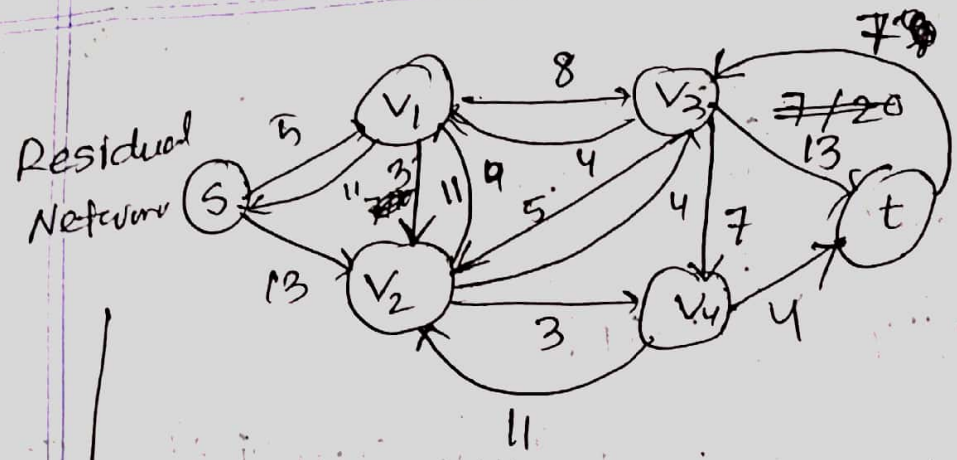

Residual network

$S \rightarrow V_1 \rightarrow V_2 \rightarrow V_4 \rightarrow V_3 \rightarrow t$ = max flow 7

total flow = $7+4 = 11$




flow




Resulting flow = 11

Flow

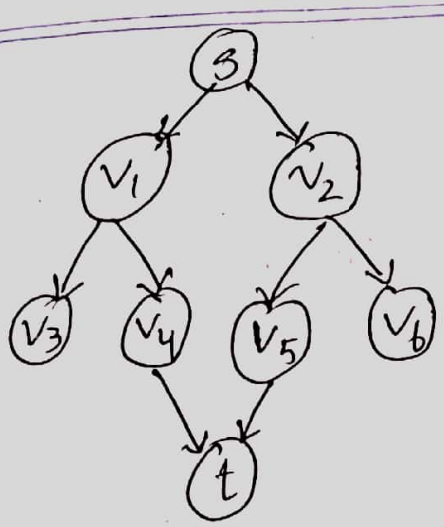
$$f(v,u) = f(u,u) - C_f(P)$$

$$= 4 - 4 = 0$$

$O(Et)$

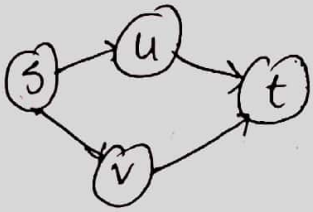
$O(E \times (V \times t))$

02/11/23




Ford = E × F* 2 million

BFS = O(V+E)



