tinyurl.com/uiuportal

powered by UIU Memes

Course: CSI227

Faculty: Swakkhar Shatabda

Note courtesy: Bushra Tabassum

Algorithm

1/2/16

Sec: NA

Swakkhar Shatabda (2008)

BSc. CSE, BUET (2007)

PhD, Brisbane, Australia

Sites.google.com/sites/sshatabda/csi227 ***

Room: 104

Phone: 01776195310

Email: swakkhar@cse.uiu.ac.bd

Counsiling

SUN - TUES → 2:00 - 5:00 PM
Room - 104

Wed - → 11:15 - 1:15
Room - 1307

Classtest → 20%. (at least 4) could be more than 4
best 2

Assignment → (at must 2/3) 5%. জমা দিলেই হবে।

Attendance → 5%.

mid → 30%.

Final → 40%.

Book:

~~Text~~ Introduction to Algorithms (3rd or so on)
T. Cormen et al.

Motive

- Number of problems
- Number of Algorithms
- How to create new Algo

Data Structure

- Store information
- operation

↓

Algorithm / computation

⇒ An algorithm is a well defined procedure that takes an instance of a problem as input and produces output

Search Problem

We are given
- a set of object
- a search key

} Abstract problem

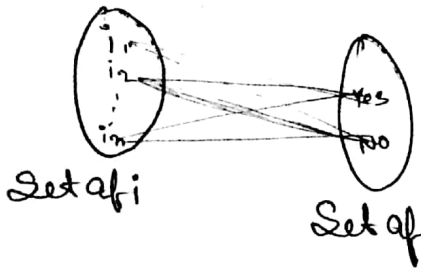
a set of object

23 17 3 2 10 12

a search key

3

} Problem instance



⇒ Problem এর Output algo দিয়েই যাচাই করা যাবে।

Sorting Problem → Data → Lab Algo

We are given
- a set of object
- that set of object → output in a sorting order

23 17 3 2 10 12

→ 2 3 10 12 17 23

6! 6 5 4 3 2 1

⇒ 6! ভাবে আমরা বের করতে পারি।

03/02/2016

elms.uiu.ac.bd

enroll key: UiU 227

course: CSI227

Problem এর defenation:-
input and output.

Application

- Internet - Search, Routing
- E-commerce - Security
- Political Compaign
- Airline
- Logitices
- Scheduling
- GPS Driving

Complexcity Analysis

Searching Problem:

Find Key

$i =$ 0 1 2 3 4 5

$A = [23 \quad 17 \quad 12 \quad 3 \quad 10 \quad 2]$

```
int findKey(int A[], int length, int key)
{
  for (int i=0, i<length ; i++)   ① ② ④
  ③ if (key==A[i])
        return i; ⑤

  return -1; ⑥
}
```

length=6
key = 3
exist / not

=> i element আছে কিনা
=> -1 হলে খুঁজে পাওয়া যায় নাই।

=> micro-instruction এর collection

=> সময় সবচেয়ে কম best case
=> সময় সবচেয়ে বেশি worstcase

=> machine depent হতে পারবে না।

=> কোন micro instraction কতবার run করে।

Runtime Analysis / complexing Analysis

(ছোট ছোট time)

	best case key=23	key=3 runtime	(T)	key=10	worst case key=55
c_1	1	1	1	1	1
c_2	1	1111	4	5	7
c_3	1	1111	4	5	6
c_4	0	111	3	4	6
c_5	1	1	1	1	0
c_6	0	0	0	0	1

* এর instance এর জন্য,

$T = c_1 + 4c_2 + 4c_3 + 3c_4 + c_5$

$T(n)$
= runtime of an algorithm on an instance of size n

Runtime

↳ machine independent

⟶ problem instance size

→best case always same

length = 10

key = 3	key = ?
1	1
1	11
1	10
0	10
1	0
0	1

length = n

array best case	array worst case
1	1
1	n+1
1	n
0	n
1	0
0	1

$T_{best}(n) = c_1 + c_2 + c_3 + c_5 = c = (\text{constant time}) = c \cdot n^0 = \Theta \cdot n^0$

$T_{worst}(n) = c_1 + (n+1)c_2 + nc_3 + nc_4 + c_6$

$= n(c_2 + c_2 + c_4) + c_1 + c_2 + c_6$

$= an + b$ (linear time)

$= an^1 + b \cdot n^0 = \Theta n^1$

$y = \overset{a}{\uparrow} 2x + \overset{b}{\uparrow} 3$

⇒ n এর highest power এর ~

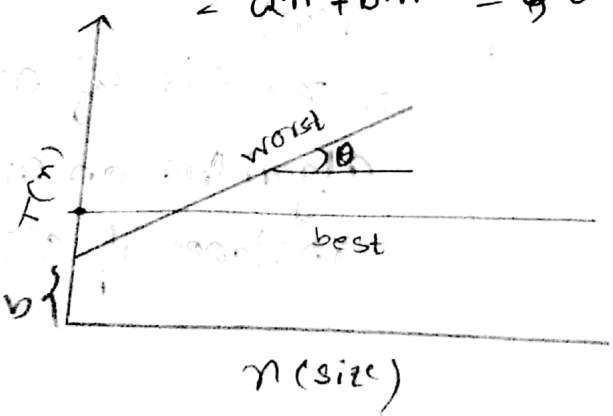
⇒ best case এর same

হবে।

⇒ worst case এর

বাড়বে,

⇒ ~ = big ~~fete~~ fute



worst case এর graph

Let,

$T(n) = n^5 + \frac{5n^{(7)}}{10017} + n^3$

$= \tilde{O}(n^7)$

Assignment - 1

⇒ runtime বের করার জন্য $\tilde{O}$ বের করতে হবে।

Algo

→ Input Instance

→ Size of a Array

Another @ program

```
int SumID Array ( int array[] int n )
{   int sum = 0;          (1)
    for ( int i=0 ; i<n ; i++ )     (2) (3) (5)
    {                                   i+=2
        sum += array [i];           (4)
    }
    return sum ;            (6)
}
```

⇒ Runtime এর Law প্রয়োগ

$c_1 - 1$

$c_2 - 1$

$c_3 - (n+1)$

$c_4 - n$

$c_5 - n$

$c_6 - 1$

$\therefore T(n) = an + b$

$= \tilde{O}(n)$

⇒ i += 2 এর জন্য, (জোড়/বিজোড়)

$c_1 \to 1$

$c_2 \to 1$

$c_3 \to \lceil \frac{n}{2} \rceil + 1$

$c_4 \to \lceil \frac{n}{2} \rceil$

$c_5 \to \lceil n/2 \rceil$

$c_6 \to 1$

$T(n) = \lceil \frac{n}{2} \rceil (a) + b$

$= \Theta\left(\lceil \frac{n}{2} \rceil\right) = \Theta\left(\frac{n}{2}\right)$

$= \Theta(n)$

Ceeling করব না,

$\lceil \frac{n}{2} \rceil + 1$

$11/2 + 1 = \lceil 5.5 \rceil + 1$

$= 6 + 1$

$= 7$

⇒ constant বাদ দিতে পারি,

⇒ i* = 2 এবং i = 1

$c_1 \to 1$

$c_2 \to 1$

$c_3 \to (\log_2 n) + 1$

$c_4 \to \log_2 n$

$c_5 \to \log_2 n$

$c_6 \to 1$

$T(n) = a \cdot \log_2 n + b$

$= \Theta(\log_2 n)$

1 2 4 8 16 32 64 ...

2^0 2^1 2^2 2^3 2^4 2^5 2^6 ...

$n = 8 \quad 4$

$n = 32 \quad 6$

K i → $2^{K-1} \geq n$

কত তম power

$\log_2 2^{K-1} = \log_2 n$

$\Rightarrow K - 1 = \log_2 n$

$\Rightarrow K = \log_2 n + 1$

$\Rightarrow i < 50$

$T(n) = c_1 + c_2 + 51c_3 + 50c_4 + 50c_5 + c_6$

⇒ n এর মান fixed হলে ① always constant. কারণ ② এর মান n এর উপর dependent.

```
⇒ int sumSquareMatrix(int** array, int n)
{ ①
  int sum=0;
  for (int ② i=0; i ③ < n; i++) ⑧
  {
      for (int ④ j=0; j ⑤ < n; j++) ⑦
      {
          ⑥ sum += array[i][j];
      }
  }
  return sum; ⑨
}
```

$c_1 - 1$
$c_2 - 1$
$c_3 - n+1$
$c_4 - n$
$c_5 - n(n+1)$
$c_6 - n^2$
$c_7 - n^2$
$c_8 - n$
$c_9 - 1$

$i=0 \; ; \; j=0$

$c_4 = 1$
$c_5 = n+1$
$c_6 = n$
$c_7 = n$

$T(n) = an^2 + bn + c$
$= \Theta(n^2)$ [Quadratic]

08/02/16

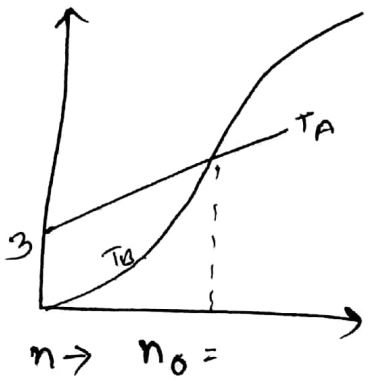
$T_A(n) = 10n + 3$

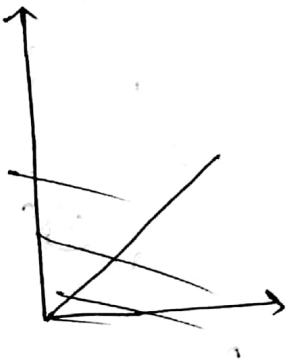
$T_B(n) = n^2$

Asymptotically? অসীমে

n এর value 0 থেকে অসীম।
ছোট, বড়, আবার।

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
$T_A(n)$	3	13	23	33	43	53	63	73	83	93	103	113	123	133	143	153	
$T_B(n)$	0	1	4	9	16	25	36	49	64	81	100	121					





for some value of $n \geq n_0$

$T_A(n) \leq T_B(n)$. c is positive integer

$10n \cdot 3 \leq n^2$, $n \geq 11$

$10n + 3 = O(n^2)$ Upper limit → এর চেয়ে আর বড় আর হবে না।
big-Oh Upper bound

$n \geq n_0$

$T_B(n) \geq c \cdot T_A(n)$

$n^2 \geq 10n + 3$

$n^2 = \Omega(10n+3)$ lower limit → এর চেয়ে আর ছোট হবে না।
big-omega lower bound

$\Omega(n)$	$O(n)$
X	Y
ছোট	বড়
minimax runtime কিভাবে যাচাই করে,	maximum runtime কিভাবে যাচাই করে,

O = big-oh ⇒ বড়

Ω = big-omega ⇒ ছোট,

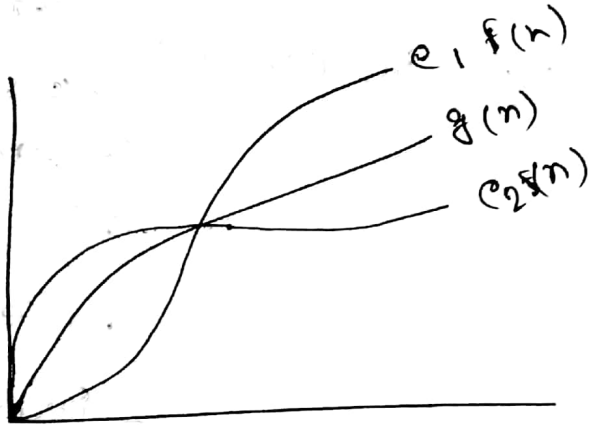
→ worst case এ ~~Hder~~ upper boundare O use.

→ best case এ lower bound use. Ω

$f(n) = O(g(n))$

$f(n) = \Omega(g(n))$

$f(n) = \tilde{\Theta}(g(n))$



$c_1 \cdot f(n) \geq g(n) \geq c_2 f(n)$

$\tilde{\Theta}$ = big theta = tight bound.

$g(n)$

$10n^2 + 3n + 5$

$\tilde{\Theta}(n^2)$

$O(n^2)$] দুটা

$\Omega(n^2)$] হইলে

true হইবে

$\tilde{\Theta}$ true.

⇒ Upperbound ও lowerbound both একসাথে true হলেই tight bound হবে।

⇒ একই কাজ বার বার করা হলো iterative

⇒ big-oh আমার চেয়ে বড়।
big-omega আমার চেয়ে ছোট।
big-theta আমার সমান।

Recursive :

```
int linear Search (int A[], int length, int key, int i)
{
  (c1) if (i >= length)
      (c2) return -1;
  else
  { (c3) if (A[i] == key)
      (c4) return i;
  return linear Search (A, length, key, i+1);
  } (c5)
}
```

$i \geq n$ হলে তখন থেমে যাবে। থেমে যাওয়া।

1. Terminating criteria/condition
basic (কাজটা কি)

2. Recurrence condition
[$i < length$ / $i < n$]
i এর value কমে হলো।

$$T(n) = c_1 + c_3 + c_5 + T(n-1)$$
$$= c + T(n-1)$$

X = 23 17 12 3 10 2

key = 2

Linear Search (X, 6, 3, 0)

↓ again call

Linear Search (X, 6, 3, 1)

↓ again call

Linear Search (X, 6, 3, 2)

↓ again call

Linear Search (X, 6, 3, 3)

Linear Search (X, 6, 3, 0)

frame

total value

⇒ ১টা function same কাজ করে but must we prefer iterative।

⇒ recursion easy but we use iterative

$\Rightarrow S(n) = 1+2+3+4+5+ \cdots + (n-2) + (n-1) + n$

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

i) $n=1$, $S(n)=1$

ii) $S(2) = 1+2 = S(1)+2$

$S(3) = 1+2+3 = S(2)+3$

$\vdots$

$S(8) = 1+2+3+4+5+6+7+8 = S(7)+8$

$S(n) = S(n-1)+n$

```
int sum (int n)
{ @if (n == 1)
      (b) return 1
  else
      return  sum (n+1) +n;
       (c)        (d)
}
```

$T(n) = \overset{c_1}{a} + \overset{c_3}{c} + \overset{c_4}{d} + T(n-1)$

recurrence formula

$= c + T(n-1)$

Substitution method

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

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

$= c + c + T(n-2)$

$T(n) = 2c + T(n-2)$

n বার এর জন্য call হয় (n-1)

$T(n-2) = c + T(n-3)$

$T(n-3) = c + T(n-4)$

$T(n-4) = c + T(n-5)$

$\vdots$

$T(k) = kc + T(n-k)$

$3c + T(n-3)$

$\Rightarrow (n-1)c + (n-(n-1))$

$\Rightarrow (n-1)c + c_1 + c_2 = O(n)$

H.W

```
void prog (int n)
{
    if (n == 1)          // c1
        return;          // c2
    else                 // c3
    {
        for (int i=0; i<n; i++)   // c4, c6
            printf ("%d", i);     // c5
        prog (n-1);               // c7
    }
}
```

best case → c_1, c_2

Worst case

$$T(n) = c_1 + c_3 + (n+1)\,c_4 + n\,c_5 + n\,c_6 + T(n-1)$$

$$= an + b + T(n-1)$$

10/02/16

Class-Test # 1

Wednesday Next

17-02-2016

Runtime Analysis

(1) Recursive

(2) loop

$T(n) = T(n-1) + an + b$

$= T(n-2) + a(n-1) + b + an + b$

$= T(n-2) + a[(n-1)+n] + 2b$

$= T(n-3) + a(n-2) + a[(n-1)+n] + 3b$

$= T(n-3) + a[(n-2)+(n-1)+n] + 3b$

$= T(n-4) + a[(n-3)+(n-2)+(n-1)+n] + 4b$

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

$T(n-2) = T(n-3) + a(n-2) + b$

$n-(n-1)$

$1 = (n-1) - n$

$2 = n-(n-2)$

$= T(1) + a[2 + \cdots + (n-2) + (n-1) + n] + (n-1)b$

$= c_1 + c_2 + a\left(\frac{n(n+1)}{2} - 1\right) + (n-1)b$

$= O(n^2)$

Merge Sort

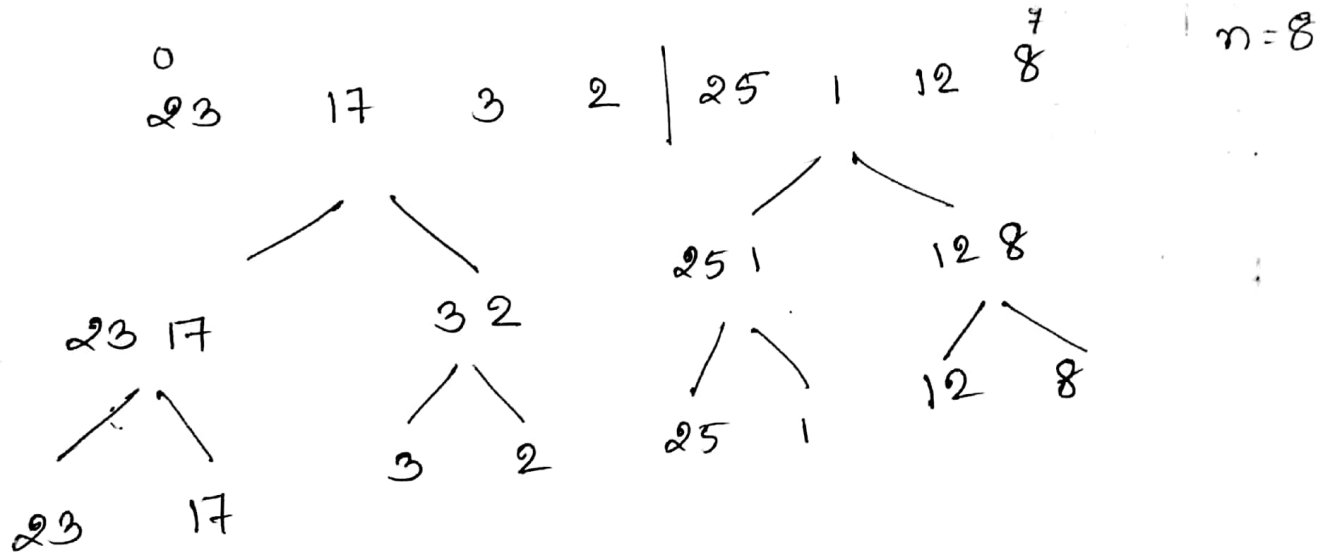
→ Sorting Problem

→ Divide and conquer

① Divide the problem into smaller but similar subproblem.

② Recursively solve the subproblem. If the size of the problem is too small then solve it.

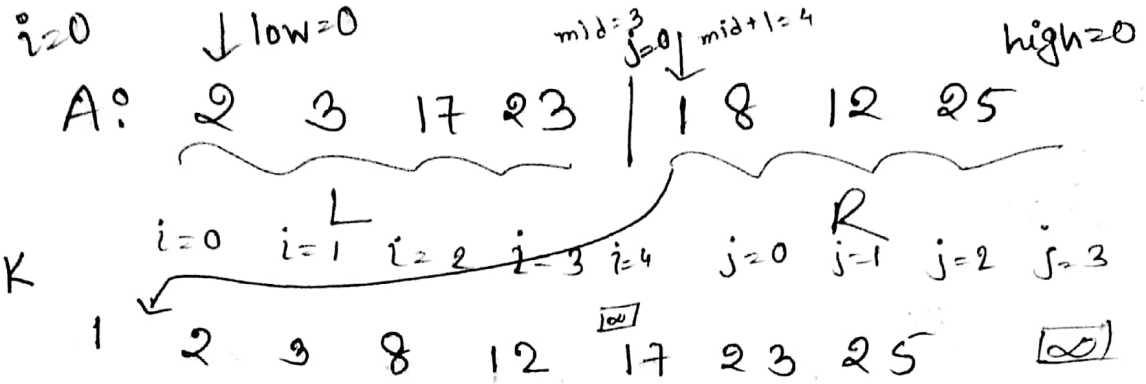
③ Combine the solutions of the sub problems.



Merge

↳ We are given two sorted Arrays

→ Merge or combine them into a single sorted array.



=> INT_MAX = ∞
सबसे बड़ा value

```
if L[i] ≤ R[j]
  {
    A[K] = L[i]
  }
    i++;
else
  {
  A[K] = R[j]
    j++;
  }
```

```
void Merge (int A[], int low, int mid, int high)
{
c1  int n1 = mid - low + 1;
c2  int n2 = high - mid;
c3  int L [n1 + 1] ; R [n2 + 1];
c4  int i, j, K;
c5  for (i=0; i<n1 ; i++)        // c6, c7
c8      L[i] = A[low + i];
    for (i=0 ; i<n2 ; i++)       // c9, c10, c11
c12     R[i] = A[mid + 1 + i];
c13 L [n1] = INT_MAX; R[n2] = INT_MAX;   <limits.h>
    for (K=low; K<=high ; K++)   // c14, c15, c16
c17     if (L[i] <= R [j])
        {
c18         A[K] = L[i];
c19         i++;
        }
    else
        {
c20         A [K] = R[j];
c21         j++;
        }
}
```

runtime

$c_1 - 1$
$c_2 - 1$
$c_3 - 1$
$c_4 - 1$
$c_5 - 1$
$c_6 - (n1+1)$
$c_7 - n1$
$c_8 - n1$
$c_9 - 1$
$c_{10} - n2+1$
$c_{11} - n2$
$c_{12} - n12$
$c_{13} - 1$
$c_{14} - 1$

$*c_{15} - n_1+n_2+1$
$c_{16} - n_1+n_2$
$c_{17} - n_1+n_2$
$c_{18} - n_1$
$c_{19} - n_1$
$c_{20} - n_2$
$c_{21} - n_2$

Heap Sort

17-02-2016

যেখে আর্ গুনিতক হে তার child

→ Tree structure

→ Tree - property

→ (1) A complete binary tree
→ (2) Heap property
- for a max heap
  parent ≥ child
- for a min heap
  parent ≤ child

→ binary heap

→ যাদের child থাকে তাদের internal nodes. এরা parents.

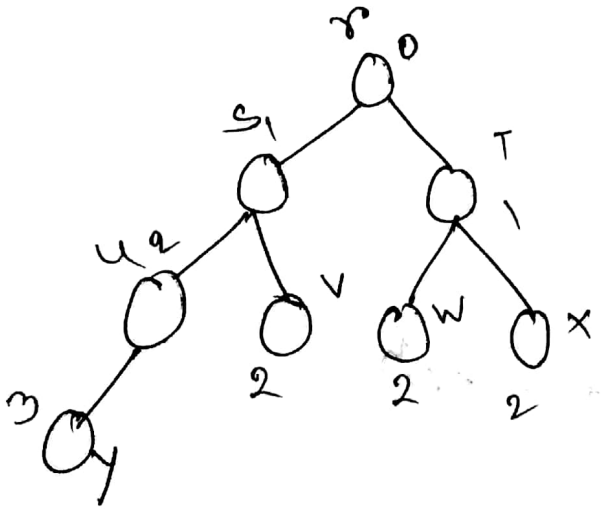
→ যাদের child নাই তারা external node. এরা leaf.

→ A complete binary tree is a binary tree that has all possible nodes in each level except the last level and the last level is completed from left to right

→ Ancestry | ancestor → পূর্বপুরুষ

successor → উত্তরসুরি

root → সবার ancestor



edge.
r-s s-u u-y

Path → a collection of edges.

একটা edge এর কোন পাশের edge এর সাথে only একটা path.

Path length = Number of edges.

Path cost =

⇒ Depth হল root point এ node থাকলে তা কত level পাবে।
থাকবে।

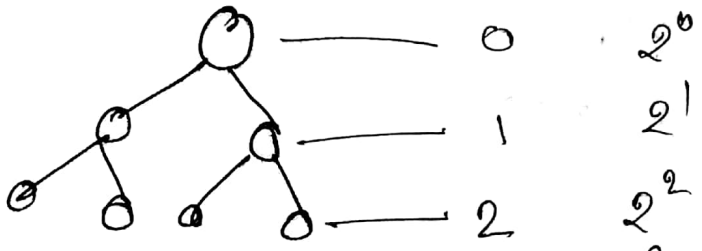
Depth → length of the path from root to a node.

Height → maximum depth.

level → প্রত্যেকটা depth হল একেকটা level.

→ Binary tree with height = 2

→ What is the maximum number of nodes possible in that tree?



