Prim's Algorithm

```
MST_PRIM(G, w, r)
O(V) ---> Q = G.V                          // create heap
O(V) ---> for each u ∈ G.V {
              u.key = ∞
              u.π = NILL
          }
O(1) ---> r.key = 0
O(V) ---> while Q ≠ ∅ {
O(V log V) ← log V ---> u = Extract_Min(Q)
V(ΔE · log V) ---> for each v ∈ G.Adj[u] {
                       if v ∈ Q and w(u,v) < v.key
                       {
                           v.π = u
O(E log V) ---> v.key = w(u,v)
                       }
                   }
          }
```

$V(\Delta E \cdot \log V) = V\Delta E \approx 2E$, $O(2E) = O(E)$

Every time we enter line 7, we select only neighbour nodes i.e. we call it ΔE (Delta E) ∴ Therefore $V\Delta E \approx 2E = O(E)$

$\boxed{V + V + V\log V + E\log V}$

Total time is

$O(|V|\log|V| + |E|\log|V|)$

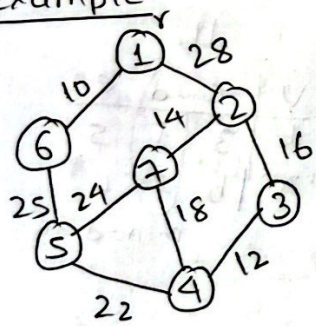
$= O(\log|V|\,(V+E))$

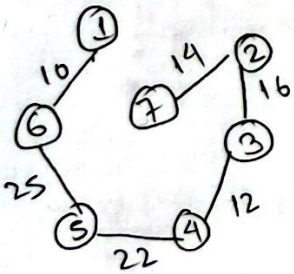
Here $E >> 1$

$= O(E\log|V|)$

$= O(E\log V)$

Example





① $Q =$

V	1	2	3	4	5	6	7
key	∞	∞	∞	∞	∞	∞	∞
π	Nill	Nill	Nill	Nill	Nill	Nill	Nill

Let say N = Nill

②

V	1	2	3	4	5	6	7
K	0	∞	∞	∞	∞	∞	∞
π	N	N	N	N	N	N	N

min = u

⇒ ③

V	2	3	4	5	6	7
K	28	∞	∞	∞	10	∞
π	1	N	N	N	1	N

min u = 6

V	2	3	4	5	7
K	28	∞	∞	25	∞
π	1	N	N	6	N

min = 5 = u

⇐

V	2	3	4	7
K	28	∞	22	24
π	1	N	5	5

min = 4 = u

V	2	3	7
K	28	12	18
π	1	4	4

min = 3 = u

⇒

V	2	7
K	16	18
π	3	4

min = 2 = u

⇒

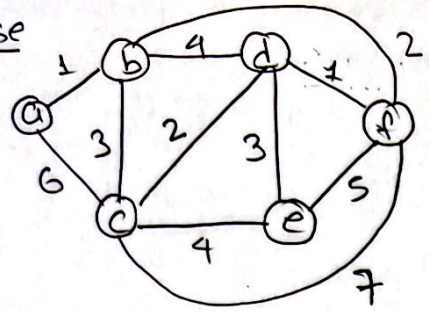
V	7
K	14
π	2

min = 7 = u

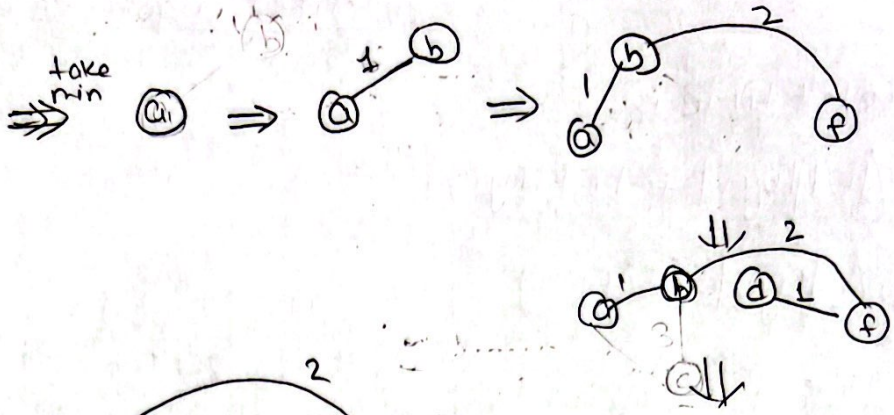
Scanned with CamScanner

Exercise

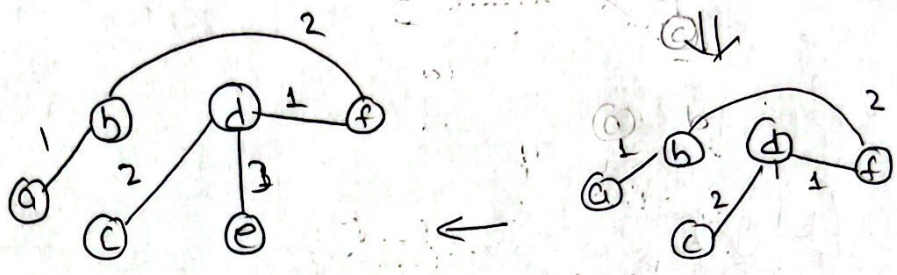
1 way



take min



6, 4, 3, 3, 5, 4



minimum cost $= 1+2+1+3+2$
$= 9$

2nd method

V	a	b	c	d	e	f
K	0	∞	∞	∞	∞	∞
π	N	N	N	N	N	N

min = a = 4

⇒

V	b	c	d	e	f
K	1	6	∞	∞	∞
π	a	a	N	N	N

min = b

⇒

V	c	d	e	f
K	3	4	∞	2
π	b	b	∞	b

min = f

V	c	d	e
K	3	1	5
π	b	f	f

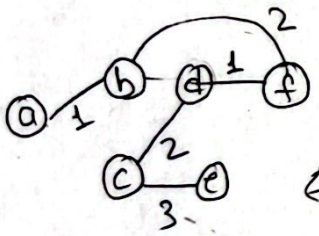
min = d = 4

V	c	e
K	2	3
π	d	d

min = c

V	e
K	3
π	d

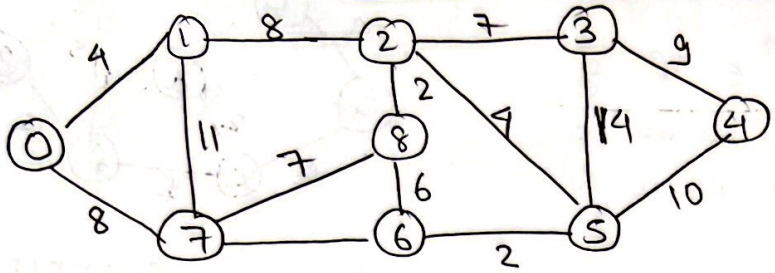
min = e

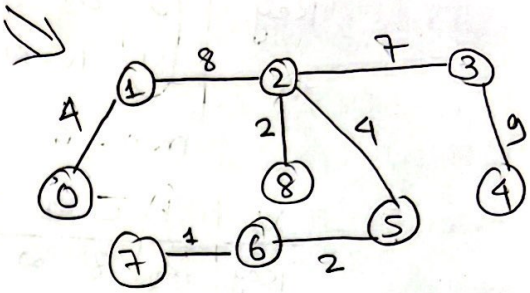


minimum cost

$1+2+1+2+3 = 9$

Prim's Exercise





minimum cost from 0 to 9 is

$4+8+2+4+1+2+7+9$

$= 37$

(v)

vertex	0	1	2	3	4	5	6	7	8
K	0	∞	∞	∞	∞	∞	∞	∞	∞
π	N	N	N	N	N	N	N	N	N

min=0

V	1	2	3	4	5	6	7	8
K	4	∞	∞	∞	∞	∞	8	∞
π	0	N	N	N	N	N	0	N

min=4

V	2	3	4	5	6	7	8
K	8	∞	∞	∞	∞	8	∞
π	1	N	N	N	N	0	N

min=2

V	3	4	5	6	7	8
K	7	∞	4	∞	8	2
π	2	N	2	N	0	2

min=8

V	3	4	5	6	7
K	7	∞	4	6	7
π	2	N	2	8	8

min=5

V	3	4	6	7
K	7	10	2	7
π	2	5	5	8

min=6

V	3	4	7
K	7	10	1
π	2	5	6

min=7

V	3	4
K	7	10
π	2	5

min=3

V	4
K	9
V	3

min=4

Scanned with CamScanner

Exam sol^n Prim's Algo

Vertex(v)	a	b	c	d	e	f	g
Key	0	∞	∞	∞	∞	∞	∞
π	N	N	N	N	N	N	N

min = a

v	b	c	d	e	f	g
Key	4	8	∞	∞	∞	∞
π	a	a	N	N	N	N

min = b

v	c	d	e	f	g
K	8	8	10	∞	∞
π	a	b	b	N	N

min = c

v	d	e	f	g
K	2	10	1	∞
π	c	b	c	N

min = f

v	d	e	g
K	2	5	2
π	c	f	f

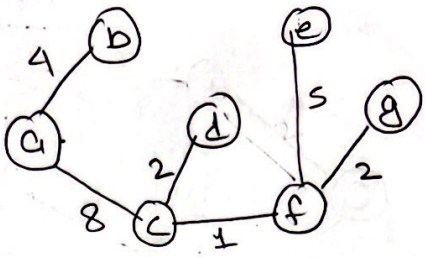
min = d

v	e	g
K	5	2
π	f	f

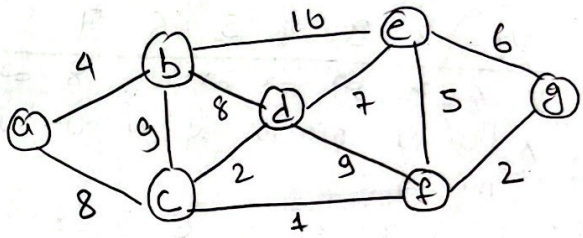
min = g

v	e
K	5
π	f

min = e



minimum spanning cost =
4+8+2+1+5+2
= 22 Ans



Scanned with CamScanner

KRUSKAL ALGORITHM

```
MST - KRUSKAL'S ALGORITHM(G, w)
{
    A = { }                                   ——— O(1)
    for each v ∈ V[G]
        do MAKE-SET(v)                        ——— O(V)
    Sort the edge of E into non
    decreasing order by wt                    ——— O(E log E)
    for each edge (u,v) ∈ E[G]
        do if FINDSET(u) ≠ FINDSET(v)
            then A = A ∪ {(u,v)}
            UNION(u,v)
    return A
}
```

