Job Sequencing Problem:

Given an array of jobs, where every job has a deadline & associated profit if job is finished by the deadline.

Every job takes a single unit of time i.e. maximum possible deadline for any job is 1 time unit.

Our objective is to maximise profit.

n: set of jobs
d_i : deadline of jobs , $d_i \geq 0$
P_i : Profit of each job , $P_i > 0$

For any job i, profit P_i is earned if and only if, the job is completed by deadline.

Constraints:

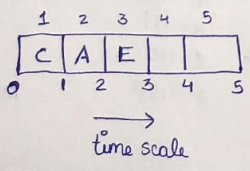
(i) One machine is available to process jobs
(ii) Each job takes single unit of time to complete

Eg.

Job	A	B	C	D	E
Deadline	2	1	2	1	3
Profit	100	19	27	25	15

S1. Sort all the jobs in descending order of profit

Job :	A	C	D	B	E
Deadline:	2	2	1	1	3
Profit :	100	27	25	19	15



100
+ 27
+ 15
142

S1. Sort the jobs in decreasing order of their profits
S2: Select the job with the highest profit
S3: Find the deadline of highest profit & put it in the index equivalent to deadline, if the slot is empty.
eg.
S4. If it is not empty, then put it in the previous one slot. If the index is greater than 0.
S5. Repeat from S2, until all the jobs have been processed.

*Longest Substring without repeating character

Given a string S, find the length of longest substring without repeating characters.

eg. S = <abcabcbb> , ans = 3, i.e. abc
S = <bbbbb> , ans = 1 , ie. b
S = <pwwkew> ans = 3 i.e. wke or kew

We will be using two pointers i & j which are initially at index 0.
We will be using frequency table or array which is used to store frequency of each char in string

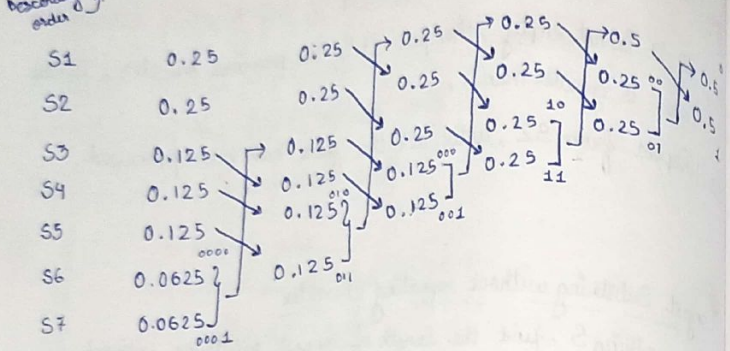
As there are a total of 256 char in extended ASCII codes, so we will be using an array count 256. COUNT[256]
Initially the value of count array is initialise to zero.

```
i=0 ; j=0 ; ans=0
for (j=0 ; j< s.size() ; ++j){
    count[S[j]]++;
    while(count[S[j]]>1){
        count[S[j]]--;
        i++;
    }
    ans = max(ans, j-i+1);
}
return ans;
```

Q) S1 S2 S3 S4 S5 S6 S7

0.25 0.25 0.125 0.125 0.125 0.0625 0.0625 → Probabilities

descending order ↑



S1 - 10

S2 - 11

S3 - 001

S4 - 010

S5 - 011

S6 - 0000

S7 - 0001

			Length
S1	0.25	10	2
S2	0.25	11	2
S3	0.125	001	3
S4	0.125	010	3
S5	0.125	011	3
S6	0.0625	0000	4
S7	0.0625	0001	4

$L(\text{Avg code length}) = \sum_{1}^{k} P_k L_k = (0.25\times2) + (0.25\times2) + (0.125\times3) + (0.125\times3) + (0.125\times3) + (0.0625\times4) + (0.0625\times4)$

$= 2.625$ bits/symbol

$H(x) = -\left(\sum_{1}^{k} P_k \log_2 P_k\right)$ // -ve has no value

$= P_1 \log_2 P_1 + P_2 \log_2 P_2 + \ldots + (P_7 \log_2 P_7)$

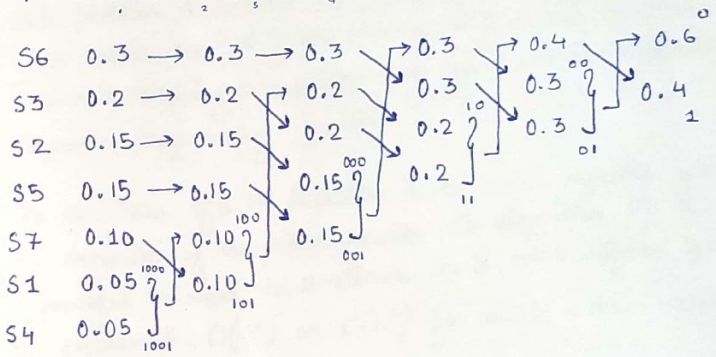
$= (0.25 \log_2 0.25) + \ldots + (0.0625 \log_2 0.0625)$

$= 2.625$ bits/symbol

$\text{Efficiency} = \frac{H}{L}\times100 = \frac{2.625}{2.625}\times100 = 100\%$

Q. Apply Huffmann code procedure for the following symbols & find the efficiency

$P = \langle 0.05, 0.15, 0.2, 0.05, 0.15, 0.3, 0.1 \rangle$



S1 - 1000

S2 - 000

S3 - 11

S4 - 1001

S5 - 001

S6 - 01

S7 - 1001

$H(x) = 2.571$

$L = 2.6$

$\text{Efficiency} = \frac{H(x)}{L}\times100$

$= \frac{2.571}{2.6}\times100$

$= 98.85\%$

Huffmann (C)

1. n = |C| // 6 no. of characters
2. Build MinHeap Q with C
3. for i ← 1 to (n-1)
   - 3.1 Allocate a new node z
   - 3.2 z → left = x = Extract_min (Q)
   - 3.3 z → right = y = Extract_min (Q)
   - 3.4 z → freq = x → freq + y → freq
   - 3.5 Insert (Q, z)
4. Return Extract_min(Q)

DYNAMIC PROGRAMMING

"Those who can not remember the past are condemned to repeat it"

"Is repeating the things for which you already have the answer, A Good thing?"

A programmer would disagree. That's what Dynamic Programming is about to always remember answers to the subproblem that you have already solved.

Given a problem which can be broken down into smaller sub-problem & smaller sub-problems can still be broken into smaller ones & if you manage to find out there are some overlapping sub-problem then you have encountered a Dynamic Programming problem.

The core idea of a DP problem is to avoid repeated work by remembering the partial results. In algorithm, DP is a powerful technique that allows one to solve different types of problem in $O(n^2)$ or $O(n^3)$ for which a naive approach would take exponential time.

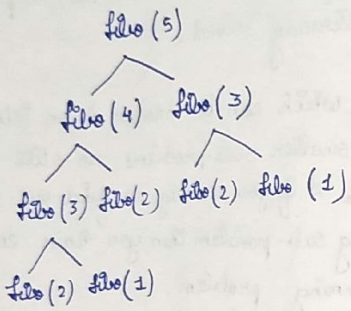
eg. 1+1+1+1+1 = on a paper, the answer is 5 here. Now write +1 in the left. So the new answer is now 6. Here we do not require recount because we remembered their were 5 no of ones.

eg. Fibonanci series

$$Fibo(n) = 1, \text{ if } n=0$$
$$Fibo(n) = 1, \text{ if } n=1$$
$$Fibo(n) = Fibo(n-1) + Fibo(n-2)$$

Recursive Code:

```
int fibo(int n){
    if(n<2)
        return 1;
    return fibo(n-1)+fibo(n-2);
}
```



DP code:

```
void fibo(){
    fiboresult[0] = 1;
    fiboresult[1] = 1;
    for(int i=2; i<n; i++){
        fiboresult[i] = fiboresult[i-1] + fiboresult[i-2];
    }
```

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

Every DP problem has a schema to be followed.

1. Show that the problem can be broken down into optimal sub-problem.
2. Recursively define the value of the solution by expressing in terms of optimal solution for smaller sub-problem.
3. Compute the value of optimal solution in bottom-up approach
4. Construct an optimal solution from the computed information.

Application

1. Matrix-Chain Multiplication
2. Longest Common Subsequence
3. 0,1 Knapsack Problem
4. All pair shortest path (Floyd-Warshall Algo)
5. Travelling salesman (TSP) problem

1. Matrix-Chain Multiplication

It can be defined as - "Find optimal parenthesis of a chain of matrices to be multiplied such that number of scalar multiplication is minimised."