2^0
2^1
2^2
2^8
2^h

max ⇒ 2.

Now, height → h

$\therefore$ max → $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \cdots + 2^h$

$= 2^{h+1} - 1$

height → 32 হলে।

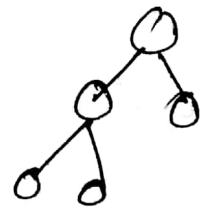
$h = 2^{23+1} - 1$

$= 2^{33} - 1$

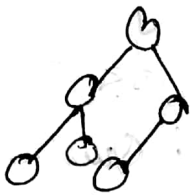
$=$

→ last level বাদ দিয়ে বাকি level complete হলে they are complete heap।

→ last level বাদ দিয়ে complete হলেও left to right complete থাকতে হবে। left থেকে কোন node বাদ দিয়ে right এ যাবে না।

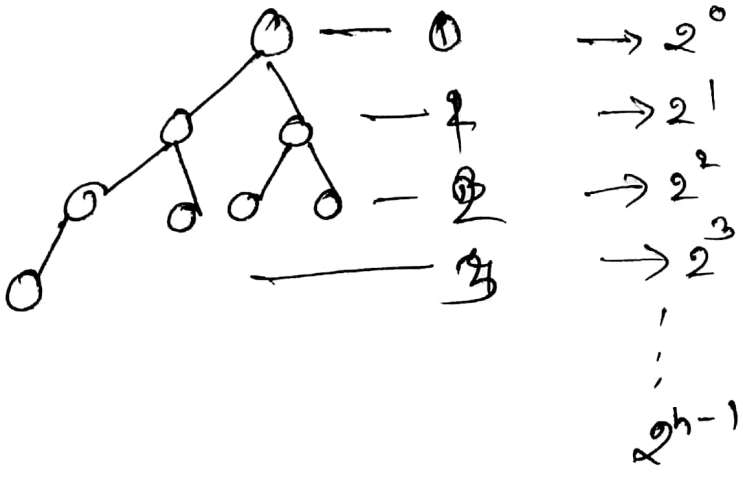


complete



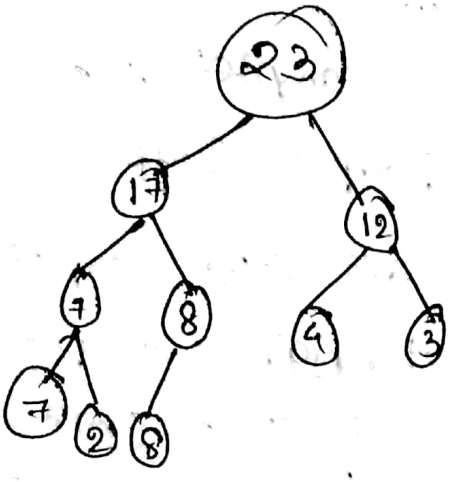
incomplete

Suppose, there's a complet binary tree with height=h=3
what is the minimum number of nodes possible?



$\therefore S = 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{h-1} + 1$

$= 2^h$



→ ~~either~~ either তার parent child এর চেয়ে বড়

→ OR তার parent child এর চেয়ে ছোট ;

→ Assignment -2

→ mid এ আসবে Last week.

→ mid Question.

	0	1	2	3	4	5	6	7	8	9
A	37	31	23	13	27	7	6 23	7	1	10

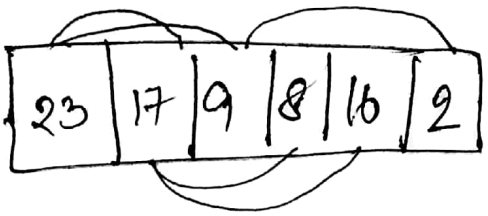
→ heapsize হলো heap এ কয়টা element আছে,

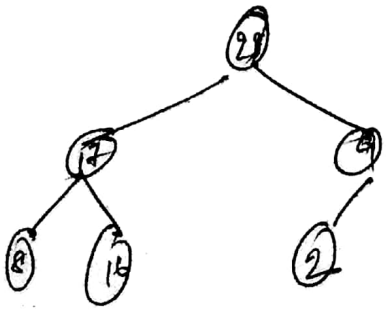
→ Array size হল বড় ।

left → $2*i+1$

right → $2*i+2$

parent? $\lceil i/2 \rceil - 1$





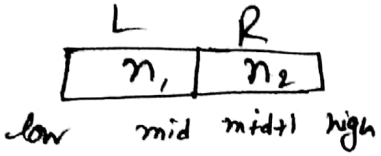
15/02/16

low=0 7=high

23 17 12 3 10 5 4 19

Merge:

$T(n) = a_1 n_1 + a_2 n_2 + a_3 \overbrace{(n_1 + n_2)}^{n} + a_4$




```
void mergesort (int A[], int low, int mid, int high)
{① if (low < high)
   {② int mid=(low+high)/2 ;
    ③ mergesort(A, low, mid);
    ④ mergesort (A, mid+1, high);
    ⑤ merge sort (A low, mid, high);
    }
}
```

runtime →

$T(n) = c_1 + c_2 + T(n/2) + T(n/2) + an + b$

$= 2 \cdot T(n/2) + an + b'$ $[\because c_1 + c_2 + b = b']$

1) $T(n) = 9T(n/3) + n$

2) $T(n) = T(2n/3) + 1$

3) $T(n) = 7T(n/2) + \Theta(n^2)$

4) $T(n) = 9T(n/3) + n^3$

→ division হল b
→ multiply না,

For merge sort:

$a = 2 ; \quad b = 2 \qquad f(n) = an + b^1$ ... 1

$n^{\log_b a} = n^{\log_2 2} = n^1$ ↕ compare

1

⇒ So same তাই tight band.

$\therefore T(n) = \Theta(n \log_2 n)$

Ans:

(1) $a = 9 ; \quad b = 3 \qquad f(n) = n^1$

$n^{\log_b a} = n^{\log_3 9} = n^2$

⇒ বড় একটার চেয়ে আর একটা,

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

(2) $a = 1 ; \quad b = 3/2 \qquad f(n) = 1$

$n^{\log_{3/2} 1} = n^0 = 1$ ↙ equal

$\therefore T(n) = \Theta(\log_2 n)$

(3) $a=7$; $b=2$ $\quad f(n)=\Theta(n^2)$

$n^{\log_2 7} = n^{2.8}$

$\therefore T(n) = \Theta(n^{2.8})$

(4) $a=9$; $b=3$ $\quad f(n)=n^3$ বড়

$n^{\log_3 9} = n^2$ ছোট

(case 3)

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

→ a = কাজ কতবার হবে

→ b = কাজ কত ভাগাভাগি করে

→ f(n) = আর extra.

Master Theorum:

→ For any recurrence of the from,

$T(n) = aT(n/b) + f(n)$; $a \geq 1$ → কত বার recursion call

$b > 1$ → array কতভাগে ভাগ

Then any of the following cases might apply: হতে,

(1) $f(n) = O(n^{\log_b a - \epsilon})$ হলে, $T(n) = \Theta(n^{\log_b a})$ $f(n)$ ছোট

(2) $f(n) = \Theta(n^{\log_b a})$ হলে, $T(n) = \Theta(n^{\log_b a} \cdot \log_2 n)$ Equal

(3) $f(n) = \Omega(n^{\log_b a + \epsilon})$ হলে, $T(n) = \Theta(f(n))$ $f(n)$ বড়

$\log_{10} 50 = ?$ $1 < ? < 3$

→ algo তে log এর কোন base দেয়া না থাকলে তা always 2.

$10^2 = 100$ ① $\log_{10} 100 = 2$

⇒ 10 এর কততম power হল

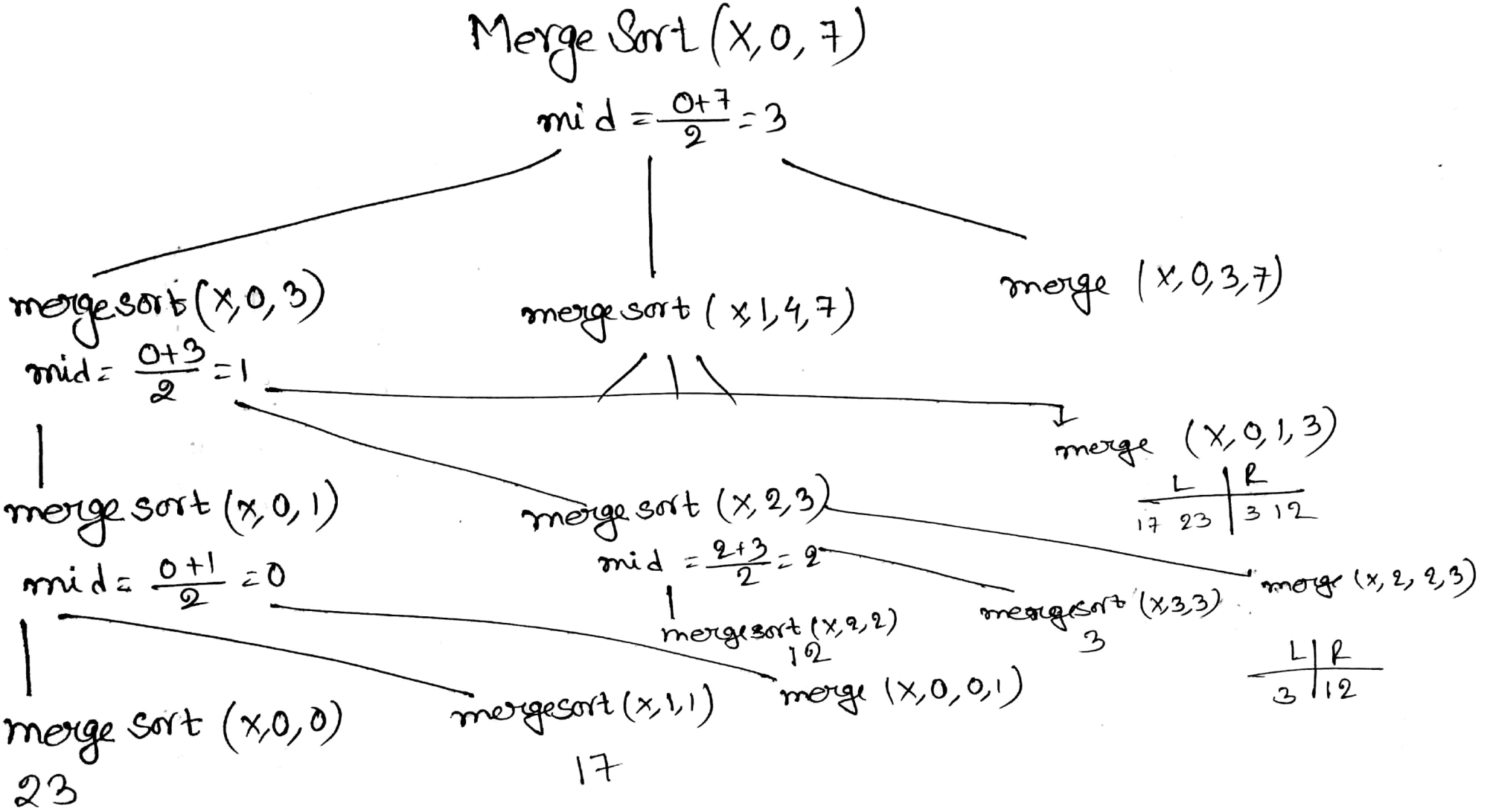
$100 = 2$ So 2.

$a^x = n$ ② $\log_a a = 1$

$\log_a n = x$ ⇒ base আর মান সমান হলে ans 1.

③ $\log 0 = 1$

⇒ কোন কিছুর উপর 0 হলে তার মান 1.



24-02-2016

Makup Class

Thursday (25-02-2016)

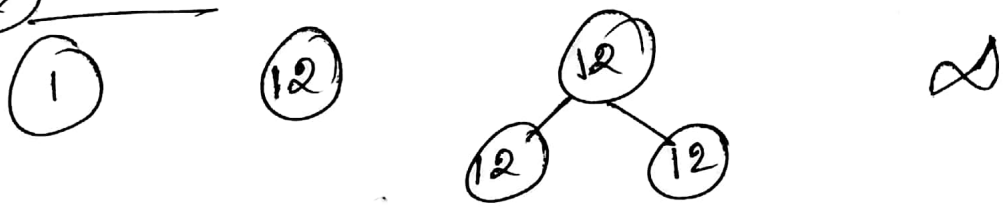
9:30 A.m - 10:50 A.M

R# 1308

Q1 (1) Give Example of a binary tree that is both max heap and min heap? How many trees?

(2) Is it possible for a max heap if be a binary search Tree.

Ans:

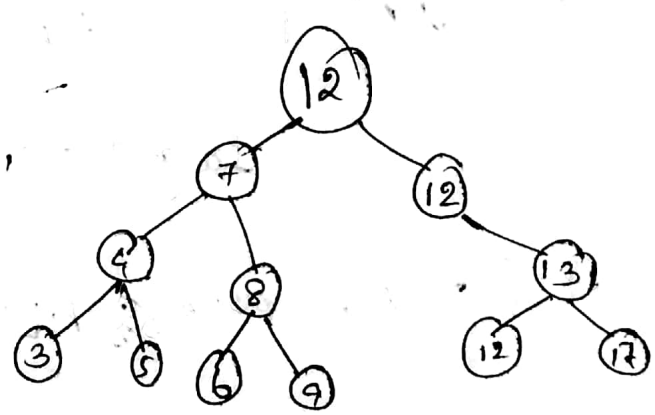


∞

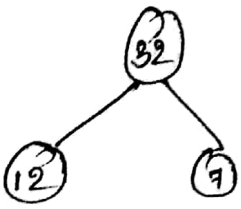
(2) A kind of binary tree,

বড় / সমান → যান দিক,

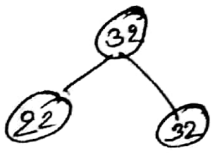
ছোট / সমান → বাম দিক



binary tree max heap.



~~bo~~ max heap
not b.t



max heap
also b.t

→ max heapify because parent থেকে হলে তাকে Sort করে থেকে নিচে নিয়ে যায় এবং parent কে all time বড় রাখে,

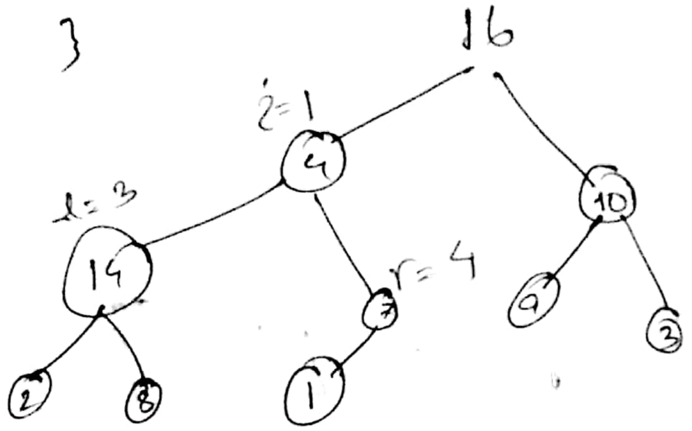
```
void Max heapify (int A[], int heapsize, int i)
{
    int l = 2*i+1;
    int r = 2*i+2;
    int largest
    if (l < heapsize && A[l] > A[largest])

        largest = l;

    if (r < heapsize && A[r] > A[largest])

        largest = r;
```

```
    if ( largest != i )
    {
        swap ( A, i, largest ) ;
        max heapify ( A, heapsize, largest );
    }
}
```



→ heapsize এর চেয়ে index কম হতে হবে তবে child. আছে বুঝাবে.

runtime :

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

$$= c \cdot h$$

$$= c \cdot \log_2 n$$

Find the runtime of the following:

1. Insertion in a sorted Array $O(n)$
2. Deletion in a Sorted Array $O(n)$
3. Update in a Sorted Array $\Theta(n)$
4. Insertion in a Sorted Linked List $O(n)$
5. Deletion in a Sorted Linked List $O(n)$
6. Update in a Sorted Linked List $O(n)$

Priority Queue

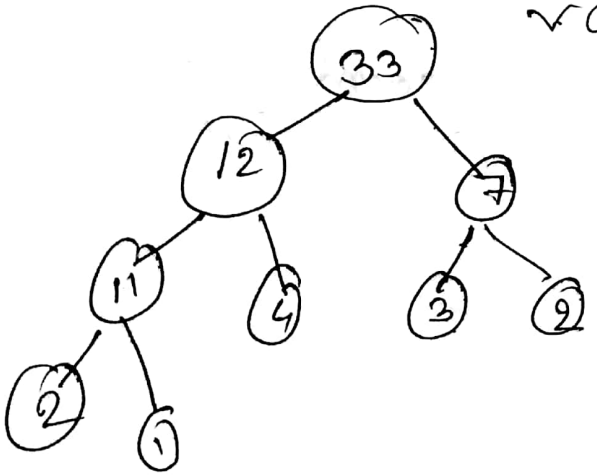
→ insert at any order

→ Deletes the largest/smallest

→ Max Priority Queue
→ Min Priority Queue

→ heap

Update



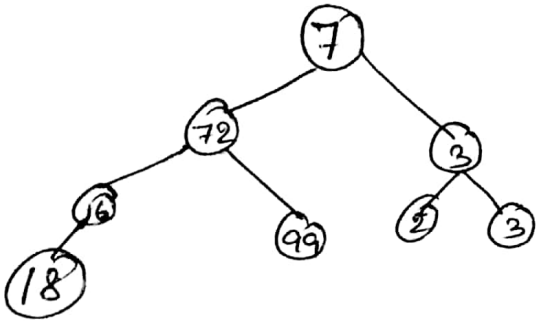

```
void Update(int A[], int heapsize, int i
                                   int new value)
{
  if (new value < A[i])
  {
     A[i] = new value;
     maxheapify(A, heapsize, i);
  }
  else
  {
     A[i] = new value i
     int p = ⌈i/2⌉ - 1;
     while( i > 0 && A[i] > A[p])
     {
        swap(A, i, p);
        i = p;
        p = ⌈i/2⌉ - 1;
     }
  }
```

$T(n) = \log_2 n$.

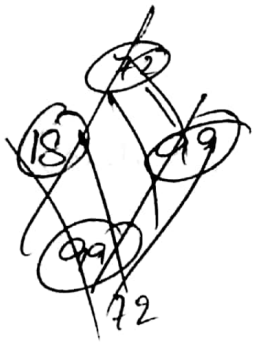
25/02/2016

How many swap operations are required to convert the following array into a max-heap?

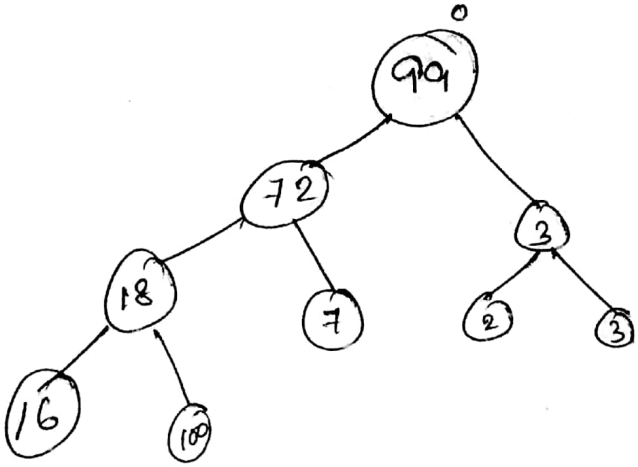
7 72 3 16 99 2 3 18



Ans: 4



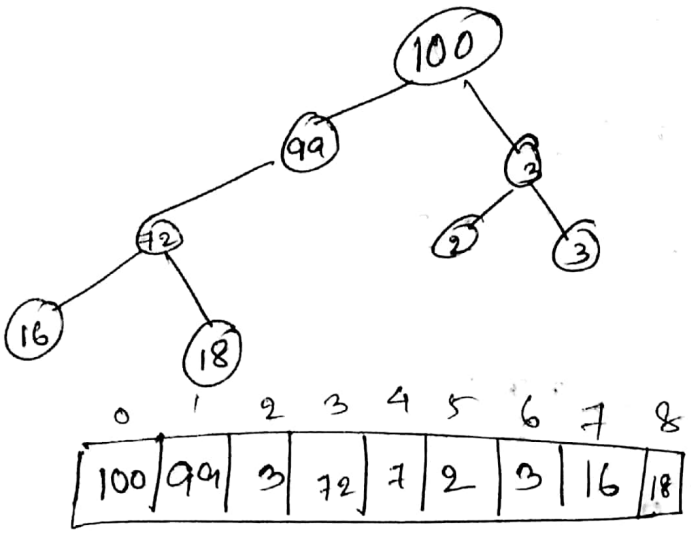
	Heap	Array / LL
(1) Update	$O(\log_2 n)$	$O(n)$
(2) Insert	$O(\log_2 n)$	$O(n)$
(3) Delet	$O(\log_2 n)$	$O(n)$



0	1	2	3	4	5	6	16	8
99	72	3	18	7	2	3	16	100

Insert

```
void insert (int A[], int heapsize, int new value)
{
  int i = heapsize;
  A[i] = new value;
  heapsize ++;
  int P = ⌈ i/2 ⌉ - 1;
while (i>0 && A[i] > A[P])
{   swap (A, i, P);
      i = P;
    P = ⌈ i/2 ⌉ - 1;
}
```



→ সবচেয়ে বড়টাকে Delet করব। 0 position।
→ heapsize 1 কমায় দিব।
→ last element কে 1st এ বসায় দিব।

Delet

```
int removemax (int A[], int heapsize)
{
    int max = A[0];
    A[0] = A[heapsize - 1];
    heapsize --;
    max heapify (A, heapsize, 0);
    return max;
}
```

Classtest -2
Wednesday
02 March, 2016

- Heap sort
- Priority Queue.
- Merge Sort
- Master Method.

Selection Sort

```
void somealgo (int A[], int length)
{
    for (int i=0; i<length-1; i++)          // 1, n, n-1
        for (int j=i+1; j<length; j++)      // n-1, n(n+1)/2 - 1, n(n-1)/2
            if (A[i] > A[j])                // n(n-1)/2
                swap (A, i, j);
}
```

best case: $O(n^2)$

worst case: $O(n^2)$

→ almost sort or sorted array হলে এটা হলে insertion sort ভাল use.

insertion sort

```
void anotheralgo (int A[], int len)
{
    for (int i=1; i<length; i++)            // 1, n, n-1
    {
        int key = A[i];                     // n-1
        int j = i-1;                        // n-1
        while (j>=0 && A[j] > key)          // n-1, n(n+1)/2 - 1
        {
            A[j+1] = A[j];                  // n(n+1)/2
            j--;
        }
        A[j+1] = key;                       // n-1
    }
}
```

→ i এর value যত বার হয় j এর value worst এ তার থেকে কম হয়,

best case: $O(n)$

worst case: $O(n^2)$

STL ও ২টা priority queue

↳ Sets (print)
→ priority queues. (only push/pop)

Greedy

① → coin change –
② → Activity change
③ → Fractional knapsaive –
④ → Egyption fraction.

~~so~~ ① 50¢ , 25¢ , 10¢ , 5¢ 1¢

change = 37¢ minimum number of coins.

