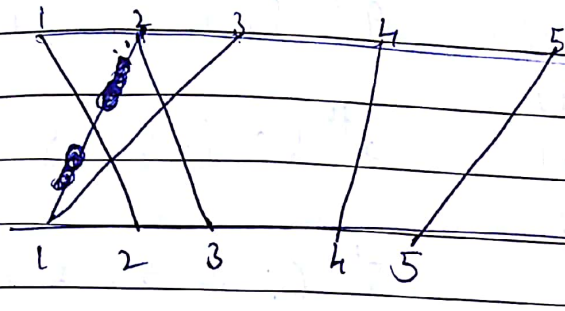
26/3/18:-

DATE: PAGE:

## Maximum Noncrossing subset of Nets:-




C [2, 3, 1, 4, 5]

4 non crossing subset i.e. 1, 2, 4 & 5

$$\text{size}(i, c_j) = \begin{cases} 0 & j < c_i \\ 1 & j \geq c_i \end{cases}$$

$$\text{size}(i, j) = \text{size}(i-1, j) \quad j < c_i$$

$$\text{size}(i, j) = \max\{(i-1, j), \text{size}(i-1, c_i) + 1\} \quad j \geq c_i$$

1) C = [8, 7, 4, 2, 5, 1, 9, 3, 10, 6]

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

not required

same follows

4 is maximum size of noncrossing subset.

qth min $C_i$ = 10
= 10 - 1
∴ (8, 9) & (7, 9)

{9, 7, 5, 3} are pins of not crossing.

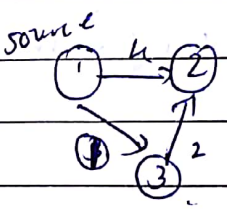
DATE: PAGE:

Trace back :-

| | | |
|---|---|---|
| | (10, 10) = (9, 10) | |
| 9 ← | (9, 10) ≠ (8, 10) | 9th Ci is 10 , i 10-1=9 |
| | (8, 9) = (7, 9) | |
| 7 ← | (7, 9) ≠ (6, 9) | 7th Ci is 9 , i 9-1=8 |
| | (6, 8) = (5, 8) | |
| 5 ← | (5, 8) ≠ (4, 8) | 5th Ci is 5 , i 5-1=4 |
| | (4, 4) = (3, 4) | |
| 3 ← | (3, 4) ≠ (2, 4) | 3th Ci is 4 , i 4-1=3 |
| | (2, 3) = (1, 3) | |

SLE :- All pair shortest path problem - (Floyds)

```
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
    for(k=1;k<=n;k++)
    {
    d[i,j]=min{d[i,j], d[i,k]+d[k,j]}
    }
```




- complexity → $n^3$

| | | |
|---|---|---|
| 99 | 4 | 1 |
| 99 | 99 | 99 |
| 99 | 2 | 99 |

Rod Cutting Problem :-

ways of cutting :- $2^{n-1}$

1)

| Length i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| profit P | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 | 24 | 30 |

DATE: PAGE:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Profit | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 | 25 | 30 |
| size. | 1 | 2 | 3 | 2 | 2 | 6 | 1 | 2 | 3 | 10 |

2 + 2 = 4 size

5 + 5 → 10 profit

consider first cut remaining arjitr.

2).

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| w | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| p | 2 | 3 | 5 | 1 | 6 | 4 | 4 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Profit | | 2 | 3 | 5 | 7 | 8 | 8 | 9 |
| Size | | 1 | 2 | 3 | 1 | 2 | 1 | 2 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 3 | 2 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 5 | 3 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 1 | 4 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 6 | 5 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 4 | 6 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 4 | 7 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |

1 seven time! are selected i.

| |
|---|
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |

7 − 1 = 6 go to 6th col

6 − 1 = 5 go to

1 − 1 = 0

DATE: PAGE:

```
int Rod cut (int T[][], int n, int P[])
{ for(i to n)
  for(i to n)
  if(j >= i)
      max(T[i-1][j], P[i]+P[i][j-i])
  else
     T[i][j] = T[i-1][j];
}
return T[n][n];
}

void Traceback (int T[][], int n)
{
   i = j = n;
   while(i>0 && j>0)
   {
     if(T[i][j] != T[i-1][j])
     {Printf("%d", i)
       j = j - i;
     }
     else
     i = i - 1;
   }
}
```