Initialization of empty set

प्रति आउट vertex है उसको singleton set बनाओ

E log E

E log V

Total time complexity

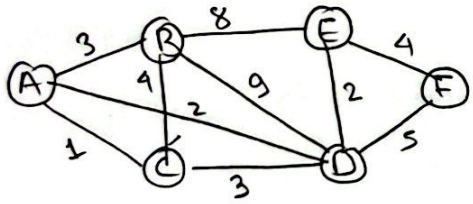
$O(1) + O(V) + O(E\log E) + O(E\log V)$

Since $|E| \le |V|^2 = \log E = O(2\log V) = O(\log V)$

$= O(E\log V) + O(E\log V)$

$= O(E\log V)$

Given Graph



$G = (V, E)$

Algorithm

① Step 1 $A = \{ \}$

② Step 2 (A) (B) (C) (D) (E) (F) → make singleton set

③ Step 3

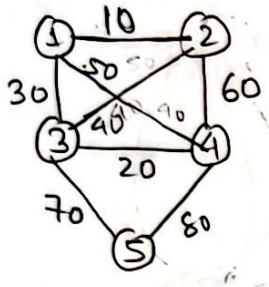
Edges E[G]	weight
A - C	1
A - D	2
E - D	2
A - B	3
C - D	3
B - C	4
E - F	4
D - F	5
B - E	8
B - D	9

Ascending with repetition

Step 4

edge = A - C
u v

KRUSKAL ALGORITHM : Minimum Spanning Tree



no. of vertices = $|V|$ = 5

no. of edges = $|V| - 1$ = 5 − 1 = 4

Sort vertices according to edge weight

(1,2) → 10

(3,4) → 20

(1,3) → 30

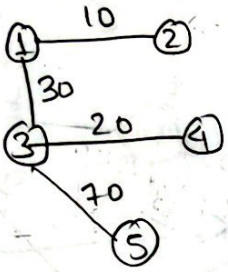
× → (3,2) → 40

× → (1,4) → 50

× → (2,4) → 60

(3,5) → 70

× → (4,5) → 80



Total weight = 10 + 30 + 20 + 70

= 130

Steps

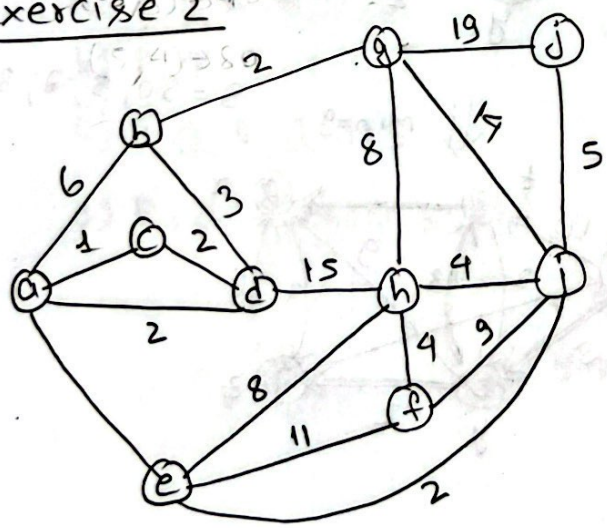
⇒ Edge sort depending on ascending order wt

⇒ connect from smallest edge but make sure that should not form cycle. (Exclude the edge that forms cycle) in spanning tree.

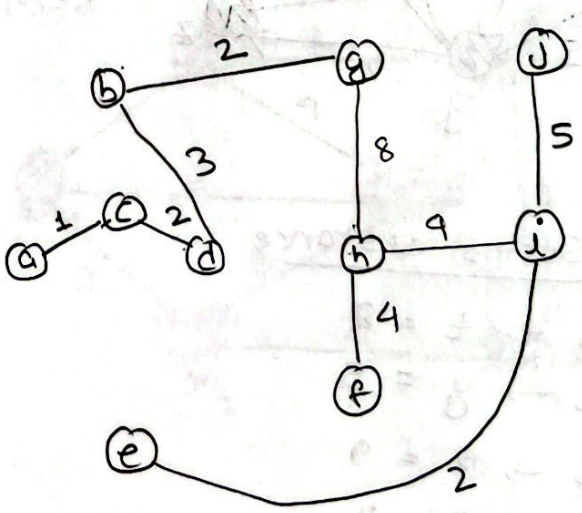
we cant not take (X) → because it forms loop in minimum spanning tree

Srgn vertex

Exercise 2



⇒



1 + 2 + 3 + 2 + 8 + 4 + 5 + 4 + 2

= 31

In exam, 1st draw all vertex and then connect depending on the minimum weight vertex.

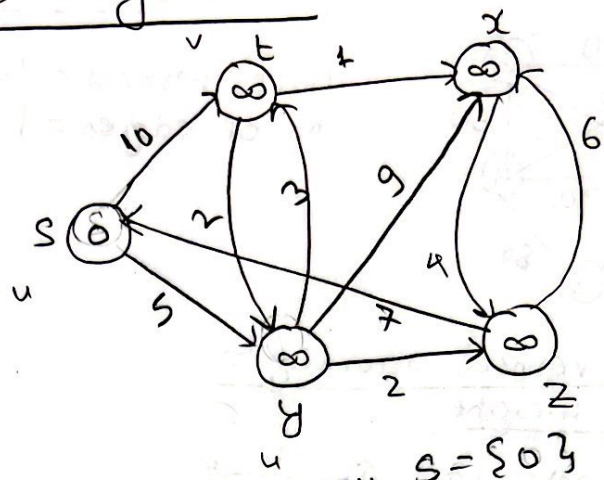
Scanned with CamScanner

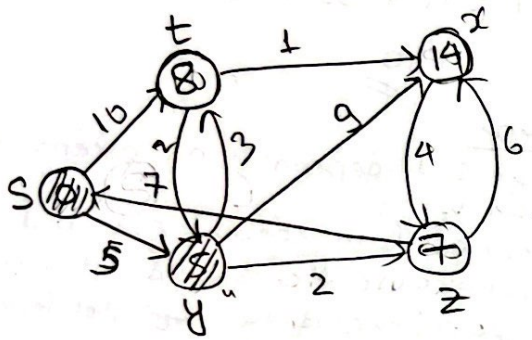
Dijkstra's Algorithm → Single source Sortest path Algorithm