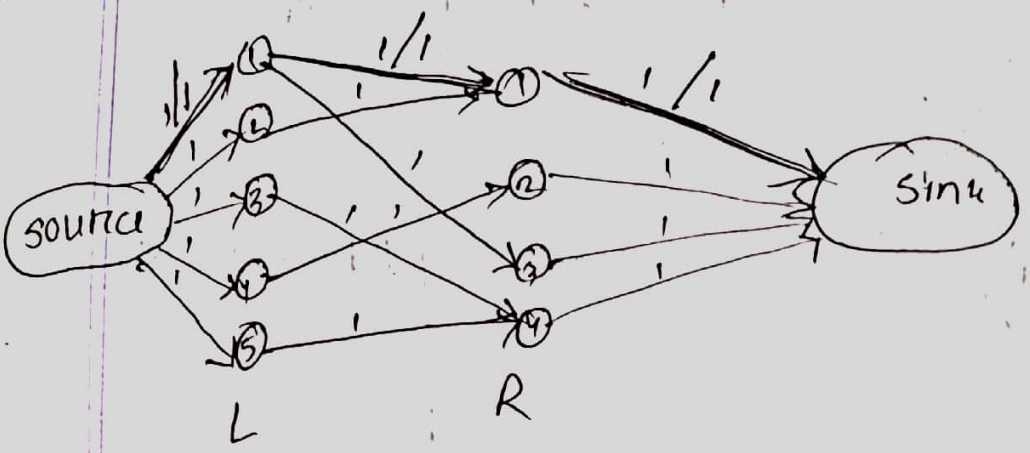

u→v যাবে না, BFS এ Gray হয়ে গেল

s→u
s→v
u→t

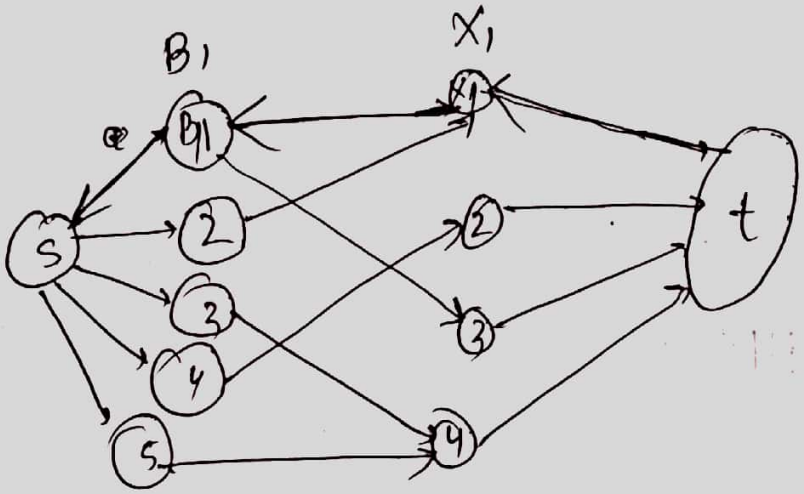
t→u→s → 1million back

06.11.23

# Matching




## 1st augmenting path:




reverse

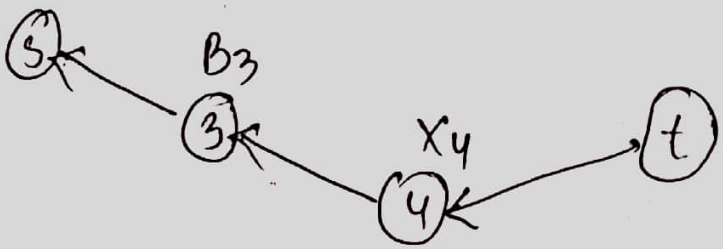
$\{B_1, X_1\}$

$\{B_3, X_4\}$

$\{B_4, X_2\}$

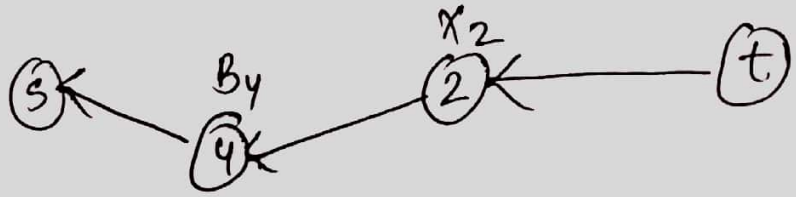
* ~~Pseudo~~ Simulation / Illustration

2nd:




$O(E * f^*)$

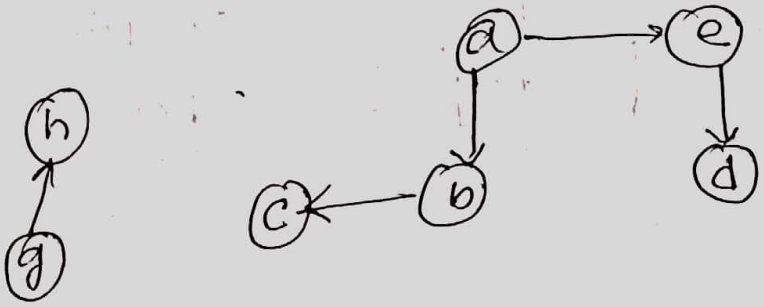
3rd:



08/11/23

DFS – soln – forman book

→




(1) call DFS — O(V+E)

(2) → O(V)

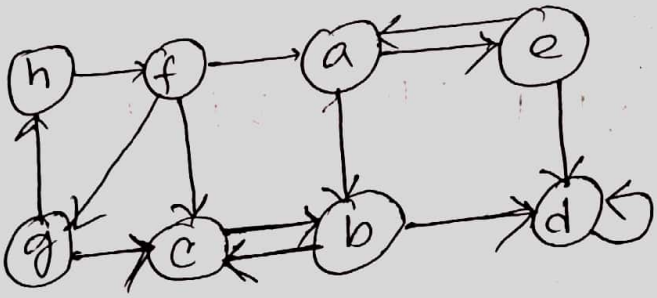
(3) → O(V+E)

(4) → O(V)

Total: time complexity
O(V+E)

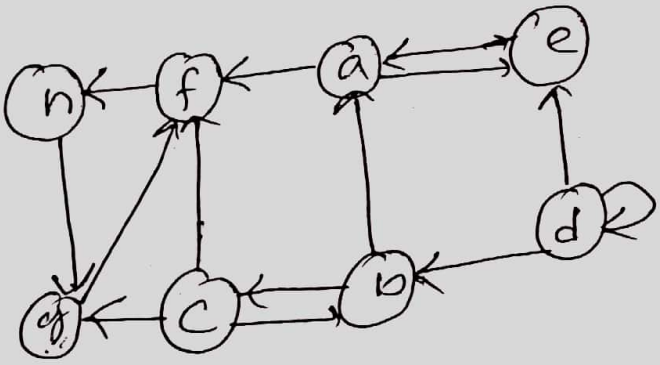
#

(1)



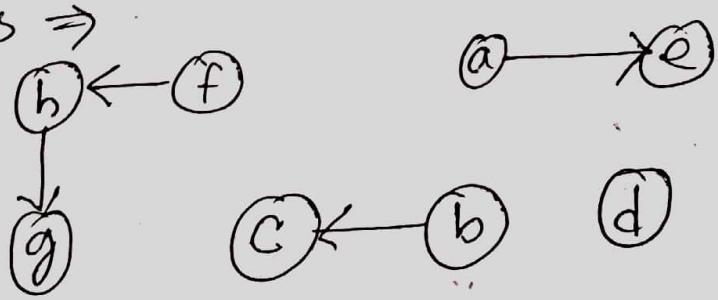

transpose

(2)