Time complexity $O(n^2)$

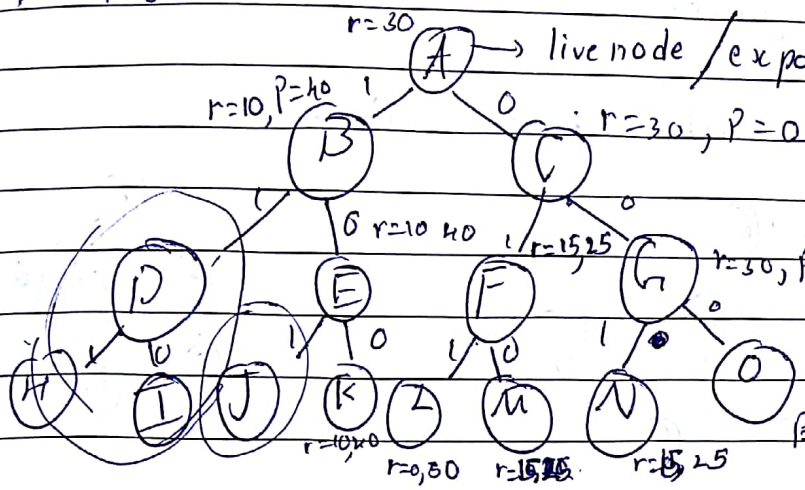
# Unit - 5 :-

DATE: PAGE:

## Back Tracking :-

→ search → Solution space.

→ tree




$w_i = [20, 15, 15]$, $P_i = [40, 25, 25]$, $C = 30$

right have 1st is not considered

5 $S_0^{th}$, $P = 40, 50, 25, 25, 0$

max is $P = 50$

Path is $[0, 1, 1]$

→ This is Knapsack backTracking method.

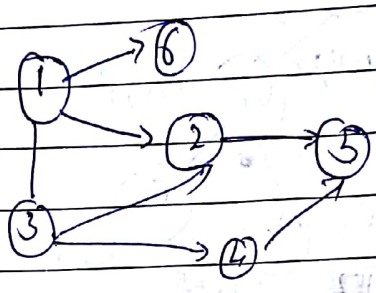
Main Gave :- Back Tracking is a systamatic way to search for a solution to a problem the technique will begin by defining a solution space for a problem. The $sol^n$ space is nothing but all possible solution for the problem & also contains atleast 1 optimal solution to the problem.

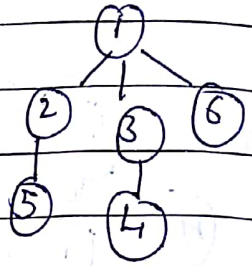
The $sol^n$ phase is organised by the key structure so that the searching for the optimal $sol^n$ is

easier. The labels on each edge of the tree represents members of the solution. All parts from root to a leaf define the element of solution phase.

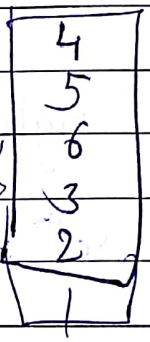
3/4/18:-

2) Breadth First search Method:-





queue




→ a[][]

→ visited[0];

→ source

→ visited[source] = 1.

q[r++] = source;

while (f<=r)

u = q[f++]

loop2 if(a[u][v]==1) && visi[v]!=1)