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times3} \quad B = \begin{bmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{bmatrix}_{3\times2}$$

$A \times B = C$

$2\times3 \quad 3\times2 \quad 2\times2$

cost = $2\times3\times2 = 12$

So $A_{p\times q} * B_{q\times r} = C_{p\times r}$

cost = $p*q*r$

eg. $A_1 = 10\times100$ $A_2 = 100\times5$ $A_3 = 5\times50$ ~~$5\times10\times50$~~

Question $(A_1\ A_2)A_3 \Rightarrow (10\times100\times5)+(10\times5\times50) = 7500$

or

$A_1(A_2\ A_3) \Rightarrow (10\times100\times50)+(100\times5\times50) = 75000$

So our job is to minimise the cost of scalar multiplication. In the above example $(A_1, A_2)A_3$ is selected

Dynamic Programming for Chained Matrix Multiplication:

S1: Characterize the structure of optimal solution

OR

Finding the optimal sub-structure

We are having a chain of matrices ie. $A_1, A_2 \ldots A_n$

And a sub structure of it will be, if we split at 'k'

eg. A_1, A_2, A_3, A_4, A_5 & $k = 3$, $(A_1 A_2 A_3)(A_4 A_5)$

$[A_1 \ldots A_k] \times [A_{k+1} \ldots A_n]$, so our cost will be

cost of $(A_i \ldots A_k)$ + cost of $(A_{k+1} \ldots A_n)$

i.e. when we place a set of parenthesis, we divide the problem into sub-problems of smaller size. Therefore, this problem has optimal sub-structure property & it can be solved using recursion.

S2: Define recursive solution

Let say the cost is $m[i,j]$, for multiplying $A_i \ldots A_j$, where A_i has the dimension $(P_{i-1} * P_i)$ and so on.

Assume we split at 'K'

$A_i (P_{i-1} * P_i)$, $A_{i+1} (P_i * P_{i+1}) \ldots$

$$m[i,j] = m[i,k] + m[k+1,j] + (P_{i-1} * P_k * P_j)$$

$$\text{So, } m[i,j] = \begin{cases} 0, & \text{if } i == j \text{ (single matrix)} \\ \min\limits_{i \le k < j} \left[m(i,k) + m(k+1,j) + (P_{i-1} P_k P_j) \right] & \text{if } i < j \end{cases}$$

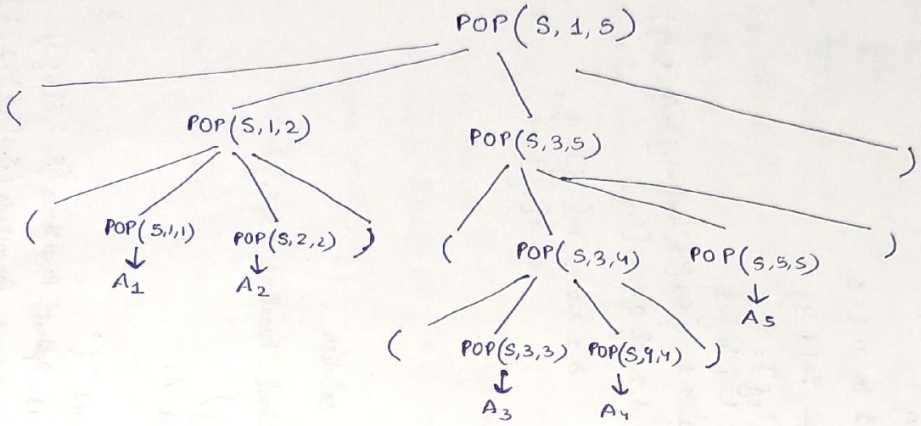
S3: Let $m[1 \ldots n, 1 \ldots n]$ a 2-d array which stores the min cost

Let $S[1 \ldots n-1, 2 \ldots n]$ a 2-d array which stores the value of 'k' for which $m[i,j]$ is optimum / minimum.

```
Algm Matrix_Chain_Order(P)  // P → an array which has the dimension of the matrix.
1. n ← Length(P) - 1
2. for i ← 1 to n
   2.1 m[i,i] = 0 , S[i,i] = 0
3. for l ← 2 to n
   3.1. for i ← 1 to n-l+1
        3.1.1 j ← i+(l-1)
        3.1.2 m[i,j] = ∞
        3.1.3 for k = i to j-1
              3.1.3.1 q ← m[i,k] + m[k+1,j] + (P_{i-1} P_k P_j)
              3.1.3.2 if q < m[i,j]
                      3.1.3.2.1 m[i,j] = q ,
                                s[i,j] = k
4. Return m, S
```

S4: Construct optimal solution

```
Algm Point_Optimal_Parenthesis (S, i, j)
1. if (i == j)
   1.1 print Ai
2. else
   2.1 print "("
   2.2 print_Optimal_Parenthesis (S, i, S[i,j])
   2.3 print_Optimal_Parenthesis (S, S[i,j]+1, j)
   2.4 print ")"
3. return 0
```


POP(S, 1, 5)
POP(S, 1, 2)
POP(S, 3, 5)
POP(S, 1, 1)
POP(S, 2, 2)
POP(S, 3, 4)
POP(S, 5, 5)
POP(S, 3, 3)
POP(S, 4, 4)
A1
A2
A3
A4
A5

$((A_1 A_2)((A_3 A_4) A_5))$

Q. Find the optimal parenthesis of matrix chain whose sequence of dimension is $P\langle 4, 10, 3, 12, 20, 7\rangle$?

Ans. $P\langle 4, 10, 3, 12, 20, 7\rangle$

$P_0\ P_1\ P_2\ P_3\ P_4\ P_5$

$A_{1\ 4\times10}\quad A_{2\ 10\times3}\quad A_{3\ 3\times12}\quad A_{4\ 12\times20}\quad A_{5\ 20\times7}$

1	0	120	264	1080	1344
2	×	0	360	1320	1350
3	×	×	0	720	1140
4	×	×	×	0	1680
5	×	×	×	×	0
	1	2	3	4	5

$m[5,5]$

	2	3	4	5
1	1	2	2	2
2	×	2	2	2
3	×	×	3	4
4	×	×	×	4

$s[4,4]$

$$m[1,2] = m[i,k] + m[k+1,j] + (P_{i-1}\ P_k\ P_j)$$
$$i=1,\ j=2,\ k=1$$
$$= m[1,1] + m[2,2] + (P_0\ P_1\ P_2)$$
$$= 0 + 0 + 4\times10\times3 = 120$$

$$m[2,3] = m[2,2] + m[3,3] + (P_1\ P_2\ P_3)$$
$$i=2,\ j=3,\ k=2$$
$$= 0 + 0 + 10\times3\times12$$
$$= 360$$

$$m[3,4] = m[3,3] + m[4,4] + (P_2\ P_3\ P_4)$$
$$i=3,\ j=4,\ k=3$$
$$= 3\times12\times20$$
$$= 720$$

$$m[4,5] = m[4,4] + m[5,5] + (P_3\ P_4\ P_5)$$
$$i=4,\ j=5,\ k=4$$
$$= 12\times20\times7$$
$$= 1680$$

$$m[1,3] = \min \begin{cases} k=1 \\ m[1,1] + m[2,3] + (P_0\ P_1\ P_3) \\ = 0 + 360 + (4\times10\times12) \\ = 360 + 480 \\ = 840 \\ k=2 \\ m[1,2] + m[3,3] + (P_0\ P_2\ P_3) \\ = 120 + 0 + (4\times3\times12) \\ = 120 + 144 = 264\ (\checkmark) \end{cases}$$

$$m[1,4] = \begin{cases} k=1 \\ m[1,1] + m[2,4] + (P_0\ P_1\ P_4) \\ = 0 + 1320 + (800) \\ = 2120 \\ k=2 \\ m[1,2] + m[3,4] + (P_0\ P_2\ P_4) \\ = 120 + 720 + 240 \\ = 1080\ (\checkmark) \\ k=3 \\ m[1,3] + m[4,4] + (P_0\ P_3\ P_4) \\ = 264 + 0 + \\ = 1224 \end{cases}$$

$i=1,\ j=4,\ k=\{1,2,3\}$

$A_1\ A_2\ A_3\ A_4\ A_5$

$A_1A_2 \quad A_3A_4A_5$

$A_3A_4 \quad A_5$

$(A_1A_3)((A_3A_4)A_5)$

$((A_1A_2)((A_3A_4)A_5))$

$T(n) = 2T(n/2) + n \log n$

2 2

$T(n) = 4T(n/2) + n^2 \log n$

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

$T(n) = 3T(n/4) + n \log n$

$3 < 4$

$\Theta(n \log n)$

Longest Common Subsequence (LCS)

In biology, we often compare the DNA of two different strings. DNA consists of a string of molecules called Bazes. Generally we compare two strings to know how similar they are

Problem Statement : We are given two strings

X of length n

Y of length m

Our goal is to produce their largest common subsequence i.e. the longest sequence of characters that appear left to right (but not necessarily in continuous block) in both strings.