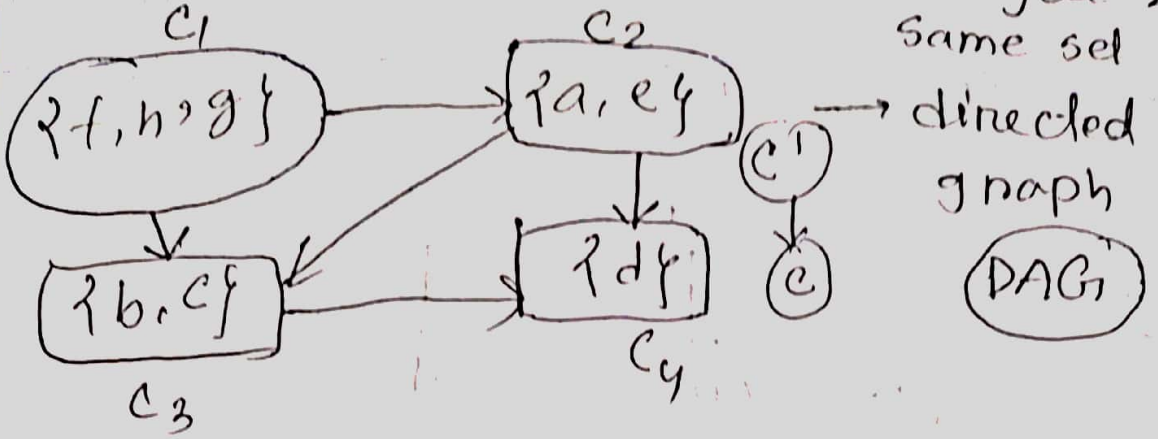



fghabced

(3) dfs ⇒



→ যদি cycle থাকতো এ same set এ চলে আসত

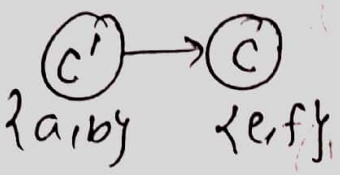



→ directed acyclic graph

DAG

* $d(C') > d(C)$ কেন?

→ C components এর graph দিয়ে বুঝা গেল,



~~1, 4, 8, 9, 11~~

13.11.23

## Greedy

overclapping Task!

↳ max subset নিতে হবে।

3টা approach

(1) Earliest start time

(2) " finish " ✗ [কোনটা efficient] → optimal soln নিতে

(3) Shortest interval.

(2) finish time ও sort করে নিতে হবে

### activity - selection Problem Greedy Approach:

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| start | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| end | 4 | 5 | 6 | 7 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

k=1;
selected task[ak];
for (m=2 to length[task]
if (finish[k] ≤ start[m])
selected task[a[m]];
k=m;
return selected task.

**Quiz**

activity selection
time complexity
O(n)

(*) shortest interval, earliest start time
⇒ optimal soln পাওয়া যায়।

Lab → earliesh finish time আসবে
→

Simulation – 10
time complexity – 5
Theory – 5

CT-2 : Topological sort, mcm

---

Divide and Conquer Approach : 15.11.16

Solving Recurrances;
i) iterative expansion
ii) Recurssion Tree
(iii) Master Theorem.

(1) merge sort
(2) Quick "
(3) Binary "
(4) max/min algorithm

$$T(n) = \begin{cases} b & n=1 \text{ or } 2 \\ 2T(n/2) & n>2 \end{cases}$$

$T(n) = 2T(n/2) + b$

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

$= 2^2 T(n/2^2) + 2b + b$

$= 2^2 [2T(n/2^3) + b] + 2b + b$

$= 2^3 T(n/2^3) + 2b + 2b + b$

$= 2^i T(n/2^i) + 2^{i-1} + \dots + 2$

we stop when, $n/2^i = 1$

$n = 2^i$

$i = \log_2 n$

$T(n) = nb + (2^i - 1)b$

$= nb + (n-1)b$

$= nb + nb - b$

$= 2nb - b$

$= b(2n-1)$

$= O(n)$

## merge sort

### Best and worst case:

$$T(n) = \begin{cases} b & \text{if } n=1 \\ 2T(n/2) + bn & \text{if } n>1 \end{cases}$$

$T(n) = 2T(n/2) + bn$

$= 2[2T(n/4) + b(n/2)] + bn$

$= 2^2 T(n/2^2) + 2bn$

$= 2^2 [2T(n/2^3) + b(n/4)] + bn$

$= 2^3 T(n/2^3) + b(n/4)] + bn$

$= 2^3 + (n/2^3) + \cancel{b(n/4)}\ 3bn$

$= 2^i T(n/i) + ibn$

$= nb + \log n \times bn$

$= b(n + n\log_2 n)$

Quicksort (Worst case)

$$T(n) = \begin{cases} b & \text{if } n=1 \\ T(n-1) + bn & \text{if } n>1 \end{cases}$$