(visited[v] = 1

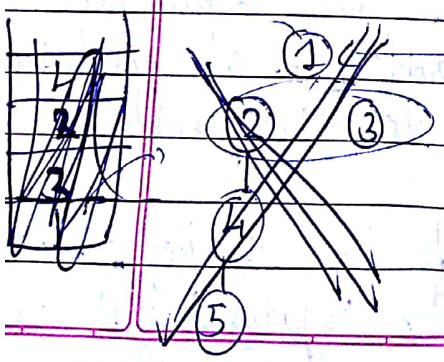
q[r++] = v;) → $O(n^2)$

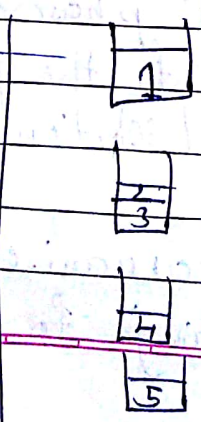
Finally:-

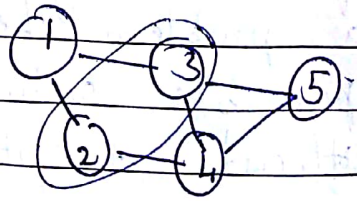
Output whose visited index is 1.

4/4/18:-

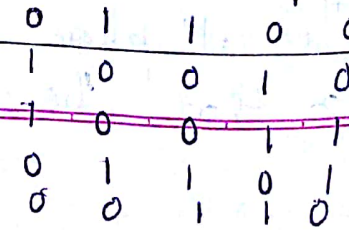
stacks 3) Depth first Search:-







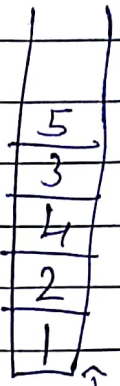
→ spanning tree



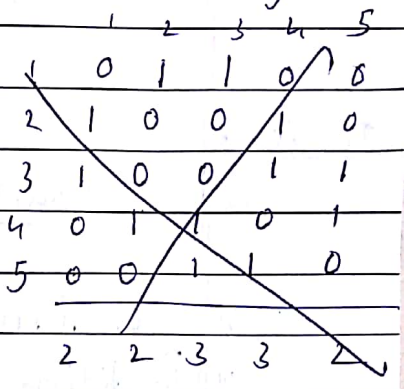
saving for visited.

DATE: PAGE:

```
dfs(int u)
{
    s[u] = 1 → printf("%d", u)
    for (v=1; v<=n; v++)
        if (a[u][v]==1 && s[v]==0)
            dfs(v);
}
```




For topology directed graph should be there.




4). n-queen problem:-

```
void nqueens(int row)
{
  if(row<n){
    for(j=0; j<n; j++){
      if(feasible(row,j)){
        a[row][j]='Q';
        nqueens(row+1);
      }
    }
  }
}
int feasible(int row, int col){
  for(i=0; i<n; i++){
    int tcol = getmarkedcol(i);
    if(tcol==col || abs(row-i)==abs(col-tcol))
      return 0;
  }
  return 1;
}
```

if(row==n) return 1;

```
int getmarked(int row) {
    for(j=0; j<n; j++) {
        if (a[row][j] == 'Q')
            return j;
    }
}
```