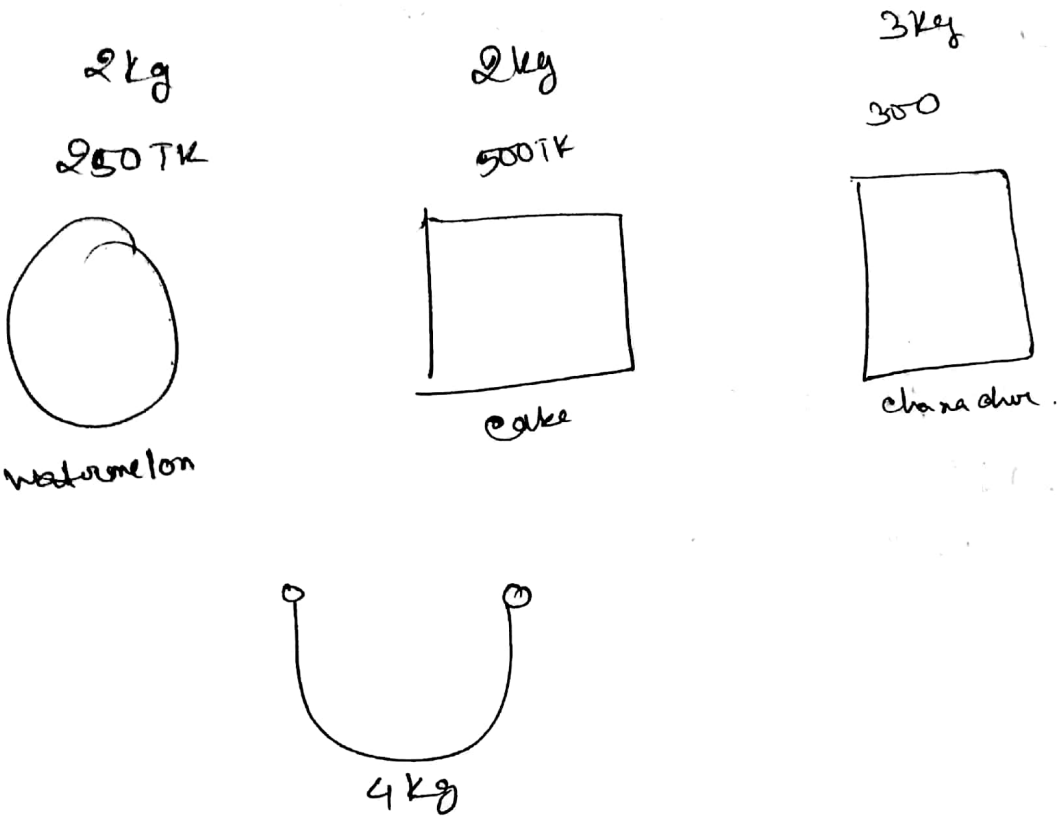
11¢ ← 11 × 1 1¢ = 11
1 10¢ × 1 1¢ = 2
2 5¢ × 1 1¢ = 3
1 5¢ × 6 1¢ = 7

Given a chang amount you have to decide which coins to give to the customer such that the total number of coins is minimized)

#

```
greedy knapsack (int w[], int v[], int W, int n, float q[])
{
    float  price = 0;
    float  load = 0;
    int  i=0;
    while (load < W)
    {
       if w[i] ≤ W-load
       {
          q[i] = 1.0;
          price += v[i];
          load += w[i];
       }
      else
       {  q[i] = (W-load) / w[i] ;
          load = W;
          price += v[i] * q[i];
       }
    i++;
    return price;
}
```

Fractional knapsack



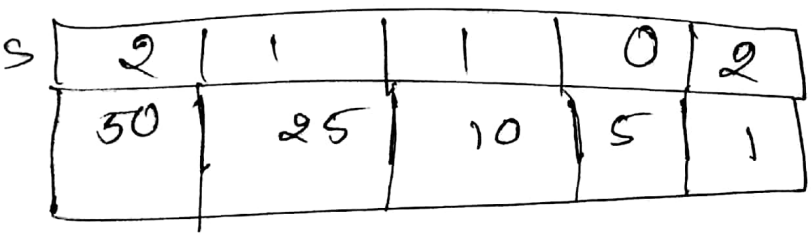
n items

W[ ] weight

V[ ] value price (~~optimal~~ unique price)

Ans: 1

```
int greedy coinchange (int amount, int c[], int n, int s[])
{
    int num=0;
   int i=0;
   While (amount >0)
   {
        s[i] = amount / c[i];
        num+ = s[i];
        amount = amount % c[i];
         i++;
   }
  return  num;
}
```



137
%
50
37/25
12/10
2/1.
1/1

Room # 1308

		overlap / conflict	maximize number of activities
A1	1 - 4		
A2	2 - 5	X	
A3	3 - 6	X	
A4	4 - 6		
A5	5 - 7	X	
A6	2 - 8	X	
A7	6 - 8		

A1, A7

A2 A5

A1 A4 A7

greedy AS(int s[], int f[], int n, int jobs[]);

greedines -

earliest finishing time

```
}
int i=0;
jobs[i] = 1;
int lastend = f[0];
while (++i < n)
{
    if (s[i] > lastend)
    {
        jobs[i] = 1;
        lastend = f[i];
    }
    i++;
}
```

29.02.16

(1) minimum করা।

Suppose you have to give an amount of change of 297 cents. You have following coins.

50 ¢, 25 ¢, 10 ¢, 5 ¢ & 1 ¢. Now decide which cents to give and how many in such a way so that total number of coins is minimized.

(2) maximized করা।

Consider the fractional knapsack problem.

item	1	2	3	4	5	6
weight	2	3	2	4	3	2
value	8	6	5	10	15	4
unit price	4	2	2.5	2.5	5	2

Suppose, knapsize = 6 kg, Decide which items to take and how much?

n

Ans: 1

C(n): minimum number of coins to give for an amount of n.

297

~~50~~

~~247~~ 47

~~25~~

22

~~10~~

2

~~5~~

2

1

0

50 / 5 | 25 / 1 | 10 / 2 | 5 / 0 | 1 / 2

$$q[i] = \frac{W\text{-load}}{w[i]} * v[i]$$

Ans: 2

item	5	1	3
~~price~~ amount	3	2	1
price	15	8	$\frac{1}{2} \times 5 = 2.5$

30¢ 50 25 10 1

25¢ } 6
1¢×5 }

10¢×3 } 3

Greedy algorithm কাজ করছে না, কেন?

fractional কনফিগুল (0-1) Knapsack দিলে এ কাজ করবে না।

⇒ Divide and conquer

↳ divide
→ Recursively Divide (প্রথম ছোট হয় প্রথম easy to sort)
→ Combine (বড় prob কে ছোট করতেছি)

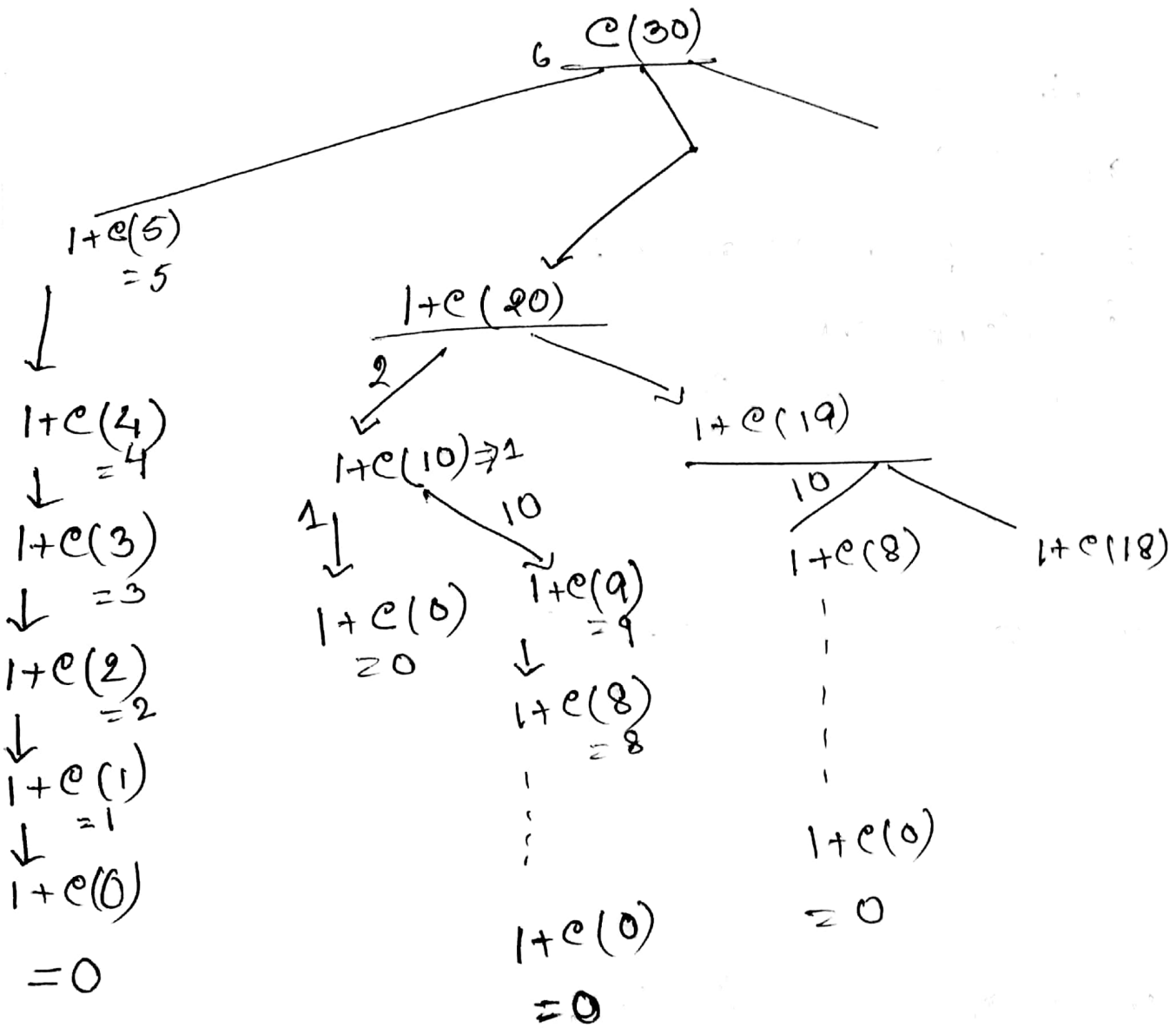
$$c(n) = \min \begin{cases} 1+c(n-50), & \text{if } n \geq 50 \\ 1+c(n-25), & \text{if } n \geq 25 \\ 1+c(n-10), & \text{if } n \geq 10 \\ 1+c(n-1), & \text{if } n \geq 1 \end{cases}$$

$$c(0) = 0$$

$$c(30) = \min \begin{cases} \frac{1}{25¢} + c(5) \\ \frac{1}{10¢} + c(20 \\ \frac{1}{1¢} + c(29 \end{cases}$$

```
int cc (int n)
{
    if (n==0)
     return 0;
    else
    {
     int min = ∞;
     if (n ≥ 50)
      } int temp = 1 + cc (n-50);
         if (temp < min)
            min = temp; }
      }
     if (n ≥ 25)
      } int temp = 1 + cc (n-25);
         if (temp < min)
            min = temp;
      }
     if (n ≥ 10)
      { int temp = 1 + cc (n-10);
         if (temp < min)
            min = temp;
     }
    if (n ≥ 1)
     } int temp = 1 + cc (n-1);
       if (temp < min)
          min = temp;
return min; }
```

C(30)



Recursive Substructure.

↳ বড় problem ছোট কিছুর উপর depend → Dynamic Programming

overlap Substructure.

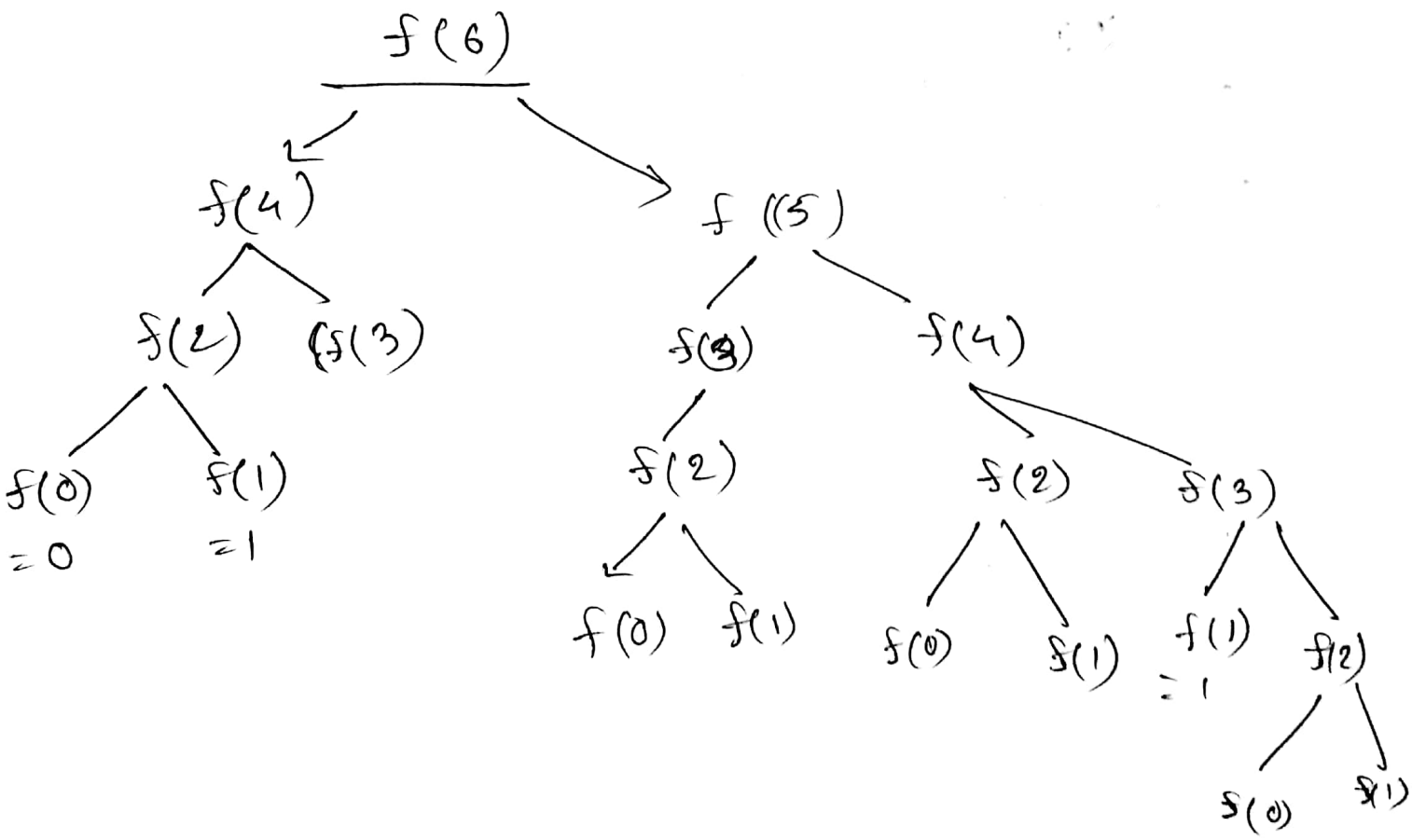
runtime বাচানোর জন্য runtime খরচ করা।

fibonacci

0 1 1 2 3 5 8 13 21 ...

$f(n) = f(n-2) + f(n-1)$ if $n \geq 2$

$f(0) = 0$; $f(1) = 1$



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

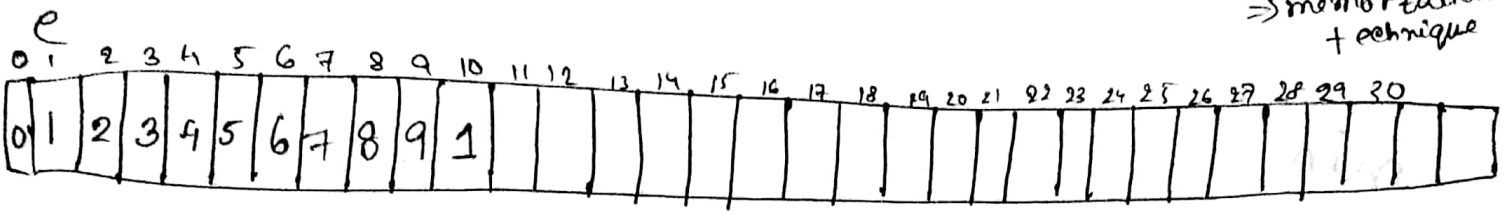
```
int f (int n)
{
    if (v[n] == -1)
    {
      if(n==0)||(n==1)
        return n;
        v[n]=n
    else
        v[n]
        return = f(n-2)+f(n-1);
    }
    return v[n];
}
```

-> (6) Call

recursive call ko yaad store

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

=> memorization technique




```
int cc(int n)
{
    if (c[n]==-1)
    {
        if (n==0)
            c[n]=0;
        else
        {
            int min=∞;

            c[n]=min
        }
        return c[n];
    }
```

02/03/16

CH#3

Wednesday Next

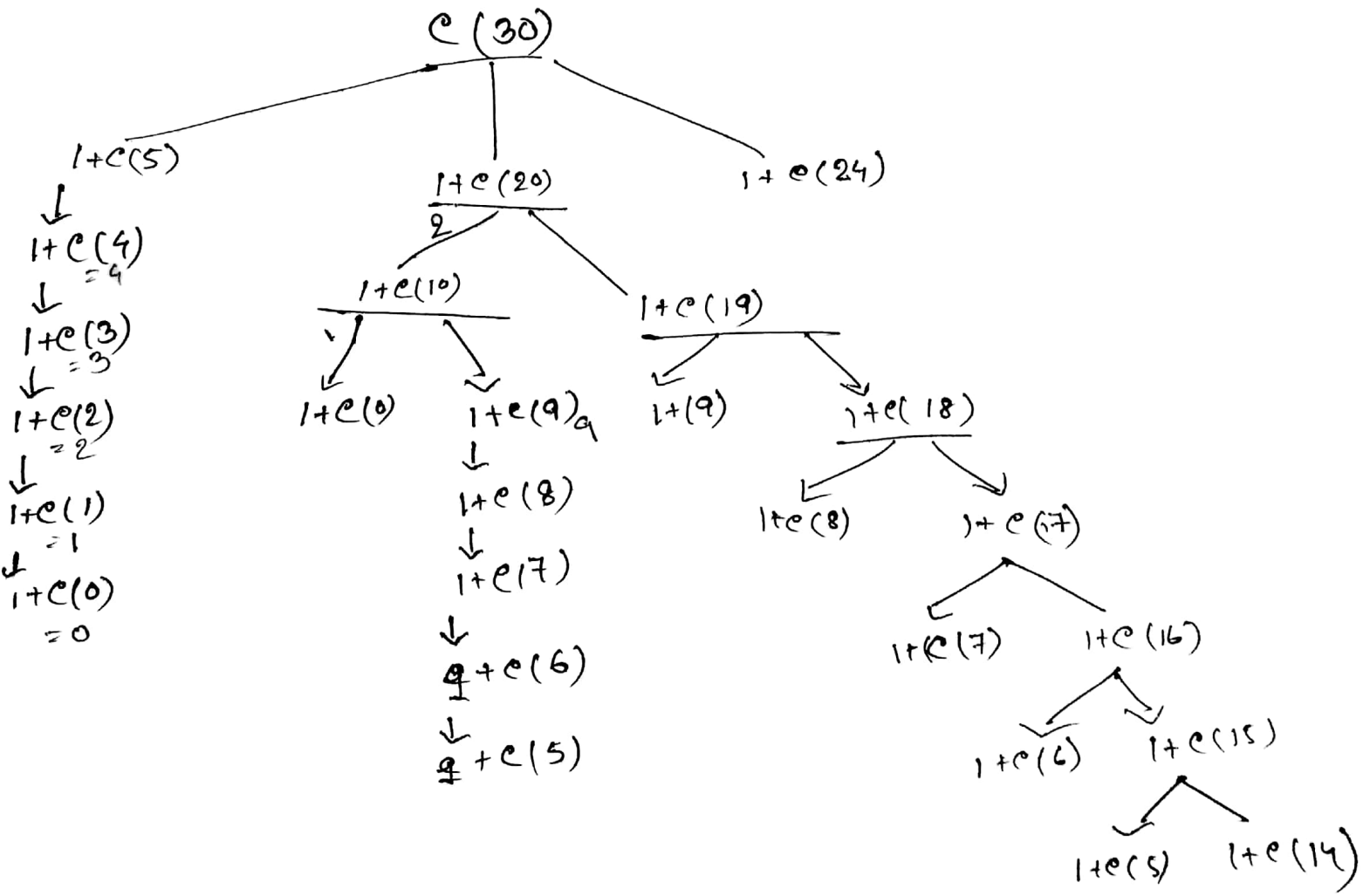
09-03-16

Syllabus - Dynamic Progamms.

→ c[n] হল যেমন মান
→ c[n] কয়টা কয়েন লাগবে।
→ s[n] ⇒ কোন coin থেকে start করবে।

Coins: 1 | 10 | 25 | 50

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
c[n]	0	1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9	10	2	3	4	5	6	1	2	3	4	5	6	3
s[n]	0	1	1	1	1	1	1	1	1	1	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	25	25	25	25	25	25	10



```
int mcc (int n)
{
    if (c[n]==-1)
    {
        if (n==0)
        {
            c[n]=0;
            s[n]=0;
        }
        else
        {
            c[n]=INT_MAX;
            if (n≥50)
            {   int temp=1+mcc(n-50);
                if (temp<c[n])
                {
                    c[n]=temp;
                    s[n]=50;
                }
            }
            if (n≥25)
            {   int temp=1+mcc(n-25);
                if (temp<c[n])
                {
                    c[n]=temp;
                    s[n]=25;
                }
            }
            if (n≥10)
            {   int temp=1+mcc(n-10);
                if (temp<c[n])
                {
                    c[n]=temp;
                    s[n]=10
                }
            }
            if (n≥1)
            {   int temp=1+mcc(n-1);
                if (temp<c[n])
                {
                    c[n]=temp;
                    s[n]=1;
                }
            }
        }
    return c[n];
}
```

c[n]: minimum number of coin

```
void PrintCoins ( int n, int S[] )
}
```

Recursive

```
    if (S[n] != 0)
    {
        printf ("%d cents", S[n]);
        printf(
        PrintCoins (n - S[n], S);
    }
```

Output:

10 cents
10 cents
10 cents

```
void IterativePrintCoins (int n, int S[])
{
    While (n)
    { printf("%d cents", S[n]);
        n = n - S[n];
    }
```

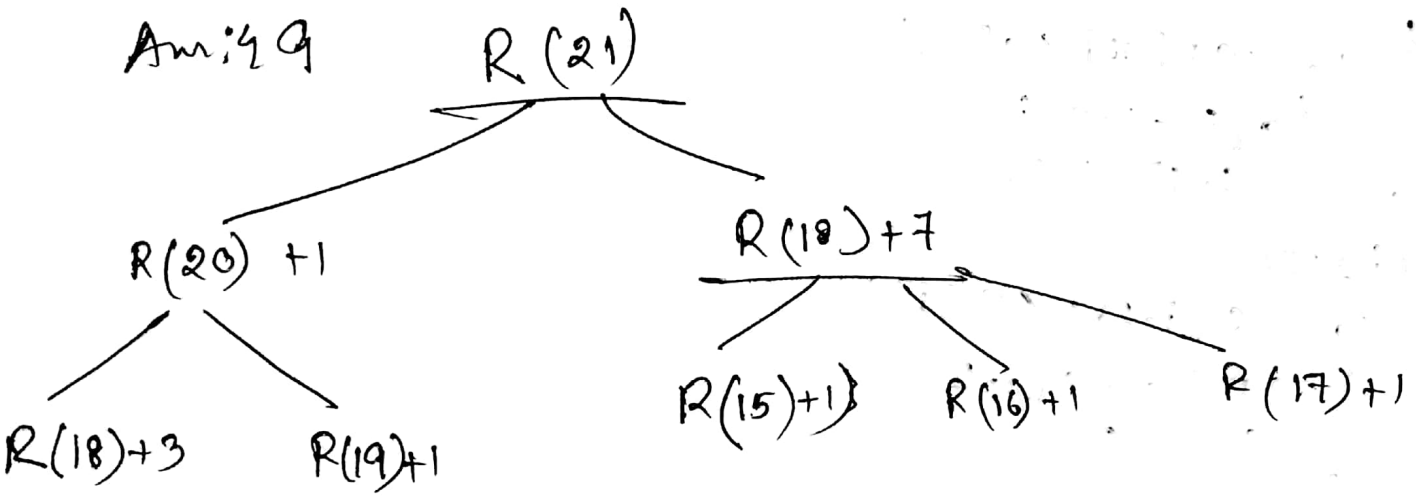
```
int iterativeCoinChangeDP(int n)
{
    c[0]=0;
    S[0]=0;
    for(int i=1; i<=n; i++)
    {
        c[i]=1+c[i-1];
        S[i]=1;
        if(i>=10)
        {
            if(1+c[i-10]<c[i])
            {
                c[i]=1+c[i-10]
                S[i]=10;
            }
        }
        if(i>=25)
        {
            if(1+c[i-25]<c[i])
            {
                c[i]=1+c[i-25]
                S[i]=25;
            }
        }
        if(i>=50)
        {
            if(1+c[i-50]<c[i])
            {
                c[i]=1+c[i-50]
                S[i]=50;
            }
        }
    return c[n];
}
```

04/03/16

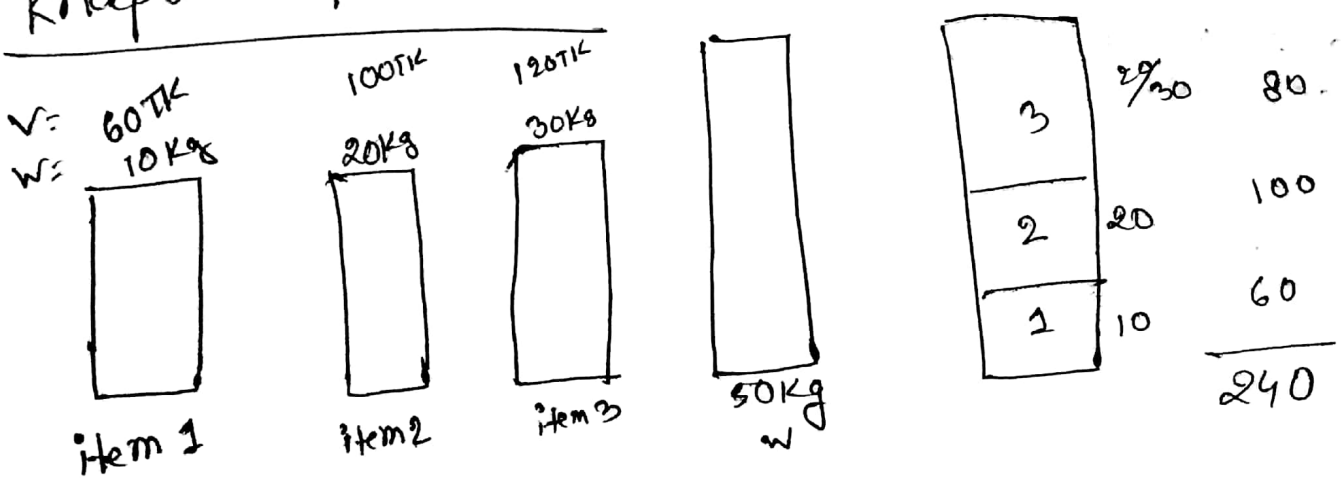
$$R(n) = \max \begin{cases} R(n-1)+1 & \text{if } n\%1 == 0 \\ R(n-2)+3 & \text{if } n\%2 == 0 \\ R(n-3)+7 & \text{if } n\%3 == 0 \end{cases} \quad \text{if } n \geq 1$$

$R(21) = ?$

Ans: 9



Knapsack problem



Solve করা যাবে।

Same problem < 0-1 knapsack → greedy দিয়ে solve করা যাবে না।

$b(n, W)$: maximum value obtained from n items and a knapsack of size W.