Relaxation

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

then $du = d(u) + w(u,v)$

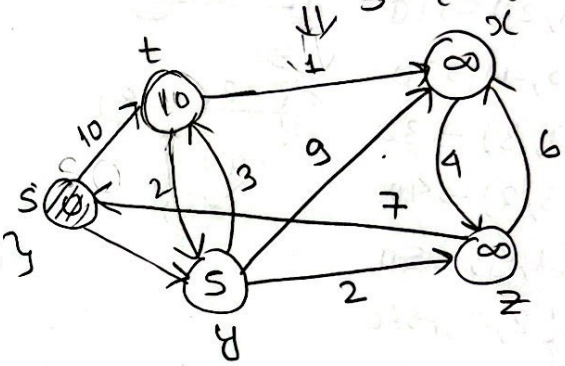




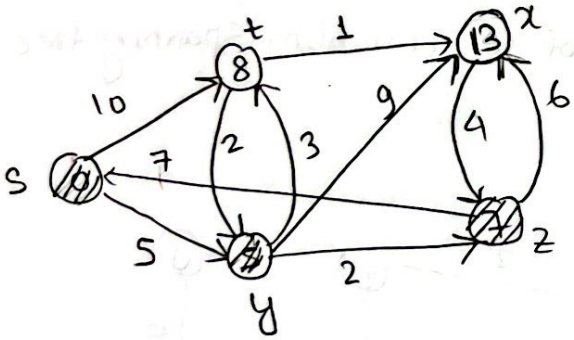
mindis = 5

$S = \{0, 5\}$

⇓ $S = \{0\}$

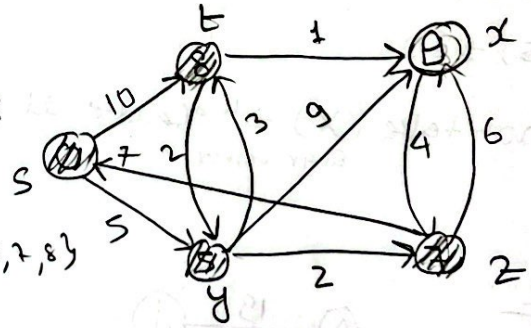


⇓ min = 7, $S = \{0, 5, 7\}$

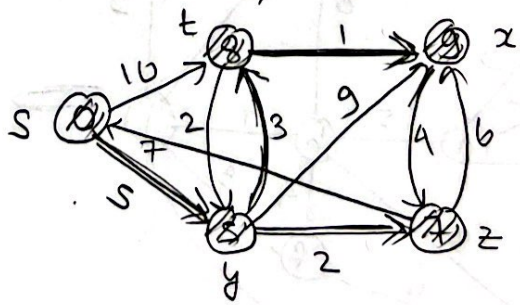


min = 8 ⇒

$S = \{0, 5, 7, 8\}$



⇓ min = 9. $S = \{0, 5, 7, 8, 9\}$



Sortest distance

$s \to t = 8$

$s \to y = 5$

$s \to x = 9$

$s \to z = 7$

Scanned with CamScanner

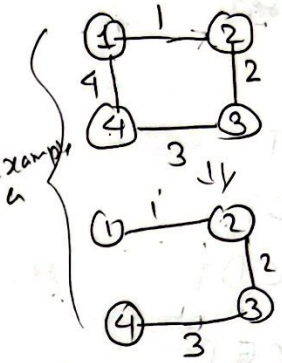
Q Let $G=(V,E)$ be any connected undirected edge-weight graph. The weight of the edge in E is positive and distinct.

✓ a) The minimum spanning tree of G is always unique

✗ b) The sortest path betn any two vertices of G is always unique.

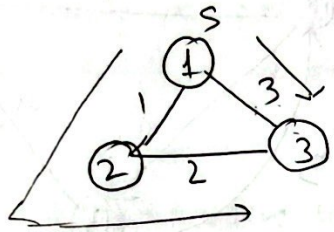
Which of the following statement is true.

example G



Spanning tree

example



$1 \xrightarrow{1} 2 \xrightarrow{2} 3 \Rightarrow 3$

$1 \rightarrow 3 \Rightarrow 3$

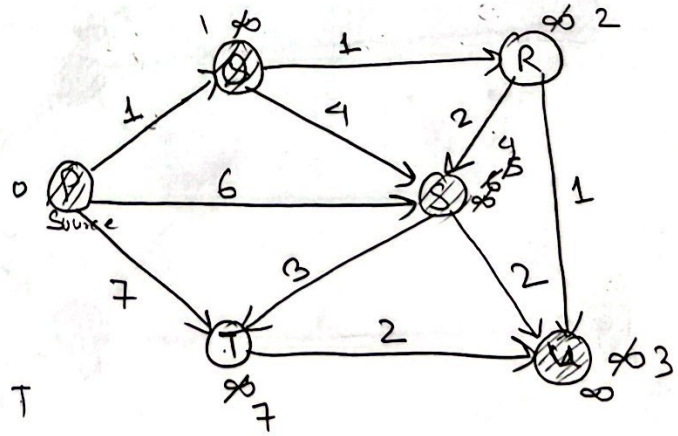
Q In what order do the nodes get included in the set of vertices for which the sortest path distances are finalized?

a) P, Q, R, S, T, U

✓ b) P, Q, R, U, S, T

c) P, Q, R, U, T, S

d) P, Q, T, R, U, S

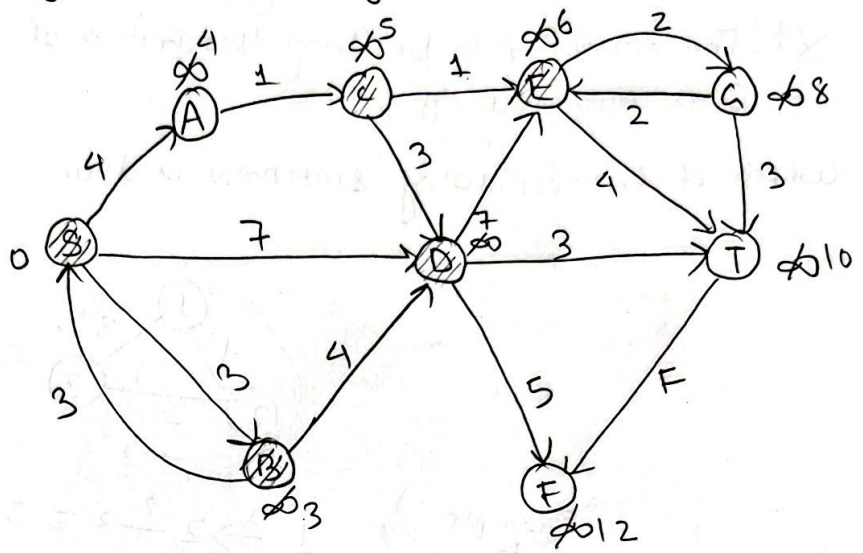


P Q R U S T

Scanned with CamScanner

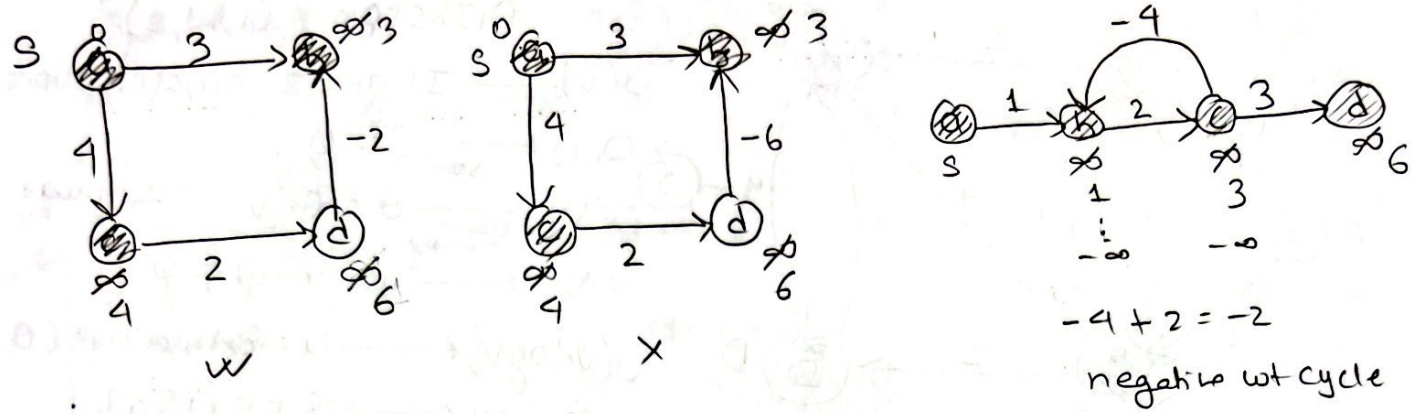
Q consider the directed graph shown in the figure below. There are multiple Sortest path betn vertices S and T which will be reported by Dijkstra's Sortest path algorithm? Assume that, in any iteration, the Sortest path to a vertex V is updated only when a strictly shorter path to V is discovered.

a) SDT
b) SBDT
c) SACDT
d) SACET



S → A → C → E → T
S → D → T X
S → B → D → X
7 7

Q Drawback of Dijkstra Algorithm



In any graph, if the weight of the edges is negative, it may or may not give correct result using Dijkstra's Algorithm.

But, if the graph contain any negative weight cycle, then it is confirmed that it will give incorrect result.

This is why Dijkstra Algorithm is applied to Grap that contains only positive weight edge.

Exercise : Compute correct shortest path distance to

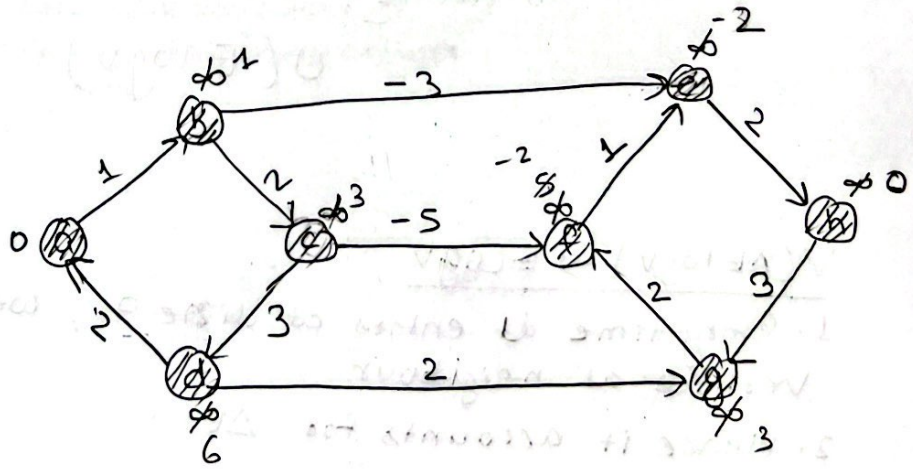
(a) only vertex a

(b) only vertex a, e, f, g, h

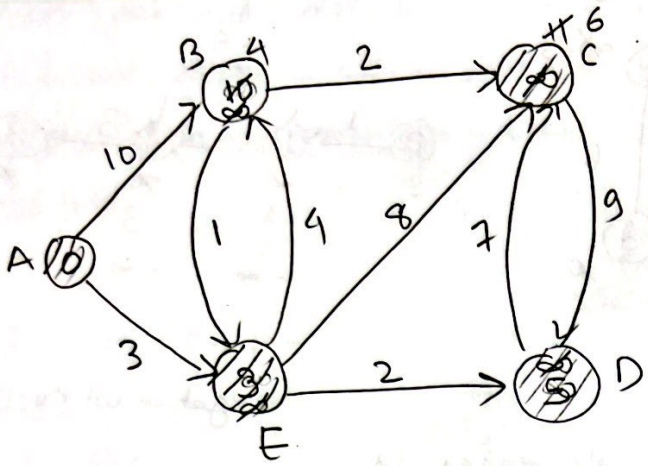
(c) only vertices a, b, c, d

(d) all the vertices ✓

Correct



Dijkstra's Algorithm Time Complexity



DIJKSTRA (G, w, s) (w: weight matrix, s: source)

