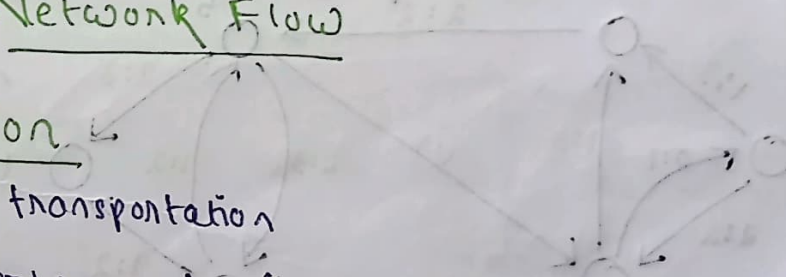
Network Flow

6.06.22

Outline

1. Introduction
2. Flow network defination
3. Flow and capacity
4. The maximum flow problem
5. Cuts in FN.
6. Residual network
7. Augumenting paths
8. Ford Fulkerson's Algo on FN
9. Example
10. Max-flow min cuts theorme
11. Application of NF
    - (i) Matching (Bipartite matching)

Network Flow



(1) Introduction

→ Model of transportation

→ Every vertex → is called switch

→ Every Edge → is called Edge

* Graph can be model as a transportation networks
  - (i) whose edges carry some sort of traffic
  - (ii) whose nodes act as a swich paring traffic between the edge.

② Defination of F.N

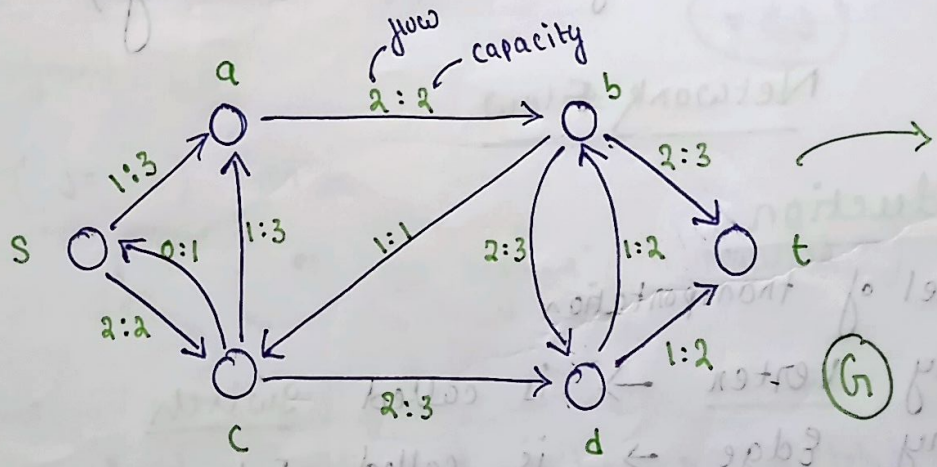
A flow network is a directed weigted graph $G(V,E)$ such that:

(i) Each and every edge of the graph associated with a non-negative capacity which is denoted by $C(u,v)$ if $(u,v) \in E$

if $(u,v) \notin E$ $C(u,v) = 0$

(ii) In this graph there are 2 distinguished vertex
- one is called denoted by (s)
- another is sink or (t)

(iii) For each vertex $v \in V$ then they have they have a path $s \rightarrow v \rightarrow t$



→ Diagram 1

Defination of flow

A flow in a flow Network G, is a real valued function

$f: V \times V \rightarrow \mathbb{R}$

that satisfy the 2 property :-

(i) Capacity Constraint for all $u, v \in V$

we require

$$0 \leq f(u,v) \leq c(u,v)$$

(ii) Flow conservation $\forall\ u \in V - \{s,t\}$

* not applicable for source and sink

$$\sum f(u,v) = \sum f(v,u)$$

incoming flow of a node v → $\sum f(u,v)$; $\sum f(v,u)$ → outgoing flow of a node v.

$$\Rightarrow \sum f(u,v) - \sum f(v,u) = 0$$

Maximum Flow Problem

Problem statement:

Given a flow network G, with flow f and and capacity c. Find a flow of max. value of G

Goal:

Arrange the traffic so as to make as efficient use as possible of the available capacity

Value of flow

The value of a flow f in a flow network G is defined by:-

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

— flow Conservatio

for Source (incoming = 0)

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

for destination (outgoing = 0)

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

Property of flow

Property 1

if $x \subseteq V$, then $f(x, x) = 0$

if $v \in V$, then $f(v, v) = 0$

Property 2

skew symmetry

if $(x, y) \subseteq V$ then $f(x, y) = -f(y, x)$

if $v, u \in V$ then $f(v, u) = -f(u, v)$

Property 3

if $x, y, z \subseteq V$, $x \cap y = \phi$

(i) $f(x \cup y, z) = f(x, z) + f(y, z)$

(ii) $f(z, x \cup y) = f(z, x) + f(z, y)$

Theorme 1

The flow value of a FN

$$|f| = f(v, t)$$

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

The value of a flow in FN is equal to the flow into the sink.

$$\sum_{v \exists v'} f(z, v) + \sum_{v \exists v} f(v, z) = |f|$$

Theorem 1

07.06.23

$$|f| = f(V,t)$$

proof.

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

by rule ① of flow property

$$= f(s \cup V-s, V) - f(V-s, V)$$

$$= f(V,V) - f(V-s, V)$$

$$= f(V, V-s)$$

$$= f(V, t \cup V-s-t)$$

$$= f(V,t) + f(V, V-s-t)$$

$$= f(V,t)$$

By applying flow conservation property

proved

Cuts

A cut (S,T) of a FN $G(V,E)$ is a partition of vertices V into 2 disjoint set S and T where

$$T = V - S \quad \text{and} \quad s \in S \quad \text{and} \quad t \in T$$

$$S \cap T = \phi$$

Net flow across the cut (S,T)

If f is a flow on a Flow network G, the net flow across

$$f(S,T) = f(u,v) - f(v,u) \quad , \forall \begin{matrix} u \in S \\ v \in T \end{matrix}$$

Capacity of a cut (S,T)

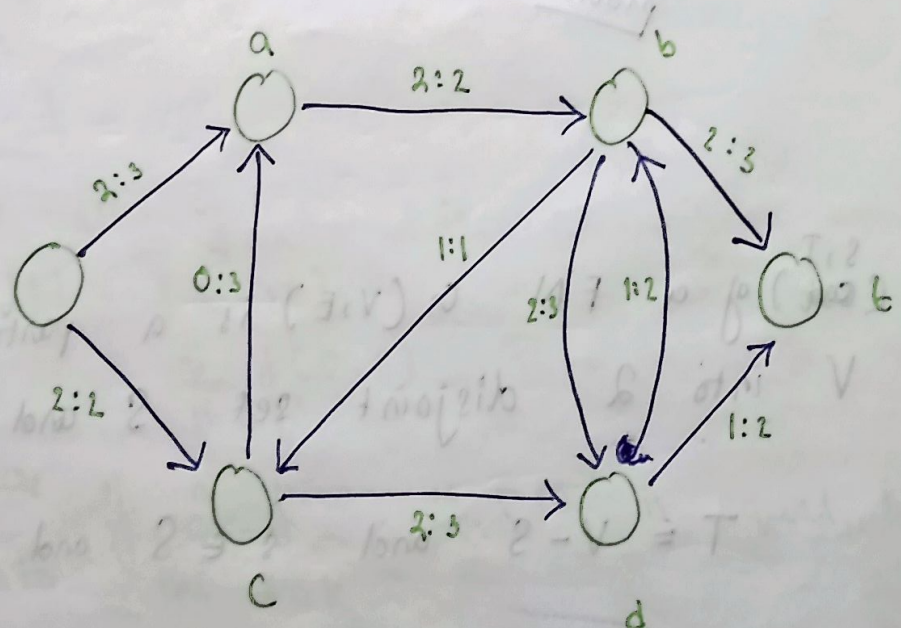
(sum of capacities of all edges coming out of cut S to T)

$$C(S,T) = \sum_{\forall u,v} C(u,v) \text{ when } u \in S \text{ and } V \in T$$

Minimum Cut

A minimum cut of a flow network G is a cut, whose capacity is minimum over all cuts of flow network.