Eg. Let $X = \langle A B A Z D C \rangle$

$Y = \langle B A C B A D \rangle$

Here the LCS is $\langle A B A D \rangle$

Dynamic Programming Approach

S1: Characterizing the LCS

S2: Let $X = \langle x_1, x_2, x_3 \ldots x_n \rangle$

$Y = \langle y_1, y_2, y_3 \ldots y_m \rangle$

And let the LCS is $Z = \langle Z_1, Z_2, \ldots Z_k \rangle$

It means that Z is appearing in both X & Y and Z_1, Z_2 are also following index sequences means, Z_2 only appear after Z_1 in both X & Y.

(1) If $(x_m == y_n)$, then $Z_k = x_n$ or y_m

then Z_{k-1} should we get from X_{n-1} & Y_{m-1}

$X = \langle x_1, x_2 \ldots x_{n-1}, x_n \rangle$

$Y = \langle y_1, y_2 \ldots y_{m-1}, y_m \rangle$

(2) If $(X_n \neq Y_m)$ & $(Z_k \neq x_n)$, then Z_k we may get from X_{n-1} or Y_m

$X = \langle x_1, x_2 \ldots x_{n-1}, x_n \rangle$

$Y = \langle y_1, y_2 \ldots y_{m-1}, y_m \rangle$

$Z = \langle z_1, z_2 \ldots z_{k-1}, z_k \rangle$

(3) If $X_n \neq Y_m$, $Z_k \neq Y_m$, then Z_k we may get from X_n or Y_{m-1}

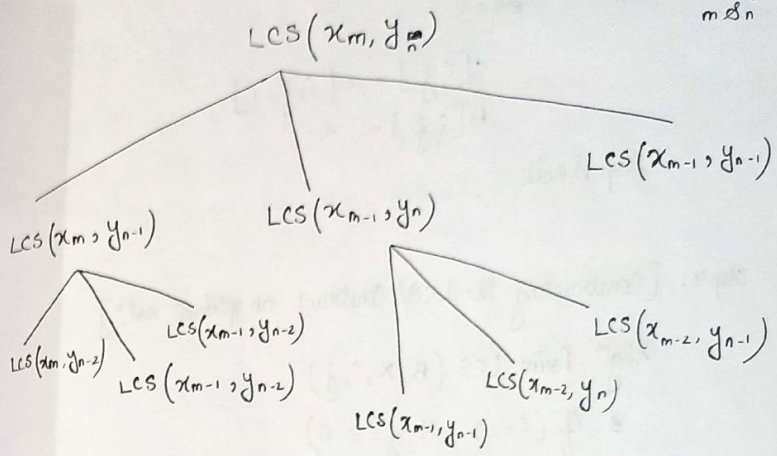
So, in this way, we are minimising the problem that means, the LCS has optimal substructure characteristics.

S2: (A recurrance solution)

Let $c[i,j]$ be the length of LCS of $[x_1, \ldots x_i]$ & $[y_1, \ldots y_j]$

$$c[i,j] = \begin{cases} 0, & \text{if } i=0 \text{ \& } j=0 \\ c[i-1, j-1] + 1, & \text{if } i,j>0 \text{ \& } x_i == y_j \\ \text{MAX}\left[c[i,j-1], c[i-1,j]\right], & \text{if } i,j>0 \text{ AND } x_i \neq y_j \end{cases}$$

Interchanged m & n



S3: [Completing the length of LCS]

Algm LCS_Length (X, Y)

m ← X.length, n ← Y.length

Let B[1...m, 1...n] & C[0...m, 0...n] be two new arrays / tables // B ← stores the symbols // C ← stores the length of LCS

for i ← 1 to m
    c[i, 0] ← 0

for j ← 1 to n
    c[0, j] ← 0

(i-1), (j-1)	(i-1), j
i, j-1	i-j

for i ← 1 to m
    for j ← 1 to n
        if $(x_i == y_j)$ then
            c[i, j] = c[i-1, j-1] + 1
            B[i, j] = "↖"
        else if (c[i-1, j] >= c[i, j-1]) then
            c[i, j] = c[i-1, j]
            B[i, j] = "↑"

else {

$c[i,j] = c[i,j-1]$

$B[i,j] = "\leftarrow"$

Stop & exit

Step 4: [Constructing the LCS/ Construct an optimal soln]

Algm Print_LCS (B, X, i, j)

1. if (i==0 || j==0)
   1.1. Return NULL
2. if B[i,j] == "↖"
   2.1. Print_LCS (B, X, i-1, j-1)
   2.2 print x;
3. else if (B[i,j] == "↑")
   3.1 Print_LCS (B, X, i-1, j)
4. else
   4.1 Print_LCS (B, X, i, j-1)
5. Stop & Exit.

Eg. $X = \langle A\ B\ A\ C\ A \rangle$ $\quad Y = \langle B\ C\ B\ B\ A \rangle$

	Col 0	B	C	B	B	A
Row 0	0	0	0	0	0	0
A	0	0↑	0↑	0↑	0↑	1↖
B	0	1↖	1←	1↖	1↑	1↑
A	0	1↑	1↑	1↑	1↑	2↖
C	0	1↑	2↖	2←	2←	2↑
A	0	1↑	2↑	2↑	2↑	3↖

↳ Length of LCS

B C A

If $(x_i == y_j)$, then store the "diagonal value + 1" & "↖"

If $(x_i \neq y_j)$, so, find the greater between top & left

(a) if top > left, then store top value & put "↑"

(b) if top == left, then store as per step (a)

(c) if top < left, then store left value & put "←"

$X = \langle A\ B\ C\ B\ D\ A\ B \rangle$ $\quad Y = \langle B\ D\ C\ A\ B\ A \rangle$

	Col 0	B	D	C	A	B	A
Row 0	0	0	0	0	0	0	0
A	0	0↑	0↑	0↑	1↖	1←	1↖
B	0	1↖	1←	1←	1↑	2↖	2←
C	0	1↑	1↑	2↖	2←	2↑	2↑
B	0	1↖	1↑	2↑	2↑	3↖	3←
D	0	1↑	2↖	2↑	2↑	3↑	3↑
A	0	1↑	2↑	2↑	3↖	3↑	4↖
B	0	1↖	2↑	2↑	3↑	4↖	4↑

↳

BCBA

$X = \langle S\ T\ U\ X\ A\ B\ D\ X\ Y \rangle$

$Y = \langle T\ A\ D\ B\ B\ D\ Y\ X \rangle$

	Col 0	T	A	D	B	B	D	Y	X
Row 0	0	0	0	0	0	0	0	0	0
S	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
T	0	1↖	1←	1←	1←	1←	1←	1←	1←
U	0	1↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑
X	0	1↑	1↑	1↑	1↑	1↑	1↑	1↑	2↖
A	0	1↑	2↖	2←	2←	2←	2←	2←	2↑
B	0	1↑	2↑	2↑	3↖	3↑	3←	3←	3←
D	0	1↑	2↑	3↖	3↑	3↑	4↖	4←	4←
X	0	1↑	2↑	3↑	3↑	3↑	4↑	4↑	5↖
Y	0	1↑	2↑	3↑	3↑	3↑	4↑	5↖	5↑

TA B D X

$P = \langle 5, 3, 2, 4, 3 \rangle$
$P_0\ P_1\ P_2\ P_3\ P_4$

1	1	2	3	4
2	×	2	3	4
3	×	×	3	4

A_1

1	0	30	70	84
2	×	0	24	42
3	×	×	0	24
4	×	×	×	0

$m[1,2] = m[1,1] + m[2,2] + P_0 P_1 P_2 \quad (5 \times 3 \times 2)$

$k=1 \quad = 30$

$m[2,3] = 0 + 0 +$

$k=2$

$m[1,3] = m[1,1] + m[2,3] + P_0 P_1 P_3$

$k=1,2 \quad = 0 + 24 + 60$

$m[1,3] = m[1,2] + m[3,3] + P_0 P_2 P_3$

$k=2 \quad = 30 + 40$

$m[2,4] = m[2,2] + m[3,4] + P_1 P_2 P_4$

$k=2,3 \quad 24 + 18 = 42$

$k=2 \quad = m[2,3] + m[4,4] + P_1 P_3 P_4$

$24 + 36$

$m[1,4] = m[1,1] + m[2,4] + P_0 P_1 P_4$

$k=1,2,3 \quad = 42 + 45 = 87$

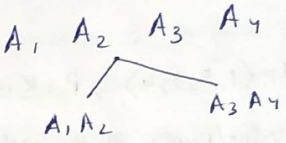
$k=1$

$k=2 \quad m[1,2] + m[3,4] + P_0 P_2 P_4$

$= 30 + 24 + 30 = 84$

$k=3 \quad m[1,3] + m[4,4] + P_0 P_3 P_4$

$= 60 + 60 = 120$



$(A_1 A_2)\ (A_3 A_4)$

0/1 Knapsack

0/1 Knapsack, means, either the whole item is selected or not selected at all.

→ Given a knapsack of capacity "k". There are "n" no. of items whose weights are $< w_1, w_2, \ldots w_n >$ & its profits are $< P_1, P_2 \ldots, P_n >$ respectively.

Eg.