$O(V) \leftarrow$ INITIALIZE SINGLE SOURCE(G, s)

$O(1) \leftarrow S = \phi$ (set)

$O(V) \leftarrow Q = G.V$ (create min heap) — Build of min heap

$V \leftarrow$ while $Q \neq \phi$

$O(V \log V) \leftarrow u =$ Extract min(Q)

$O(1) \leftarrow S = S \cup \{u\}$

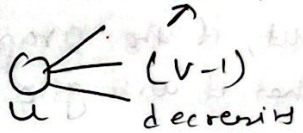
$O(E \cdot \log V) \leftarrow V(\Delta E \log V)$ for each vertex $v \in G.Adj[u]$ ($V-1$)

d[u]

Relax(u, v, w)

$S = \{A, E, B, D, C\}$

This is the order of Deletion from the given algorithm



Time complexity $= \underbrace{O(V)}_{\text{initialization}} + \underbrace{O(1)}_{\text{set case}} + \underbrace{O(V)}_{\text{Build heap}} + \underbrace{O(V \log V)}_{\text{min Extract V times}} + \underbrace{O(E \log V)}_{\text{relax times}}$

$= O((V+E) \log V)$

$E > V$

$= O(E \log V)$

internally call decrease function.

Relaxation [illegible] call [illegible] depending on the number of edges.

$V(\Delta E \log V) \rightarrow E \log V$

1. Every time it enters at line 7, we get small number of vertices as neighbour
2. Hence it accounts for ΔE
3. Hence it takes $|V|(\Delta E \log V)$

So, the total running time is $O(|V| \log V + |E| \cdot \log V)$

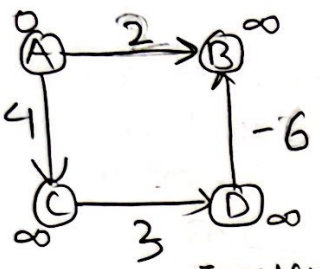
$= O((|V| + |E|) \log V)$

Since, the number of edges is higher then the number of vertices, it becomes $O(E \cdot \log V)$

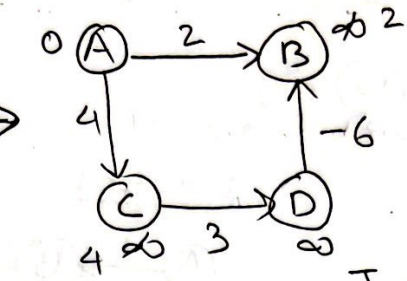
= Total running time of Dijkstra Algorithm

Scanned with CamScanner

Bellman Ford Algorithm

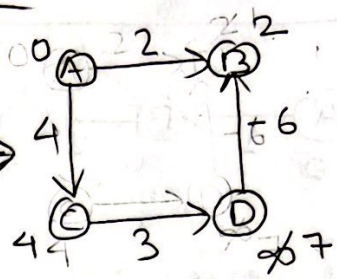


Input graph



O/P graph

1st time



Input → Output

2nd time

WC

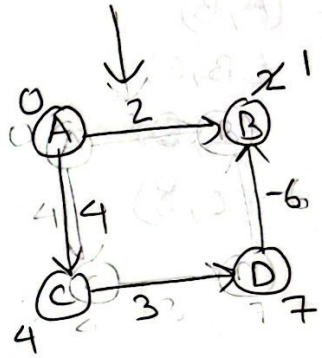
$V = 4$

relax $= V - 1$

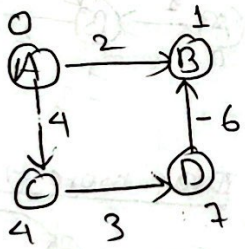
$= 4 - 1$

$= 3$

(A,B) (A,C) (C,D) (D,B)



3rd time



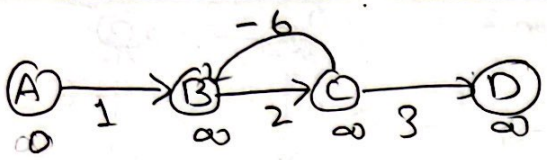
← Check if the correct answer for negative wt

no change means this algorithm for this cycle giving the correct answer.

	DJ	BF
+ve wt	✓	✓
-ve wt	✓/✗	✓
-wt cycle	✗	✗

Scanned with CamScanner

DrawBack of Bellman Ford Algorithm

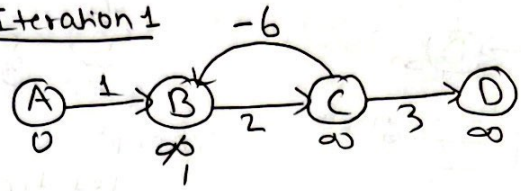


no. of vertex = 4

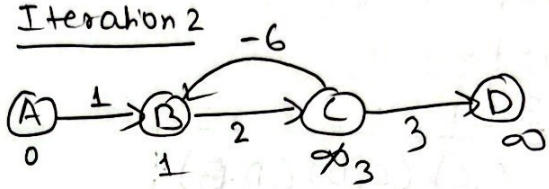
no. of Iteration = 4 - 1 = 3
time to select the vertex

(A, B)
(B, C)
(C, D)
(C, B)

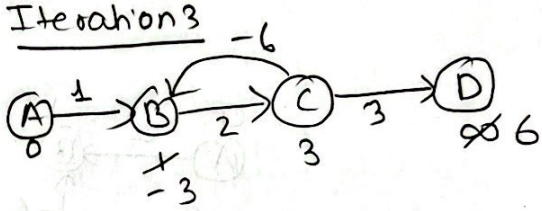
Iteration 1



Iteration 2



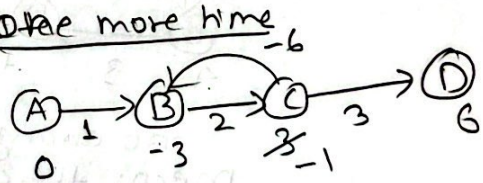
Iteration 3



A/c to algorithm the sortest path is

but Check whether correct or not

Take more time

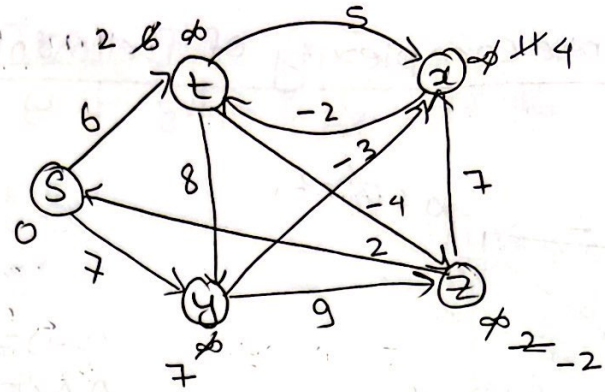


The graph contains negative wt cycle

$-6 + 2 = -4$

Scanned with CamScanner

Exercise



Number of vertex = 5
Number of relaxation for each vertex = 5 − 1 = 4

(t, x)
(t, y)
(t, z)
(x, t)
(y, x)
(y, z)
(z, x)
(z, s)
(s, t)
(s, y)

Shortest Distance

s → t => 2
s → y = 7
s → z = -2
s → x = 4

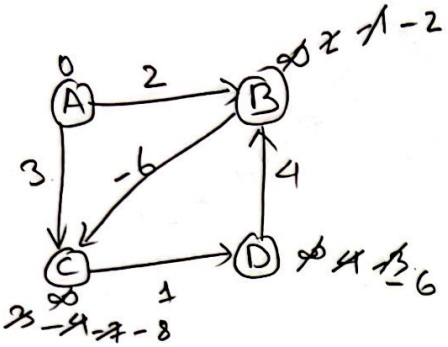
Q How to detect negative wt cycle in graph?

→ If we relax nodes for V. time (one more times), then if the value changes means there is negative cycle in graph and if the value is not change then means that is negative wt.

Thus, Bellman Ford Algorithm is capable to detect whether the negative wt is just the negative wt or forming a cycle.