complexity $\Rightarrow n * O(n^2) + n!$

$\Rightarrow O(n^3) + n!$

$\Rightarrow O(n!)$

$n * [(n-1) + O(n^2)]$

14/18:-

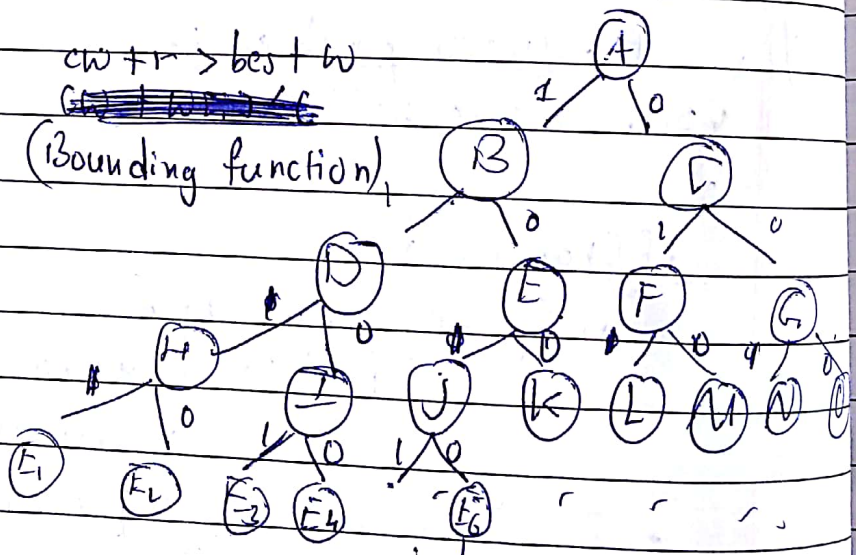
5) Container Loading:-

1) sum of subset problem. $\sum_{i=1}^{n} w_i \leq c_1 + c_2$

2) Partition.

cw + r > best w

(Bounding function)




best w = 10 x[10 0]

cw + w[i] > best w ;

best w = 11 x[1 0 0]

x[0 1 1]

gives 11 itself so no update.

First Backtracking soln :-

DATE: PAGE:

```
int loading (int i){
    if (i > n){
        if (cw > bestw)
            bestw = cw;
        return 0;
    }
    if (cw + w[i] <= c)
    {
        x[i] = 1;
        cw = cw + w[i];
        loading (i+1)
        cw = cw - w[i];
    }
    x[i] = 0;
    loading (i+1);
}
```

Second backtracking sol^n :- (optimal)

```
int loading (int i ){
    if (i > n ){
        if (cw > bestw)
            bestw = cw;
        return;
    }
    r = r - w[i];
    if (cw + w[i] <= c)
    {
        x[i] = 1;
        cw = cw + w[i];
        loading (i+1);
        cw = cw - w[i];
    }
```

r + cw > bestw

DATE:

```
if(cl > bestw)
{
x[i]=0;
loading(i+1);
}
r=r+w[i];
}
```

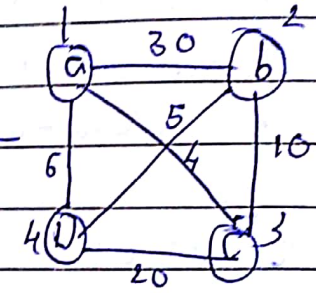
11/4/18:

6) Travelling salesman problem:-

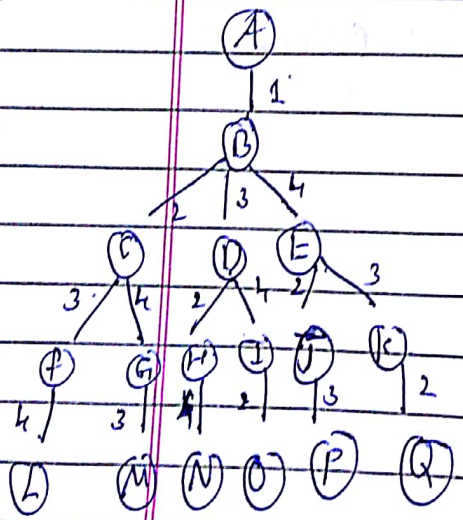
x - x Path.

1 cycle - Tour (closed walk)

cycle <- min cost




Possible sol = $\frac{n!}{n} = (n-1)!$ (To eliminate giving cycle same)




Permutation logic Using backs tracking

```
void perm(int list[], int i){
  if(i==n){
    for(j=1;j<=n;j++)
      cout<<a[j];
  }
  else
  {
  for(j=i;j<=n;j++)
  {
  swap(list[i], list[j])
  perm(i+1)
  swap(list[i], list[j])
  }
}
```

## Actual program :-

## Starting step

```
Perm(int a[], int n)
{
 if(i==n)
  {
  Pf(List[i]);
  }
 else
  {
  for(j=i; j<=n; j++)
   {
   swap(a[i],a[j]);
   perm(i+1);
   swap(a[i],a[j]);
   }
  }
}
```

Store cities name is x[i] array.

x[i] = i.

cc

best of C.

initially bestc = 999.