Given $k = 8$, $W = \langle 1, 5, 3, 4 \rangle$, $P = \langle 15, 10, 9, 5 \rangle$

$T(i, j)$ = Max value/profit of the selected items, if we are allowed to take items 1 to i and we have a weight restriction of j.

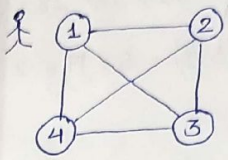
The matrix 'T' has $(n+1)$ rows & $(\frac{k}{\omega}+1)$ columns

$$T(i, j) = \text{Max}\left\{ T(i-1, j), (\text{value}_i) + T(i-1, j-(\text{weight}_i)) \right\}$$

Item 1 is visited

P	W	Row \ Column	0	ω=1	ω=2	3	4	5	6	7	8
		i=0	0	0	0	0	0	0	0	0	0
15	1	i=1	0	15	15	15	15	15	15	15	15
10	5	2	0	15	15	15	15	15	25	25	25
9	3	3	0	15	15	15	24	24	25	25	25
5	4	4	0	15	15	15	24	24	25	25	29

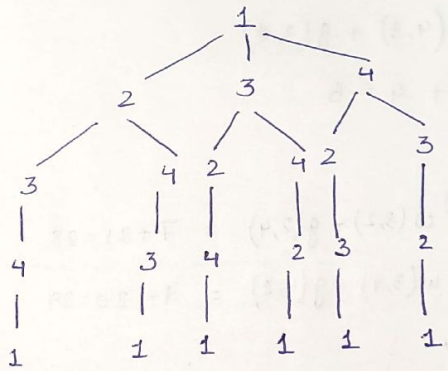
TSP (Travelling Salesman Problem)



	1	2	3	4
1	0	16	11	6
2	8	0	13	16
3	4	7	0	9
4	5	12	2	0

Given a set of cities & the distance b/w every pair of cities, the problem is to find the minimum distance covered by visiting other cities exactly one except the source. This problem can be solved using DP & branch & bound

DP



$$g(i, S) = \min \left(\omega[i, j] + g(j, S - j) \right)$$

i → starting vertex

S → set of vertices the salesman will travel

$$g(1,\{2,3,4\}) = \min \begin{cases} w(1,2) + g(2,\{3,4\}) = 16 + 22 = 38 \\ w(1,3) + g(3,\{2,4\}) = 11 + 28 = 39 \\ w(1,4) + g(4,\{2,3\}) = 6 + 17 = 23 \end{cases}$$

$$g(2,\{3,4\}) = \min \begin{cases} w(2,3) + g(3,4) = 13 + 14 = 27 \\ w(2,4) + g(4,3) = 16 + 6 = 22 \end{cases}$$

$$g(3,4) = \min \{ w(3,4) + g(4,\phi) \} \qquad g(x,\phi) = w(x,1)$$

$$\min (9 + 5)$$

$$= 14$$

$$g(4,3) = w(4,3) + g(3,\phi)$$

$$= 2 + 4 = 6$$

$$g(3,\{2,4\}) = \min \begin{cases} w(3,2) + g(2,4) = 7 + 21 = 28 \\ w(3,4) + g(4,2) = 9 + 20 = 29 \end{cases}$$

$$g(2,4) = w(2,4) + g(4,\phi) = 16 + 5 = 21$$

$$g(4,2) = w(4,2) + g(2,\phi) = 12 + 8 = 20$$

$$g(4,\{2,3\}) = \min \begin{cases} w(4,2) + g(2,3) = 12 + 17 = 29 \\ w(4,3) + g(3,2) = 2 + 15 = 17 \end{cases}$$

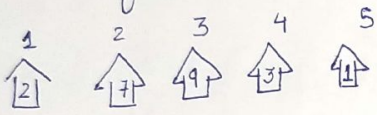
$$g(2,3) = w(2,3) + g(3,\phi) = 13 + 4 = 17$$

$$g(3,2) = w(3,2) + g(2,\phi) = 7 + 8 = 15$$

$$1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$$

The House Robber Problem

You are a thief who has found a block of houses to roab. Each house i, has a non-negative $V(i)$, worth of value inside, that you can steal. However, due to security system ~~reason~~ of the house, you will get caught if you roab two adjacent houses. What is the maximum value you can steal from the block?



Edit Distance

Secret to Bisect

Minimum no: of changes (Insertion & Deletion)

Disjoint Sets

→ Two sets are said to be disjoint sets if they have no common elements.

For eg. $S1 = \langle A, B, C \rangle$
$S2 = \langle P, Q, R \rangle$

Here S1 & S2 are disjoint sets.

→ Disjoint set data structure contains a set of disjoint sets where $S = \{S1, S2, \ldots Sn\}$ such that ~~$S1 \cap S2 = \phi$~~ $S_i \cap S_j = \phi$

→ Each set is identified by a representative. For eg -

$S1 = \{A, B, C\}$, $S2 = \{P, Q, R\}$

Here A & P are representative of S1 & S2 respectively.

Operations

There are in general three disjoint operation -

(i) Make_set(X) - It creates a new set whose only member is X for time being.

(ii) Union (X, Y) - It combine the sets which contains the element X & Y. The union operations is possible if the representative of the member are not same.

$X = \{A, B, P\}$
$Y = \{M, N, Q\}$
} Union (B, N) $= \{A, B, P, M, N, Q\}$

(iii) Find_set(X) - It will return the representative of the set in which X is a member.

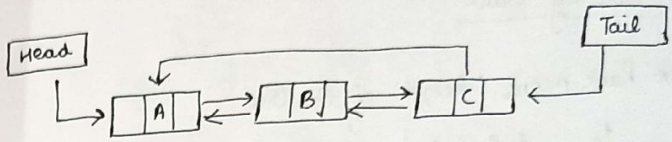
Eg. $X = \{A, B, C, D\}$
Find_Set(C) = {A}

Representation of Disjoint Sets -

(i) Linked List (ii) Tree

(i) Linked List

Let $S1 = \{A, B, C\}$, $S2 = \{P, Q, R\}$

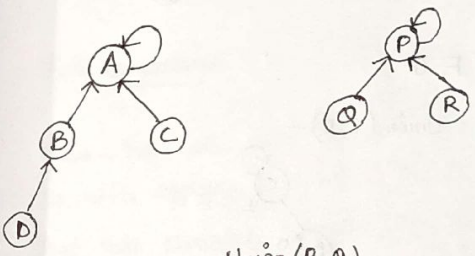


* First element of the linked list is the representative

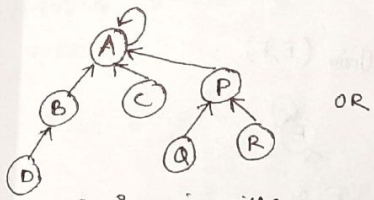
(ii) Tree Representation

* Root of the tree is representative
* Each node points to its parent node.
* Root points to itself.

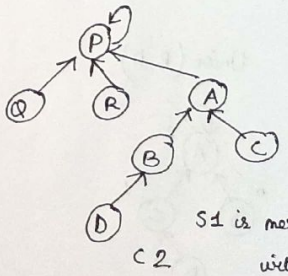
$S1 = \{A, B, C, D\}$ $S2 = \{P, Q, R\}$



Union (B, Q)



OR



S2 is merging with S1
C1

C2 S1 is merging with S2.

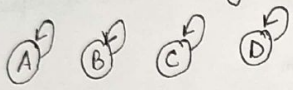
C1 is preffered over C2, as in C1 height of the tree doesn't increase.

∴ This is called Union by Rank.

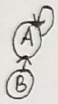
Union by Rank

* Rank means 'height' of the tree

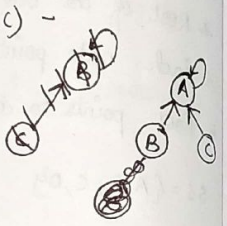
Eg. A, B, C, D, initially their are 4 disjoint sets



Union (A, B) –

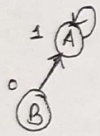


Union (B, C) –

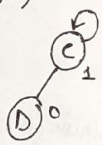


Eg – A, B, C, D, E, F, G

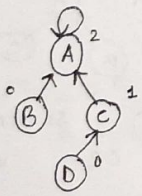
Union (A, B) –



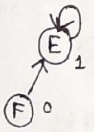
Union (C, D) –



Union (B, D)

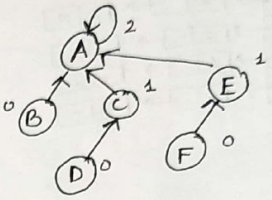


Union (E, F)

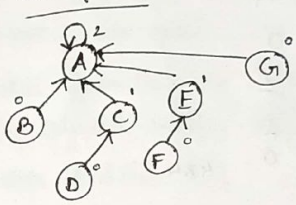


Union (B, E)

While merging two trees, ranks of the two trees are compared higher smaller
than root having higher rank is combined with root having smaller rank.