Scanned with CamScanner

Time complexity of BellmanFord Algorithm



(A,B), (A,C) (C,D) (D,B) (B,C)

0 A	1 B	2 C	3 D
0	∞	∞	∞

```
Bellman-Ford(G,w,s)
O(V) <-- Initialize-single source(G,s)
         for i=0 to |G.V|-1          -> V-1
(V-1)E(O1) <-   for each edge (u,v) ∈ G.E
O(VE)
                O(1) <-- Relax(u,v,w)
```

```
for each edge (u,v) ∈ G.E
    if d(v) > d(u) + w(u,v)
        return false;
return true;
```

O(E) <- relax one more time to check if resort path is true

Check if it is negative wt or forming negative wt cycle.

dominating one

$O(VE) + O(E)$

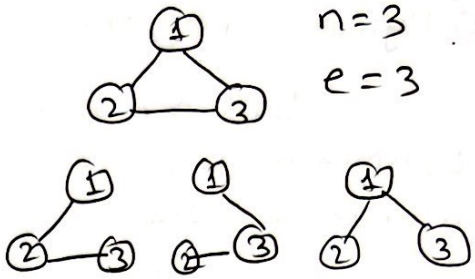
time complexity $O(VE)$

Scanned with CamScanner

Spanning Tree

A connected subgraph H of given graph G is said to be spanning tree iff

1) H should contain all vertex of G
2) H should contain (n−1) edge
3) H should be connected
4) Total number of spanning tree n^{n-2}



$n=3$
$e=3$

Q How many maximum no. of edges can we remove while creating spanning tree

$\Rightarrow$ $E-V+1 = 3-3+1 = 1$

AIGORTTHM

```
A = ∅
while A does not form a spanning tree {
    find an edge (u,v) that is safe for A
    A = A ∪ {(u,v)}
    return A;
}
```

Scanned with CamScanner

Dynamic Programming

1) Greedy method
2) Dynamic Programming

Both are used for solving optimization problem
Optimization problem are those which requires either minimum or maximum result.

1) Greedy method -> Here we try to follow the predefined procedure to get best result.
eg Kruskal's algo -> always select the minimum edge wt and that gives the best result.
Dijkstra sortest path algorithm -> always select the sortest path vertex and continue relaxing the vertices, so, we get a sortest path.

2) Dynamic programming -> we try to find out all possible solution and then pick up the best/optimal solution. This approach is different and time consuming compared to greedy algorithm.
For any problem, there may be many feasible solution. so, we try all solutions and pick up the optimal solution. Dynamic programing are solved using Iteration. It follows principle of optimality.
principle of optimality says that a problem can be solved by taking sequence of decisions to get optimal solution. DP saves computation in each iteration.

In greedy approach, decision is taken one time, where as in Dynamic programing, we take decision in every steps while solving the problems.

-> DP -> 1) we should be able to break the original problem to smaller subproblems that have same structure
2) optimal substructure of the problem -> The solution of the problem must be a composition of subproblem itself.
(Subproblems itself needs to be optimal)
3) Subproblem overlap -> Optimal subproblems to unrelated problems can contain subproblems in common.

Scanned with CamScanner

0/1 Knapsack Problem

Object	1	2	3	4
Profit	1	4	5	7
Weight	1	3	4	5

$W = 7$

object	Profit	Weight	w=0	w=1	w=2	w=3	w=4	w=5	w=6	w=7
n=0	0	0	0	0	0	0	0	0	0	0
n=1	1	1	0	1	1	1	1	1	1	1
n=2	4	3	0	1	1	4	5	5	5	5
n=3	5	4	0	1	1	4	5	6	6	9
n=4	7	5	0	1	1	4	5	7	8	9

total wt
$7 - 4 = 3$
remaining wt
$3 - 3 = 0$

Total object	1	2	3	4
	0	1	1	0

$$V[i,j] = \max\left[V[i-1,j],\ v_i + V[i-1, j-w_i]\right]$$

if $j - w_i \geq 0$

Time complexity $= (n+1)(w+1)$

$O(nw)$

Scanned with CamScanner

$a = \log_b b \quad b = 2^a$

Runtime of 0/1 Knapsack

Runtime of 0/1 Knapsack using above solution is $O(nw)$ where n is the number of items and w is the limit on the weight

$$\text{size} = \log_b w$$
$$w = 2^{\text{size}}$$

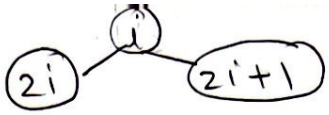
where size = number of bits to represent 'w'
b = base

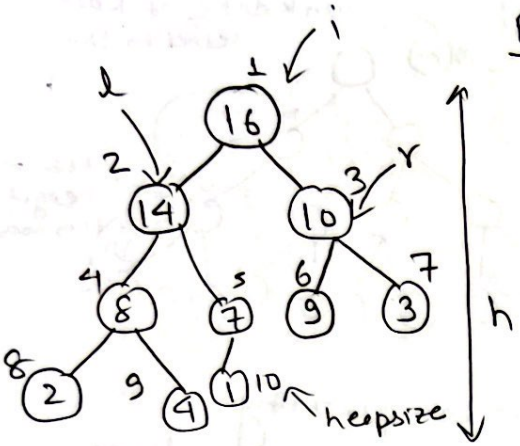
$$\therefore T(n) = O(n \cdot w)$$
$$= O(n \times 2^{\text{size}})$$
$$= O(nw)$$

Explanation: In the DP approach, we create a 2D array (often referred to as a memoization table) of size $(n+1) \times (w+1)$ to store the solutions to subproblems. You iterate through each item and each possible capacity of the knapsack, filling in the table based on the optimal choices. This leads to a time complexity of $O(nw)$

Scanned with CamScanner

MAX Heapify





$2\log n + \log n = 3\log n$

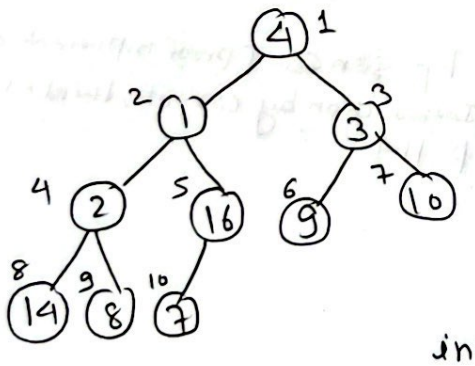
$O(\log n)$

node = i
parent = i/2
Left child = 2i
right child = 2i+1

```
MAX-HEAPIFY(A, i)
    l = LEFT(i)          — 2i
    r = RIGHT(i)         — 2i+1
    if (l ≤ A.heapsize and A[l] > A[i]
        largest = l
    else largest = i
    if (r ≤ A.heapsize and A[r] > A[largest])
        largest = r
    if largest ≠ i
        Exchange(A[i], A[largest])
        MAX-HEAPIFY(A, largest)
```

Build Heap

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7



```
Build Max Heap(A, i)
O(1)      ←— A.heapsize = A.Length
O(n/2)    ←— for i = [A.length/2] down to 1
O(log n)  ←— max Heapify(A, i)
```

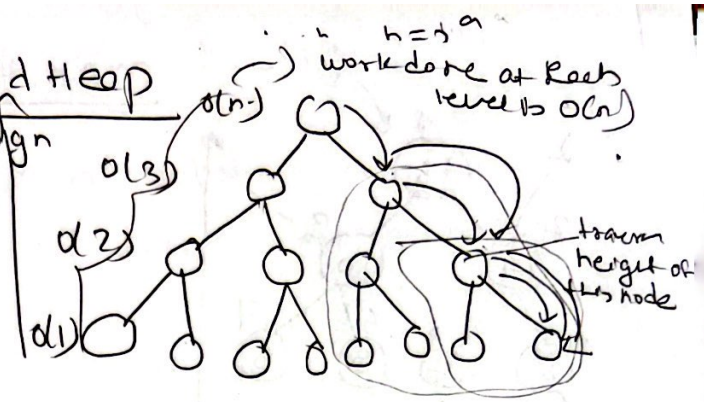
internal node	leaf node
1 to n/2	n/2 + 1 to n

$\Downarrow$

$O(n \log n)$

Scanned with CamScanner

Analyzing Build Heap



Wq The running time of Heapify is
$O(n)$ ← a tighter bound
$O(n\log n)$ ← a loose bound

The height of each level of tree is different. So, calculating the time complexity at each node at a particular level and finally summing them will give a tighter bound of $O(n)$

The number of nodes at each level of tree is

$$= \left\lceil \frac{n}{2^{h+1}} \right\rceil$$ where n is the total number of nodes in tree

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

$$= \sum_{h=0}^{\log n} \frac{n}{2^h \cdot 2} c \cdot h \Rightarrow \frac{cn}{2} \sum_{h=0}^{\log n} \frac{h}{2^h} =$$

$$\leq \frac{cn}{2} \boxed{\sum_{h=0}^{\infty} \frac{h}{2^h}}$$

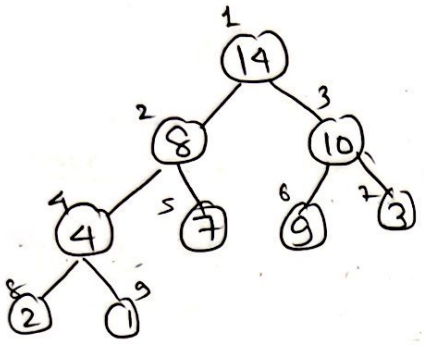
2 This is hp series [proof is present a Introduction by corrnen, third edit p: 114

$$\leq \frac{cn}{\cancel{2}} \cdot \cancel{2}$$

$$\leq cn$$

Scanned with CamScanner

Heap Deletion



max=14

```
Heap Extract max(A) {
    if (A.Heapsize < 1)
        "Heap Underflow"
    else
        max = A[1]
        A[1] = A[A.heapsize]
        A.heapsize = A.heapsize - 1
    Max Heapify (A, 1)
    return max
}
```

constant

logn → Max Heapify (A, 1)

c → return max

Time complexity = $O(\log n)$

Heap Sort

- BuildHeap takes $O(n)$
- Each of $(n-1)$ calls to heapify() takes $O(\log n)$ time (takes $O(\log n)$ time)
- Thus the total time taken by HeapSort()

$$= O(n) + (n-1)\,O(\log n)$$
$$= O(n) + O(n \log n)$$
$$= O(n \log n)$$

A = [1 --- n]

```
HeapSort(A) {
    BuildHeap(A);                    O(n)
    for(i = length(A) downto 2)      O(n)
    {
        swap(A[1], A[i]);            O(1)
        heap_size(A) -= 1;           O(1)
        max Heapify(A,1);            O(logn)
    }
}
```