(n = Num of item, W = size of knapsack)

item	W[i]	V[i]
1	2	3
2	4	6
3	3	7
4	5	8
5	9	10

W = 10

$b(0, w) = 0$
$b(n, 0) = 0$
→ Termenating

$b(5,10) \rightarrow 10 + b(4,1)$
$\rightarrow b(4,10)$
} ২য় অংশ আর নেব না।

$b(4,10) \rightarrow 8 + b(3,5)$
$\rightarrow b(3,10)$

Suppose ⇒
$b(3,7) \rightarrow 7 + (2,4)$
$\rightarrow b(2,7)$

$b(n, w)$ if $w[n] \le w$

$$\max \begin{cases} v[n] + b(n-1, w - w[n]) \\ b(n-1, w) \end{cases}$$

এই Law follow করবে।

else $w[n] > w$

$b(n-1, w)$

```
int ROI (int n, int w)
    if (b[n][w] == -1)
{   if (n == 0 || w == 0)
        return 0;
    b[n][w] = 0
    else
        if (w[n] <= w)
        return b[n][w] = max(v[n] + ROI(n-1, w-w[n]), ROI(n-1, w));
    else
        return ROI(n-1, w);
}   return b[n][w];
}
```

Ans:

R01 (5, 10)

10

10 + (4, 1)

(4, 10)

(3, 1)

8 + (3, 5)

(3, 10)

(2, 1)

(1, 1)

(0, 1)

= 0

=> কোন কয়টা item এর w, knapsack এর w এর থেকে ছোট হলে তা আরাজরি আরেকটা।

→ জায়গা না নিয়ে আমরা just উপরের মানটা।

W →

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
✓1	0	0	3	3	3	3	3	3	3	3	3
X2	0	0	3	3	6	6	9	9	9	9	9
✓3	0	0	3	7	7	10	10	13	13	16	16
✓4	0	0	3	7	7	10	10	13	15	16	18
5	0	0	3	7	7	10	10	13	15	16	18

আমরা নিচ্ছি

আমরা নিচ্ছি না

1, 3

মানুষের উপর OR 2 এর আছে।

```
int IterativeKnapsack01(int n, int w)
{
    for (int i=0; i<=w; i++)
        b[0][i]=0;

    for (int i=0; i<=n; i++)
        b[i][0]=0;

    for (int i=1; i<=n; i++)
        for (int j=1; j<=w; j++)
        {
            if(w[i]<=j)
                if(b[i-1][j-w[i]]+v[i] > b[i-1][j])
                    b[i][j] = b[i-1][j]
                else
                    b[i][j] = b[i-1][j];
            else
                b[i][j] = b[i-1][j];
        }
    return b[n][w];
}
```

b[i][j]

b[i-1][j]

b[i-1][j-w[i]]+v[i]

item	W[i]	V[i]
1	3	25
2	2	20
3	1	15
4	4	40
5	5	50

	0	1	2	w_3 → 3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	25	25	25	25
2	0	0	20	25	25	45	45
3	0	15	20	35	40	45	60
4	0	15	20	35	40	55	60
5	0	15	20	35	40	55	65

b(5, 6)

b(4, 1)

```
void printItems(int n, int w)
{
    if(n!=0)
    {
        if (b[n][w] == b[n-1][w]
        {
            printf("Item %d not taken", n);
            printitems(n-1, w);
            n--;
        }
    else
        {
            printf("Item %d taken ", n);
            printItems(n-1, w-w[n]);
            n--;
            w-- = w[n];
        }
}
```

⇒ iterative

FINAL

21/03/16

Graph

→ What is Graph?

↳ A pair ; $G = (V, E)$ (V = vertex, E = Edge)

→ V = set of nodes / vertices

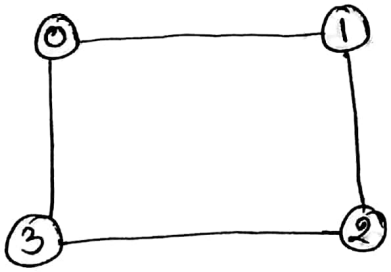
→ E = set of arcs / edges

→ A data structure that represents relationship between elements.

∨

→ (V, E) = Empty হতে পারে।

→ Facebook Friendship



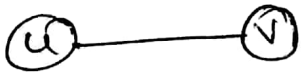
undirected

→ Edge অন্তত ২'টা vertex এর মধ্যে হতে হবে।

$V = \{0, 1, 2, 3\}$

$E = \{(0,1), (1,0), (1,2), (2,1), (2,3), (3,2), (3,0), (0,3)\}$

$= \subseteq (V \times V)$



$(u, v), (v, u)$

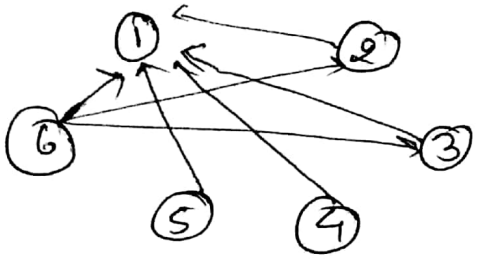
→ সমার্থক ২ দিকের।

integer multiple

একমুখী → Directed

6 is the multiple of 1

1 is not the multiple of 1

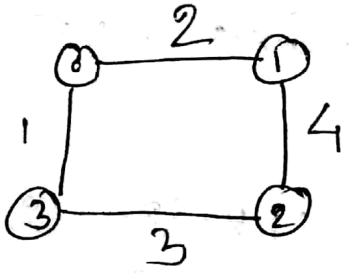


directed

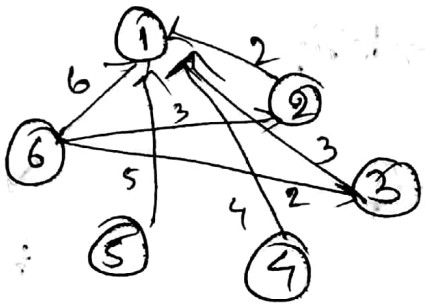
What is weight ?/level

→ real number mapping of edges.

→ একটা edge একটা number.



undirected
weighted



Directed
weighted

$|V| = 10$

minimum = 0

maximum = 100

→ একটা node হতে।

minimum = 0

maximum = ∞

3

$|E|_{max} = V \times V$ → Self-loop থাকে

Graph কত space store?

space

↳ Adjacency Matrix $O(|V|^2)$

→ Adjacency list $O(|V|+|E|)$

→ Adjacency → Neighbour

→ Matrix → 2D Array

→ Adjacency Matrix

row = |V|

colum = |V|

→ row = colum সমান।

→ Adjacency list এর edge।

→ 0 = edge নাই

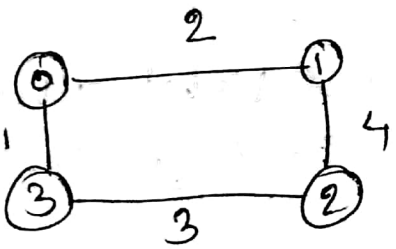
→ edge থাকলে 1।

→ edge এর weight দেয়া থাকলে ঐ box এ edge এর weight থাকবে

→ V

u \ v	0	1	2	3
0	0	2	0	1
1	2	0	4	0
2	0	4	0	3
3	1	0	3	0

Undirected weighted → Adjacency matrix



Degree

→ number of edges.

→ Undirect graph এর edge সমান।

|V| = 10

|E| = 16

integer এর Size = 2

জায়গা কত নিবে ?

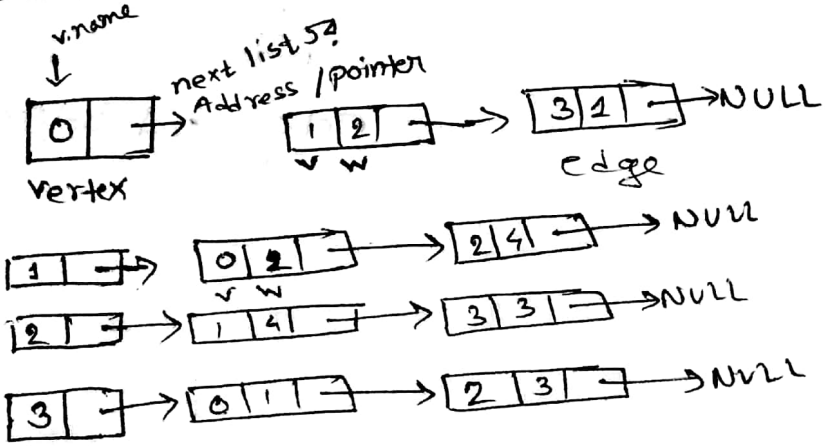
10 × 10 = 100 × 2 = 200 byte.
matrix

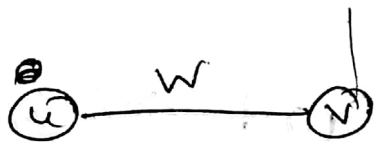
	2i	32i
10i	1P	16P
10P	E	48
V		
20		

68 × 2 = 136 byte.
list

Adjacency List

↳ vertex এর তার সাথে যুক্ত।





→ edge এর 2টা part

→ এইসাথে কে ও সাথে যুক্ত,

→ এই Linked List টাই হল Adjacency list,

4i 4P

2i × 1P × 8

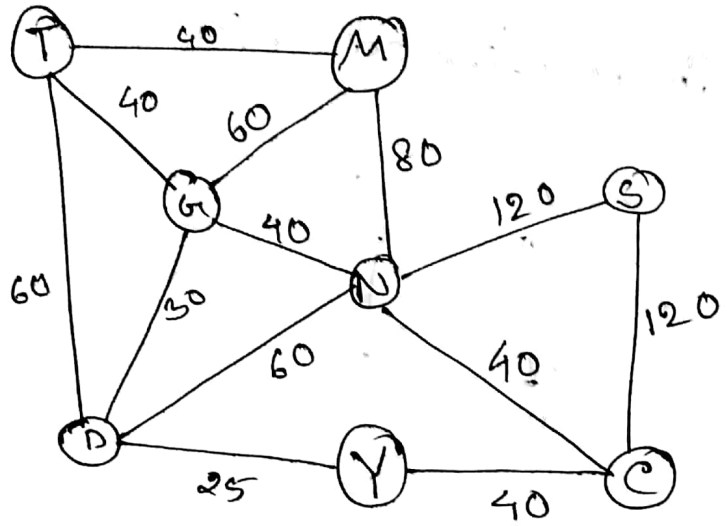
= 16i, 8P

Total ⇒ 20i, 12P

b

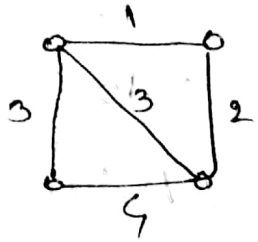
23/03/14

(1)



Road Network

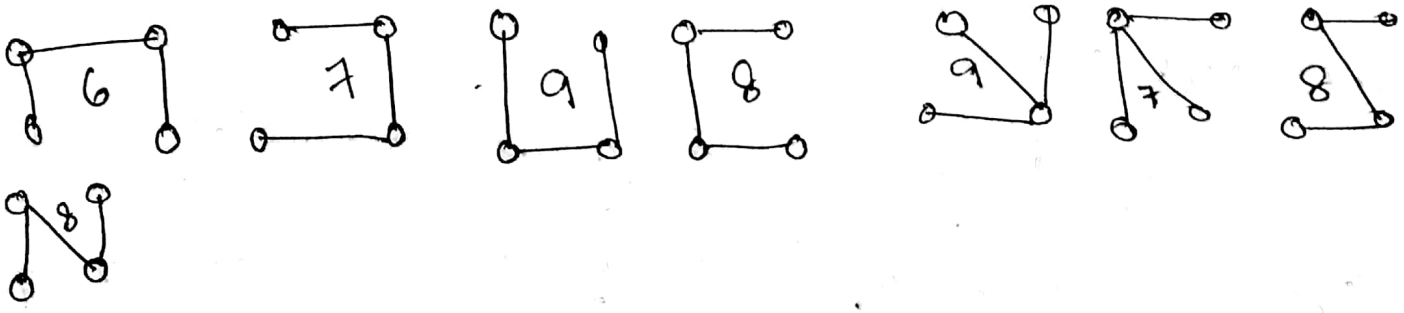
(2)



1) Connected Graph
→ একটা path থাকে atleast

2) Minimum Cost

(3) 4টা Node ... 3টা edge হলেই হবে।



$T = (V, E')$

$E' \subseteq E$

→ Spanning Tree
↳ acyclic ∧ Subgraph
connected

→ এদের মধ্যে সবচেয়ে ছোট যে (cost/weight) দিয়ে তা হল minimum Spanning Tree. (MST)

→ সব V থাকবে but সব E থাকবে না,

6

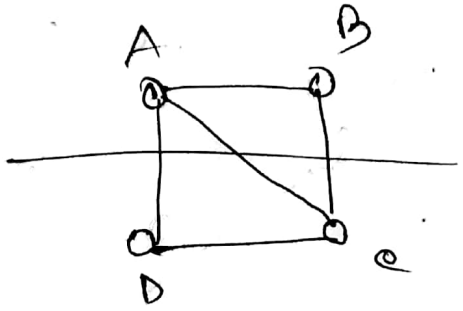
PRIM + KRUSKAL

both greedy, both minimized করে।

$V = \{A, B, C, D\}$

$S = \{A, B\}$

$V - S = \{C, D\}$



→ Cut of a graph $G=(V, E)$ is a disjoint partion of the vertices $S, V-S$।

Crossing Edge

→ যারা একটা cut এর ২ পাশের edge।

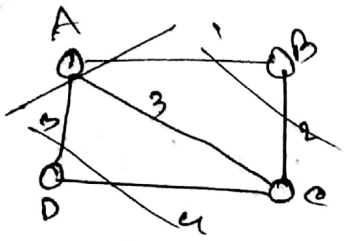
→ এই পাশের edge এর সাথে এ পাশের edge এর মধ্যে cut পরলে।

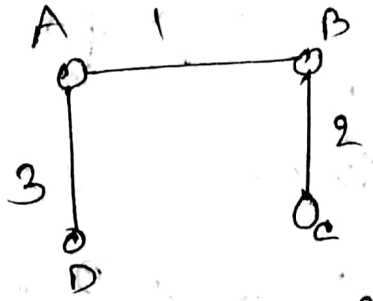
→ Crossing Edge বাদ দিলে graph connected হবে না

→ Crossing Edge এর মধ্যে যেইটা কম (cost/weight) এইটা নিব।

→ minimum edgeটা হল → Light edge.

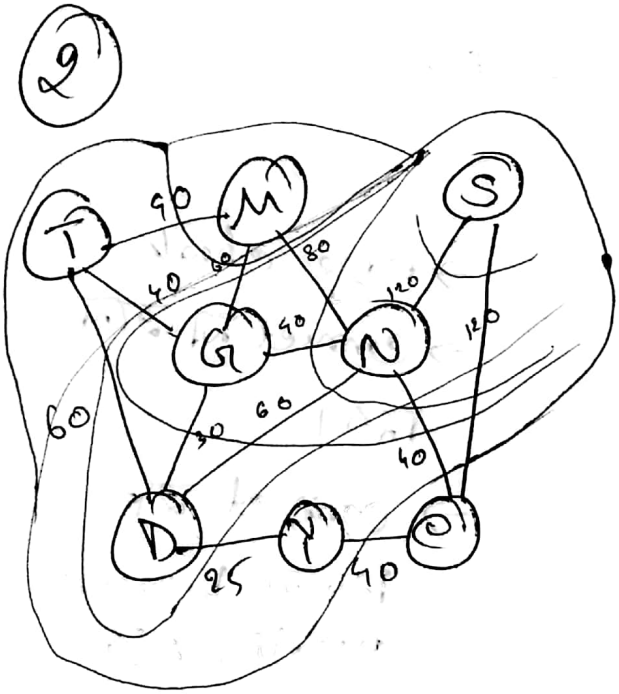
1

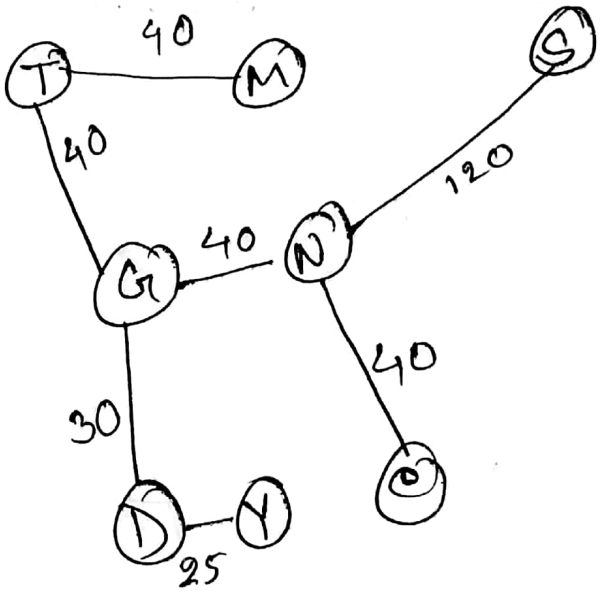




minimum spanning tree.

→ cut (মুছে) এবং minimum গাছ।





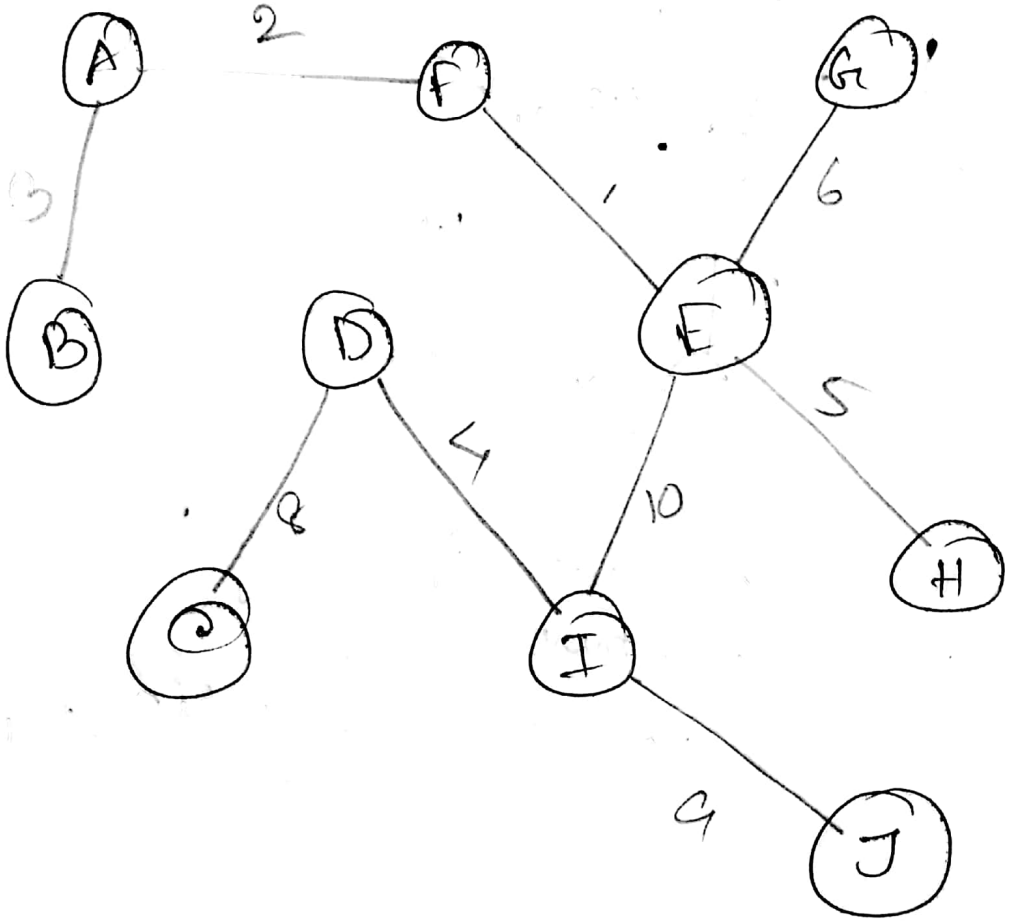
→ minimum spanning tree
→ minimum cost
→ connected acyclic graph

→ is MST of a graph unique?

→ Ans: NO. Because এমন কিছু edge থাকে যাদের edge একই হতে পারে। So MST তখন unique না।

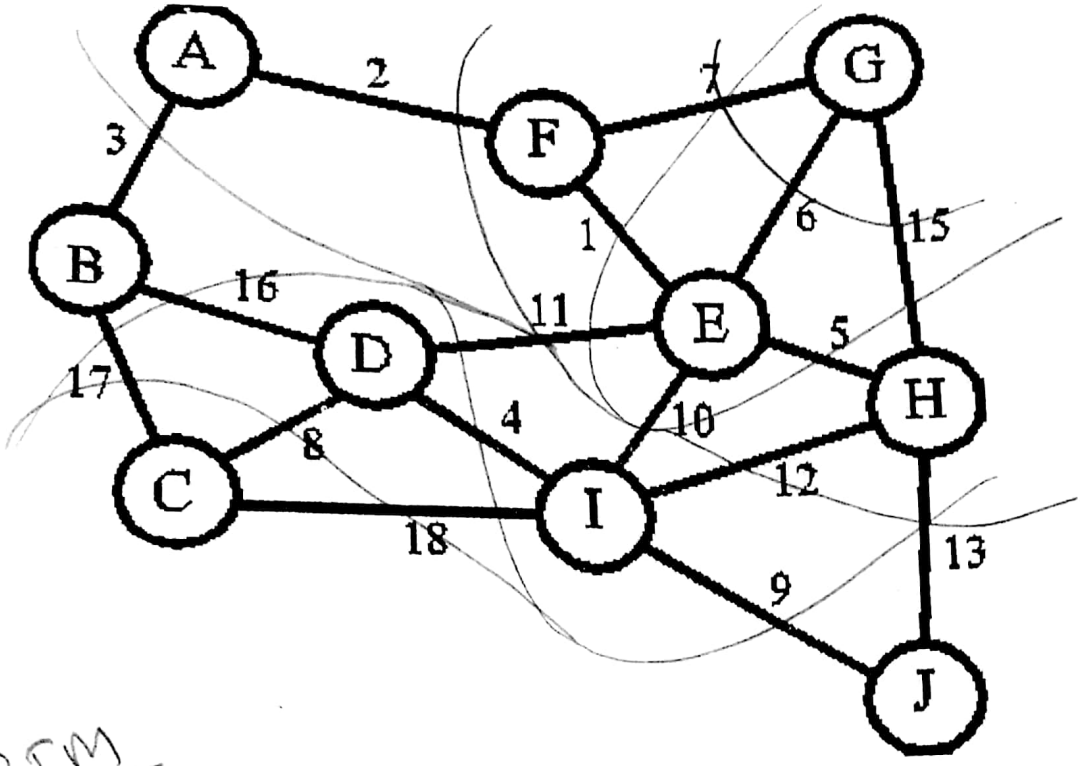
→ KRUSKAL is more greedy.