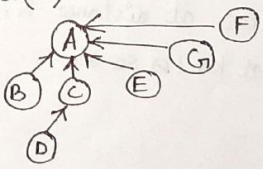
Union (A, G)



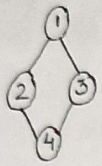
Path Compression

Find – Set needs more time for some specific leaf nodes
So, while applying find-set for any leaf node, we make the leaf node connected to the root directly, so that next time performing find-set operation on same leaf node seems faster.

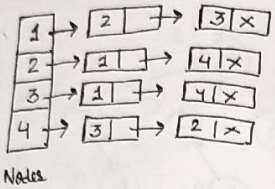
Find – set (F) –



Graph



Linked List



Adjacency Matrix

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

4×4

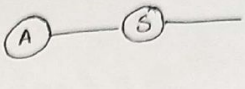
Traversing

Breadth First Search (BFS) (Queue)

Depth First Search (DFS) (Stack)

BFS

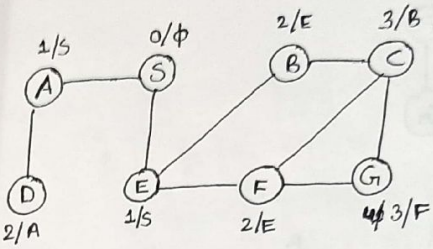
Given a graph $G=(V,E)$ in which S is a source vertex, BFS will find out every vertex that are reachable from S. It also computes the distance from S to every reachable vertex. The name of algo is BFS because this algo discovers all vertices at distance k, before exploring the vertices at k+1 & so on.



x/y

x → distance from source

y → Parent of this vertex



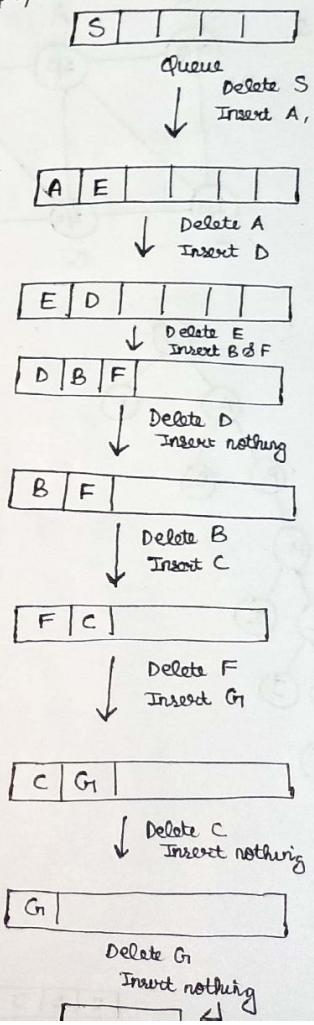
The starting vertex is S

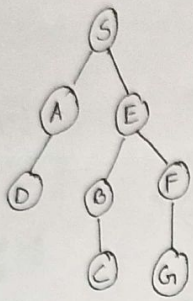
Insert S into queue

Delete from the queue & insert all its adjacent vertices into queue

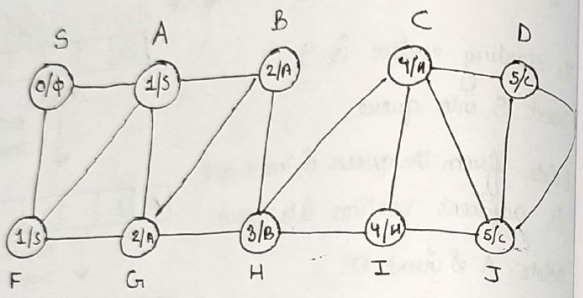
Delete A & insert D

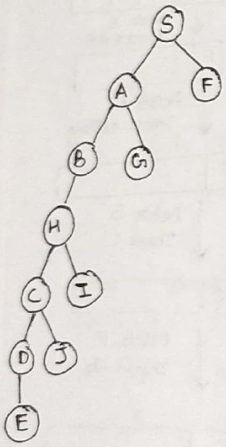
Delete E & insert B & F

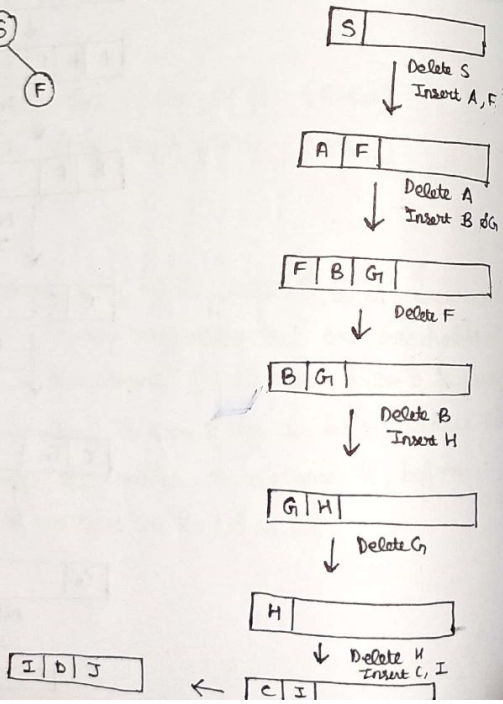



S
A
E
D
B
F
C
G1

Q.


S
A
B
C
D
0/$
1/S
2/A
4/A
5/C
1/S
2/A
3/B
4/H
5/C
F
G1
H
I
J


S
A
F
B
G1
H
C
I
D
J
E


S
Delete S
Insert A, F
A F
Delete A
Insert B & G1
F B G1
Delete F
B G1
Delete B
Insert H
G1 H
Delete G1
H
Delete H
Insert C, I
I D J
C I

BFS algo^m

BFS use colour schemas i.e. white, grey & black to keep track of progress.

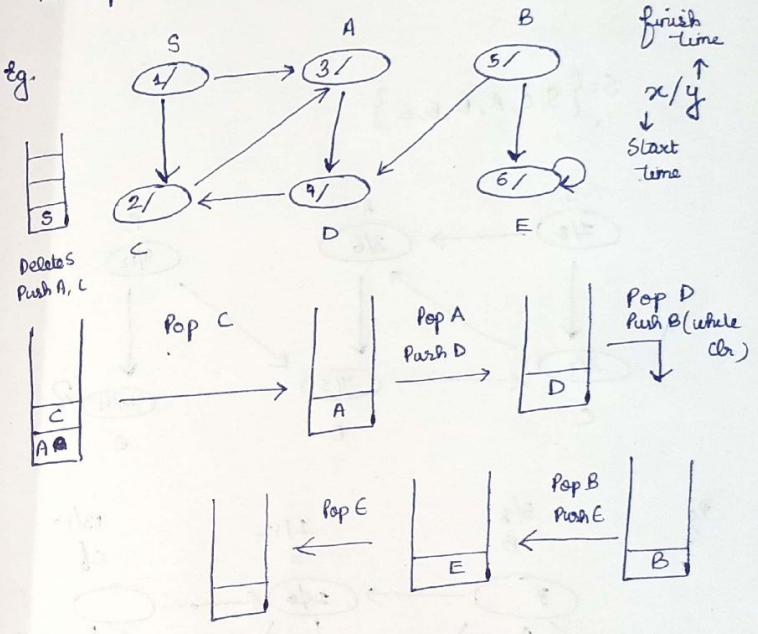
BFS(G, S) // G → Adjacency Matrix, S → Source vertex

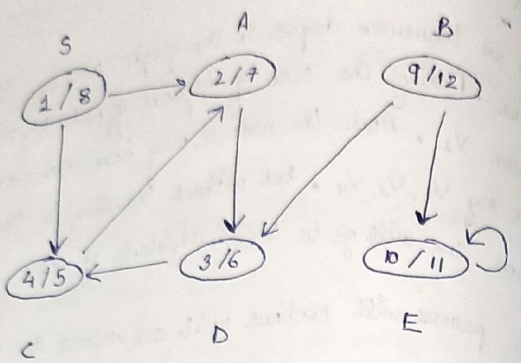
1. for each vertex ∀ U ← V[G] − {S}
   - 1.1 do colour [U] ← WHITE
   - 1.2 d [U] ← ∞
   - 1.3 P [U] ← NIL
2. colour [S] ← GRAY
3. d[S] ← 0 , P[S] ← NIL
4. Q ← ϕ // Queue 'Q' is initialized to NULL
5. Enque (Q, S)
6. While (Q ≠ ϕ)
   - 6.1 ∀ U ← dequeue (Q)
   - 6.2 For each vertex V ← adj [U]
     - 6.2.1 if colour [V] ← WHITE
       - 6.2.1.1 colour [V] ← GRAY
       - 6.2.1.2 d[V] ← d[U] + 1
       - 6.2.1.3 P[V] ← U
       - 6.2.1.4 Enqueue (Q, V)
   - 6.3 colour [U] ← Black
7. Stop & Exit

DFS (Depth First Search)

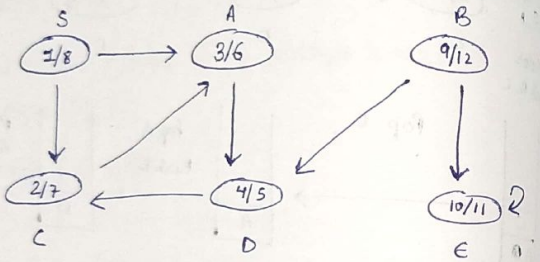
In DFS we traverses deeper in the graph wherever it is possible. During the scan of adjacent of node U, we find a vertex V_1. And the node U, may have other adjacent let's say V_2, V_3, V_4, but without travelling to these vertices, we will go to V_1 & adjustants of V_1.