Example



① $S = \{a, b\}$ $T = \{s, c, d, t\}$

cut (S,T) is a valid cut in the FN G

② $S = \{ s, a, b, c \}$ $T = \{ d, t \}$ ..... G

③ $S = \{ s, t \}$ $T = \{ a, b, c, d \}$

④ $S = \{ s, a \}$ $T = \{ b, c, d, t \}$

Cut validity

① $S = \{ a, b \}$, $T = \{ s, c, d, t \}$

here $s \notin S$, $t \in T$, $S \cap T = \phi$

, $T = V - S$

↓

Not valid cut

② $S = \{ s, a, b, c \}$, $T = \{ d, t \}$

here $s \in S$, $t \in T$, $S \cap T = \phi$

, $T = V - S$

valid cut

③ $S = \{ s, t \}$, $T = \{ a, b, c, d \}$

here $s \in S$, $t \notin T$, $S \cap T = \phi$

$T = V - S$

↓

not valid cut

④ $S = \{ s, a \}$, $T = \{ b, c, d, t \}$

here $s \in S$, $t \in T$, $S \cap T = \phi$

$T = V - S$

valid cut

Net flow across cut (S,T)

(2) $S = \{s, a, b, c\}$ $T = \{d, t\}$

$f(S,T) = f(a,b,d) + f(b,t) + f(c,d) - f(d,b)$

$= 2 + 2 + 2 - 1$

$\boxed{f(S,T) = 5}$

(3) (4) $S = \{s, a\}$ $T = \{b, c, d, t\}$

$f(S,T) = f(s,c) + f(a,b) - f(c,a)$

$= 2 + 2 - 0$

$\boxed{f(S,T) = 4}$

Capacity of a cut

(2) $S = \{s, a, b, c\}$ $T = \{d, t\}$

$C(S,T) = C(b,d) + C(c,d) + C(b,t)$

$= 3 + 3 + 3$

$= 9$

(3) (4) $S = \{s, a\}$ $T = \{b, c, d, t\}$

$C(S,T) = C(s,c) + C(a,b)$

$= 2 + 2$

$= 4$

For some case $f(S,T) = C(S,T)$

Theorme

For a given flow f in a flow network G, the net flow across the cut is same as

$$\boxed{f(S,T) = |f|}$$

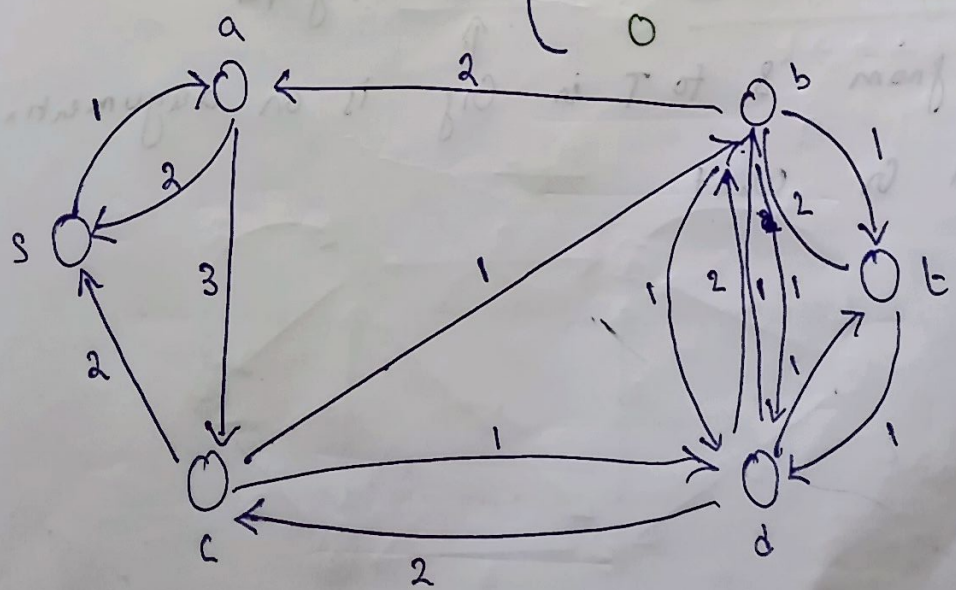
The value of any flow f is a F N G is bounded from the above theorme by the capacity on

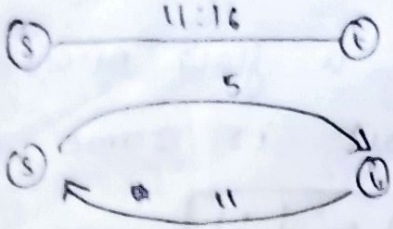
$$C(S,T) = |f|$$

Residual Networks

let f be a flow on F N , G the residual network $G_f(V, E_f)$ is the network on graph with strictly positive residual capacity $C_f(u,v)$

$$C_f(u,v) = \begin{cases} C(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (u,v) \notin E \\ & (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$





(i) $|V_f| = |V|$

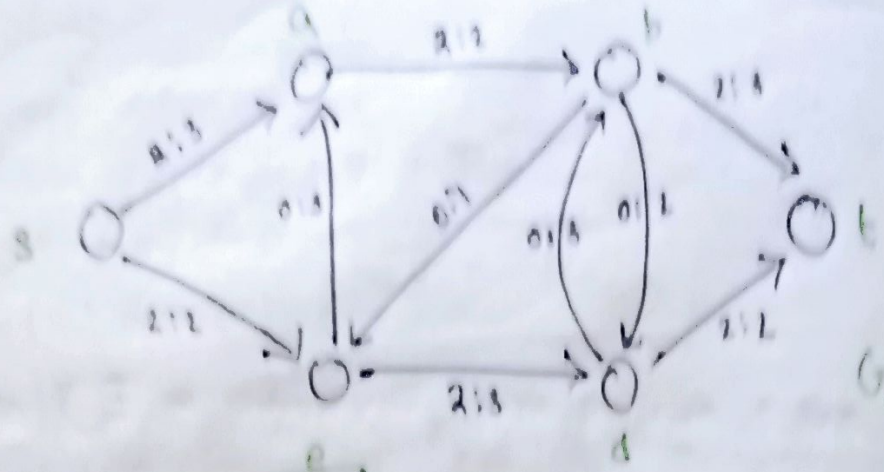
(ii) $|E_f| \leq 2|E|$.

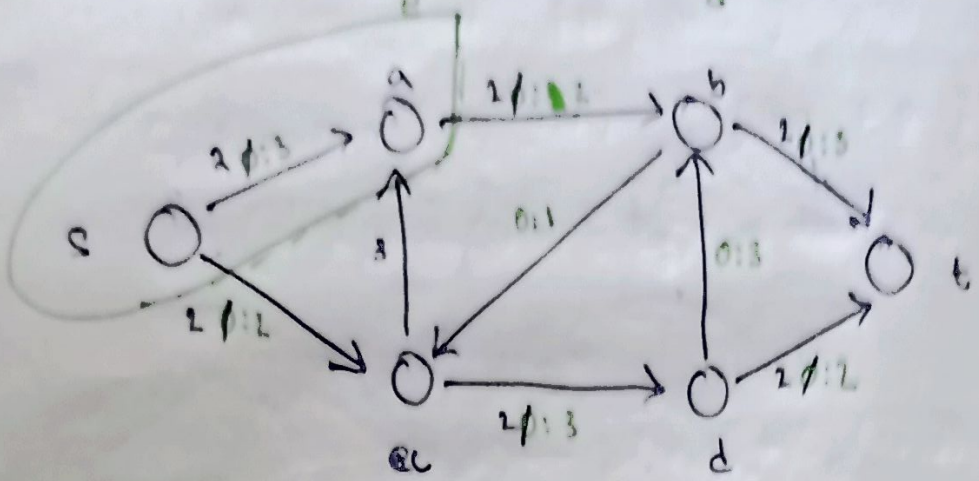
Augumenting Path (P)

residual graph

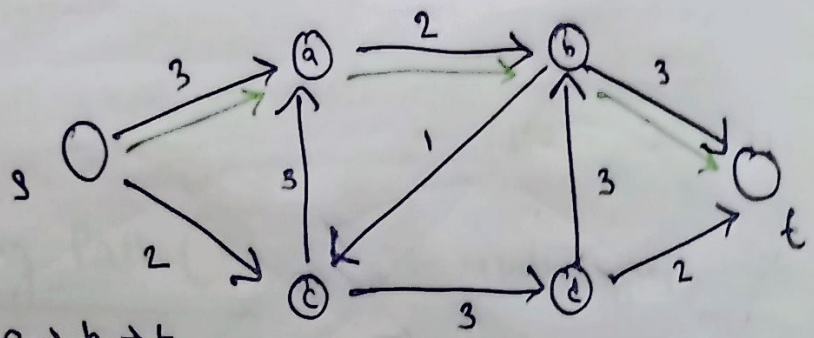
Any path from S to T in G_f is an augumenting path (P) in G w.r.t.