→ Sorted oreder এ নেয়া হল KRUSKAL

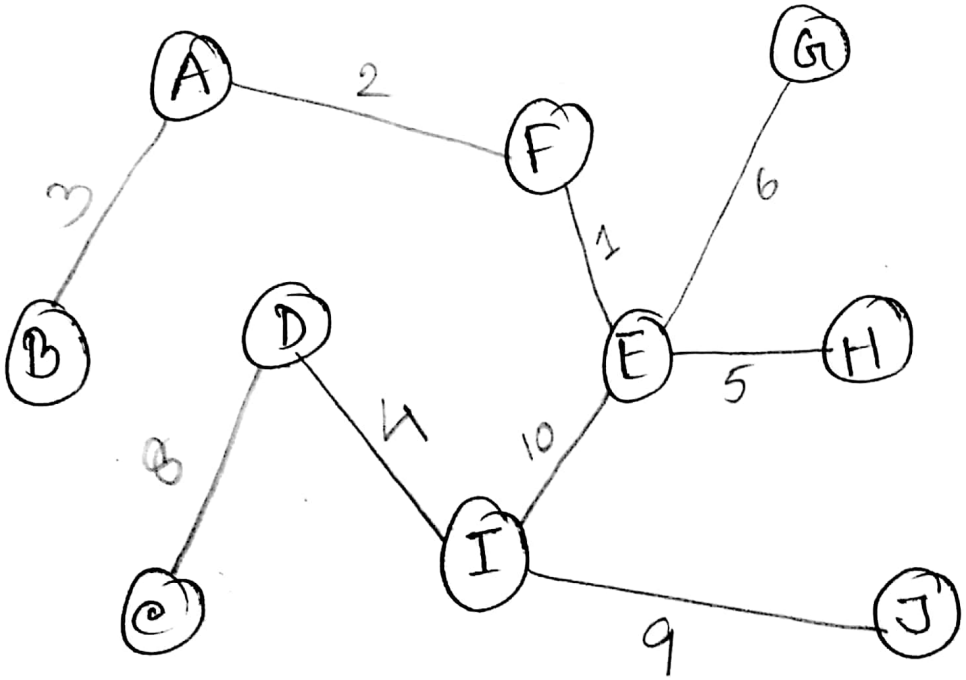


→ edge cheak
→ connect না হলে নিব।
→ connect হলে নিব না because cyclic হবে।

→ Safe edge যা connect করে এবং যাতে (যাতে connected না, mean to cyclic হবে না।



PRIM

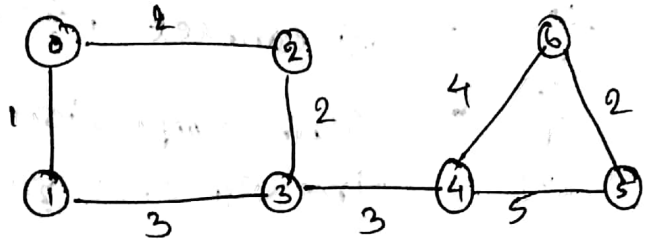


3+2+8+4+10+1+6+5+9

COST → 48

10 Minimum Spanning Tree (MST) 4/4/16

Give a graph $G = (V, E)$



* nodes are some entity in a network.
  V
* edges are communication line.
  E
* weights are repair costs.

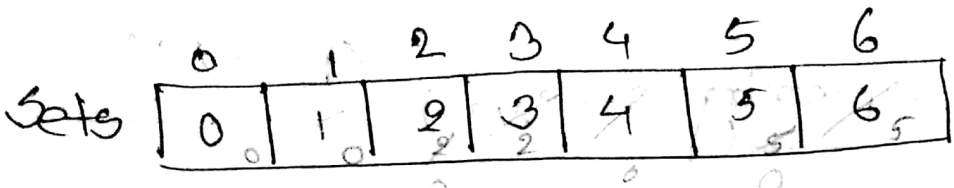
$T = (V, E')$ Such That (1) T is connected, acyclic

$E' \subseteq E$ (2) cost of $T = \sum w(u, v) \in E$ is minimized

→ connected components

Disjoint sets (S)

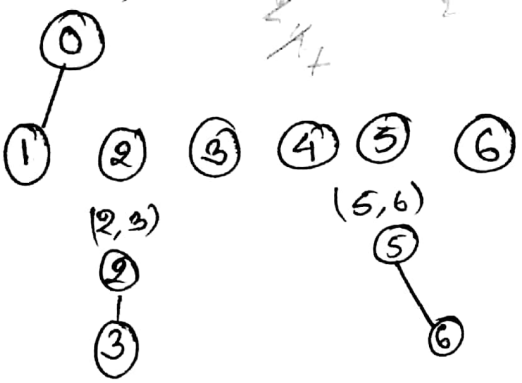
→ যারা একই সময়ে connected তারা একই set-এ.
↳ union sets (দুইটা আলাদা হবে)
↳ find set (কে কোন set এ)



→ লক্ষ্য থাকবে আমি কোন Set এর member।

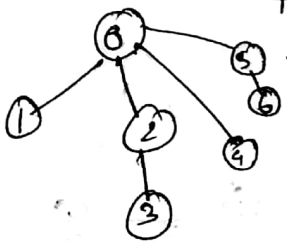
→ Parent একই হলে they are connected so আর connect করব না।

(0,1)

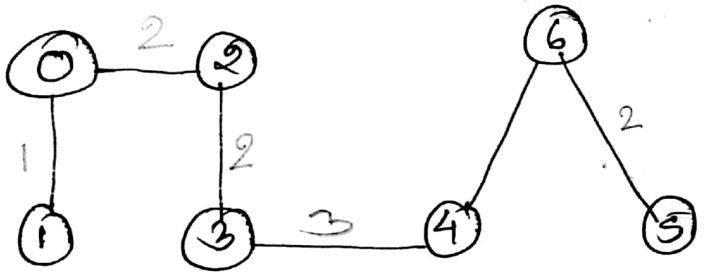


(2,3)

(5,6)



→ Path compression → tree এর size ছোট করা।



↔ mst cost

```
int size[NV];
int sets[NV];

void makesetAll()
{
    for(int i=0; i<NV; i++)
    { set[i]=i;
      size[i]=1;
    }
}
```

O(|V|) linear

```
void unionsets(int u, int v)
{
    int s1 = findset(u);
    int s2 = findset(v);
  if( s1 != s2)
  {
    if(size(s1) < size[s2])
    { sets[s1]=[s2];
      size[s2] += size[s1];
    }
  else
    { sets[s2]=[s1];
      size[s1] += size[s2];
    }
  }
```

O(log|V|)

Tree এর root টাকে return করে

```
int findSet(int v)
{
  if(v != sets[v])
  {
    sets[v]=findsets(sets[v]);
  }
  return sets[v];
}
```

O(log|V|)

→ Amortized technique

↳ কিছু কাজ প্রথমে বেশি করে রাখে পরে কাজ কম করতে হয়।

1. c_1

2/3. $(c_2 + c_3)|V|)$

4. $|E| \log_2 E|$

5. $|E|$

6. $|E) \log_2 |V|$

7. $|E|$

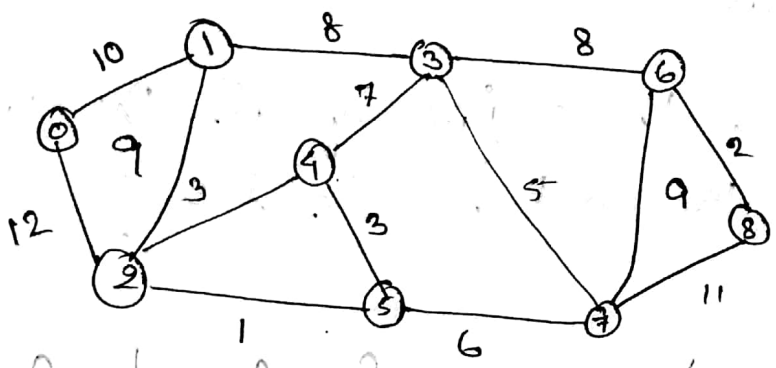
8. $|E| \log_2 |V|$

$\therefore T = |E| \log_2 |E| + |E| \log_2 |V|$

12

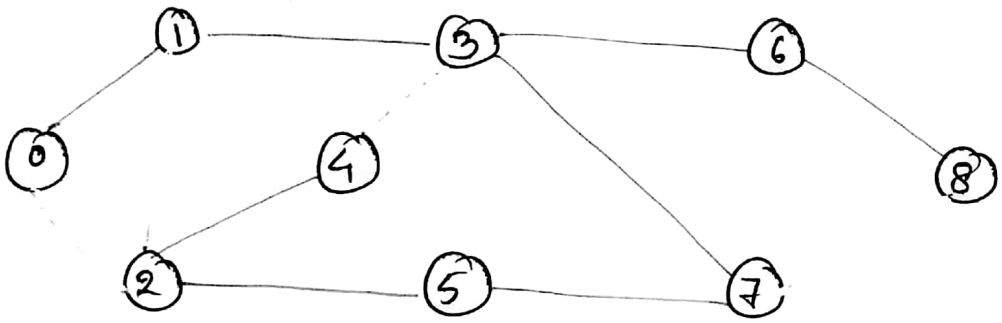
KRUSKAL

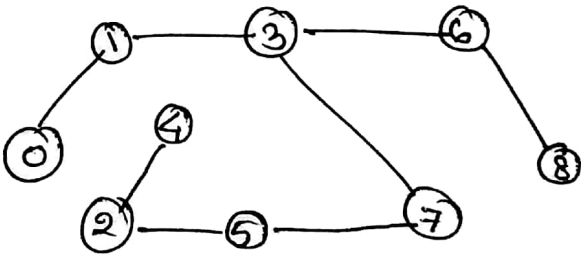
14

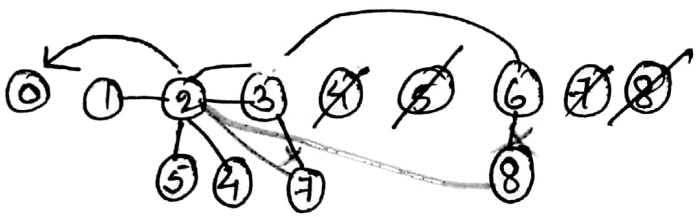


	0	1	2	3	4	5	6	7	8
S.No	0	~~1~~ 2	2	~~3~~ 2	~~4~~ 2	~~5~~ 2	~~6~~ 2	~~7~~ 3	8
Size	1	1	~~1~~	~~1~~ 2	1	1	1	1	1

~~2~~ ~~3~~ ~~5~~ 9







Suppose, you have a graph with 100 vertices. The cost of the MST is 600.

Now, if you increase the weight of all the edges by 2, what will be the cost of the MST?

Ans:

100 টার জন্য 99 টা edge.

So 2 করে বাড়ে আর 198 টা বাড়বে.

So MST = 798.

15

06/04/16

16

Prim algo data structure :

1) Q, Priority Queue (min)
2) Key, Array (crossing edge, weight)
3) π, Array (Parent / Predecessor)

আমার আগে কে

Prim's Algo

· Runtime ·

```
PRIM (V, E, w, r)
  Q = ∅                                          ——— c
  for each u ∈ V                                 ——— |V|
     key[u] ⇐ ∞
     π[u] ⇐ NIL
  INSERT (Q, u)                                  ——— |V| log2|V|
Decrease-key (Q, r, 0) // K[r] ⇐ 0                ——— log2|V|
while Q ≠ 0                                      ——— |V|
  u ⇐ Extract-Min(Q)                             ——— |V| log2|V|
  for each v ∈ Adjacent[u]                       ——— |E|
  if v ∈ Q and w(u, v) < key[v]                  ——— |E|
    π[v] = u                                     ——— |E|
  Decrease-key (Q, v, w(u, v))                   ——— |E| log2|V|
```

Runtime: $|E| \log_2|V| + |V| \log_2|V|$

D G N M T C S Y

Nil ⇒ -

- Key এর মানের priority queue insert,
- crossing edge দ্বারা,
- যেই Node এর সাথে অন্য যেই Node গুলো connected আছে তার মান আগের মানের চেয়ে ছোট হতে হবে, Then update,

Update এর Law

→ Decrease করা key তে

→ Parent হিসাবে নিজের বসা,

	D	G	N	M	T	C	S	Y
Key	∞	∞	∞	∞	∞	∞	∞	∞
π	—	—	—	—	—	—	—	—

update

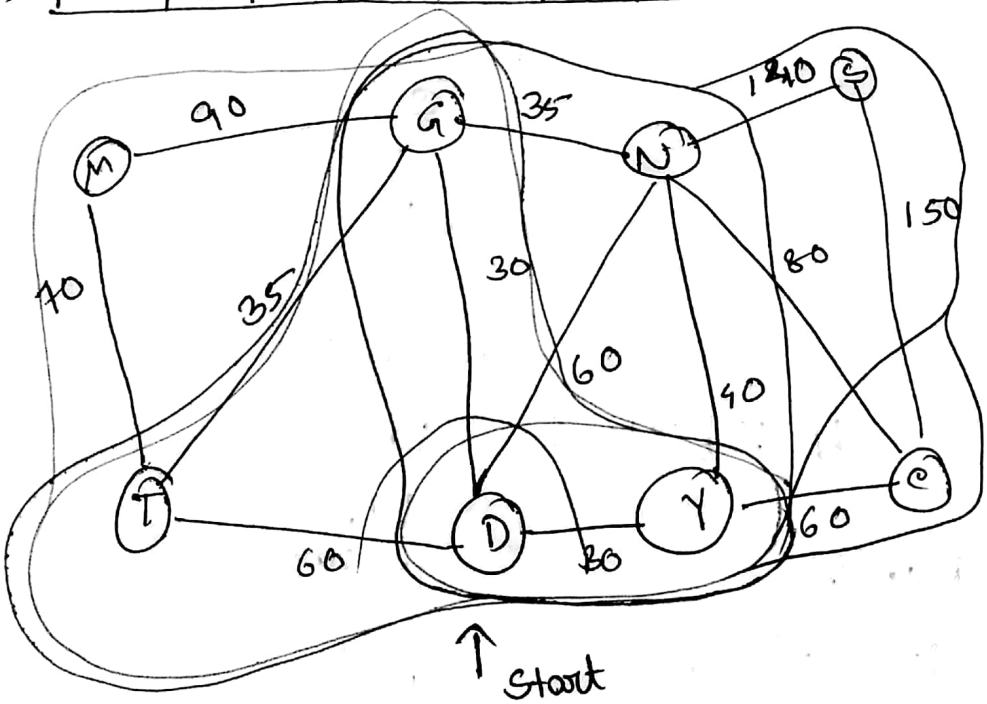
	D	G	N	M	T	C	S	Y
D	0	30	60	∞	60	∞	∞	30
		D	D		D			D

Priority queue থেকে this 30 connected

	D	G	N	M	T	C	S	Y
Y	0	30	40	∞	60	60	∞	30
		D	Y		D	Y		D

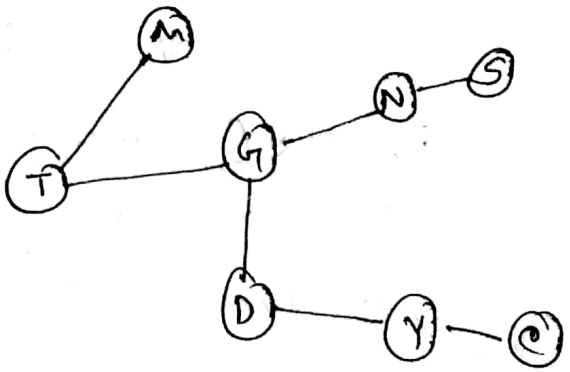
G	0	30	35	90	35	60	∞	30
		D	G	G	G	Y	-	D

T	0	30	35	70	35	60	∞	30
		D	G	T	G	Y	-	D



2	∅	30	35	70	35	60	140	30
		D	G	T	G	Y	N	D

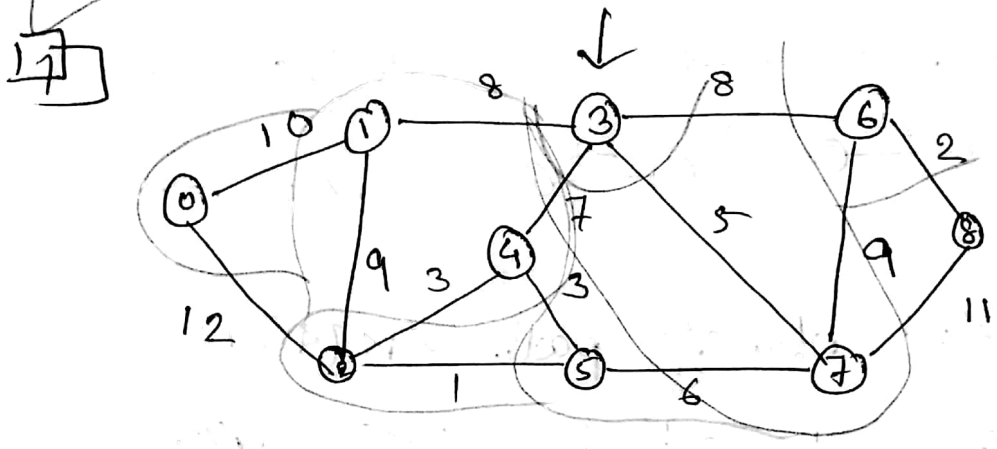
MST



Kruskal → $|E| \log_2 |E| + |E| \log_2 |V|$

Prim → $|E| \log_2 |V| + |V| \log_2 |V|$

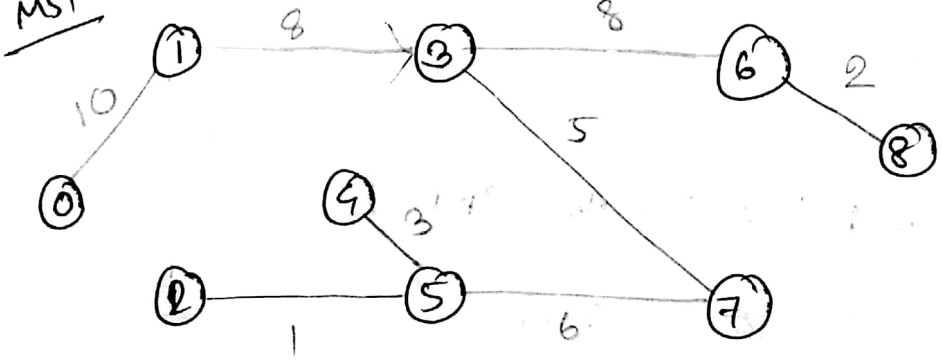
Prim better, we can see from the runtime.



consider source/start = 3

find MST and show the array key & π

MST



	0	1	2	3	4	5	6	7	8
key	∞ 12 10	∞ 8	∞ 1	∞	∞ 7 3	∞ 6 4	∞ 8	∞ 5	∞ 11 2
π	2 1	3	5		3 5	7	3	3	7 6

Final Ans :-

Key	10	8	1	0	3	6	8	5	2
⇑	1	3	5	-	5	7	3	3	6

positive cycle ④ —3— ② —1— ⑤ —3— ④ = 7 > 0 +ve

negative cycle ④ —3— ③ —(-1)— ⑤ —(-3)— ④ = -1 < 0 -ve

☐ cycle neg হলে Shortest path থাকবে না,
cycle pos হলে Shortest path থাকতে পারে,

Shortest Path Problem.

→ Single Source

1) Simple (Breath Frist Search) (BFS)
   ↳ Unweighted (কাজ করবে)

2) Dijkstra
   ↳ weighted, edge weights positive (একটাও Neg হলে কাজ করবে না)

3) Bellman-ford
   ↳ negative cycle থাকলে কাজ করবে না,

4) DAG-6 Shorted Path
   ↳ DAG (Directed Acyclic Graph), weight.

07/04/16

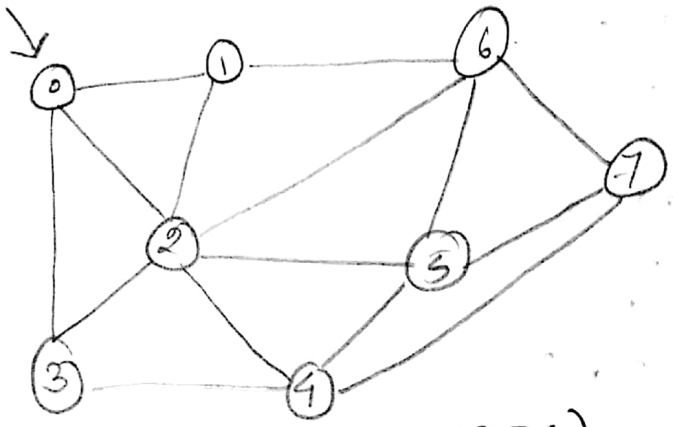
(20)

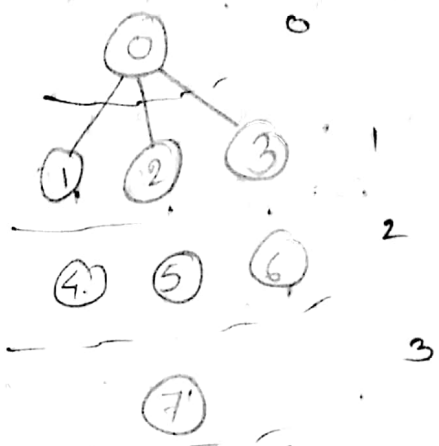
Shortest Path Problem:

Input:

1) A graph, unweighted
   $G = (V, E)$
2) Source vertex

Output: Shortest Paths





Breadth first Search (BFS)

1) Queue
   - First In First Out
   - FIFO
2) is Visited
   - Array
3) TT, Parent, predecessor

	0	1	2	3	4	5	6	7
isvisit	0	0	0	0	0	0	0	0
π	-1	-1	-1	-1	-1	-1	-1	-1

⇒ -1 কেউ খুঁজে পায় নাই।
⇒ 0 = visit করে নাই।

0 1 2 3 6 4 5

→ insert / push / nq
→ delete / pop / dq

After visit 0

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

→ Source always -1 হয়ে থাকে কারণ তার কোন parent নাই।

A.V.1

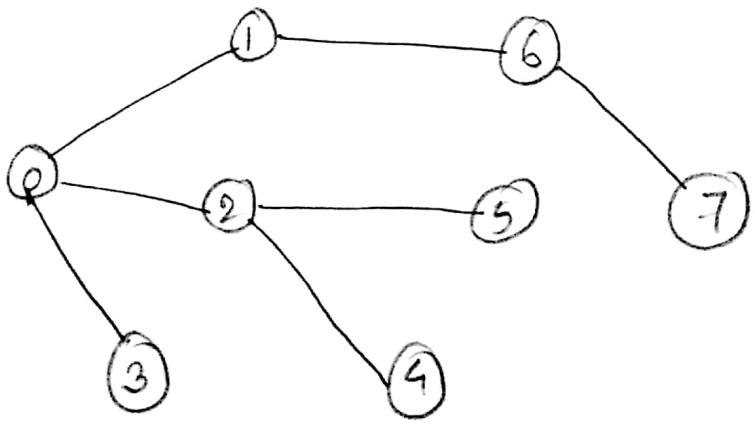
1	1	1	1	0	0	1	0
-1	0	0	0	-1	-1	1	-1

A.V.2

1	1	1	1	1	1	1	0
-1	0	0	0	2	2	-1	-1

A.V.6

1	1	1	1	1	1	1	1
-1	0	0	0	2	2	1	6



→ only যাদের parent যাদের তাদের edge এইটাই Shortest Path.

```
void PrintShortestPath(int v, int pi[])
{
    if (pi[v] ≠ -1)
        printShortestPath(pi[v], pi)
    printf("%d", v);
}
```

→ যার জন্য বের করব তার parent তার parent এভাবে আগাবো source পর্যন্ত।

PSP(7)

| parent call

PSP(6)

print(7)

|

PSP(1) print(6)

|

PSP(0)

print(1)

|

print(0)

H.W| Print Shortest Path এর iterative function.