This process will continue until all vertices are discovered.

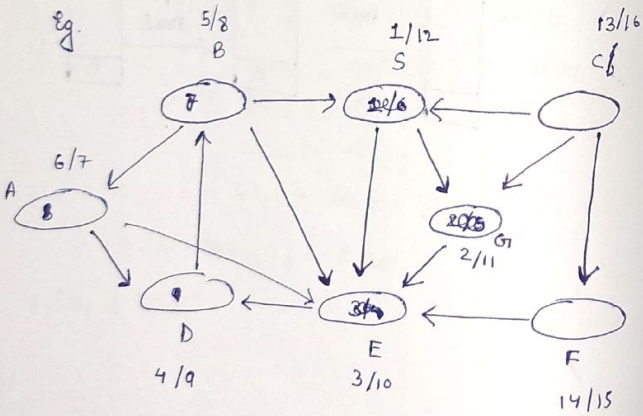


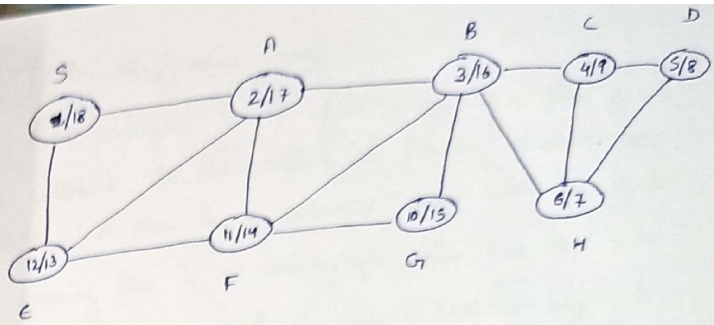

S
A
B
1/8
2/7
9/12
4/5
3/6
10/11
C
D
E

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


S
A
B
1/8
3/6
9/12
2/7
4/5
10/11
C
D
E

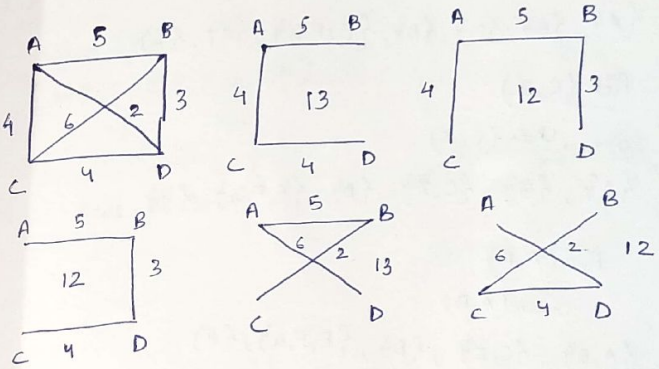
Eg.


5/8
B
1/12
S
13/16
6/7
A
2/11
G1
D
E
F
4/9
3/10
14/15


A
B
C
D
S
2/17
3/16
4/9
5/8
6/7
10/15
11/14
12/13
G1
F
H
E

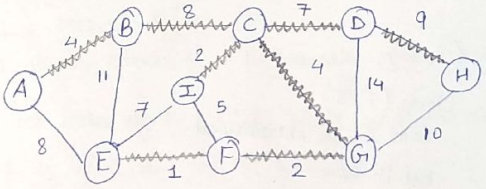
MST (Minimal Spanning Tree)

The MST problem is to find a tree of a given graph that contains all the vertices & has a minimum total weight. A tree is a connected graph with no cycle. A spanning tree is a sub-graph which has the same set of vertices wrt graph & doesn't contain any cycle.



Kruskal Algm (Greedy Algm)

eg.



Sort all edges in non-decreasing order.

(E,F) (F,G) (C,I) (A,B) (C,G) (F,I) (E,I) (C,D) (B,C) (A,E) (D,H) (G,H) (B,E) (D,G)

As per the algo, initially all ... sets.
Pick the edge with smallest weight & apply the union fi...

Pick (E,F) Union (E,F)

{A}, {B}, {C}, {D}, {E,F}, {G}, {H}, {I}

Pick (F,G)

Union (F,G)

{A}, {B}, {C}, {D}, {E,F,G}, {H}, {I}

Pick (C,I)

Union (C,I)

{A}, {B}, {C,I}, {D}, {E,F,G}, {H}

Pick (A,B)

Union (A,B)

{A,B}, {C,I}, {D}, {E,F,G}, {H}

Pick (C,G)

Union (C,G)

{A,B}, {C,I,E,F,G}, {D}, {H}

Pick (F,I)

Union (F,I)

Same leader, not allowed

Not Allowed

Pick (E,I)

Union (E,I)

Same leader not allowed

Pick (C,D)

Union (C,D)

{A,B}, {C,I,E,F,G,D}, {H}

Pick (B,C)

Union (B,C)

{A,B,C,I,E,F,G,D}, {H}

Pick (A,E)

Union (A,E)

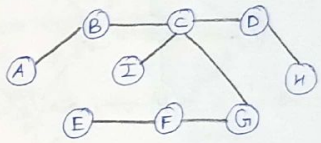
Not allowed

Pick (D,H)

Union (D,H)

{A,B,C,I,E,F,G,D,H}

Adds upto 37

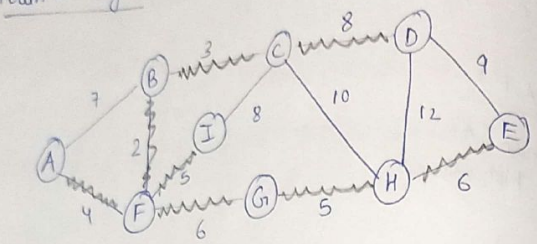


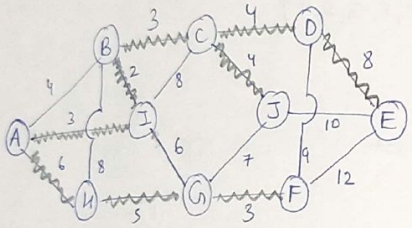
= 37

Algm Kruskal_MST (G, W)

1. A ← NULL
2. For each vertex V ∈ V[G]
   2.1 Make_set(V)
3. Sort all the edges of E in increasing order w.r.t W
4. For each edge (U,V) ∈ E, picked up in increasing order.
   4.1 if find_set(U) ≠ find_set(V)
      4.1.1 A ← A ∪ {(U,V)}
      4.1.2 Union (U,V)
5. Return A

Prim's Algo^m





Single Source Shortest Path

Dijkstra's Algm

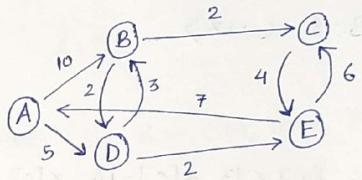
It can be used to solve, a single source shortest path problem of a directed weighted graph.

→ $G(V, E)$, for the case in which all the edges are non-negative, i.e. $W(u, v) \geq 0$

→ It has less time complexity as compared to Bellman Ford Algo^m

→ The Belman Ford Alg^m is used to solve, if the graph is having a -ve weight

It use minimum Priority Q for implementation.