Example





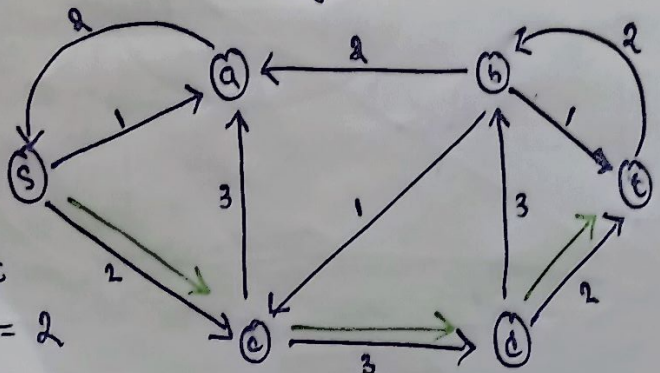
G_{f1}



augument path → s → a → b → t

bottle neck capacity = 2

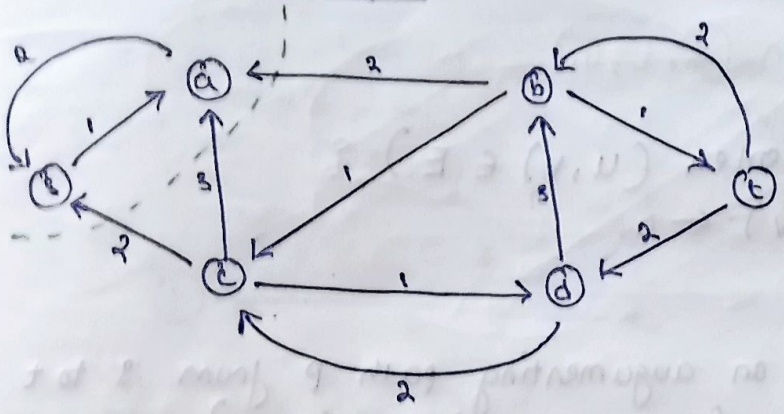
G_{f2}



augument path → s → c → d → t

bottle neck capacity = 2

Gf 3



Final Residual Graph

→ no more augumenting paths.

Maxflow = 2+2 = 4

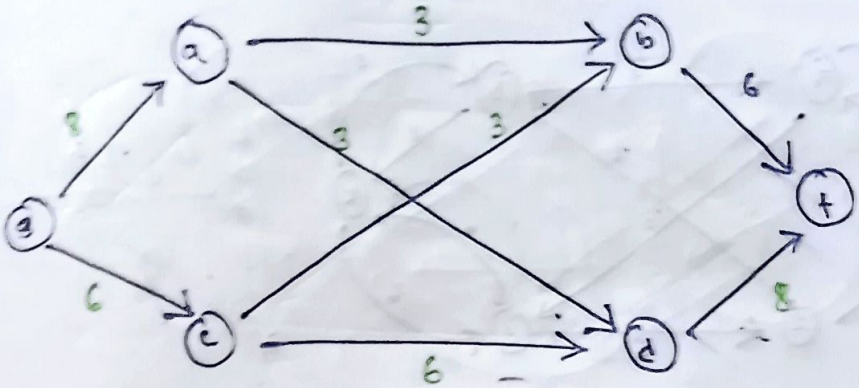
min cut (S, T)

S = { s, a } , T = { b, c, d, t }

~~cap~~ C(S,T) = C(s,c) + C(a,b) = 2+2 = 4

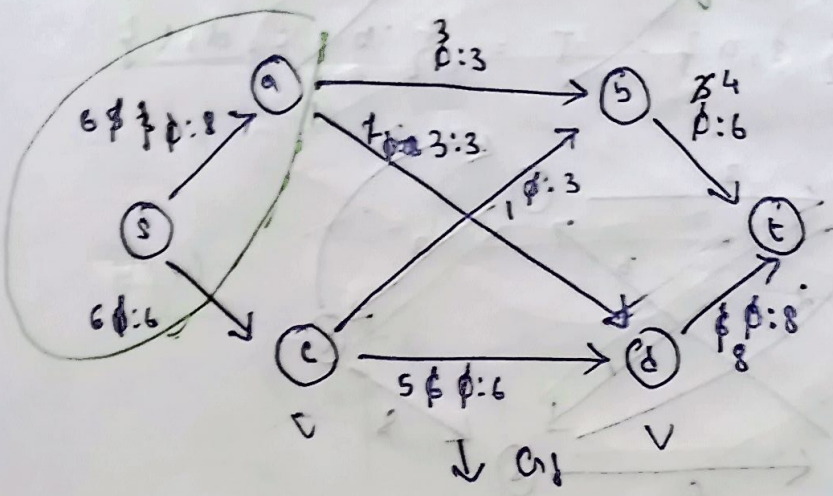
```
Ford-Fulkerson (s, t, G)
{
    for (each vertex (u,v) ∈ E) {
        f(u,v) ← 0
    }
    while ( ∃ an augumenting path p from s to t
            in Gf of the FN, G. )
    {                                              O(|f|)
        Cf(P) = min { Cf(u,v) ; (u,v) ∈ P }
        for (each edge (u,v) ∈ Ef in path p)       O(E)
        {
            if( (u,v) ∈ E)
                f(u,v) ← f(u,v) + Cf(u,v);
            else f(v,u) ← f(v,u) - Cf(u,v);
        }
    }
}
```

Time Complexity = $O(|f| \cdot E)$



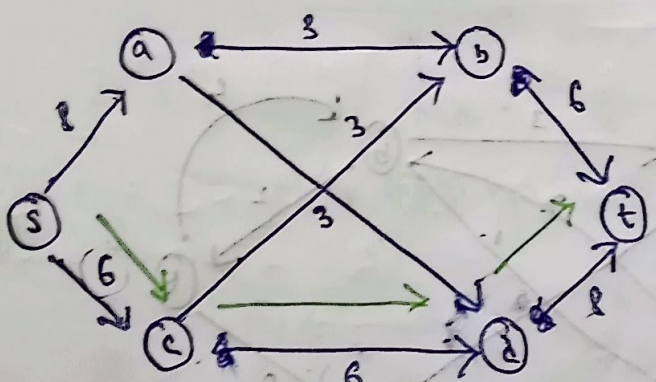
G

Find the ~~flow~~ max flow of the flow network G by applying Ford Fulkerson Algo.