Queue
isvisited
T

```
void BFS( int G[][], int s)
{
   for(int i=0; i<nv; i++)
   {
      isvisited [i]=0;
      pi[i]= -1;
   }
   isvisited [s] = 1;
   Queue Insert(s);
   while (Queuesize() != 0)
   {  int u = Queue Delete;
      for (int v = 0; v<nv ; v++)
         if(G[u][v]==1 && isvisited [v]==0)
         {
            pi [v] = u;
            isvisited [v]= 1;
            Queue Insert (v);
         }
   }
}
```

runtime: $O|V| + |E|$

Depth first Search (DFS)

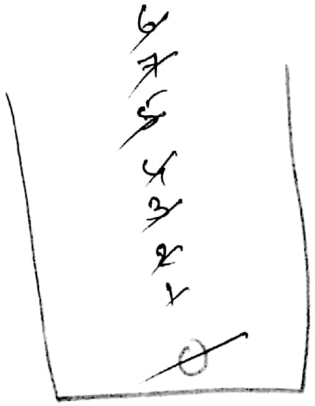
→ code Same কিন্তু Difference.

```
void DFS(int G[][], int s)
{   for (int i=0; i<nv; i++)
    {   isvisited[i]=0;
        pi[i]=-1;
    }
    is visited[s]=1;
    Stack Insert(s);
  while (Stack.size()!=0)
  {  int(u = Stack Delete);
     for (int v=0; v<nv; u++)
       if (G[u][v]==1 && isvisited[v]==0)
       {   pi[u]=u;
           isvisited[v]=1;
           Stack Insert[v];
       }
    }
}
```

DFS

→ cycle detection
→ topological sorting
→ connected component.
→ Last in first out

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



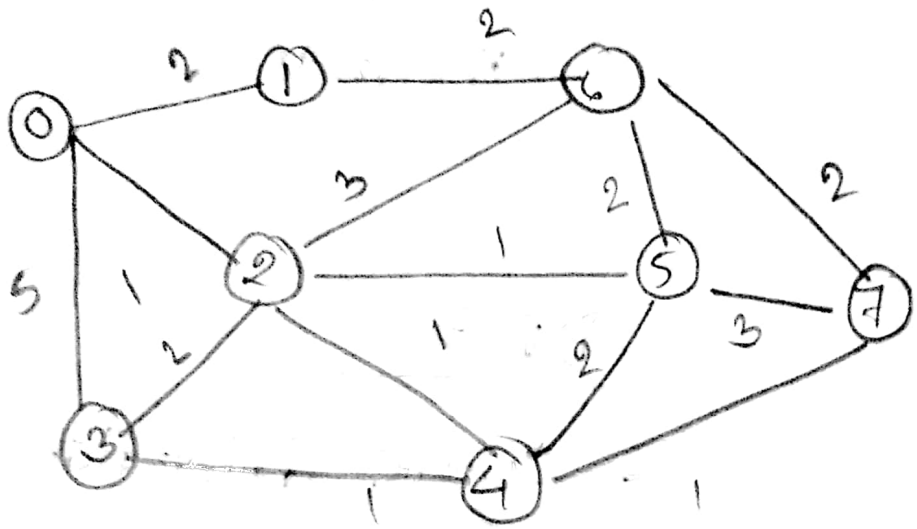
isvi 0,3

1	1	1	1	1	0	0	0
-1	0	0	0	3	-1	-1	-1

is vis 4,7

1	1	1	1	1	1	1	1
-1	0	0	0	3	4	7	4

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



Dijkstra's Algorithms:

↳ min Priority Queue

→ is visited

→ π

→ d / distance (short distance)

```
void Dijkstra (int G[][], int s)
{
  for (i=0; i<nv; i++)
  {
    isvisited[i]=0;
    pi[i] = -1;
    d[i] = INT_MAX;
    MinPQInsert(i, d[i]);
  }
  d[s]=0;
  MinPQ DecreaseKey(s,0);
  While (Min PQ.size()!=0)
  {
    int u = Min PQ.Delete);
    isvisited[u]=1;
    for (int v=0; v<nv; v++)
      if (G[u][v]==1 && isvisited[v]==0)
      {
        if (d[u]+G[u][v] < d[v])
        {
          pi[v]=u;
          d[v]=d[u]+G[u][v];
          Min P.Q Decrese (v, d[u]);
        }
      }
  }
}
```

→ খুব্বু distance আনি না তাই d = ∞ করে দেব।

→ 2নম্বর নিজে বের হবে তখন সে নিজেকে visited করে shortest path নিয়ে বের হবে।

→ Better কিছু পেলেই only Update করব।

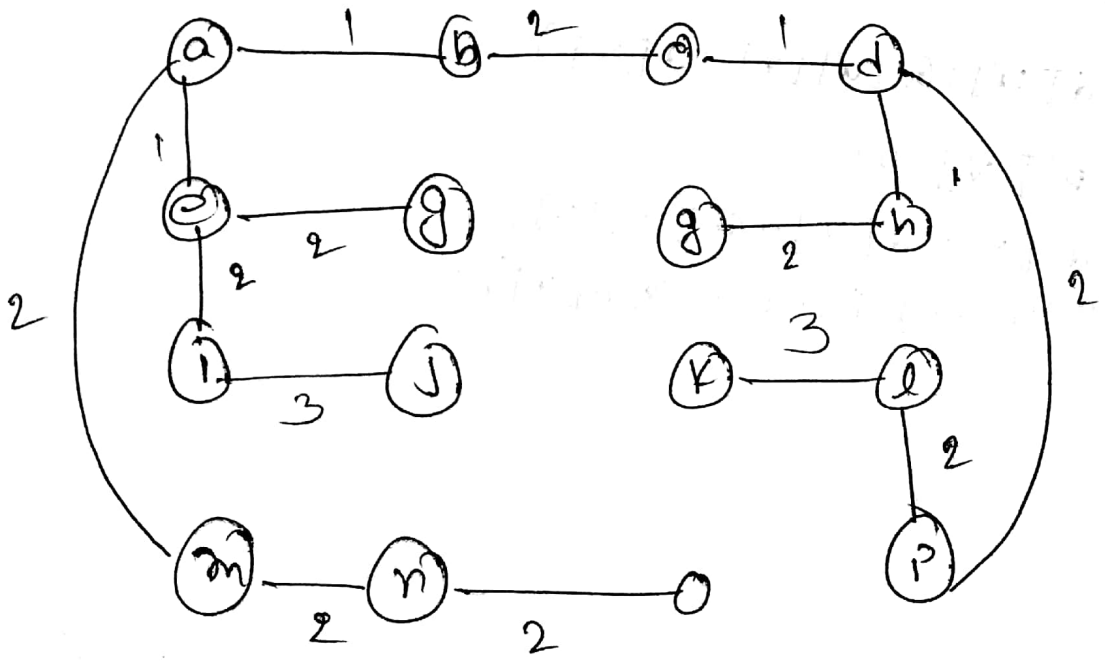
26

0	1	2	3	4	5	6	7	
0	0	0	0	0	0	0	0	isvisited
-1	-1	-1	-1	-1	-1	-1	-1	π
∞	∞	∞	∞	∞	∞	∞	∞	d

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

যে যখন বের হবে তখন সে নিজেকে visited করবে।

11/04/16

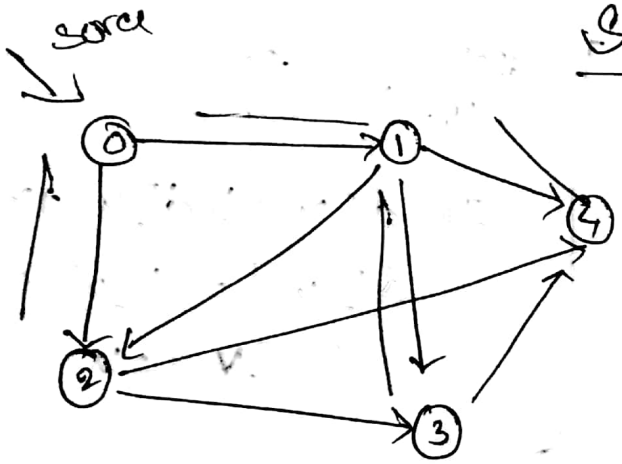


MST = 28।

List Better

16 x 2

28 x 3



Shortest Path

BFS

যে queue তে ঢুকবে তা আর কখনো update হবে না।

Queue: (FIFO)

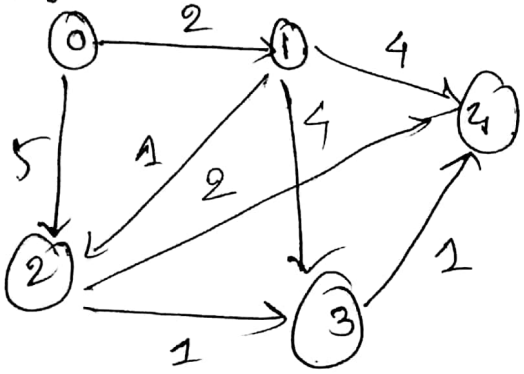
~~0~~ +2 3 4

isvisited

0	1	2	3	4
0	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1
-1	~~-1~~ 0	~~-1~~ 0	~~-1~~ 1	-1

Source 0

Dijkstra



→ যে Queue থেকে বের হল তা আর কখনই update হবে না।

d = estimate of shortest path distance

min Priority Queue

(0,0), (1,∞), (2,∞), (3,∞), (4,∞)

	0	1	2	3	4
isvisited	1	1	1	1	1
π	-1	0	1	2	2
d	∞	2	3	4	5

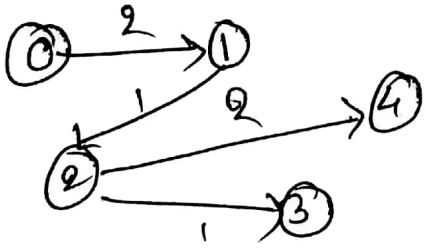
If $d[v] > d[u] + w$

$d[v] = d[u] + w$

$\pi[v] = u$

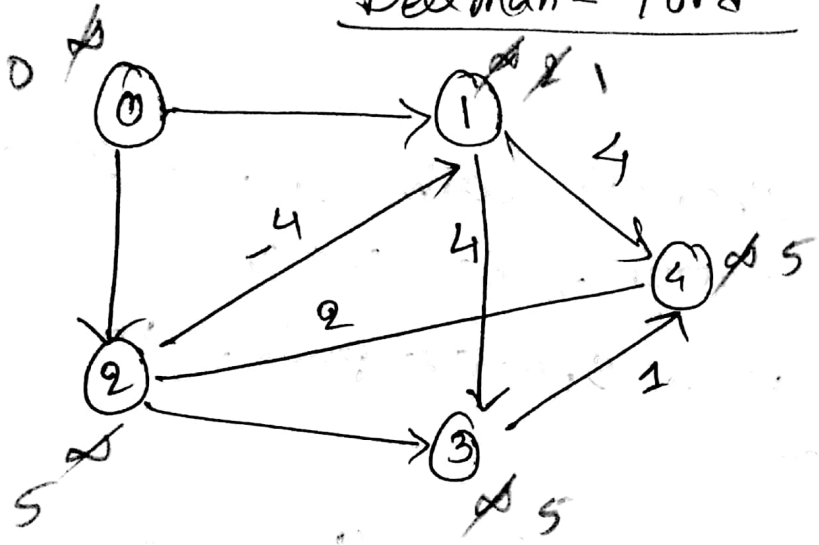
relax an edge

(0,0), (1,2), (2,3), (3,4), (4,5)



Shortest Path.

Bellman - Ford



Shortest Path

minimum path
or

Shortest path

→ Shortest paths are not unique

→ Shortest paths from a single node are organized as tree.

→ Path cost example time, cost, penalties.

→ Generalized BFS for weighted graphs.

Variants

1) Single source : Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$

2) Single destination: Find shortest paths to a given destination vertex.

3) Single pair : Find shortest path from u to v

4) All pairs: Find shortest path from u to v for all $u, v \in V$

Negative weights?

→ are ok →

→ Neget cycles are not ok → Shortest path pasible না

→ Neget cycles are ok if and only if not reacheable from source.

cycle?

→ positive হলেও দাম, prob নাই।

Outputs of the Algo

For each vertex $v \in V$

→ $d[v] = \delta(s, v)$

initially $d[v] = \infty$

Reduces as algorithm progresses, maintain $d[v] \geq \delta(s, v)$

$d[v]$ is a shortest path estimate

→ $\pi[v]$ = predecessor of v on a shortest path from s

if no predecessor, $\pi[v] = NIL$

π induces shortest path tree

Initialization

```
INIT - SINGLE-SOURCE (V, S)

    for each v ∈ V
        d[v] = ∞
        π[v] = NIL
```

RELAX

```
if d[v] > d[u] + w
```

```
Relax (u, v, w)
    if d[v] > d[u] + w(u,v)
        d[v] = d[u] + w(u,v)
        π[v] = u
```

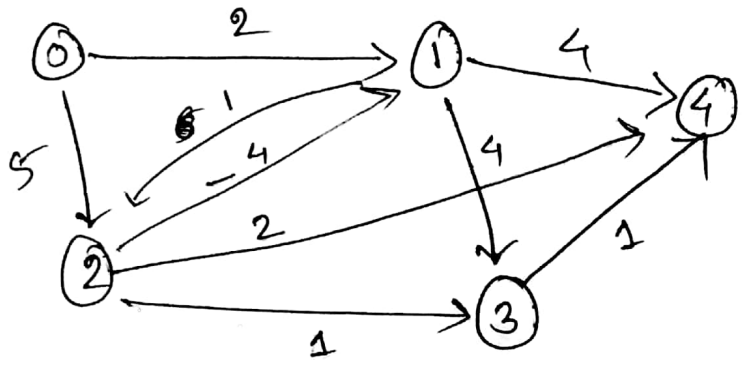
Bellman ford - Algorithm.

```
1. INIT - Single Source (V, S)
2. for i = 1 to |V| - 1
3.   for each edge (u,v) ∈ E
4.     Relax (u,v,w)
5. for each edge (u,v) ∈ E
6.     if d[v] > d[u] + w(u,v)
7.       return FALSE
8. return TRUE
```

(n-1) বার

*Bellman ford - এ 4 বার লুপ, 4 বার লুপ চলল shortest path নাই।

H.W



→ Bellman-ford কাজ করবে না
→ neget cycle থাকে

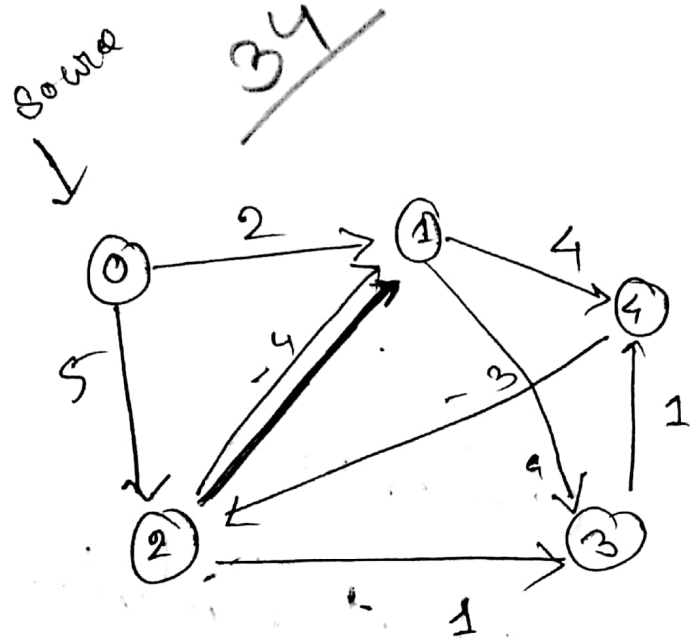
- Triangle inequality: For all $(u,v) \in E$, we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$

- Upper Bound Property: We always have $d[v] \geq \delta(s,v)$. Once $d[v] = \delta(s,v)$, it never changes.

- No Path property: If $\delta(u,v) = \infty$ then $d[v] = \infty$ always

13/04/16

34



i) Detect the cycle
ii) print the cycle

i) find shortest path

	0	1	2	3	4
d	0	∞	∞	∞	∞
π	-1	-1	-1	-1	-1

⇒ Bellman & source 0

⇒ n-1 বার হয়, কাজ করবে।

	0	1	2	3	4
d	0	~~∞~~ 2	~~5~~ 2	~~∞~~ 6 ~~5~~	~~∞~~ 5
π	-1	~~2~~ 2	~~4~~ 4	~~2~~ 3	~~-1~~ 1

1st try

	0	1	2	3	4
d	0	~~-2~~ -2	~~-1~~ -1	~~5~~ 2	2
π	-1	2	4	~~2~~ 1	1

2nd try

	0	1	2	3	4
d	0	~~-2~~ -5	~~-1~~ -4	0 -1	-1
π	-1	2	4	~~2~~ 1	1

3rd try

4thtym

	0	1	2	3	4
0		-5 -8	-4 -7	-2 -4	-4
-1		2	4	3 2	1

→ 5th tym এ যদি relax হয় then Shortest Path নাই, cycle আছে।

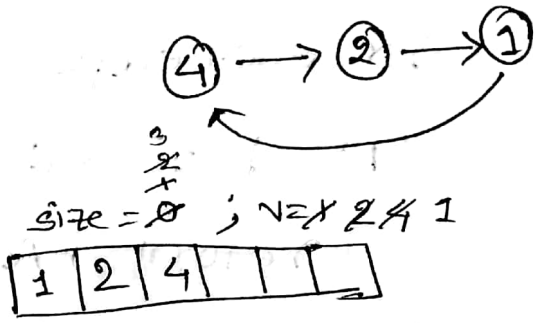
(i) detect a cycle.

→ There is a cycle is this graph because at the 5th tym the graph's edge is updated.

35

(ii) Print the cycle :-

```
void printNegativeCycle(int v)
{
    int cycle[nv];
    int size = 0;
    while(linearSearch(cycle, size, v) == -1)
    {
        cycle[size++] = v;
        v = pi[v];
    }
    cycle[size] = v;
    while(size >= 0)
        printf("%d -", cycle[size--]);
}
```



Bellman - ford Algorithm


```
Bellman- ford (V, E, W, s)
INIT . SINGLE - SOURCE (V, s)          |V|
for i = 1 to |V| - 1                   |V|-1
    foreach edge (u,v) ∈ E             (|V|-1)|E|
        RELAX (U, V, FW)               (|V|-1)|E)
for each edge (u,v) ∈ E                |E|
    if d[v] > d[u]+w(u,v]              |E|
        return FALSE                   1
return True
```

runtime: $|V| \times |E|$

runtime: V^3

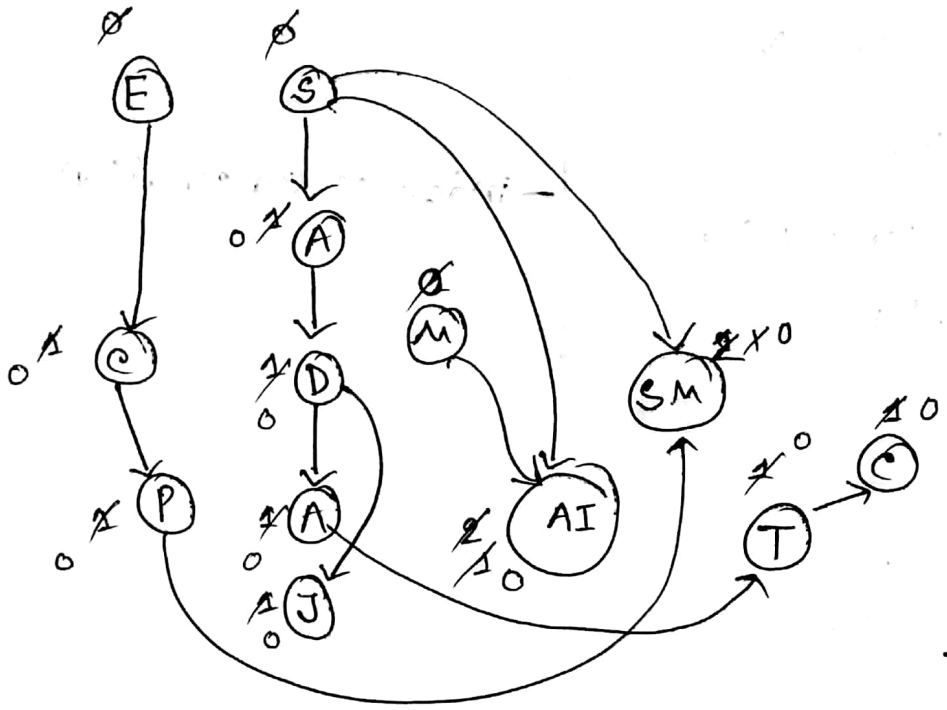
Dence graph → ঘন গ্রাফ → V^2 → Adjacency list → tym ways

Sparse graph → ছড়ানো ছিটানো → V → Adjacency Matrix → Space ways

runtime: V^2

DAG (Directed Acyclic Graph)

→ graph undirected হলে cycle থাকবেই।



→ edge থাকতখনই থাকবে যখন pre-requisite হবে।

→ precedence থাকতে হবে।

Output
→ Sequence / ordering

→ Topological sorting / ordering কাজ করে না তখন বুঝবো cycle আছে।

↳ indegree বের করব।
↳ আমার দিকে যে আসছে

→ যাদের indegree '0' তাদের আগে কেয়ে করা।

→ indegree বের হয়ে যাবে এবং তাদের সাথে যে edge connected তার indegree 1 করমায় দিবে।

Output

E - M - S - C - A - P - D - A - J - T - AI - SM - C

Topological Sort (G = (V, E))

{

V — for each vertex $v \in V$
    indegree [v] = 0

E — for each edge $(u, v) \in E$
    indegree [v] ++

— indegree calculation

c — T = { }, Q = { }

V — for each vertex $v \in V$
    if indegree [v] == 0
        Q . insert (v)

V — While (Q is not empty)

}

V — u = Q . remove ( )
    T = T ∪ {u}

|E| — for each vertex $v \in$ Adjacentlist (u)
    indegree [v] − − ;
    if indegree [v] == 0
        Q . insert (v) ;

}

runtime: |V| + |E|

DAG - Shortest Paths (V, E, W, S)

topologically sort the vertices V+E

INIT-SINGLE-SOURCE (V, S) V

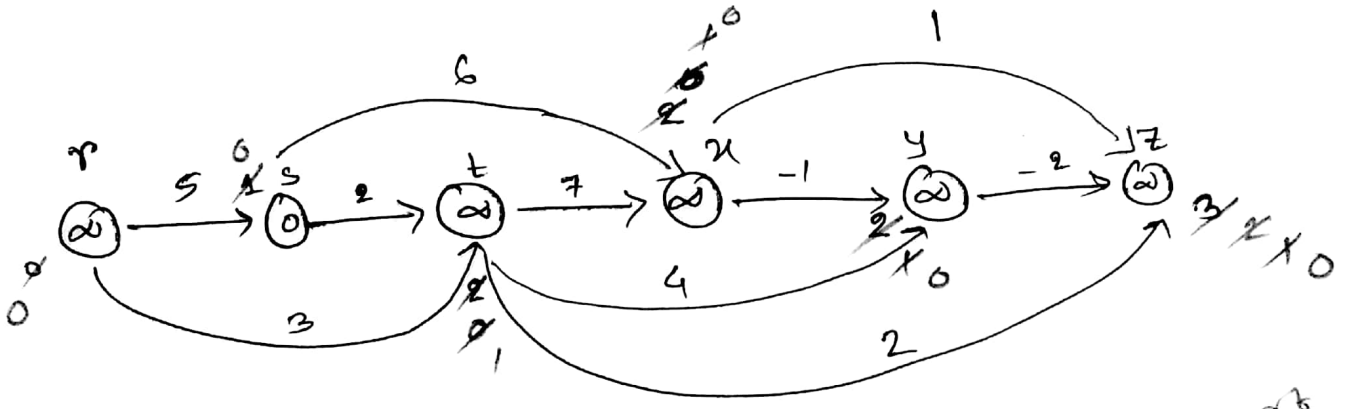
for each vertex u taken in topological sorted order V

  for each vertex v ∈ Adjacent[u] E

    RELAX (u, v, W) E

Runtime: V+E

→ একটা edge একবার relax করবে।



	r	s	t	x	y	z
d	∞	~~∞~~ 0	~~∞~~ 2	~~∞~~ 6	~~∞~~ ~~6~~ 5	~~∞~~ ~~4~~ 3
π	-1	-1	~~x~~ s	~~x~~ s	~~x~~ ~~s~~ x	~~x~~ ~~t~~ y

	r	s	t	x	y	z
d	∞	0	2	6	5	3
π	-1	-1	s	s	x	y

Topo sort
↳ much better than Bellman ford
↳ But only DAG ই use.