Scanned with CamScanner

Q Sum of subset problem

$\underbrace{\{3,2,1,7\}}_{n}$ sum = 6 (K)

Brute Force Time complexity $O(2^n)$

Solving using Dynamic programing

- Optimal substructure
- Overlapping subproblem

$SS(n,s) =$
SumofSubset

False $n=0$, $S \neq 0$ (no. of element, set)

True $n=0$, $S=0$

True $n \neq 0$, $S=0$

$SS(n,s) = SS(n-1,s)$ if $S<V[n]$

$SS(n-1, S-V[n])$ OR $SS(n-1,s)$ } if $S \geq V[n]$

$\{3,2,1,7\}$ Sum=6

we can not include

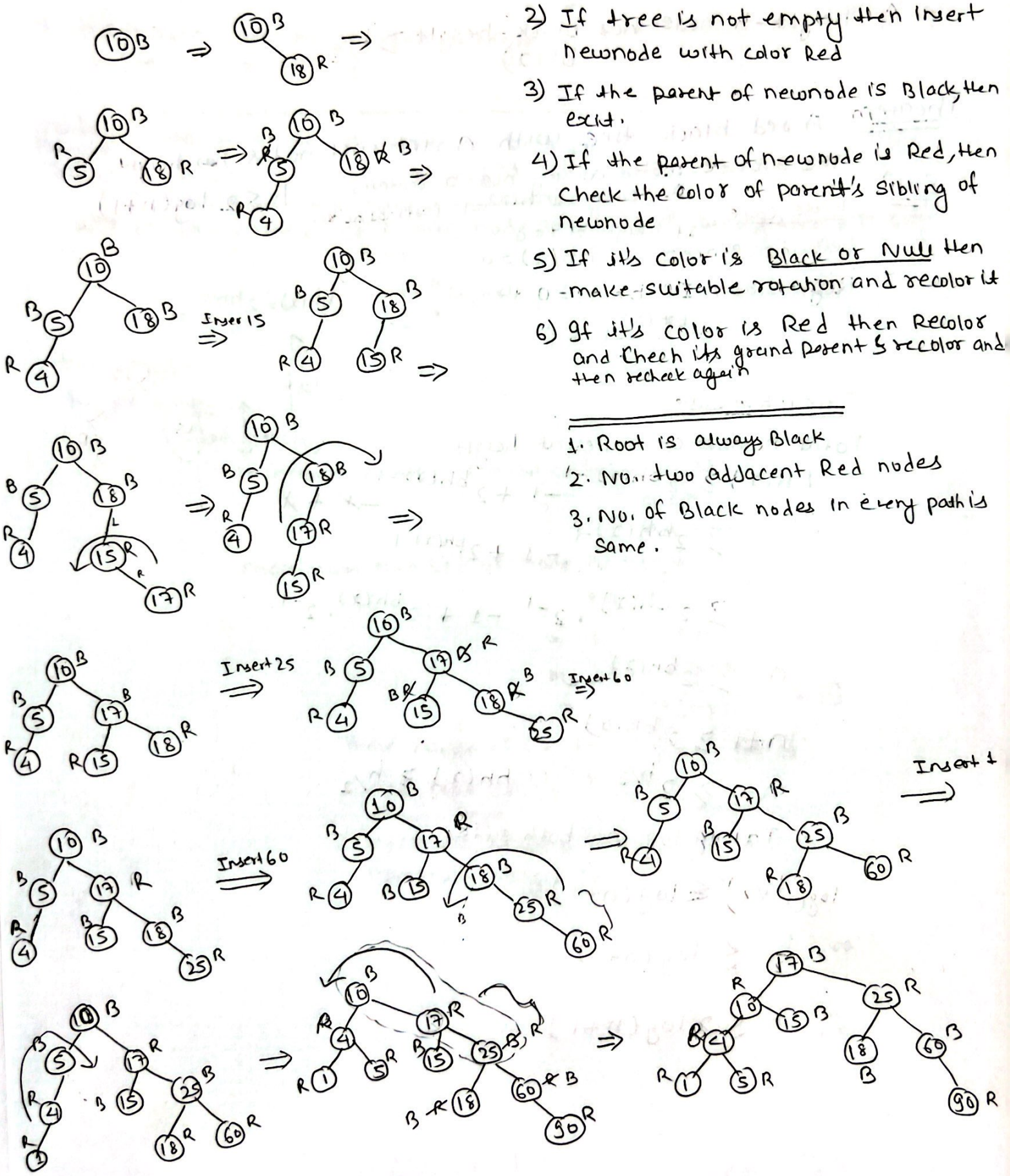
RB Insert

10, 18, 5, 4, 15, 17, 25, 60, 1, 90

1) If tree is empty then create newnode as root node with color Black
2) If tree is not empty then Insert newnode with color Red
3) If the parent of newnode is Black, then exit.
4) If the parent of newnode is Red, then Check the color of parent's sibling of newnode
5) If it's color is Black or Null then make suitable rotation and recolor it
6) If it's color is Red then Recolor and Check its grand parent & recolor and then recheck again

---

1. Root is always Black
2. No. two adjacent Red nodes
3. No. of Black nodes in every path is same.



Scanned with CamScanner

Q What is the minimum black height of a node with height h.?

→ A height-h node has black-height $\geq h/2$
$bh(x)$

Theorem A red black tree with n internal nodes has height $h \leq 2\log(n+1)$

Sol^n The subtree rooted at any node x contains at least $2^{bh(x)}-1$ internal nodes.

Base Case

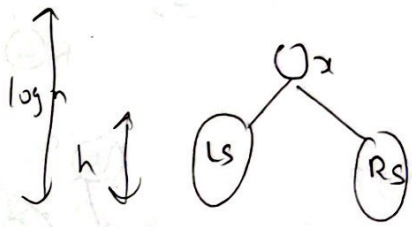
If height of $x = 0$

Leftside : since $x = 0$, $bh(x) = 0$

Right side : If $bh(x) = 0$ then,

$$2^{bh(x)} - 1 = 0$$

$$bh(x) = 2^{bh(x)} - 1$$



Inductive Case

Total number of nodes at height

$$bh(x) \geq 2^{bh(x)-1} - 1 + 2^{bh(x)-1} \quad -X + X$$

$$\geq 2^{bh(x)-1} - 1 + 2^{bh(x)-1}$$

$$\geq 2^{bh(x)} \cdot 2^{-1} - 1 + 2^{bh(x)} \cdot 2^{-1}$$

$$n \geq 2^{bh(x)} - 1$$

$$n+1 \geq 2^{bh(x)}$$

$$n+1 \geq 2^{h/2} \quad \because bh(x) \geq h/2$$

Taking log on both sides

$$\log(2^{h/2}) \leq \log(n+1)$$

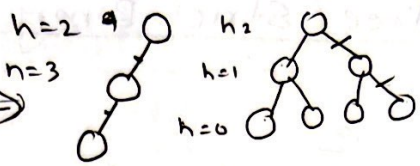
or, $\frac{h}{2} \leq \log(n+1)$

$$\therefore h \leq 2\log(n+1)$$

Scanned with CamScanner

Binary Tree

minimum number of nodes at ht h binary tree $(n) = h+1$



$h = 2$

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

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

$= 2^3 - 1 = 8 - 1 = 7$

maximum number of nodes at height h is $n = 2^{h+1} - 1$

node ~~height~~ range

The total no. of nodes in binary tree range = $h+1 \le n \le 2^{h+1}$

कोई भी binary tree मे इससे कम (h+1) node बनाएगे तो वो binary tree नही होगी and

" " " " ज्यादा 2^{h+1} " " " " " "

formula

height range

We know,

minimum number of node at height h is :

$n = h+1$

$h = n-1$ —(1)

maximum number of node at height h is

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

or, $2^{h+1} = n+1$

or, $h+1 = \log(n+1)$

or, $h = \log(n+1) - 1$ —(1)

There range of height in binary tree is

$$\underbrace{\log(n+1)-1}_{\text{minimum height}} \le h \le \underbrace{n-1}_{\text{maximum height}}$$