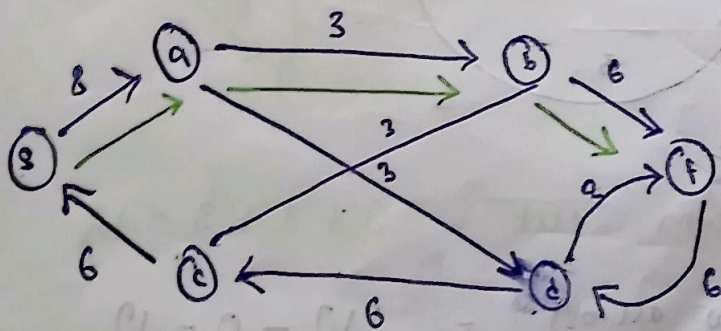


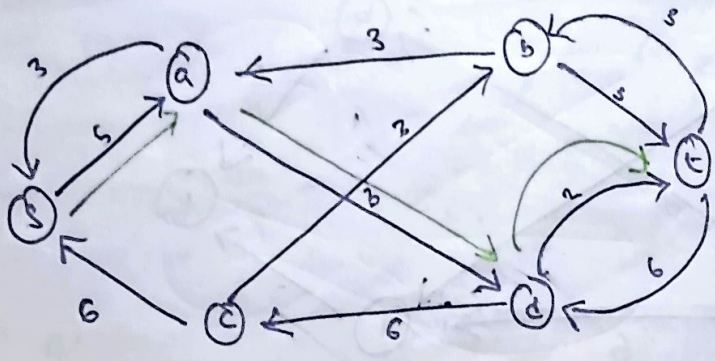
$|f| = 0 + 6 + 3 + 2 + 1$
$= 12$

G_f



G_f 1

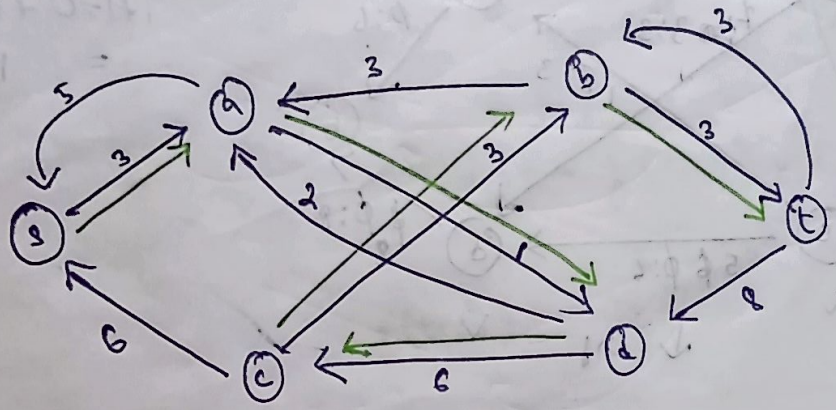




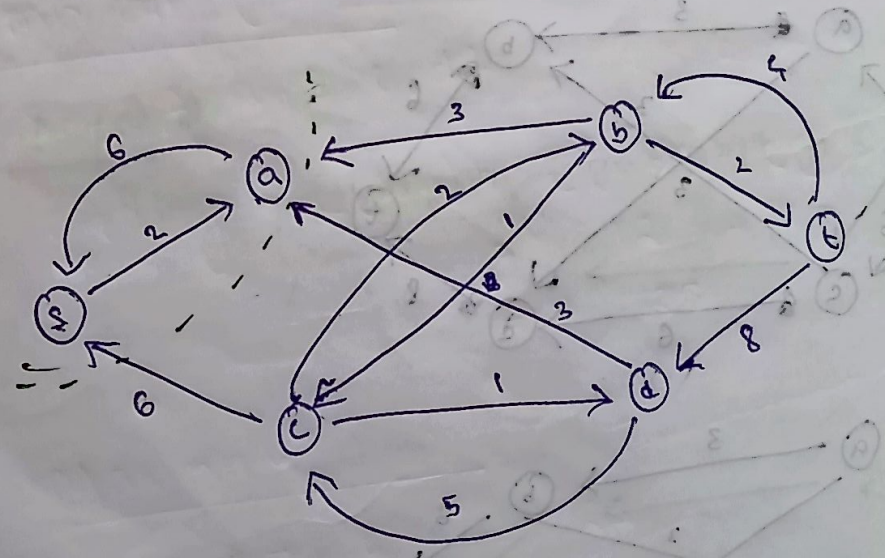
$G_f 3$

Minimum cut (S, T)

$S = \{s, a\}$ $T = \{b, c, d, t\}$



$G_f 4$

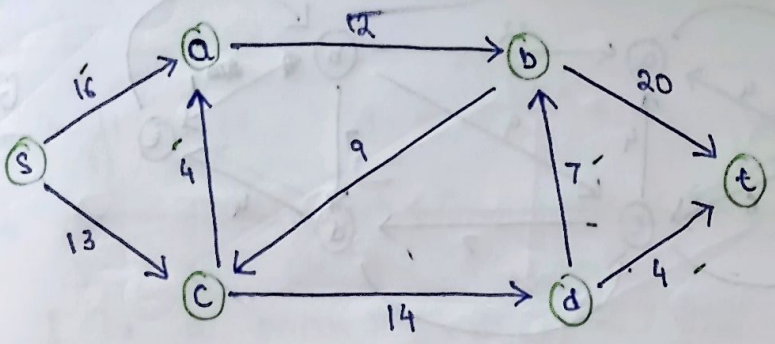


$S = \{s, a\}$ $T = \{b, c, d, t\}$

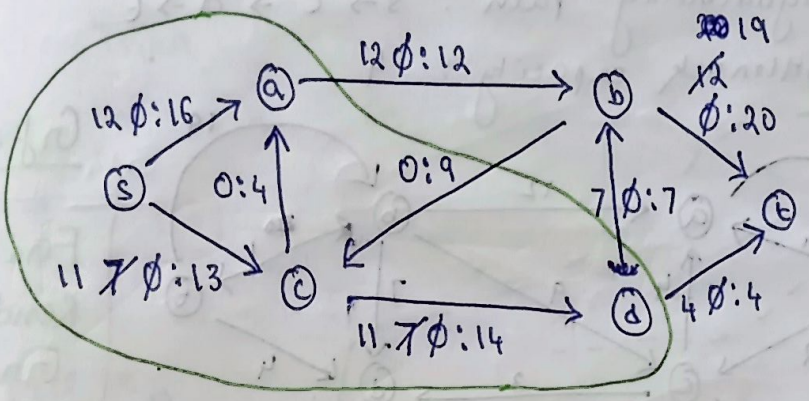
Minimum cut = $6 + 3 + 3 = 12$

net flow across cut = $12 - 0 = 12$

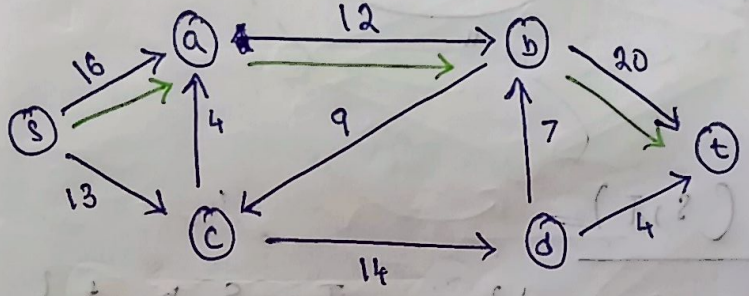
Example 4



G



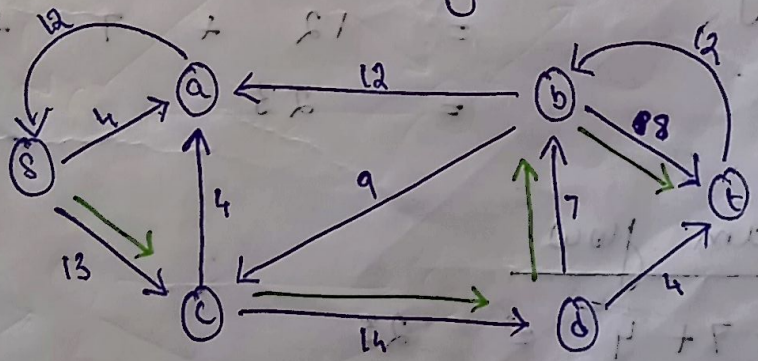
G



G_f 1

Augmenting path : $s \to a \to b \to t$

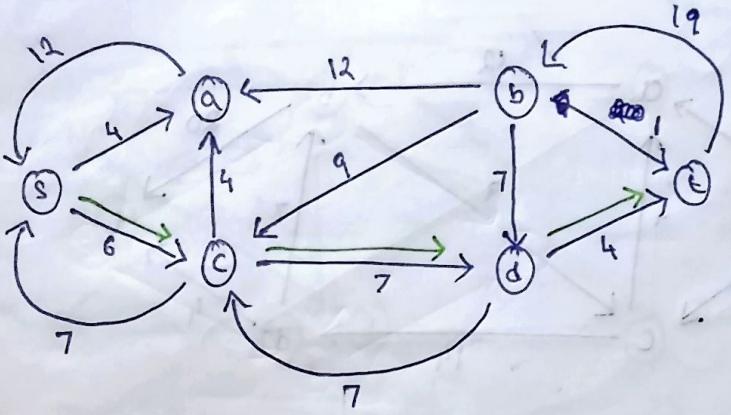
bottleneck capacity : 12



G_f 2

Augmenting path : $s \to c \to d \to b \to t$

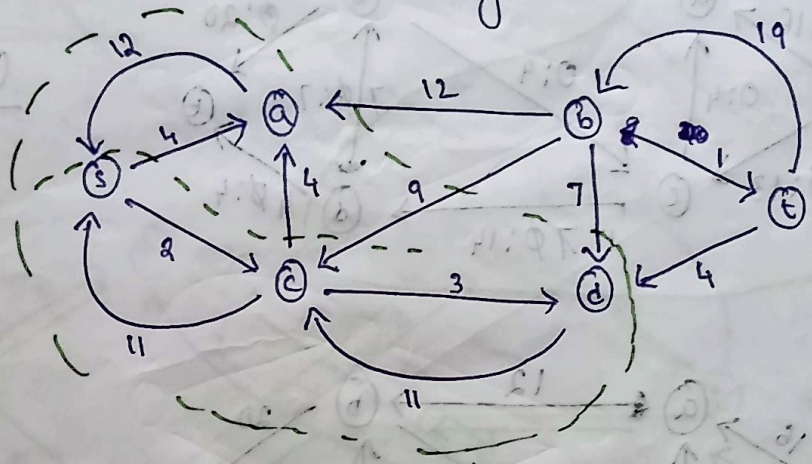
bottleneck capacity : 7



Grf 3

augamenting path : $s \to c \to d \to t$

bottleneck capacity : 4



Grf 4

Final Residual Graph

Min cut (S, T)

$S = \{a, s, c, d\}$ $\quad T = \{b, t\}$

capacity $C(S,T) = C(a,b) + C(d,b) + C(d,t)$

$= 12 + 7 + 4$

$= 23$

Maximum flow

$|f| = 12 + 7 + 4 = 23$

Max-flow min-Cut theorme

let f is a flow in FN, $G(V,E)$ with capacity c

then the ~~given stat~~ following conditions are equivalent:

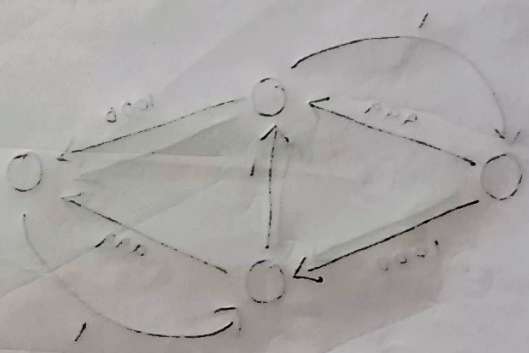
(i) f is a max flow in a FN, G

⇕

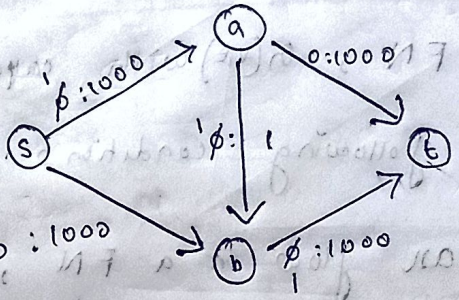
(ii) Then there is no augumenting path in Residual ⇕ network G_f of FN, G.