Dijkstra S.P (V, W, E, S)

INIT-SINGLE-SOURCE(V, S)

$S = \emptyset$

$Q = V$

While $Q \neq \emptyset$

u = Extract-Min (Q)

$S = S \cup \{u\}$

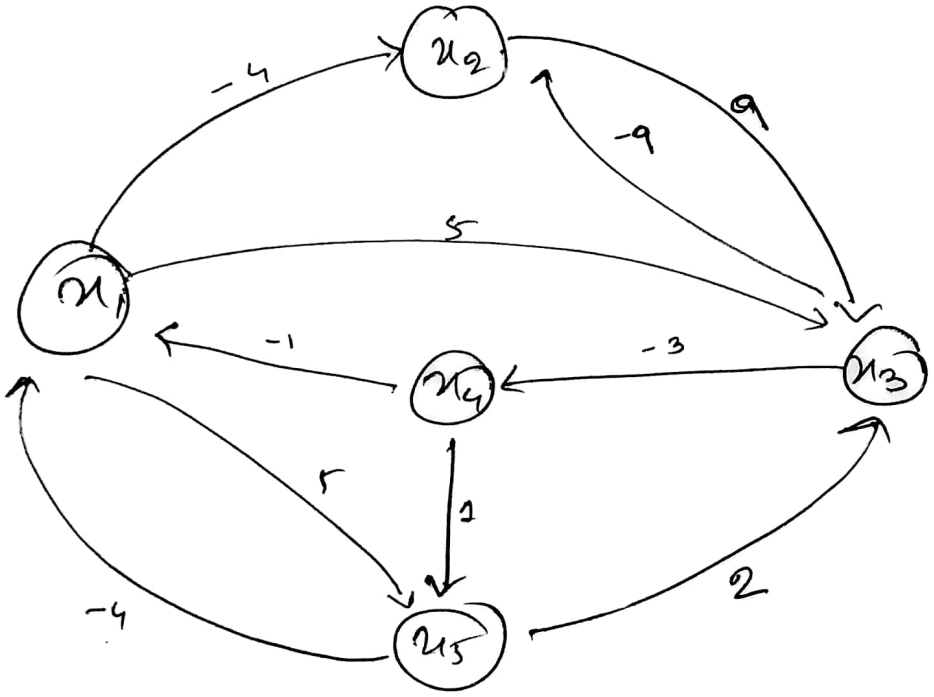
for each vertex $v \in$ Adjacent [u]

Relax (u, v, w)

C.T

Question Solve

find Shortest path



(a) $x_1 \rightarrow x_5$

(b) $x_3 \rightarrow x_1$

1

18/04/16

Comparison Sort:

→ Those sorting algorithms that compares two elements to sort.

→ Bubble Sort, Quick Sort, heap sort, merge sort, Selection sort.

$n^2 \rightarrow n \log_2 n$

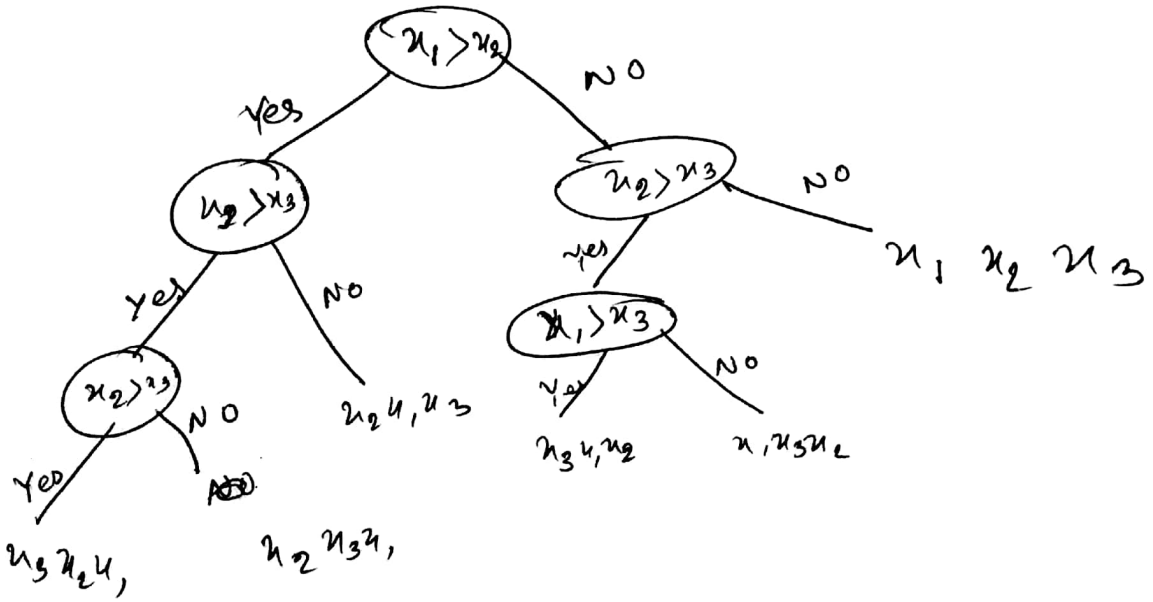
↓

?

Ω (?)

→ The only operation that may be used to gain order information about a sequence is comparison of pairs of elements.

a_1 a_2 a_3 Compair



"Decision Tree"

→ polinomial এ নিব না হইলে বের হবে না 2 solve.

leafe → 6 টা

combination → 6 টা

element যত তত (!)

→ $n \to n!$

→ $3 \to 3!$

→ $6 \to 6!$

→ $12 \to 12!$

hight → $\log_2 n$

Any binary tree of height h has $\leq 2^h$ leaves.

$T(n) \geq h \geq \log_2 (n!)$

Sterlings Appoximation

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

~~$e^x = 1 + \frac{1!}{x^1} + \frac{2!}{x^2} + \frac{3!}{x^3} + \dots \infty$~~

$e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \infty$

$e^1 = 2.71 \dots$

$h \geq \log_2 \left(\frac{n}{e}\right)^n$

$\geq n \log_2 n - n \log_2 e$

$h = \Omega(n \log_2 n)$

Scanned by CamScanner

③ Counting Sort:

- How many elements are there smaller or equal?
- All the elements are in a range [0 --- k]

A

4	0	1	3	2	6	2	3	1	2

C

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

→ কত করটা আছে

C

0	1	2	3	4	5	6	7	8	9
1	3	6	8	9	9	10

→ আমার থেকে ছোট / সমান আছে কয়টা

B

0	1	2	3	4	5	6	7	8	9
0	1	1	2	2	2	3	3	4	6

K = maximum

```
void Counting Sort ( int A[], int B[], int n)
{
    int c[K+1];
    for (i=0; i<=K; i++)
        c[i] = 0;                          → K
    for (i=0; i<n; i++)
        c[A[i]] = c[A[i]] + 1;             → n
    for (i=1; i<=K; i++)
        c[i] = c[i] + c[i-1];              → K
    for ( i=n-1; i>=0; i--)                → n   (Assending / Decending)
        B[c[A[i]]-1] = A[i];
        c[A[i]] = c[A[i]]-1;
}
```

Runtime: $O(n)$ → K এর চেয়ে ছোট

$O(n+K)$ → K/n সমান

→ n এর value K এর value থেকে বড় only তখনই we can use counting sort.

15

→ Counting Sort কেন Stable Sort ?

→ element same থাকলে তারা যদি তাদের order বজায় রাখে তবে তাদের Stable Sort বলে।

— if there are two elements

$A[i] = A[j]$, $i < j$, and after sorting

$i \to i'$; $j \to j'$ if $i' < j'$, then we call it a Stable algorithms.

So we use decending sort Here.

→ All algo are Stable.

→ Counting Sort তখনই apply করা যাবে যখন element কম হবে।

→ prim ত যে কোন side থেকে start করে Node + edge cut করে then we find mst.

→ Krustal ত ছোট edge গুলো connect করতে করতে we find mst.

→ BFS → unweight
Dijkstra → weight + positive
Bellman ford → weight + neget (neget cycle হলে don't work)

DAG → Indgree (Tropological sort) then → relax.

Decision Tree

→ Abstraction of any comparison sort.
→ Represents comparisons made by
  → a specific sorting algorithm.
  → on inputs of a given size
→ Abstracts away everything else: control and data movement
→ We are counting only comparisons.

(6)

20/04/16

□ Radix Sort

Radix Sort (A, d)

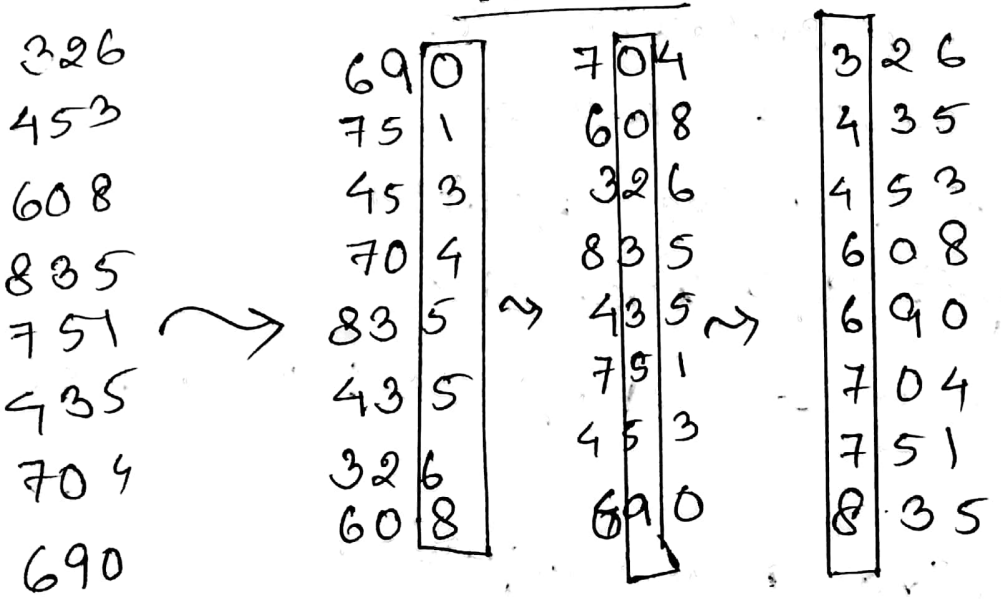
for i = 1 to d

use a stable sort to sort array A on digit i

326 690 704

Sort এর rule.

Sorted



→ সব integer সমান সংখ্যক Digit.

→ Integer না হলে Binary করব.

→ উদাহরণে সবারই ৩টা।

→ ধরা হলে সবারই ৪টা।

→ যে আগে আছে সে আগে থাকবে।

→ linear runtime

→ number of digit যদি বেশি হয়, range ও হয়।

(7)

Backet Sort (linear time Runtime প্রতি সমান ভাবে Same number এর element যায়)

→ Divide [0,1) into n equal-sized buckets.

→ দশমিক দিতে হবে।

→ Backet এ নিব। কোন Backet এ নিব তা hassing ~~নিয়ম~~ use হয়।

⇒ দশমিক সংখ্যাকে 10 দ্বারা গুন করব।

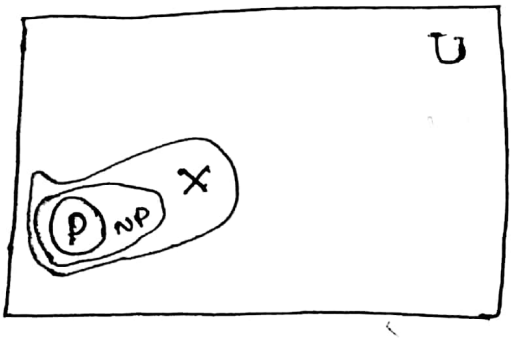
→ দশটা Backet হলে 10টা empty linked list.

```
Bucket Sort (A)
Let B[0 ... n] be a new array
n = A.length
for i = 0 to n-1
    make B[i] an empty list
for i = 1 to n
    insert A[i] into list B[⌊nA[i]⌋]
for i=0 to n-1
    Sort list B[i] using insertion sort.
```

8 → Complexity Class

Easy Problems / Tractable Problems

P = {a set of problems for which there exist a correct algorithm that solves the problem in polynomial time}



→ Polynomial time

n^c
n^0
n^1
n
$n \log_2 n$
n^3
n^{100}
$n^{3\sqrt{n}}$
⋮

X = {a set of problem for which there exist a correct algorithm that sortes the problem in exponential time}

→ Problem हल
P का member,
→ MST
SORTING
shortest Path
greedy knapsack

→ exponential

e^n ⋮ n^n
2^n
3^n
4^n
⋮

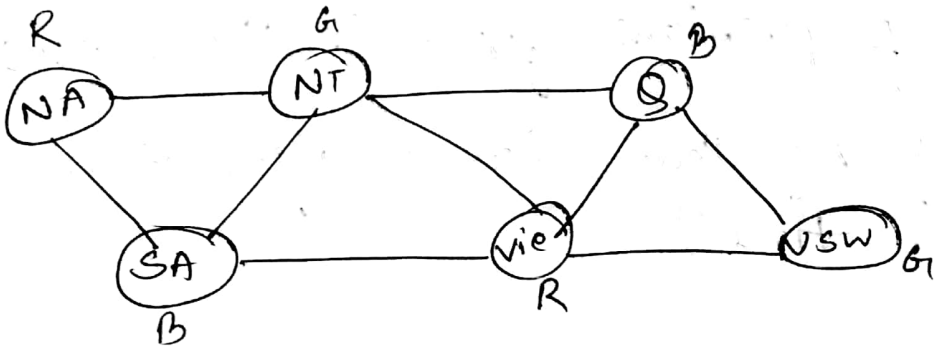
$2^n > n^{100}$

$P \subseteq X$

exponential Graph

→ Graph Coloring

→ Map colouring



TR = { a set of problem for which there one no correct algorithm that runs in finite time }

```
int main()
{
while (1)

}
   int *p= malloc ()

  return 0;
}
```

→ HALTING PROBLEM

input: A program

Output: Yes / NO

if it halt

if it doesn't halt,

(19) Verification Problem

Sorting

1 3 2 5
input

1 2 3 5
output

NP = { a set of problems for which there exist a correct Algorithm that verifies it in polynomial time }

P ⊆ NP ✓

NP ⊆ P ✗

P = NP ?

P ≠ NP ?

(11)

25/04/16

Decision Problem:

Given an input instance of a problem, decide whether a ~~sol~~ solution exist or not.

Verification Problem:

Given an input instance and a ~~sol~~ solution of a problem, decide whether that problem maps to the solution or not.

NP

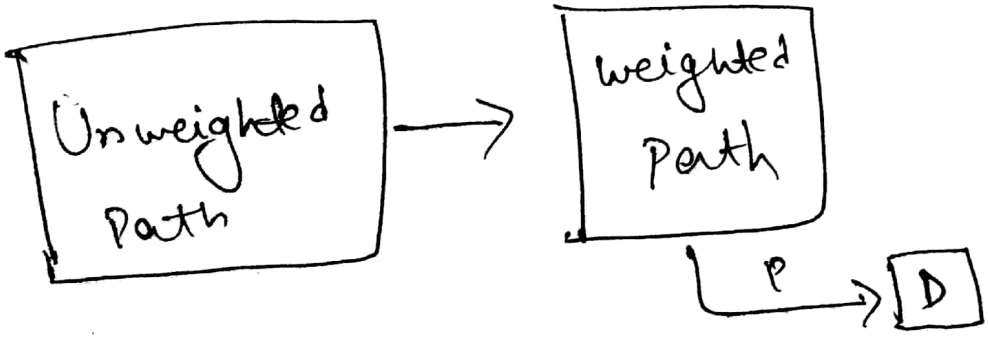
একটা problem P হলে তা NP হবেই।

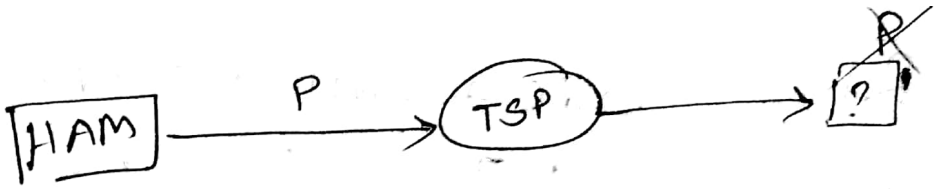
NPC (NP-complete) (tough problem)

A problem $X \in NPC$ iff (if and only if) following are true:

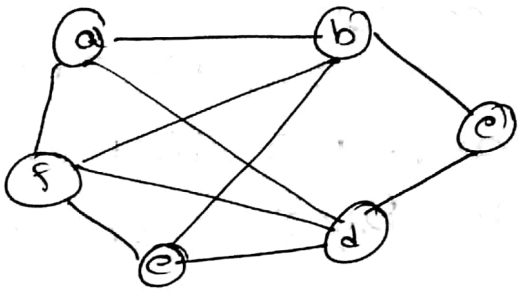
i) $X \in NP$

ii) $Y \in NPC$; $Y \leq_P X_i \forall Y$





Hamiltonian Cycle / Travelling Sales Pearell Problem (TSP)



→ a - b - c - d - e - f - a

→ b - a - f - e - d - c - b

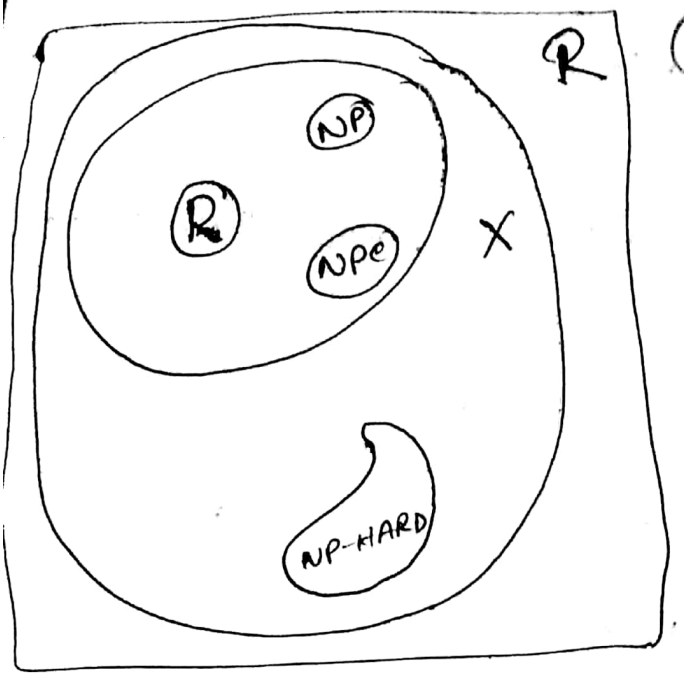
→ a - b - c - d - f - a ×

→ TSP → Polinomial time a Slove হয় না, So Hamiltonian ও Slove হয় না।

→ NP → Compendium → NP problem এর list দেওয়া আছে।

NP-HARD

যে কোন একটা Solve হলে সব Solve হয়ে যায়, varify করা যাবে না।



→ NPC এর একটা এদিক ওদিক হলে তারা P হয়ে যাবে।

→ Reduce এর জন্যই তার Hard হয়।

→ Reduce হল একটা থেকে অন্যটাতে যাওয়া যাবে and. কিন্তু বাদ দিলে তা যাতে ফিরত পাওয়া যায়।

⇒ Donald Knuth

↳ Art of Computer Programme

Algo Prefer

P

X — Time, correctness

R

NP

NPC

NP-Hard

তারপর করব

Approximate Algorithm

n - queen Problem

↳ a $n \times n$ chess grid

→ n queens.

$n \geq 4$

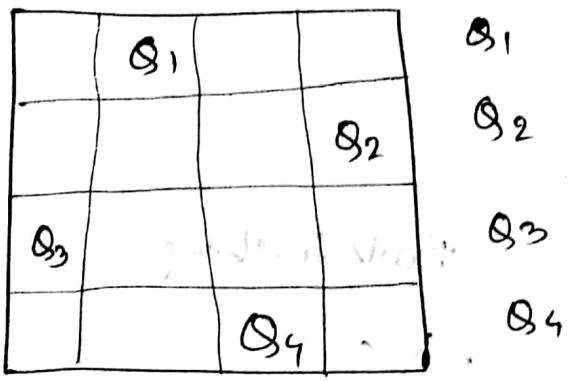
→ Place n queens on a $n \times n$ grid such that no two queen attack each other.

attack ↓

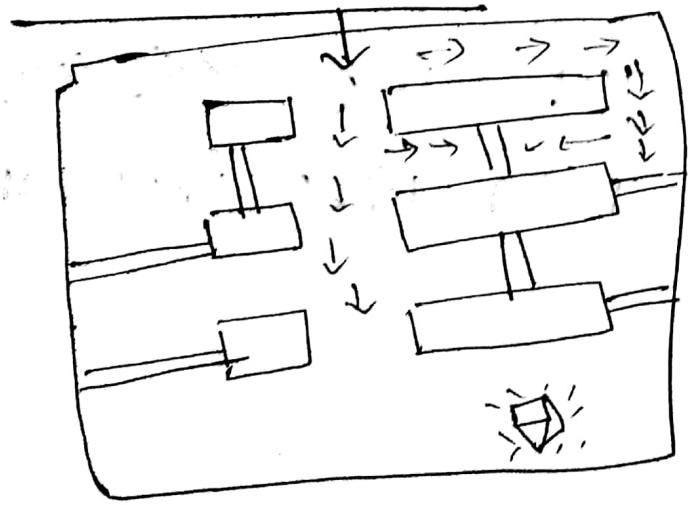
conflict

1) row conflict.
2) column conflict.
3) diagonal conflict.

$n = 4$



Maze (গোলক ধাঁধা)



↳ Back-tracking এর ভুল হলে ফিরে আসাকে বলা হয়।

~~Row~~ (15)

→ row conflict থেকে আসার জন্য Queen গুলোকে row তে fixed করে দিব।

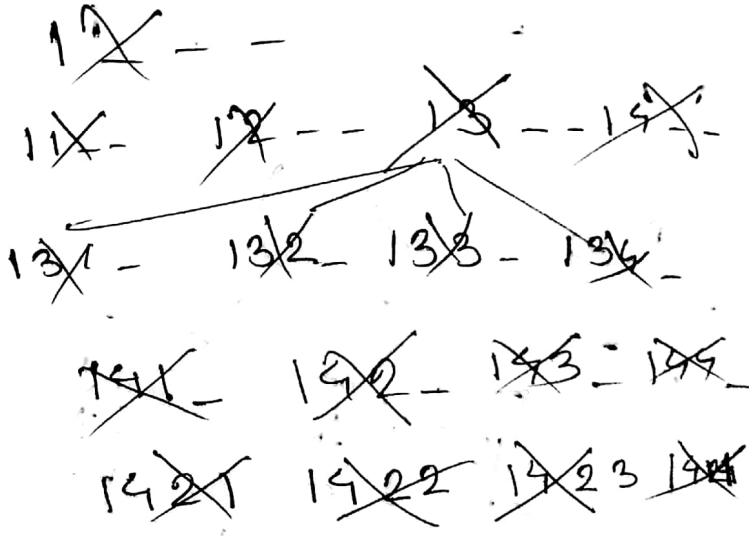
$Q_1 \rightarrow$ row 1
$Q_2 \rightarrow 2$
$Q_3 \rightarrow 3$
$Q_4 \rightarrow 4$

→ Column conflict

Q_1	Q_2	Q_3	Q_4
2	4	1	3
0	1	2	3

Q_1	Q_2	Q_3	Q_4
~~1~~	~~2~~		

Back tracking.