Let A be the source

Vertex	Adj(V)	d[V]	π[V]
A	B:10, D:5	0	NIL
B	C:2, D:2	~~∞~~ 8	~~A~~ D
C	E:4	~~∞~~ 6 ~~8~~ 10	~~E~~ B
D	B:3, E:2	~~∞~~ 5	~~A~~
E	C:6, A:7	~~∞~~ 7	~~A~~ D

Initially the Min Priority Queue =

MPQ ⟨A:0, B:∞, C:∞, D:∞, E:∞⟩

S1: Delete A from MPQ

Relax (A, B, 10) : $0+10 < \infty$, $d[B] = 10$, $\pi[B] = A$

Relax (A, D, 5) : $0+5 < \infty$, $d[D] = 5$, $\pi[D] = A$

MPQ ⟨D:5, B:10, C:∞, E:∞⟩

S2: Delete D

Relax (D, B, 3) : 5+3 < ~~∞~~ 10 , d[B] = 8, π[B] = D

Relax (D, E, 2) : 5+2 < ∞ , d[E] = 7 , π[E] = D

MPQ ⟨E:7, B:8, C:∞⟩

S3: Delete E

Relax (E, C, 6) : 7+6 < ∞ d[C] = 13, π[C] = E

Relax (E, A, 7) : 7+7 < 0

MPQ : ⟨B:8, C:13⟩

S4: Delete B

Relax (B, C, 2) : 8+2 < 13 d[C] = 10 π[C] = B

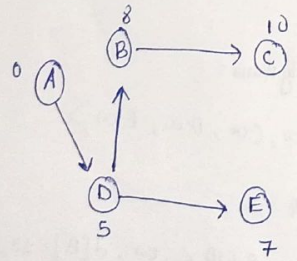
Relax (B, D, 2) : 8+2 < 5

MPQ : ⟨C:10⟩

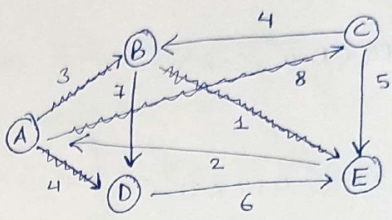
S5: Delete C

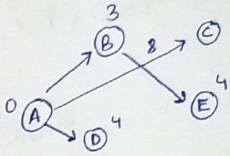
Relax (C, E, 4) : 10+4 < 7

MPQ : ⟨Empty⟩



Q.





Vertex	Adj[v]	d[v]	π[v]
A	B:3, D:4, C:8	0	NIL
B	D:7, E:1	~~∞~~ 3	~~∞~~ A
C	B:4 ; E:5	~~∞~~ 8	~~∞~~ A
D	E:6	~~∞~~ 4	~~∞~~ A
E	A:2	~~∞~~ 4	~~∞~~ B

MPQ: ⟨A:0, B:∞, C:∞, D:∞, E:∞⟩

Delete A

Relax (A, B, 3) : 0+3 < ∞ d[B] = 3 , π[B] = A

Relax (A, D, 4) : 0+4 < ∞ d[D] = 4 π[D] = A

Relax (A, C, 8) : 0+8 < ∞ d[C] = 8 π[C] = A

MPQ : ⟨B:3, D:4, C:8, E:∞⟩

Delete B

Relax (B, D, 7) : 3+7 < 4

Relax (B, E, 1) : 3+1 < ∞ d[E] = 4 , π[E] = B

MPQ : ⟨D:4, E:4, C:8⟩

Delete D

Relax (D, E, 6) = 4+6 < 4

MPS ⟨ E:4 , C:8 ⟩

Delete E

Relax (E, A, 2) = 4+2 < 0

MPS ⟨ C:8 ⟩

Delete C

Relax (C, B, 4) : 8+4 < 3

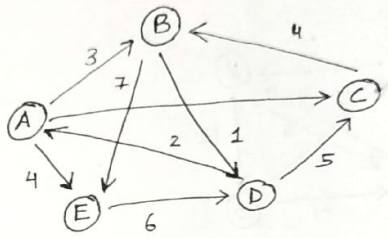
Relax (C, E, 5) : 8+5 < 4

All Pairs Shortest Path Problem

Floyd - Warshall Algorithm

This alg^{m} is used to find the shortest path b/w any two pair of vertices.

Eg.



→ We need to build a weight matrix i.e. it represents the cost between two vertices with no intermediate vertices.

W^{0} =

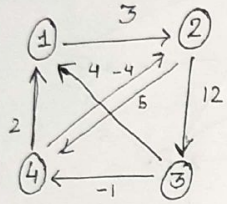
	A	B	C	D	E
A	0	3	8	∞	4
B	∞	0	∞	1	7
C	∞	4	0	∞	∞
D	2	∞	5	0	∞
E	∞	∞	∞	6	0

→ Let P is the path from i to j with all intermediate nodes. The path P may be broken down to two paths viz through k.

intermediate nodes

$$d_{ij}^{(k)} = \begin{cases} w_{ij}, & k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & k \geq 1 \end{cases}$$

Eg.



$$W^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & \infty & \infty \\ 2 & \infty & 0 & 12 & 5 \\ 3 & 4 & \infty & 0 & -1 \\ 4 & 2 & -4 & \infty & 0 \end{array}$$

Via node 1

$$W^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & \infty & \infty \\ 2 & \infty & 0 & 12 & 5 \\ 3 & 4 & 7 & 0 & -1 \\ 4 & 2 & -4 & \infty & 0 \end{array}$$

$$d_{23}^1 = \min\left(d_{23}^0, d_{21}^0 + d_{13}^0\right)$$

$$= (12, \infty)$$

$$= 12$$

$$d_{24}^1 = \min\left(d_{24}^0, d_{21}^0 + d_{14}^0\right)$$

$$= 5, \infty$$

$$= 5$$

$$d_{32}^1 = \infty, \infty, 4+3$$

$$= 7$$

$$d_{34}^1 = -1, 4+\infty$$

$$= -1$$

$$d_{42}^1 = -4, 2+3$$

$$d_{43}^1 \quad \infty, 2+\infty$$

Via Node 2

$$W^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 15 & 8 \\ 2 & \infty & 0 & 12 & 5 \\ 3 & 4 & 7 & 0 & -1 \\ 4 & 2 & -4 & 8 & 0 \end{array}$$

$$d_{13}^2 = \infty, 3+12$$

$$d_{14}^2 = \infty, 3+5$$

$$d_{31}^2 = 4, 7+\infty$$

$$d_{34}^2 = -1, 7+5$$

$$d_{41}^2 = 2, -4+\infty$$

$$d_{43}^2 = \infty, -4+12$$

Via Node 3

W_3 =

	1	2	3	4
1	0	3	15	8
2	16	0	12	5
3	4	7	0	-1
4	2	-4	8	0

$d^3_{12} = 3, 15+7$

$d^3_{14} = 8, 15+-1$

$d^3_{21} = \infty, 12+4$

$d^3_{24} = 5, 12+-1$

$d^3_{41} = 2, 8+4$

$d^3_{42} = -4, 8+7$

Via Node 4

W_4 =

	1	2	3	4
1	0	3	15	8
2	7	0	12	5
3	1	-5	0	-1
4	2	-4	8	0

$d^4_{12} = 3, 8+-4$

$d^4_{13} = 15, 8+8$

$d^4_{21} = 16, 5+2$

$d^4_{23} = 12, 5+8$

$d^4_{31} = 4, -1+2$

$d^4_{32} = 7, -1+-4$

↖ Final answer

Back Tracking

It is a general algo^m methodology for finding all or some solutions to some problems that incrementally builds the solutions.

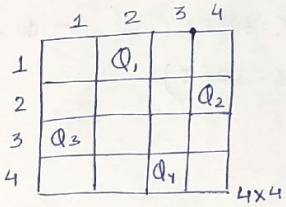
N - Queens Problem

We are given N no: of queens to be placed on N×N chess board, so that no two queens attack each other.

$N \geq 4$

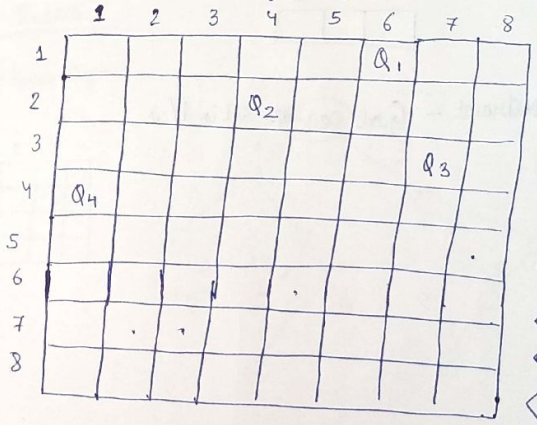
4 - Queens Problem

N = 4, So chess board size is 4×4.



<2, 4, 1, 3>

Q. Solve 8-Queen Problem for a feasible sequence <6, 4, 7, 1>



<3, 5, 2, 8>

<8, 2, 5, 3>

Sum of sub-set problems

$S = \{5, 10, 12, 13, 15, 18\}$

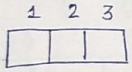
Sum = 30

Sol: First sort in ascending order

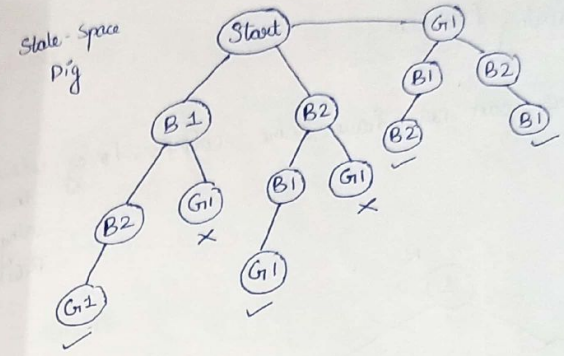
SubSet	Sum (30)	Action
{5}	5 <= 30	Add next no
{5,10}	15 <= 30	— do —
{5,10,12}	27 <= 30 (F)	Backtrack
{5,10,13}	28 <= 30 (F)	— do —
{5,10,15}	30 <= 30 (T)	

Ex – B1, B2, G1

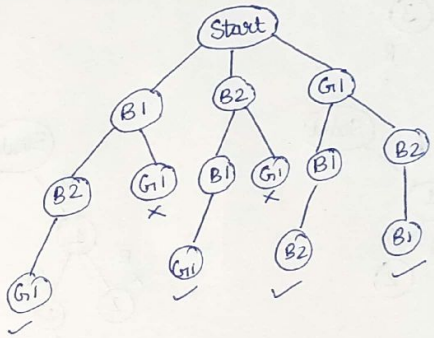
Allow them to sit in three chairs.