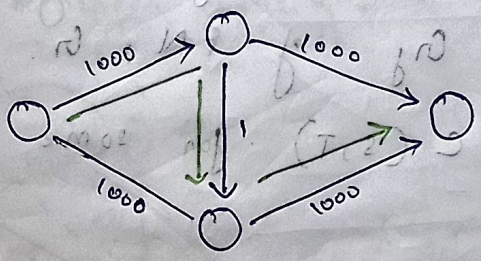
(iii) $|f| = c(S,T)$ for some cut (S,T) in FN G.

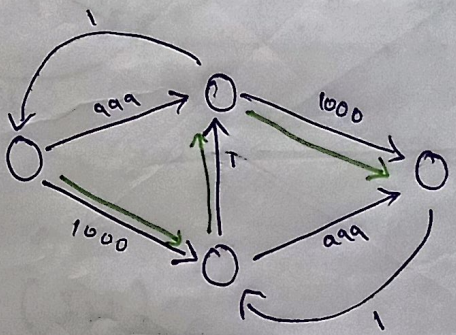
* prove



Drawback of Ford Fulkerson :







Application of Max flow Problem

(1) Bipartite Matching

List of Job J_i

Rules

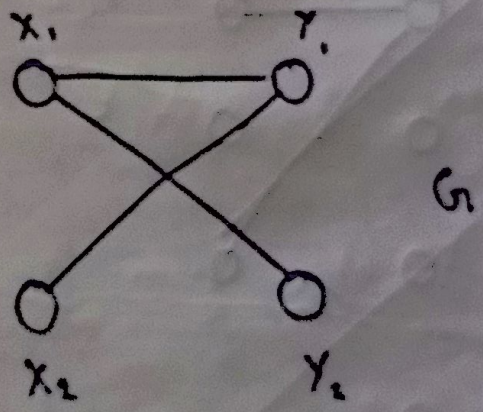
→ each candidate must be given atmost 1 job and each job must be give to one candidate

Goal

→ assign the candidate to job such that max no. of job are filled.

Example 1

13.06.22



$M_1 = \{(x_1, y_1)\}$

Application of Max flow Problem

(i) Bipartite Matching

List of Job J_i

Rules

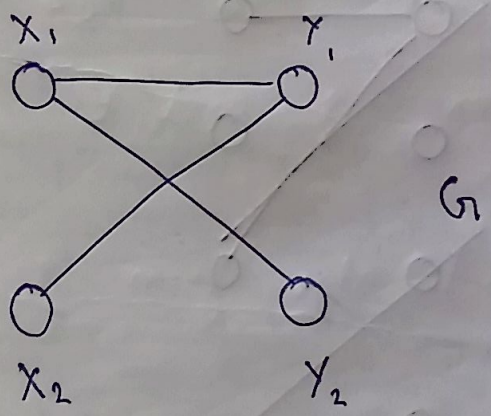
→ each candidate must to be given atmost 1 job and each job must be give to one candidate

Goal

→ assign the candidate to job such that max no. of job are filled.

Example 1

13.06.22



$M_1 = \{(x_1, y_1)\}$

Matching

Def: A matching in Graph G (V, E) is a set of edges $M \subseteq E$ with the property that each node appears atmost in ~~one~~ one edge of M. The M is a matching.

Perfect matching: (ii) Each and every ~~edge~~ vertex should be connected to exactly one edge of M.

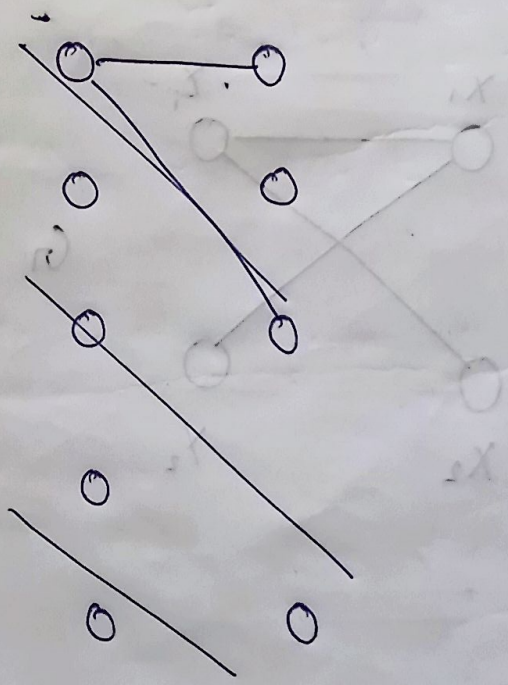
Greedy Method

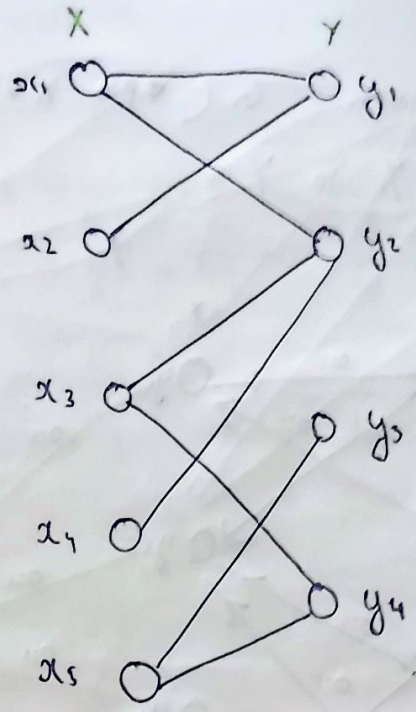
→ # keep adding edge into M, till no more edges can be added

Example 2.

$m = \{e\}$ (a, b)