$T(n) = T(n-1) + bn$

$= T(n-2) + b(n-1) + bn$

$= T(n-3) + b(n-2) + b(n-1) + bn$

$\vdots$

$= T(n-i) + b(n-(i-1)) + \ldots b(n-2) + b(n-3) \ldots$

$= T(n-i) + b((n-i+1) + \ldots (n-2) + n-1 + n)$

$n - i = 1 \Rightarrow n = i+1$

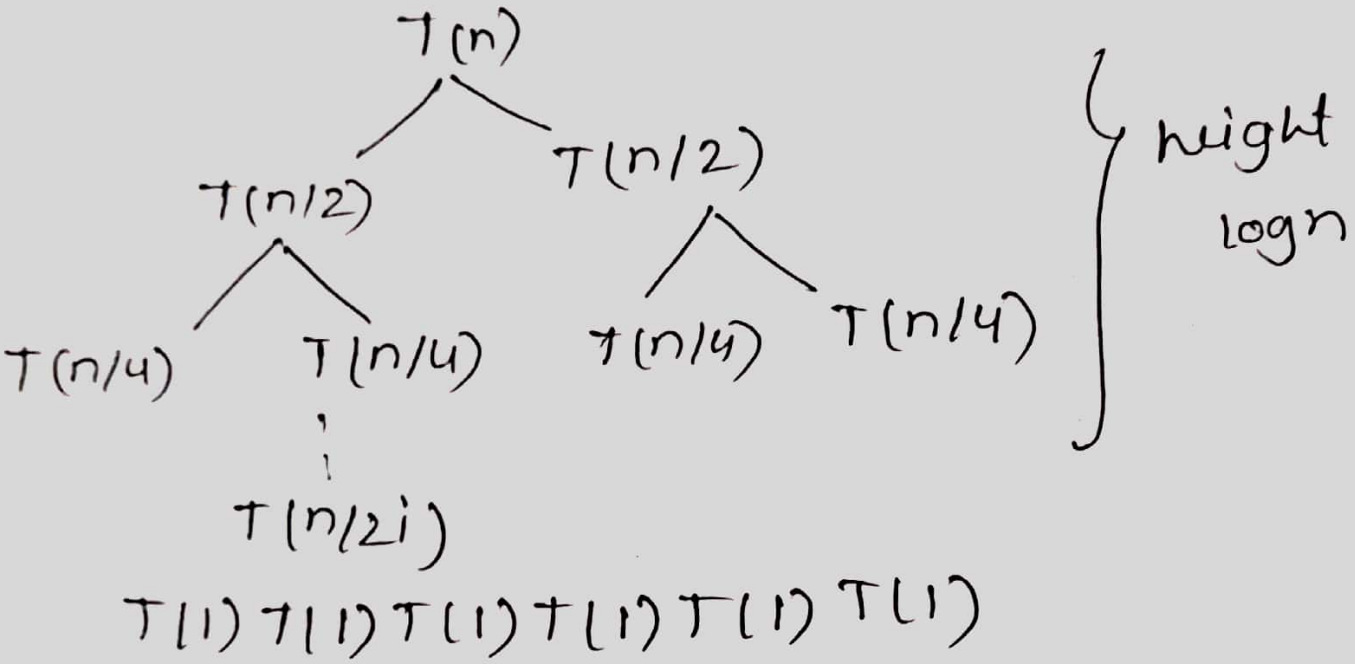
$\Rightarrow i = 1-n$

$= b + b(2 + \ldots + (n-2) + (n-1) + n)$

$= b + b\left[\frac{n(n+1)}{2}\right] - 1 \quad = b + bn^2 = O(n^2)$

# Recursion Tree

$$T(n) = \begin{cases} b & n=1 \\ 2T(n/2) + bn & n>1 \end{cases}$$




| Num of nodes per level | Time for each node | Total time |
|---|---|---|
| 1 | bn | bn |
| 2 | b(n/2) | bn |
| 4 | b(n/4) | bn |
| $2^i$ | $b(n/2^i)$ | bn |
| n | b | bn |
| | | $\log_2 n$ terms |