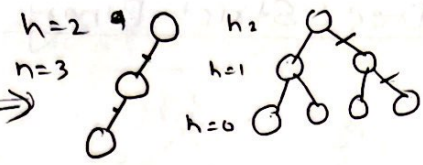
∴ minimum height of binary tree is $\log n$

maximum height of binary tree is $n-1$

Scanned with CamScanner

Binary Tree

minimum number of nodes at ht h binary tree $(n) = h+1$



$h = 2$

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

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

$= 2^3 - 1 = 8 - 1 = 7$

maximum number of nodes at height h is $n = 2^{h+1} - 1$

node ~~height~~ range

The total no. of nodes in binary tree range = $h+1 \le n \le 2^{h+1}$

कोई भी binary tree मे इससे कम (h+1) node बनाएंगे तो वो binary tree नही होगी and

" " " " ज्यादा 2^{h+1} " " " " " "

formula

height range

We know,

minimum number of node at height h is :

$n = h+1$

$h = n-1$ —(1)

maximum number of node at height h is

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

or, $2^{h+1} = n+1$

or, $h+1 = \log(n+1)$

or, $h = \log(n+1) - 1$ —(1)

There range of height in binary tree is

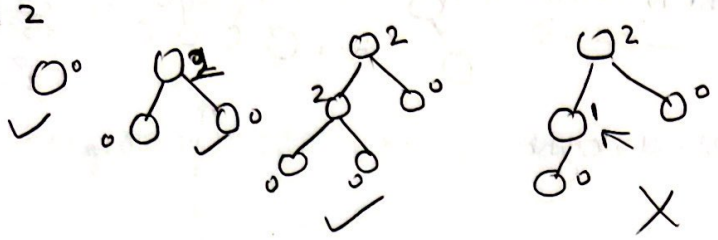
$$\underbrace{\log(n+1) - 1}_{\text{minimum height}} \le h \le \underbrace{n-1}_{\text{maximum height}}$$

∴ minimum height of binary tree is $\log n$

maximum height of binary tree is $n-1$

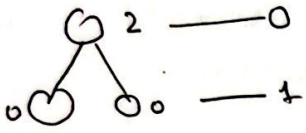
Scanned with CamScanner

Full Binary Tree / Strict Binary Tree → no of children is atmost 0 and 2

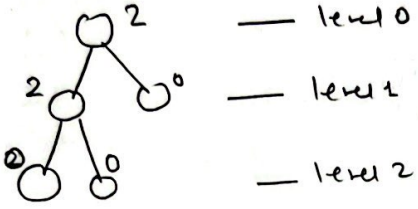


Complete Binary Tree

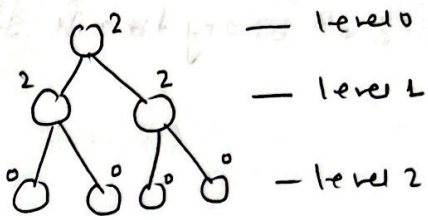
* All leaf nodes are at same level
* Depth of all leaf nodes are same



Full BT

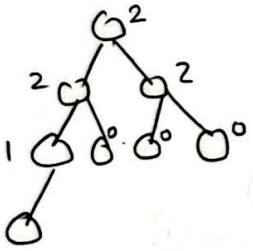


Full BT ✓
Complete Binary Tree ✗

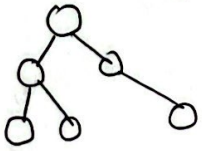


Full BT ✓
Complete BT ✗ coz last level is not complete
i.e. All leaf node are at the same level.

Almost Complete Binary Tree

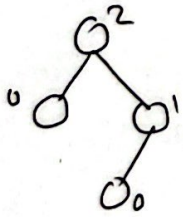


Def^n → left to right fill



Full BT ✗ (coz 1 present)
Complete BT ✗
Almost Complete BT ✓

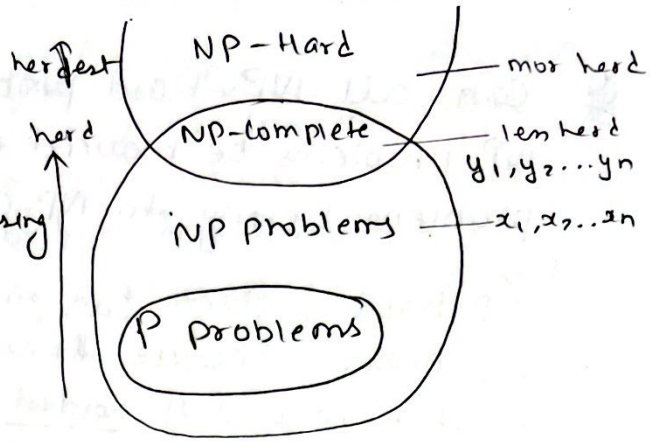
Almost Comp BT ✓



BT → ✓ either 2 or 0 leaves
CBT → ✗ (leaves are at different level
FBT → ✗
Almost CBT → ✗ (no left to right leaves)

Scanned with CamScanner

NP-complete



1) P problems : The problems that are solvable in polynomial time using deterministic algorithm is called P-Problems

eg: linear search $- n$

Binary search $- \log n$

This problem can also be verified in polynomial time.

Deterministic algorithm means all steps in the algorithm is known

2) NP problem (Non deterministic polynomial problem)

The problem that can not be solved in polynomial time but their sol^n can be verified in polynomial time using non-deterministic algorithm.

eg Traveling Sales Person problem., Knapsack $- 2^n$

③ NP complete Problem -

When all problems from NP can be reduced to an NP problem y in polynomial time then y is called NP-complete problem

NP-Complete Problems (say Y) can be solved if and only if there is a NP-Problem (say X) that can be reduced into Y in polynomial time. All NP-complete problem are NP-hard.

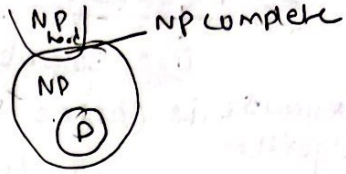
④ NP Hard

a) Np hard problem are problem for which a solution that can not be verified in polynomial time.

b) It also include NPComplete problems i.e can verify solutions in polynomial time.

Scanned with CamScanner

Q Can all NP-Hard problems be reduced to NP Problems? Can all NP problems be reduced to NP-Hard problems? Does power set problem belong to NP-Complete problem?



No,

→ NP-hard problem can not be reduced to NP problems because these problems are at least as hard as the hardest problems in NP in terms of computational complexity. In order word, they are at least as difficult to solve as the most challenging problems in NP.

→ Yes, all problems in NP can be reduced to NP-hard problems because NP hard problems are at least as hard as NP complete problems (where all NP problems can be reduced).

→ The power set problem which involves finding the set of all subset of a given set, is not a NP-complete problem. It's actually not even in NP. NP completenen is a concept primarily associated with decision problems where you are trying to find a solution that satisfies certain criteria, not problem that involve counting or generate subsets like the power set problem.

In other words, powerset problem is solved in dynamic programming in exponential time and it is verified in exponential time.

Scanned with CamScanner

Bonus Question :

Suppose that we have a hash table with n slots, with collisions reduced by chaining and suppose that n keys are inserted into the table. Each key is equally likely to be hashed to each slot. Let M be the maximum number of keys in any slot after all the keys have been inserted. Argue that the probability Q_k that exactly k keys hash to a particular slot is given by

$$Q_k = \left(\frac{1}{n}\right)^k \left(1-\frac{1}{n}\right)^{n-k} \binom{n}{k}$$

(n, k) means the number of ways k items can be taken from n items.

Sol^n A particular key is hashed to a particular slot with probability of $1/n$. Suppose we select a specific set of k keys. The probability that these k keys are inserted into the slot in question and that all other keys are inserted elsewhere is

$$\left(\frac{1}{n}\right)^k \left(1-\frac{1}{n}\right)^{n-k}$$

Since there are $\binom{n}{k}$ ways to choose our k keys, we get

$$Q_k = \left(\frac{1}{n}\right)^k \left(1-\frac{1}{n}\right)^{n-k} \binom{n}{k}$$

Q MST algorithms by Prim or Kruskal use Greedy strategy. Greedy strategy in general does not guarantee an optimal result. Explain how Greedy strategy in the case of Prim or Kruskal still produce an optimal result.

→ prim or kruskal still produce optimal result because it chooses minimum value of edges and vertex and also use decision to choose minimum path. In case of prim it visit through each node and select the minimum one.

Scanned with CamScanner

Q What is the running time of 0-1 Knapsack problem using Dynamic Programming used in # a above?. Explain your answer in good detail.

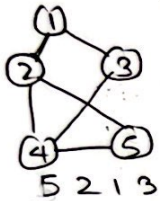
Q Consider Universal Hash function $h(x) = \sum a_i . x_i \bmod$. Prove that the probability of collision is $1/m$ where there are m slots in the hash table. key X is represented as $a_0x_0 + a_1x_1 + \ldots$ where x_i values are less than m and a_i values are selected randomly from $\{0, 1, 2, \ldots m-1\}$

Scanned with CamScanner

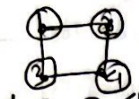
Hamiltonian Cycle and vertex Cover

The Hamiltonian cycle and vertex cover problems have already been shown to belong to NP.

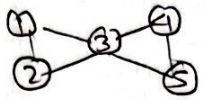
Hamiltonian path is a path in an undirected or directed graph that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path that is a cycle.