a - c



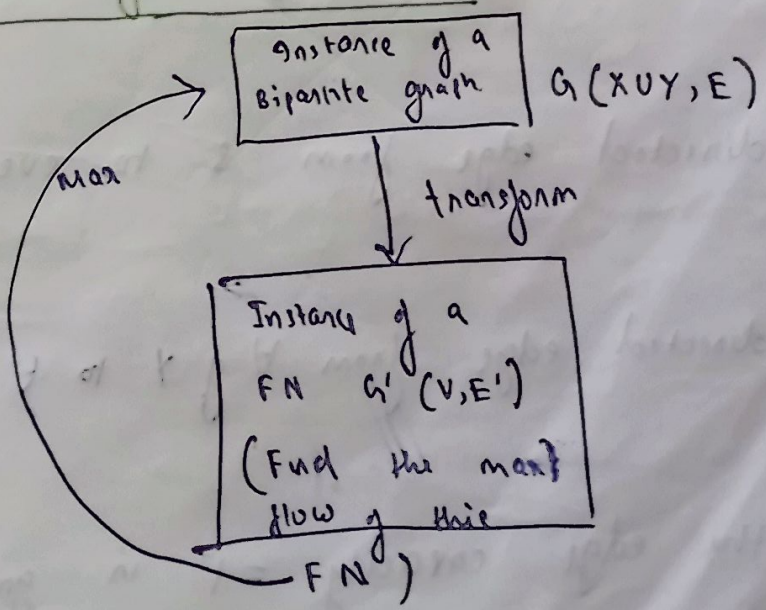


✗ $M = \{(x_1, y_1), (x_3, y_2), (x_5, y_3)\}$ $|M| = 3$

✓ $M = \{(x_1, y_2), (x_2, y_1), (x_3, y_4), (x_5, y_3)\}$ $|M| = 4$

✓ $M = \{(x_4, y_2), (x_1, y_1), (x_3, y_4), (x_5, y_3)\}$ $|M| = 4$

③ Maximum flow problem

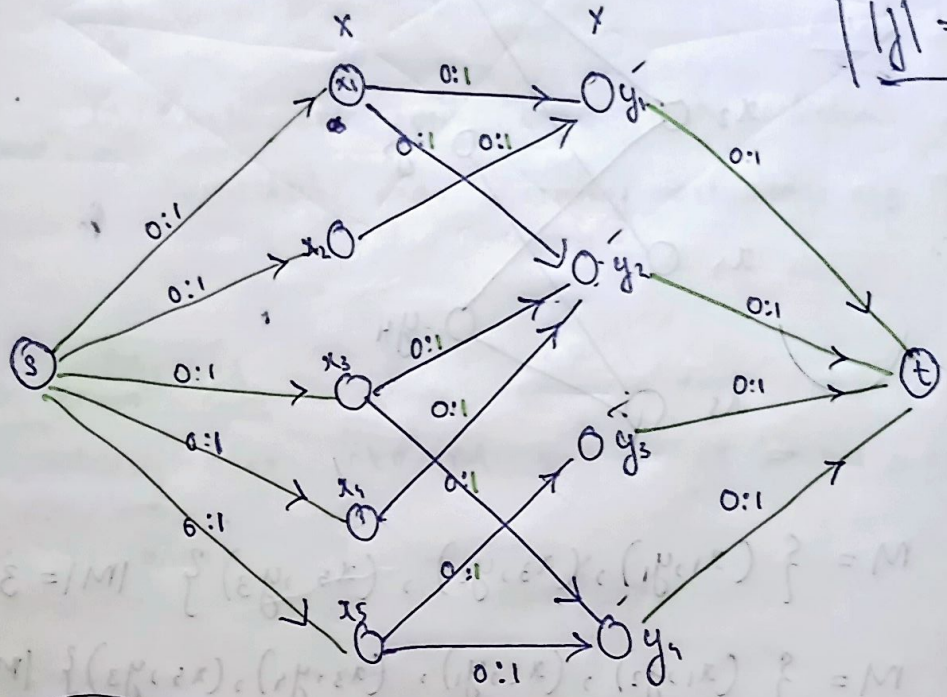


Step 1

Given Bipartite Graph $G(X \cup Y, E)$ direct the edge from X to Y

$|d| = 4 = |M|$

$|y| = |M|$



P_1 : $s \to x_2 \to y_1 \to t$

P_2 : $s \to x_4 \to y_2 \to t$

P_3 : $s \to x_3 \to y_4 \to t$

P_4 : $s \to x_5 \to y_3 \to t$

Step 2

Add to vertex s and t

Step 3

Add a directed edge from s to every vertex in X

Step 4

Add a directed edge from $\forall$ of Y to t

Step 5

Set all the edge capacity = 1 in graph made $G'(V', E')$

Step 6

Apply Ford - Fulkerson method

$P_1 = \{ s \to x_3 \to y_3 \to t \}$ $\quad c_j(P_1) = 1$

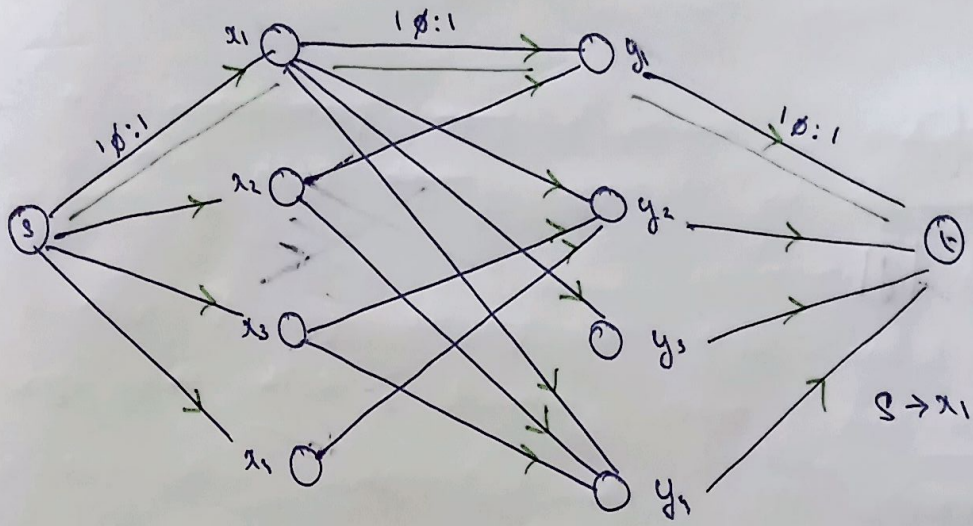
$P_2 = \{ s \to x_3 \to y_4 \to t \}$ $\quad c_j(P_2) = 1$

$P_3 = \{ s \to x_2 \to y_1 \to t \}$ $\quad c_j(P_3) = 1$

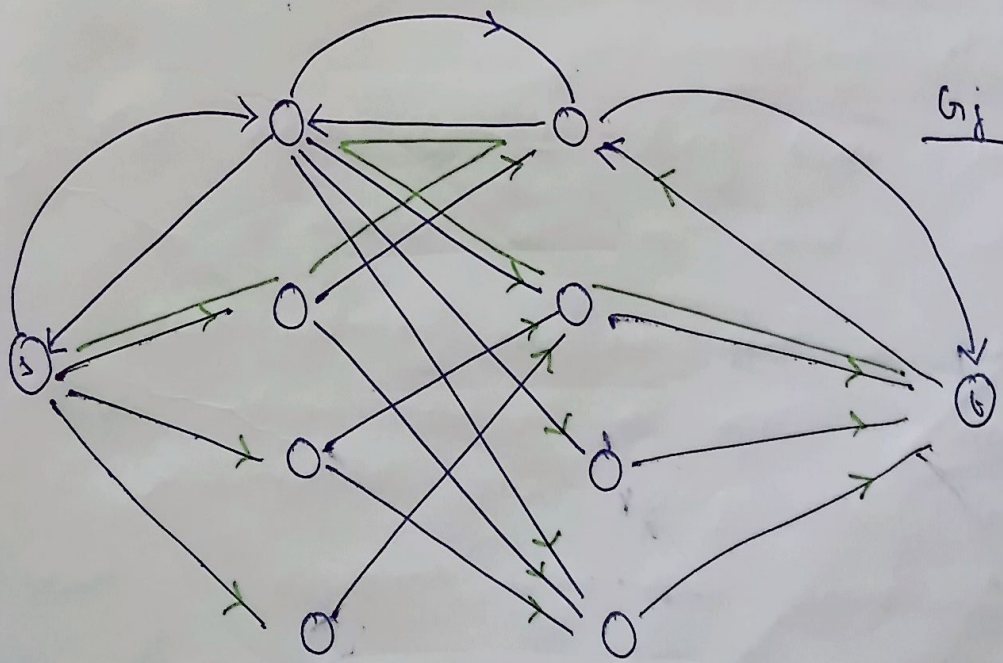
$P_4 = \{ s \to x_1 \to y_2 \to t \}$ $\quad c_j(P_4) = 1$