Rubayet mam

# The Master Theorem

15.11.2023

$a \geq 1, \quad b > 1$

$T(n) = aT(n/b) + f(n)$ → data store করতে যতটুকু time লাগে

* Master Theorem → cookbook

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

according to theorem, $a = 4, b = 2, f(n) = n$

$g(n) = n^{(\log_b a)}$ → divide conqure করা

$= n^{(\log_2 4)}$

$= 2n$ ~~$n^{\log_2 4}$~~

~~$f(n) = n$~~ $= n^2$

$f(n) = n$

$g(n) = n^2$

$f(n) << g(n)$ , $T(n) = O(g(n))$ → $O(n^2)$ → $n << n^2$

$f(n) >> g(n)$ , $T(n) = O(f(n))$

$f(n) \sim g(n)$ , $T(n) = O(g(n) \log_b n)$ / $O(f(n) \log_b n)$

# $T(n) = 4T(n/2) + 8n + 5$

(i) $f(n) = 8n + 5 \longrightarrow O(n)$

~~$a = 8$~~

$g(n) = n^{\log_2 4} = n^2 \longrightarrow O(n^2)$

(ii) $T(n) = 9T(n/3) + 5n\sqrt{n} + 8\log n$

$9T(n/3)$

$g(n) = n^{\log_3 3^2} = n^2$

$f(n) = 5n\sqrt{n} + 8\log n$

$= n^{3/2} + \log n$

↓

$n^{3/2}$

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

(iii) $T(n) = 7T(n/4) + 3n^2 + 4n$

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

$g(n) = n^{\log_b a}$

$= n^{\log_4 7}$

$= \log_4 7 < \log_4 16$

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

(iv) $T(n) = 20T(n/4) + 3n^2 + 4n + 5 \Rightarrow Tn = O(g(n))$

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

$g(n) = n^{\log_4 20}$ $\Big\}$

(v) $T(n) = 9T(n/3) + 5n^2 + 7$

$f(n) = n^2$
$g(n) = n^2$ $\Big\} T(n) = O(n^2)$

(*) Master theorem → defn

$$T(n) = \begin{cases} b \\ 2T(n/2) + bn \end{cases}$$

$= n\log_2^2$

$= n\log n$

(*) $T(n) = \begin{cases} b \\ T(n-1) + bn \end{cases}$

$a = 1$, $b = 1$ [b এর value 1 এর থেকে বড় হতে হবে]

→ Iterative/recursion দিয়ে solve করতে হবে।

20.11.23

## Flow network

Ford → time complexity O|P এর উপর depend

→ flow 0 ধরে নিতে হয়.

→ augmenting flow

→ residual network

(*) capacity - augmenting path

→ DFS

Edmond → BFS

Capacity কম হলে → Ford

" বেশি " → Edmond

min cut → max flow এর জন্য

Bipartate

22.11.23

## P-class of Problems

time → $O(n^k)$

vertex এর degree even হলে Euler possible.

" " না।

(*) inorder → Binary tree হতে হবে।

(*) Graph দিয়ে দিবে Euler দিয়ে solve + inorder, preorder, postorder বের করতে হবে।

np → i/p sequence এর জন্য varify করতে পারলি

npc → reduction

27.11.23

NP-Complete Problem (cont.)

$S = \{3, 5, 10, 2, 1, 21\}$

$n = 7$

N-P Hard Problem: → [Polynomial time এ solve করা যায় না]

$NPC \subseteq NP\text{-Hard}$

→ Halting problem

Co-P : $P = CO\text{-}P$

* $CO\text{-}NP \stackrel{?}{=} NP$

* CO-P is a subset of CO-NP

$P = CO\text{-}P$

$P \subseteq NP$

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

~~$CO\text{-}P \stackrel{?}{=}$~~

~~CO~~ $NP \stackrel{?}{=} CO\text{-}NP$

$NPC \subseteq NP\text{-hard}$

$NPC \subseteq NP$

$COP \subseteq CO\text{-}NP$

$P \subseteq CO\text{-}NP$

$P \subseteq NP \cap CO\text{-}NP$

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

## Approximation Algo!

$C^*$ → optimal soln

$C$ → near "

$\rho$ → approximation ration

maximization Prob → $C^* \geq C$

minimization " " $C \geq C^*$