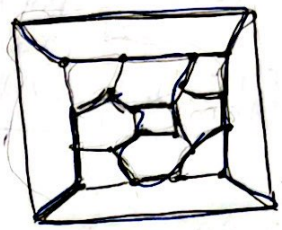
5 2 1 3 4 5 ✓

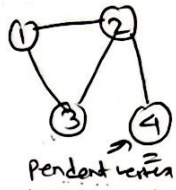


1 2 3 4 1
2 3 4 1 2 ✓



1 2 3 4 5
2 1 3 4 5
(Here no hamiltonian cycle)
✗

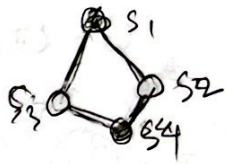




Pendant vertex
1 2 3 4 1
2 3 4 1 2
✗

Vertex Cover

Def : Given a graph $G = (V, E)$,
$S \subseteq V$ is a vertex cover if every edge is an incident on a vertex in S.

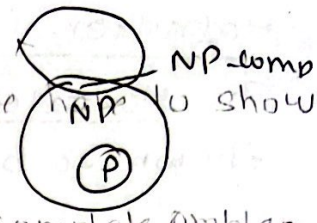


proposition Given a graph G, the size of a matching in G is at most the size of a vertex cover.

proof : For every edge in the matching at least one vertex must be in the cover. Also, all these vertex must be distinct by definition of matching

Scanned with CamScanner

Q prove Clique is NP-Complete

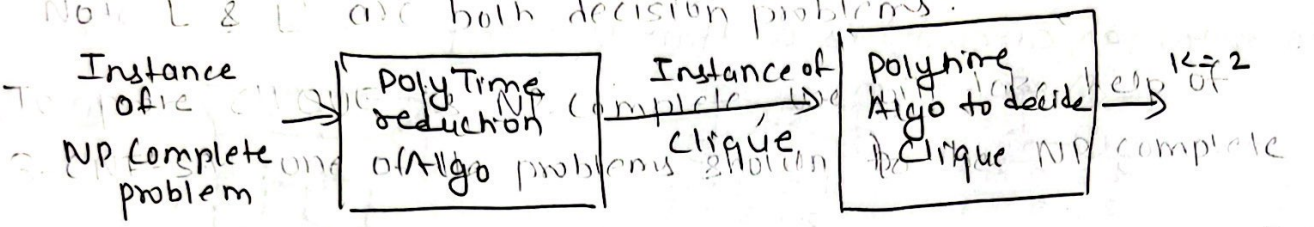


-> To prove a clique is NP-complete

1st: Clique $\in$ NP

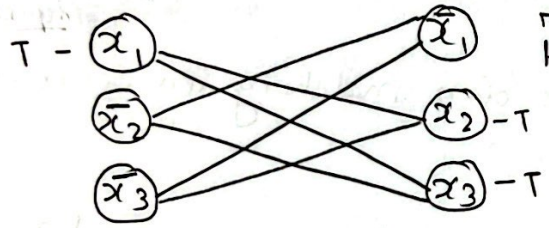
2nd: NP problem can be reduced to clique decision problem

3-CNF-SAT

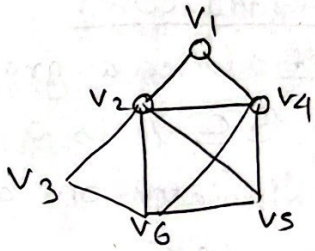


3-CNF-SAT

$\phi = (x_1 \vee \bar{x}_2 \vee \bar{x}) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$ K=2



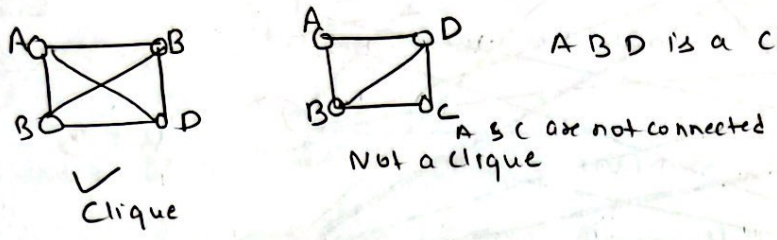
thus, K = 2



- K = 3
K = 4

Scanned with CamScanner

Clique → A clique is an undirected graph $G=(V,E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. The size of clique is the number of vertices in it.



The clique problem is an optimization problem of finding a clique of maximum size in a graph.

Decision version of clique problem → Does a clique of size k exist in a graph.

The formal way of defining it

CLIQUE = $\{(G,k)$: G is a graph containing a clique of size k$\}$

Proof The CLIQUE Problem is NP complete

Step 1 : we have to show clique is an NP Element/problem when an algorithm/sol^n that can not be solved in polynomial time but can be verified in polynomial time called NP problem.

clique $\in$ NP

Step 2 ; Take any instance from NP complete and reduce to NP

instance has boolean variable & operation $\wedge, \vee, \neg$ ← 3-CNF-SAT $\propto$ CLIQUE

3-CNF-SAT: boolean problem

$\propto$: reduce in polynomial time

CLIQUE: graph problem

we will construct a graph G such that ϕ is satisfiable if and only if G has a clique of size k

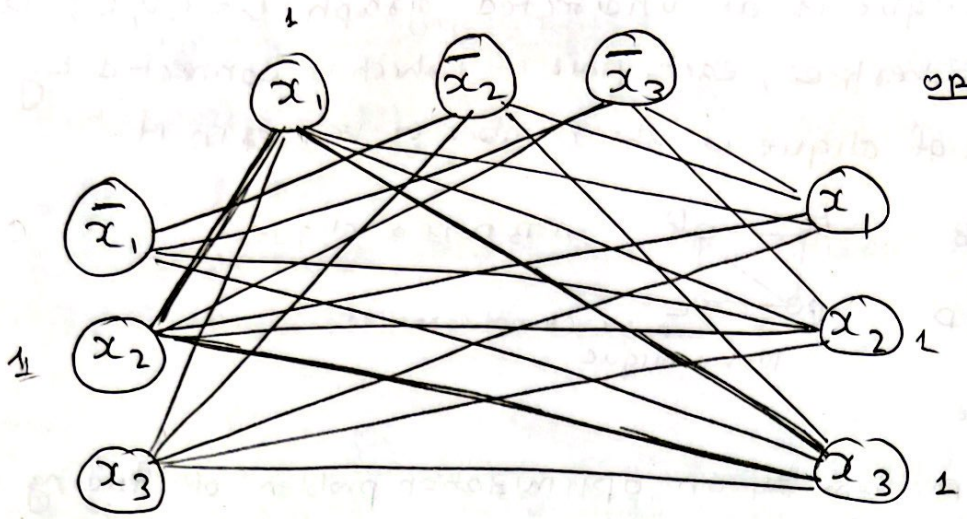
Let us conside on example

$$\phi = (\underset{1}{x_1} \vee \underset{0}{\bar{x}_2} \vee \underset{0}{\bar{x}_3}) \wedge (\underset{0}{\bar{x}_1} \vee \underset{1}{x_2} \vee \underset{1}{x_3}) \wedge (\underset{1}{x_1} \vee \underset{1}{x_2} \vee \underset{1}{x_3}) \quad k=3$$

$= 1 \wedge 1 \wedge 1$ ↗ An assignment of literals such that ϕ evaluates to true.

$= 1$

Scanned with CamScanner



Clique problem goal

optimization → is to find the the maximum size of clique

decision → Does a clique of size k exists in a graph?

$k = 3$

$k = 3$

So, which shows that clique belong to NP and 3-CNF-SAT can be reduced to clique in polynomial time. Thus clique is NP complete problem.

The formal way of defining is

Clique = $\{(G, k) : G$ is a graph containing clique of size $k.\}$

2-CNF SAT → Polynomially solvable

VS

3-CNF SAT → NP-complete

⇒ P is also in NP → Any problem in P is also in NP because if a problem can be solved in polynomial time, its certificate can be verified in polynomial time.

$$P \subseteq NP$$

⇒ $P = NP$ or $P \neq NP$

This is the question that has not yet been proved.

By seeing the veen diagram, it is most likely that P is not NP.

→ We took an arbitrary instance of 3-CNF SAT and reduce it to CLIQUE and not the other way around.

Scanned with CamScanner

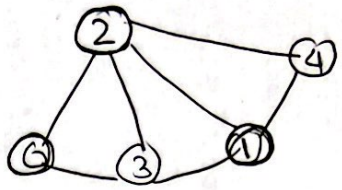
Important point

1. we took an instance of 3-CNF-SAT and reduced it to CLIQUE and not the other way around
2. we took an arbitrary instance of 3-CNF-SAT and reduced it to an instance of CLIQUE
3. we have shown that a specific instance of Clique is hard
4. This is fine because if we show that a specific instance of CLIQUE is hard -> a more general case will be hard (because the general case will include the specific case)

Scanned with CamScanner

Q MINIMUM VERTEX COVER

is NP complete problem



{0, 1, 2}

Initialize current_best = |vertexList|

For every possible subset S of V

If S is a vertex cover

Try to update current_best

Return current_best

Q Clique and SubsetSum is in NP

Clique = { <G, K> G is a graph that has K clique

SS = { <S, t> S = {$x_1, s_2 \ldots x_k$}

a subset of S adds to t

They both belongs to NP class

two approach : ① non deterministic polynomial approach

② using reduction method → we can verify deterministicly in polynomial time whether or not a given certificate is verified in polynomial time or not

certificate = set of K vertices

Algorithm/verifier = check all $\binom{K}{2}$ pairs of vertices

certificel = the subset of S

Algo/verifier = Add all element of subset to check if it is equal to B

Scanned with CamScanner