Example 3

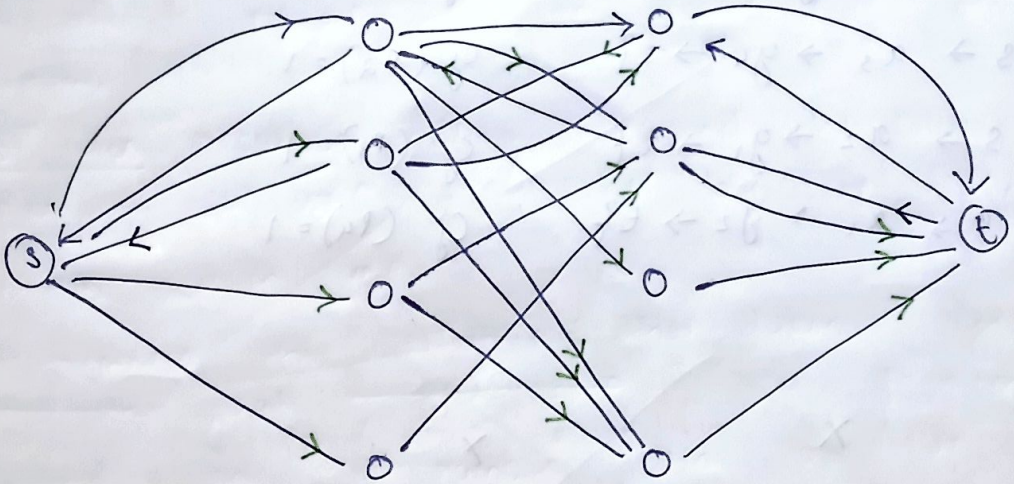
X X



$s \to x_1 \to y_1 \to t$



G_j 1



G_2

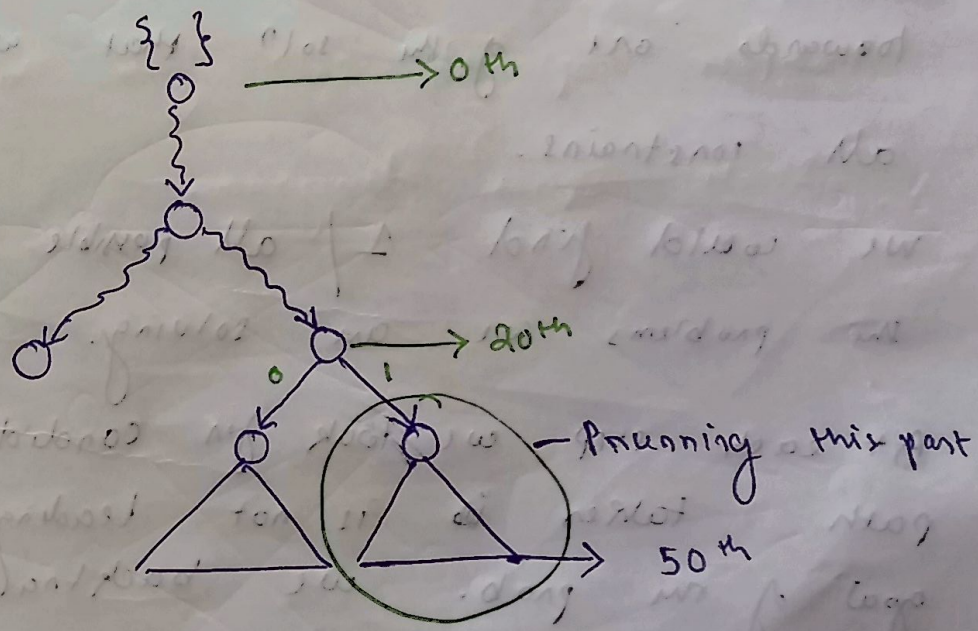
Theorme

(i) $\cdot |A|$

Backtracking

(i) Introduction
(ii) Control Abstraction.
(iii) General Algo for Backtracking Method
(iv) Constructing All subsets of a given set.
(v) Constructing All permutation of a given set
(vi) Constructing all paths in a graph G.
(vii) x Subset sum
(viii) x N-Queen } by using backtracking.
(ix) x Sudoku

Introduction



→ Backtracking is an algo. design technique for listing all possible soln for combinatorial searching problem.

→ It uses recursive calling to find the solution by building a soln step by step increasing values with time.

→ It removes the solutions that doesnot give rise to the goal ~~of the soln~~ of the problem based on constrains given to solve problem.

→ Backtracking is similar to ~~prev~~ permutation n combinatn where we try different ~~root~~ path / soln of the same problem.

→ We start with a possible partial soln of the prob. and move ahead with this aproach towards one of the soln that will satisfy all constrains.

→ We could find 1 / all possible soln's for the problems we are solving.

→ At each step - we look for candidate if the path taken is not leading us to goal of the prob. we backtrack 1 level back and start with new candidate if that level again doesn't lead to correct soln we backtrack to 1 level further up till we reach the root, we say that soln is not available for combination with said constrains.

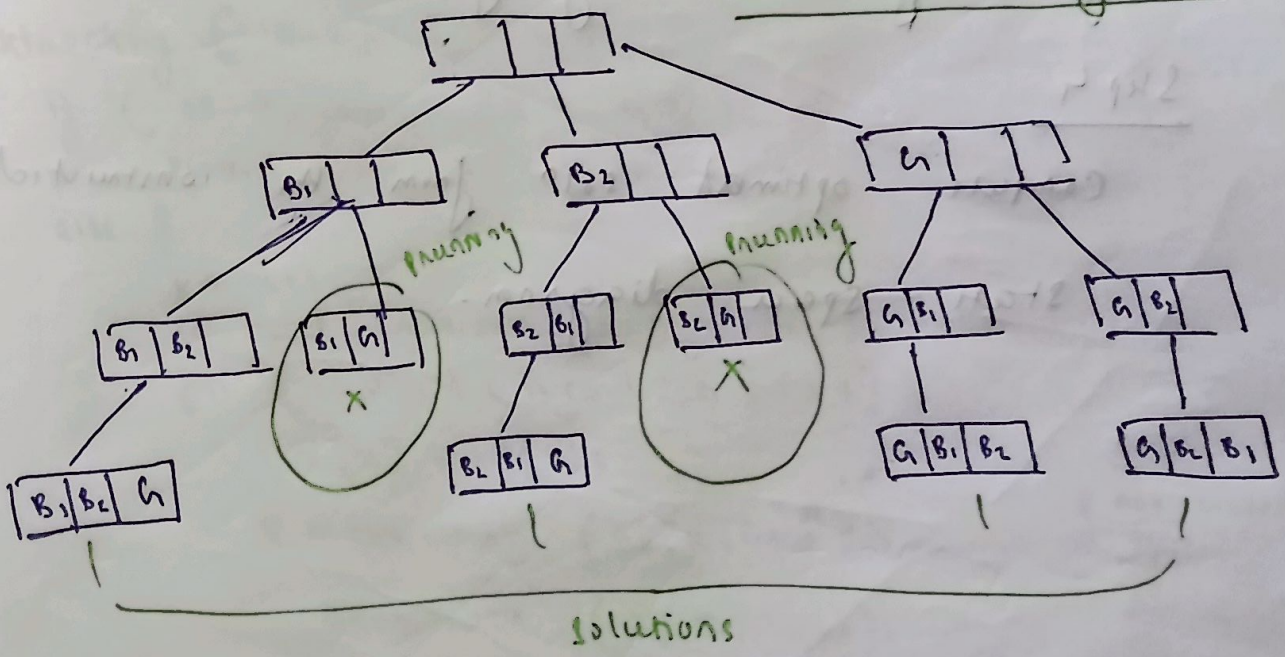
→ If we find a potential candidate which will lead us to goal of problem which will become part of partial soln and that will be used as a part of final soln.

Problem 2 (B_1, B_2, G)

We want to find all combination 2 boys, 1 girls in 3 seat

Constrains :- girls should not be in middle of 2 boys

State Space Diagram



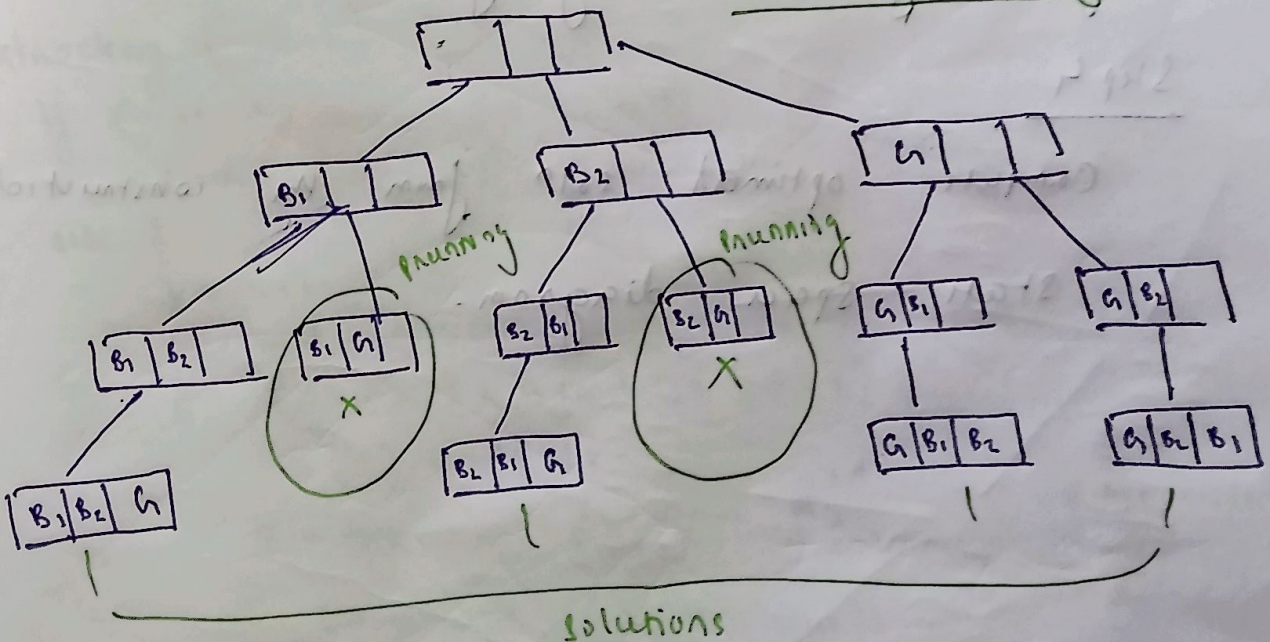
→ If we find a potential candidate which will lead us to goal of problem which will become part of partial soln and that will be used as a part of final soln.