cc = 0.

```
tsp(int i, int a[][], int n){
 if(i==n){
```

→ written next page.

```
 }
 else
  {
  for(j=i; j<=n; j++){
   if(a[x[i-1]][x[j]] != 999 &&
   cc + a[x[i-1]][x[i]] < bestc ||
   bestc == 999)
    {
    cc += a[x[i-1]][x[j]];
    swap(x[i], x[j]);
    tsp(i+1);
```

```
cc -= a[x[i-1]][x[i]];
swap(x[i], x[j]);
}
}
```

0 —— 0 —— 0 ——

```
if(i==n){
if(a[x[n-1]][x[n]]!=999 && a[x[n]][x[1]]!=999 &&
cc+a[x[n-1]][x[n]]+a[x[n]][x[1]]<bestc
bestc==999)
{
for(j=1;j<=n;j++){
bestx[j]=x[j];
bestc=cc+a[x[n-1]][x[n]]+
a[x[n]][x[1]];
}}
}
else
```

Complexity O(n-1)!

0/1 Knapsack:-

n=4, $w_i$ = [2, 3, 4, 1] C=5

$P_i$ = [9, 10, 12, 14]

$P_i/w_i$ (Decreasing order)

4.5 3.33 3 14

∴ $P_i$ [14, 9, 10, 12] $w_i$ = [1, 2, 3, 4]

1st, 2nd, 3rd → object

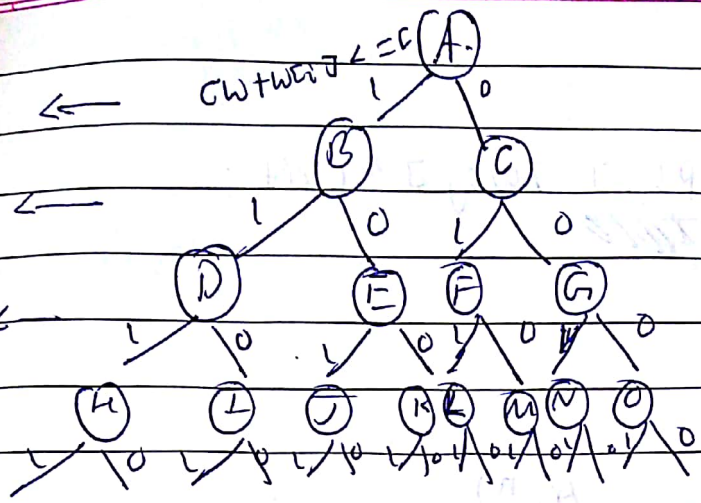
Capacity left for 3rd object is c=2, w=3

Fraction ⇒ $\frac{P_i}{w_i} \times 2 = 6.66$

take fraction of 3rd object i.e. weight 3

∴ 14+9+6.66 = 29.66

∴ total value.




```
Knap(int i)
{
else{ if(cw+w[i]<=c)
    {
    cw+=w[i];
    cp+=p[i];
    Knap(i+1);
    cw-=w[i];
    cp-=p[i];
    }
 if(bound(i+1)>bestc)
   Knap(i+1);
 }
}
```

```
if(i==n)
{
 if(cp>bestc)
 {
  bestc=cp;
  return;
 }
}
```

```
int bound(int j)
 {
 int cleft=c-cw;
 int b=cp;
while(j<=n && w[j]<=cleft)
 {
  b=b+p[j];
  j++;
 }
```

If queen (assign to cp then return

```
if (j <= n)
  {
    bt = p[j] /w[j] * cleft;
  }
}
```

complexity is $O(2^n)$.

# Unit :VI :-

## NP Hard and NP complete Problem :-

| Polynomial Time | Exponential → NP Hard problem |
|---|---|
| (Divide) search { Linear - n | 0/1 - O(2^n) |
| Binary - log n | TSP |
| sort { Bubble - n^2 | Graph |
| Merge - n log n | Container loading. |

1) Reduction ( Reducing the Problem statement from exponential )
2) Non deterministic Polynomial time form.