2 - - -

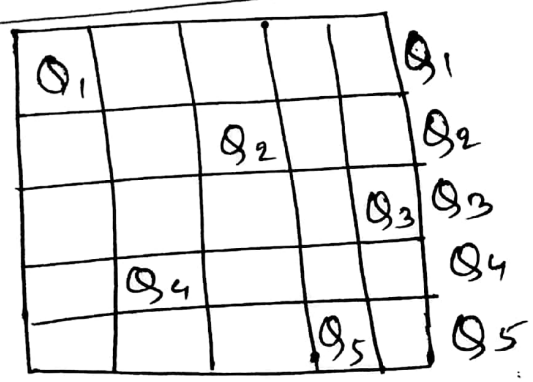
~~21~~ - - ~~22~~ - - ~~23~~ - - 24 - -

241 - ~~242~~

~~2413~~ ~~2414~~ 2413

Q_1	Q_2	Q_3	Q_4
2	4	1	3

5 Queen Problem



Q_1	Q_2	Q_3	Q_4	Q_5
1	3	5	2	4

1 - - - -

~~11~~ - - ~~12~~ - - 13 - - -

~~131~~ - - ~~132~~ - - ~~133~~ - - ~~134~~ ... 135 ..

~~1351~~ - 1352 -

~~13521~~ ~~13522~~ ~~13523~~ 13524

```
int X[n];
void nqueen (int n, int K)
{
   for(int i=1; i<=n; i++)
    {
     if (place(K,i)==1)
     {
       X[K-1]=i;
      if (K==n)
      { print Solution ();
        return;
      }
      else
        nqueen (n, K+1);
     }
    }
print ("NO solution");
}
```

n = n queen
K = Kth queen

→ K তম queen কে i তম ঘরে বসাতে পারি i তে then 1 return.
না গেলে 0,
→ আগের কোয়ে আগে conflict করে নাকি তাও check.

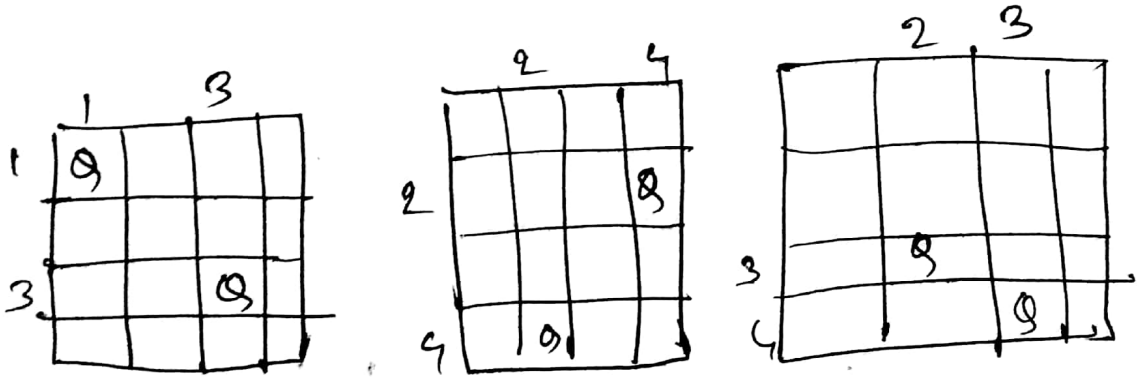
Back-tracking

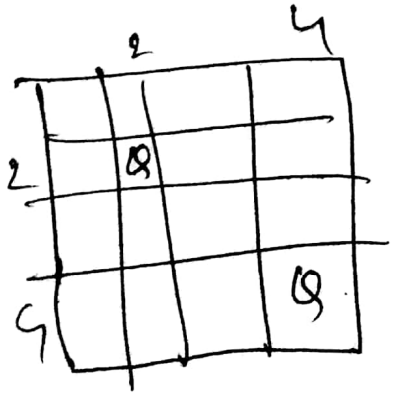
Runtime: n^n

```
int place (int k, int i)
{
    for (int j=1; j<k ; j++)
    {
      if (x[j-1]==i)
          return 0;
      else if (x[j-1]==i || abs(k-j)==abs(i-x[j-1]))
          return=0;
    }
    return 1;
}
```

column conflict

Diagonal conflict





→ row ~~and~~ column ~~and~~ difference Same.

Syllabus

1. Graphs
   ↳ Adjacency lists & Matrix ☐ Concept থাকলে হবে, clear.
2. Disjoint Sets.
3. Minimum Spanning Tree.
   → prim & Kruskal → pseudocode → runtime → complex / কে কোথায় use করে।
4. DFS & Topological Sort.
5. Shortest Path Algorithms
   ~~BFS~~ (1), Dijkstra (3), DAG (2) Shortest Path → order runtime better.
   Bellman-Ford (4). → neget cycle থাকলে
   Shortest path না থাকার posiblity বেশি।
   → Shortest path unique নাকি graph এর উপর depend. edge different হলে unique.
6. Sorting in Linear time
7. Complexity classes (P, NP, NPC, NP-head, X, R)
8. Backtracking Algorithms. (Queen প্রশ্ন)

→ times of Space → Adjacency list.

Runtime

1) Adjacency list → $O(|V|+|E|)$
2) Adjacency Matrix → $O(|V^2|)$
3) Prim → $|E| \log_2 |V| + |V| \log_2 |V|$
4) Kruskal → $|E| \log_2 |E| + |E| \log_2 |V|$
5) Bucket → $O(n^2)$ (Insertion, Quick)
6) Counting → $\Theta(K+n)$
7) Radix → $\Theta(d(K+n))$
8) Bellmanford → $O(EV)$
Merge / heap → $\Theta(n \log_2 n)$ $O(n \log_2 n)$
9) BFS → $O|V| + |E|$

Short Tips

প্রজায়গা কেমন হিসাব করে?

V এর জন্য সূত্র হল 1i, 1P
E এর জন্য সূত্র হল 2i, 1P

thm size দেওয়া থাকবে। (V+E) করে Total number কে size দিয়ে গুন দিব। That's my space!

এর মুল হল Adjacency list এর ~~জন্য~~ So adjacency list better.

$\frac{2 \times 2 = 4}{\text{Matrix}}$ $\frac{2+1 = 3}{\text{list}}$ ✓ better

2) Prim and Kruskal is for minimum spanning tree! Kruskal a little bit greedy.

Kruskal এ ছোট edge গুলো আগে আগে নিয়ে acyclic connected graph করে mst বের করা হয়।

Prim এ একটা source থেকে edge cut / crossing

edge এর সাপেক্ষে mst বের করা হয়।
It was ~~edge~~ algo greedy and acyclic connected graph.

MST of a graph it's unique because their are same edge থাকে weight same even if ২ থাকে edge ৩ same হতে পারে So that mst it's unique হবেনা।
⇒ more edge with the same weight. So mst not unique!

Graph এ vertices 100, MST - 600।
if weight increase all the edges by 2, will mst change?

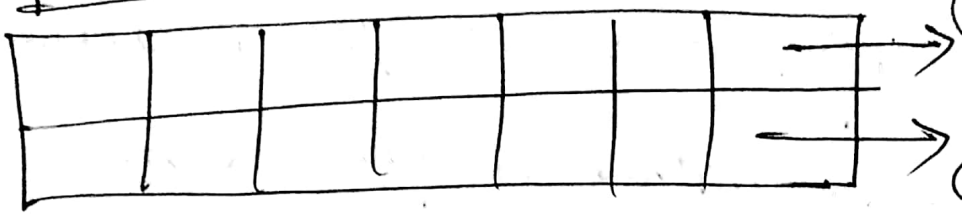
→ vertex 100 So edge 99
if ~~2~~ 2 করে বাড়ে So 198 টা।

99
×2
198

So mst = 798।
mst will be change!

=> Prim ⊖ better from the runtime.
v can easily count but e can't.

Prim



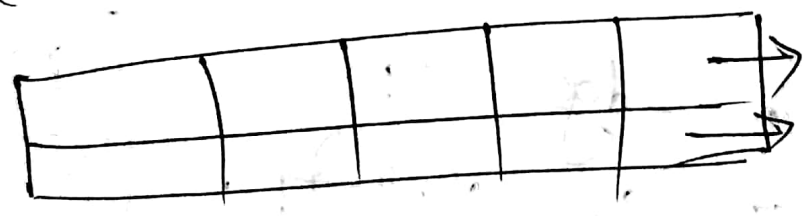
initial হবে ∞ দ্বারা
Key/value (K)
parent (π)
Nil থাকা থাকবে

Kruskal

Box array নাই।
edge weight সব দেখব। কোনো দোষ তা হিসাব করে a cyclic connect graph বের করব যা আমার mst।

৩) neget cycle হলে Shortest path নাই।

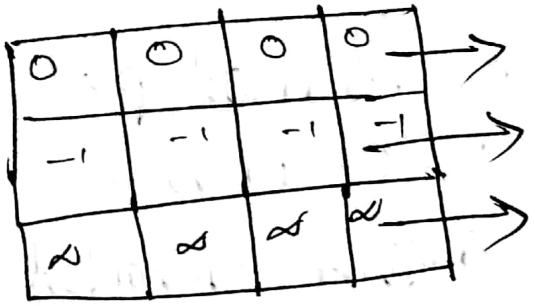
i) BFS → unweighted। একবার যে node হতে যেই node এ এসে তা visited।



isvisited (visit করে নাই (initial 0)
π (-1 দ্বারা initial)
↳ কাউকে খুজে পায় নাই।

Note: সবার parent থাকলে তাহলে Shortest Path।
Source always (-1)।

ii) Dijkstra → Directed, weighted, একবার relax করবে Direction ways, weight always positive হতে হবে।



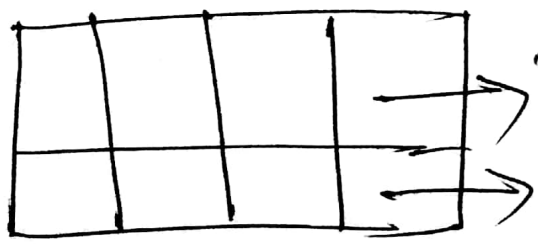
isvisited (key/value) → 0 initial থাকে। → যার যার থাকে বের হবে তাকে visit করা হবে।

π (parent) → -1 থাকে।

d (distance/weight) → ∞ থাকে।

→ আরো যা পাইবে only তার থাকে হো পেলেই update হবে অন্যথায় না।

(iii) Bellmanford → Directed, weighted (negative weight), vertex ⇒ n ; relax হবে n-1 বার, n-1 বার relax complete হয়ে গেলে Shortest path আছে। তার relax n বার হলে S.P নাই। n বার relax হলে negative cycle আছে।



π (parent) → (-1)

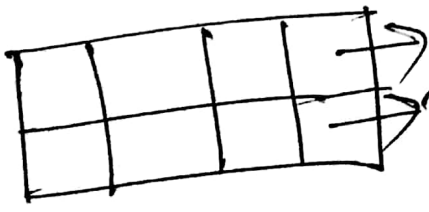
d (distance/weight) → ∞

→ যতবার relax তবার parent/distance change.

Shortest Path aren't unique !

Suppose there is two edge with the same weight but we can chose only one if we chose a-b which is ~~need~~ our shortest path & if we can also choose b-e which is also shortest path so that shortest ~~part~~ path aren't unique.

iv) DAG: Directed, weighted (negat হতে পারে) But কখনো কোন cycle হবে না, means child parent এ যাবে না, DAG হলে graph কে tropological sorting করব, then sort করে তাকে 1 ~~bost~~ বার relax করে দেব



8) Sorting in linear time:

i) Compair Sort → হবে Sorting এ compair

ii) Counting Sort → Stable Sort, order sequence maintain।

$A[i] = A[j]$; $i < j$, $i \to i'$; $j \to j'$

So $i' < j'$।

(iii) Radix Sort :→ Row থেকে শুরু 1st Row, 2nd Row then 3rd Row ways solve করবে।

(iv) Backet Sort → একটা array কে Same equal size এ divide করবে [0, 1), then (.) দিয়ে, প্রত্যেক সংখ্যাকে 10 দ্বারা করে Backet এ দিতে গুণ করে hassing করে দিব।

যত Backet হবে তত empty linked list।

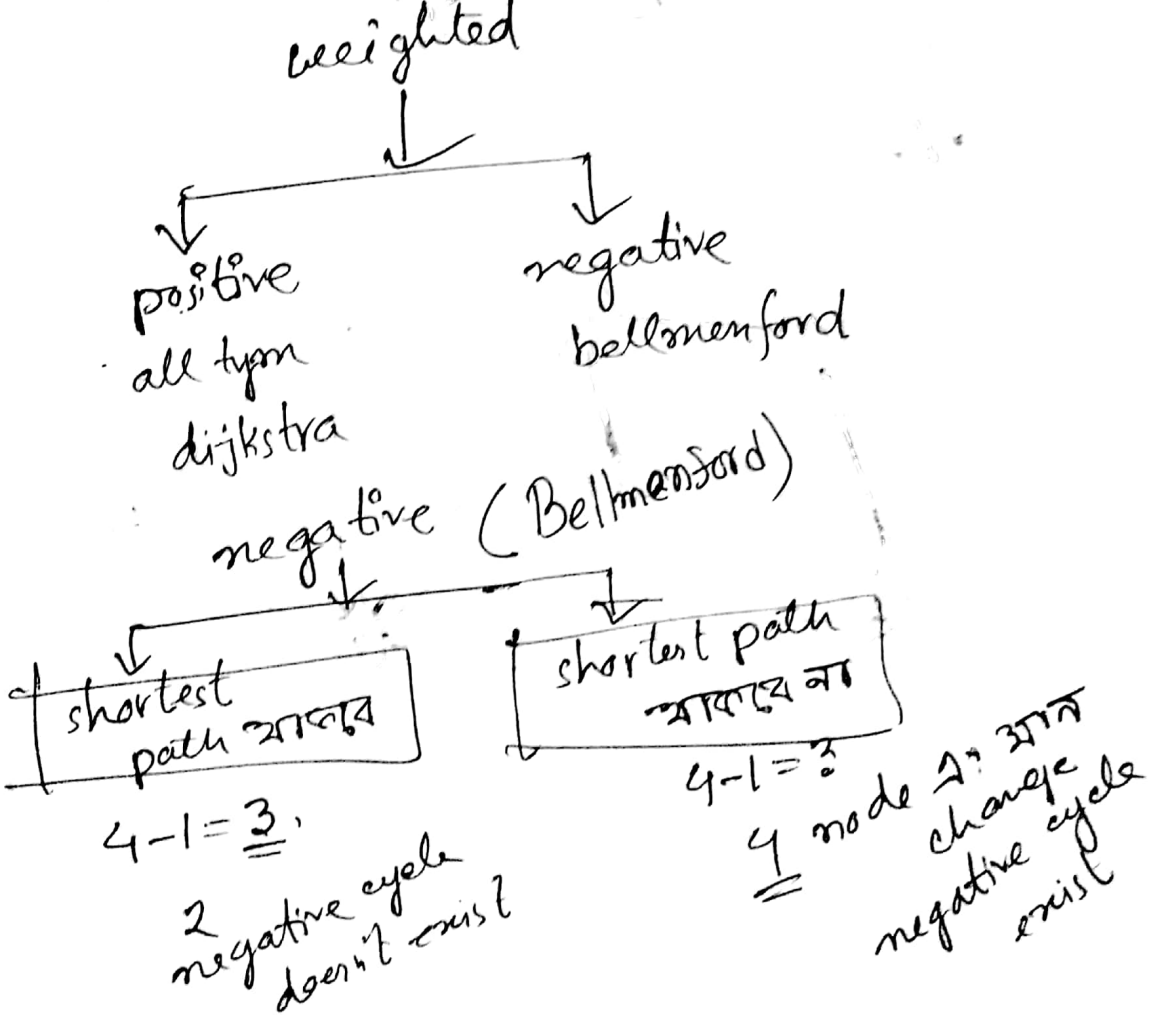
Shortest Path :

(i) BFS, DFS
(ii) Dijkstra
(iii) bellmen ford
(iv) DAG

{BFS: unweighted
{DFS: unweighted

{Dijkstra: weighted (positive)
{~~bellmenford~~: weighted (negative)

(4)



Counting Sort :—

কোনটা বসাবো তা নিব A থেকে

A:

0	1	2	3	4	5	6	7	8	9
0	2	3	4	1	0	2	5	4	5

C:

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

* কোন element কয়টা করে আছে

কোন ঘরে বসাবো তা নিব C_2 থেকে।

C_2:

0	1	2	3	4	5
~~2~~	~~3~~	~~5~~	~~6~~	~~8~~	~~10~~
~~1~~ 0	2	~~4~~ 3	5	~~7~~ 6	~~9~~ 8

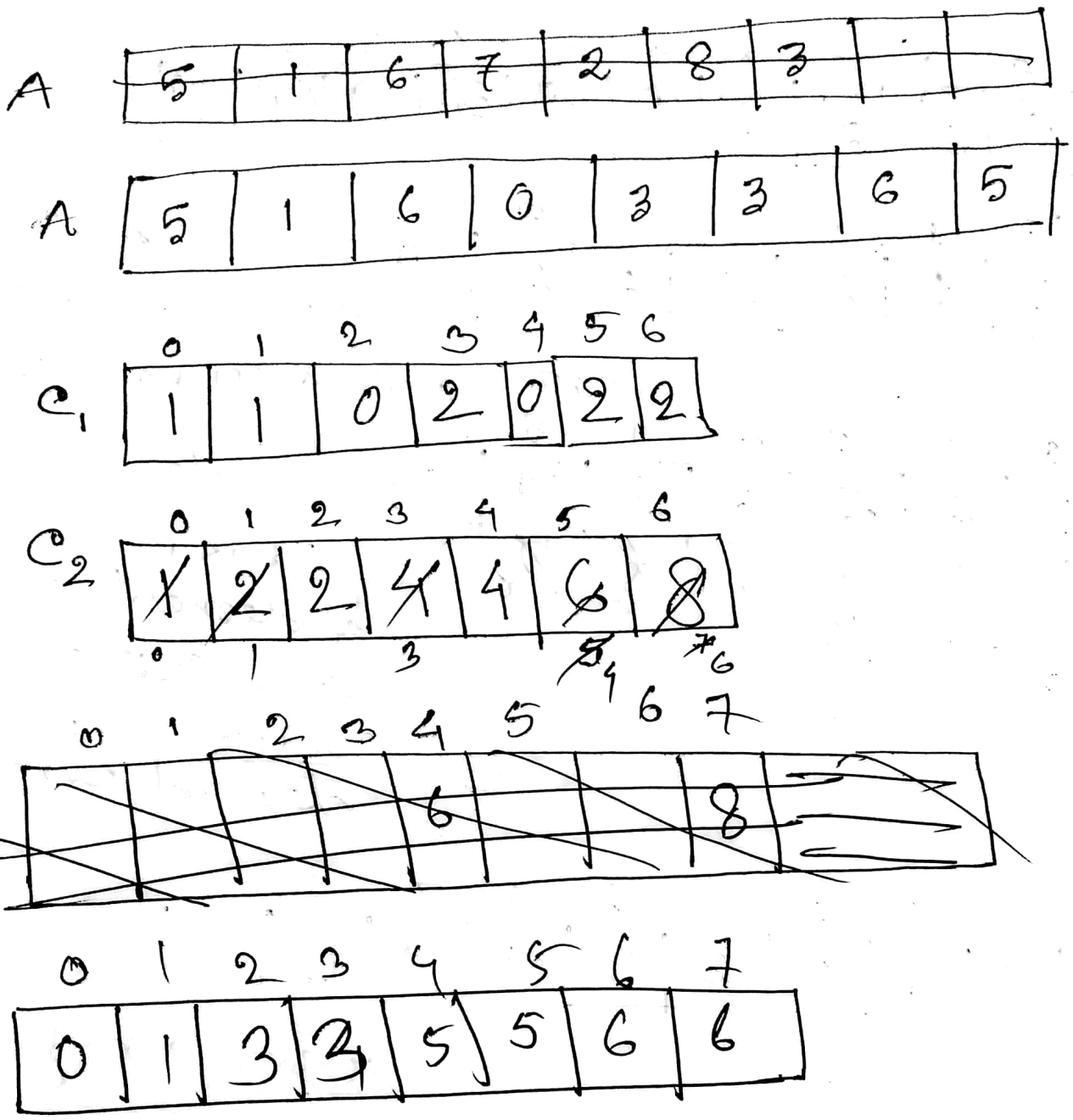
প্রতিটা element নিব। ঐ element এর চেয়ে ছোট অথবা সমান কয়টা element আছে

decrease করলাম because A তে কোন কিছু যদি 2nd tym থাকলে তা আবার B তে যাবে.

B:

0	1	2	3	4	5	6	7	8	9
0	0	1	2	2	3	4	4	5	5

* given array last থেকে element নেয়া strt ঐ element টা C_2 তে কয়টা আছে B তে বসাবো সাথে সাথে decrease



Short tips

i) P={a set of problems for which ~~they~~ there exist a correct algorithm that can solve in polynomial time}

ii) X={a set of problems for which there exist a correct algorithm that can solve in exponential time}

iii) R={a set of problems for which there hasn't exist a correct algorithm that can solve infinite time}.

iv) NP={a set of problems for which there exist a correct algorithm that varifies it in polynomial time}.

$P \subseteq NP$; $NP \subseteq PX$

$\therefore P \neq NP$

⊡ a set of 'P' problems are always 'NP' problems but a set of 'NP' problems wasn't always the 'P' problems.

(v) NPC ⇒ A problem $X \in NPC$ if and only if the following are true.

i) $X \in NP$

ii) $Y \in NPC$; $Y \leq X$; $\forall Y$

(৬) Traveling Salesman

→ Polynomial time এ solve হবে না।

→ একটা graph কে কত way তে travel করা যায়।

→ তার মানে যে node যাবে সেই node এই ফিরে আসবে।

But এ node এ, এ way তে 2nd time travel করা যাবে না। কিন্তু সব node visit করা লাগবে।

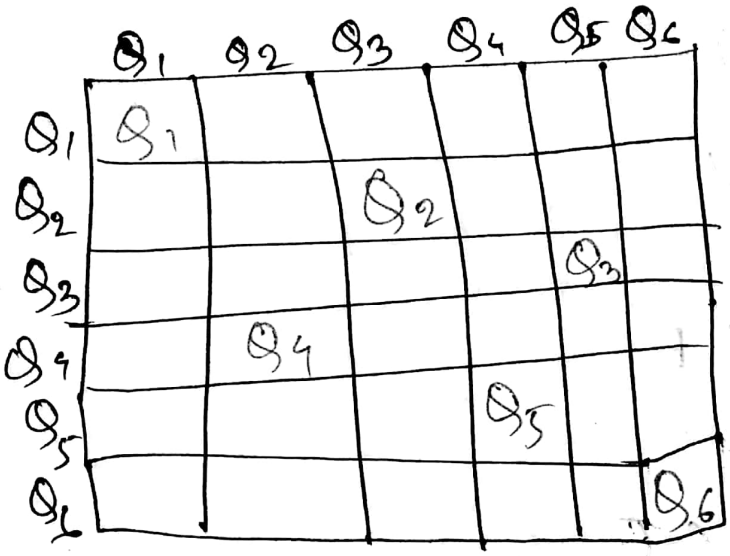
9) Queen Problem / Backtracking

→ n queens or যত বলবে তা $n_1 \geq n_2$ হবে।

→ Conflict Problems
  ↳ row conflict
  ↳ column conflict
    ↳ diagonal conflict.

→ Backtracking হল → কোন একটা way ধরে অগ্রসর করল যদি ভুল হয় ঐ way টা, mean to say goal এ reached না হয় then return আসবে backtracting বলে।

6 Queen Problem



1 - - -

11 - - - 12 - - - 13 - - -

131 - - - 132 - - - 133 - - 134 - - 135

1351 - - - 1352 - -

13521 - 13522 - 13523 -

135284 -

135241 135242 135243

135244 135245 135246

Ans: 135246