Problem 2 (B_1, B_2, G)

we want to find all combination 2 boys, 1 girls in 3 seat

constrains :- girls should not be in middle of 2 boys

State Space Diagram



Strategy

Step 1

Identify all the constrains (implicit, explicit) of given problem

Step 2

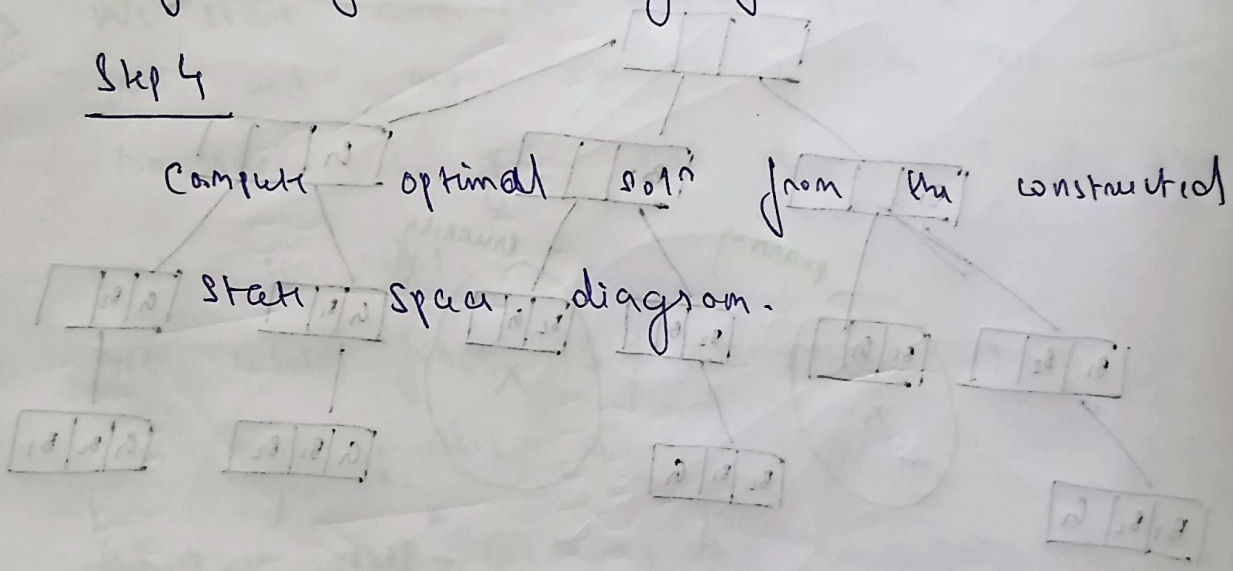
define an appropriate bounding funct^n based on given constrains.

Step 3

Construct a state space diagram or tree by using all bounding func^

Step 4

Compute optimal soln from the constructed state space diagram.



```
Backtracking (a, k) {
    if ( a is a possible soln of the prob.) then
        print (a);
    else {
        k ← k+1
        compute Sk
        while (Sk ≠ φ) {
            Sk ← Sk - ak ;
        }
        Backtracking(a, k);
    }
}
```

```
Backtracking ( a, k) {
    if ( is_a_solution (a, k, input))
        process_solution (a, k);
    else {
        k ← k+1
        construct_candidate (a, k, input, c, ncandidates);
        for (i←0, i < ncandidates, i++) {
            a[k] = c[i];
            # make_move (a, k, input);
            Backtrack (a, k, input);
            # unmark_move (a, k, input);
        }
        if (finished)
            return true;
    }
}
```

not needed

Construct All Subsets

How many subsets are there of an 'n' element set

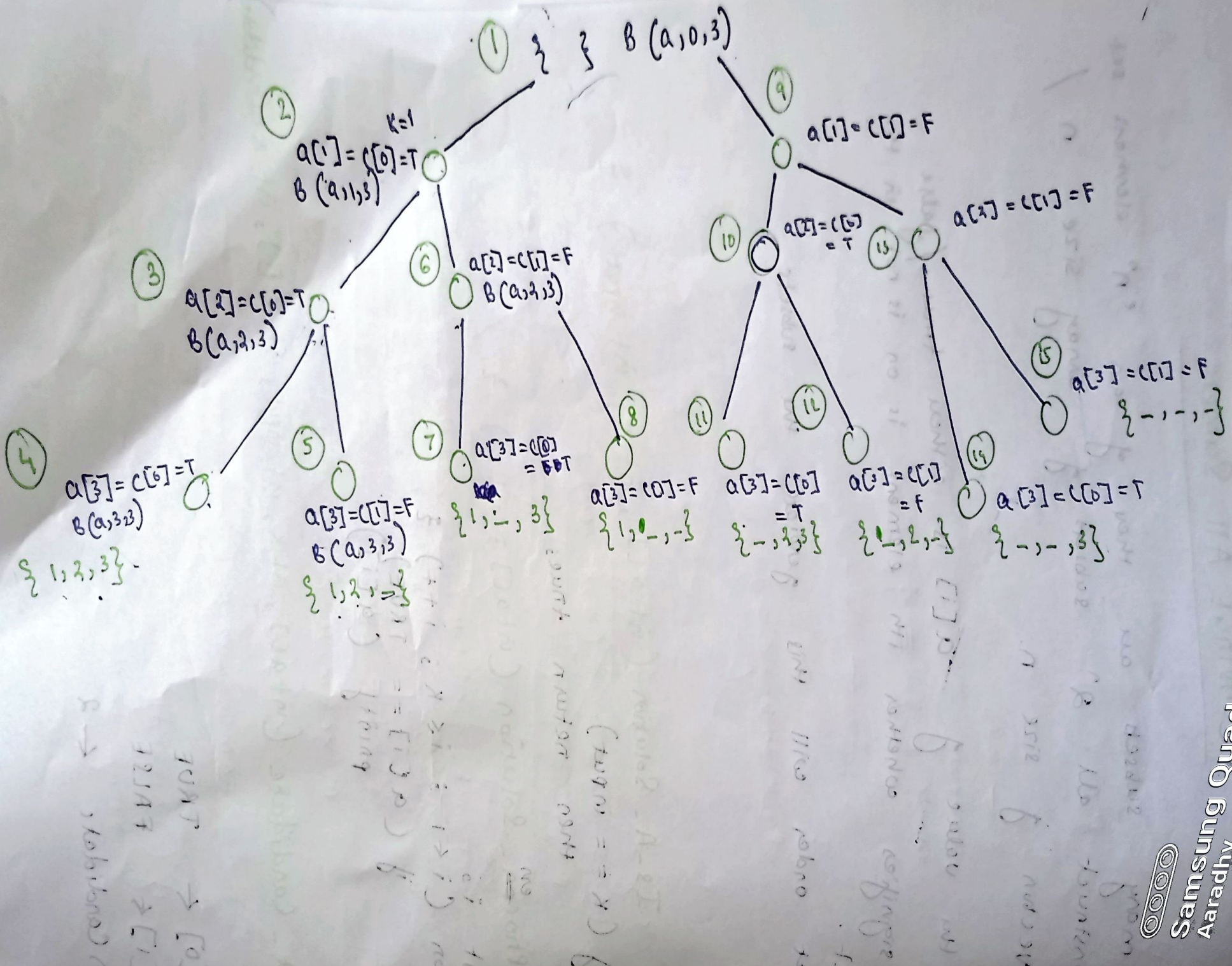
To construct all 2^n subsets of array size n / vector of size n.

→ when the value of a[i] is either true / false

→ it signifies whether ith element is on is not in the subset.

→ what order will this generate the subsets

```
boolean Is_A_Solution (int a[], int k, int input) {
    if (k == input)
        then return true;
}
void Process_Solution (int a[]; int k) {
    int i;
    for (i <- 1 ; i <= k ; i++) {
        if (a[i] == TRUE)
            printf (a[i])
}
Construct_Candidate (int a[], int k, int input, int c[], int *nCandidate)
{
    c[0] <- TRUE
    c[1] <- FALSE
    *nCandidate <- 2
}
```


1
{ } B (a, 0, 3)
2
K=1
a[1] = c[0] = T
B (a, 1, 3)
3
a[2] = c[0] = T
B (a, 2, 3)
4
a[3] = c[0] = T
B (a, 3, 3)
{ 1, 2, 3 }
5
a[3] = c[1] = F
B (a, 3, 3)
{ 1, 2, - }
6
a[2] = c[1] = F
B (a, 2, 3)
7
a[3] = c[0] = T
{ 1, -, 3 }
8
a[3] = c[0] = F
{ 1, -, - }
9
a[1] = c[1] = F
10
a[2] = c[0] = T
11
a[3] = c[0] = T
{ -, 2, 3 }
12
a[3] = c[1] = F
{ -, 2, - }
13
a[2] = c[1] = F
14
a[3] = c[0] = T
{ -, -, 3 }
15
a[3] = c[1] = F
{ -, -, - }

```
bool is_a_solution (int [], int k, int n)
{
    if (k == n) then return T;
}

void process_solution (int a[], int k) {
    int i;
    for (i=1; i<=k; i++)
        printf ("%d", a[i])
}

void construct_candidate (int a[], int k, int n, int c[], int *nCandidate)
{
    Bool Pe_sol [MAX]
    int i
    for (i=1; i<=n; i++) {
        Pe_sol [a[i]] ← T
    }
    *nCandidate ← 0
    for (i=1; i<=n; i++) {
        if (Pe_sol [i] == F) {
            c [*ncandidate] ← i
            *nCandidate ++
        }
    }
```