Constraint – Girl can not sit in b/w

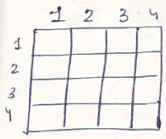


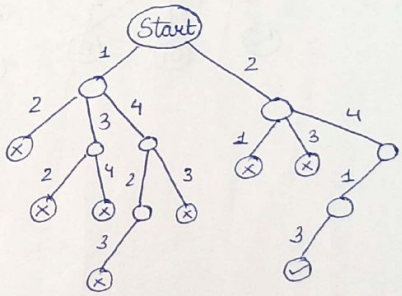
Branch n Bound (BFS)



4 - Queens

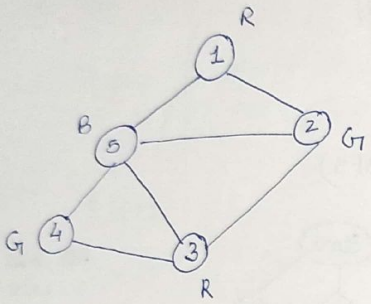
State Space Diagram

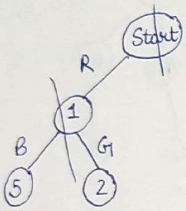


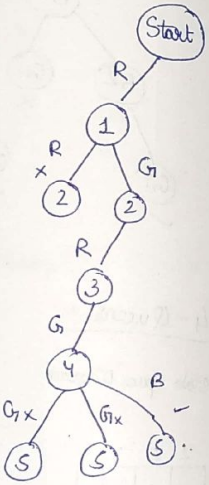


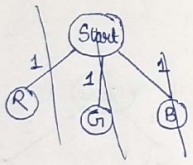
Graph Colouring Problem

Neighbours can not have same colour. For eg colour the vertices using R, G, B

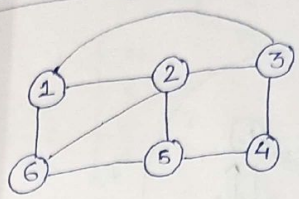


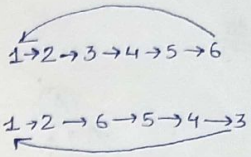




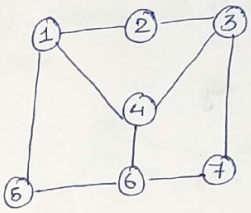


Hamiltonian Cycle (Back-tracking)

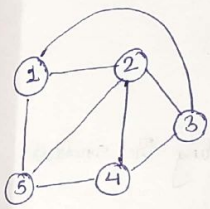


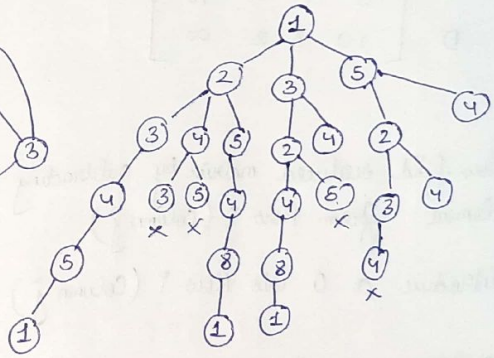


So, the above graph has hamiltonian cycle



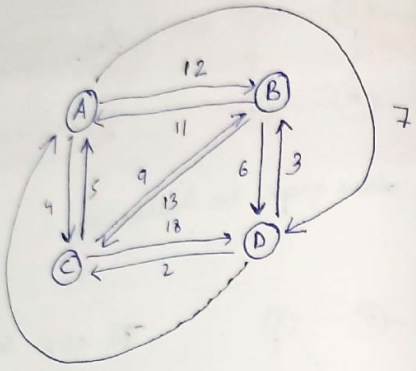
This graph does not have any hamilton cycle





TSP (Branch n Bound)

eg



To city

S1:

From city \ To city	A	B	C	D
A	∞	12	4	7
B	11	∞	13	6
C	5	9	∞	18
D	10	3	2	∞

S2 Now find reduced matrix by subtracting the smallest element from Row i (Column j)

Introduce a 0 into Row i (Column j)

(a) Subtract 4 from Row 1 & continue

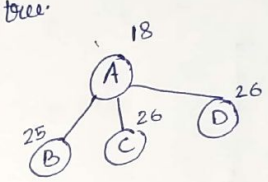
	A	B	C	D
A	∞	8	0	3
B	5	∞	7	0
C	0	4	∞	13
D	8	1	0	∞

(c) Subtract 1 from Column 2 (∵ No 0 present in Column 2)

	A	B	C	D
A	∞	7	0	3
B	5	∞	7	0
C	0	3	∞	13
D	8	0	0	∞

Total Reduced Cost = 4 + 6 + 5 + 2 + 1 = 18

The reduced cost is nothing but the root value of search tree.



To find the cost of B, make A's row & B's column all infinity & we can not move from B to A again so BA is ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	7	0
C	0	∞	∞	13
D	8	∞	0	∞

B's cost = C(A, B) + Reduction cost + New reduction cost

= 7 + 18 + 0

= 25

For C, make A's row & C's column ∞.

	A	B	C	D
A	∞	∞	∞	∞
B	5	∞	∞	0
C	∞	3	∞	13
D	8	0	∞	∞

	A	B	C	D
A	∞	∞	∞	∞
B	5	∞	∞	0
C	∞	0	∞	10
D	8	∞	0	∞

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	0
C	∞	0	∞	10
D	3	∞	0	∞

C's cost = C(A,C) + Red^n cost + New Red^n cost

= 0 + 18 + 8 = 26

For D, make A's row & D's column ∞

	A	B	C	D
A	∞	∞	∞	∞
B	5	∞	7	∞
C	0	3	∞	∞
D	∞	0	0	∞

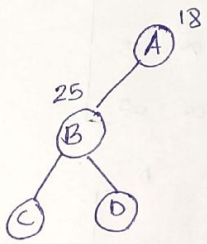
	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	2	∞
C	0	3	∞	∞
D	∞	0	0	∞

D's cost = C(A,D)

= 3 + 18 + 5 = 26

At this stage, we find that the path A to B is most promising as it has the min cost.

∴ We expand the node B.



For C's cost, B's row & C's column node it ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	7	0
C	0	∞	∞	13
D	8	∞	0	∞

A & B row = ∞

C, B = ∞

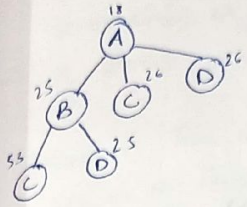
C, A = ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	0
C	∞	∞	∞	13
D	8	∞	∞	∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	∞	∞	13
D	8	∞	∞	∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	∞	∞	0
D	0	∞	∞	∞

C's cost = 7 + 25 + (13+8) = 53



D's cost (A → B → D)

Write again the B's cost matrix (of the previous level)

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	7	0
C	0	∞	∞	13
D	8	∞	0	∞

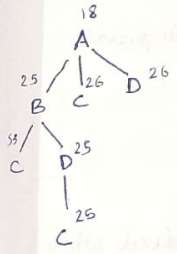
A → row → ∞
D → col → ∞
(D, B) → ∞
(D, A) → ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	0	∞	∞	∞
D	∞	∞	0	∞

D's cost = 25 + (0+0) + 0 = 25

cost upto B ↑ Reduction cost ↑ cost (B, D)

As at node D we find the smallest value compare to node C. Expand node D which value is 25.



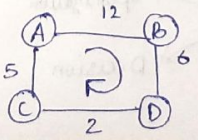
Find C's cost (A → B → D → C)

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	0	∞	∞	∞
D	∞	∞	0	∞

A, B, D → row → ∞
C → col → ∞
(C, D) → ∞
(C, B) → ∞
(C, A) → ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	∞	∞	∞
D	∞	∞	∞	∞

C's cost = 25 + (0, 0) + ∞ 0 = 25



P, NP, NP-Hard, NP → complete

$$2^n > n^k$$

Polynomial time -

The time complexity of an alg^n $O(n^k)$ where k is an integer. This alg^m can be solved in polynomial time.

For eg. Linear Search
Binary Search
Matrix Chain
BFS, DFS
Merge Sort etc

All the mentioned algo^m we can solve using polynomial time.

In general, for polynomial time alg^m, K can be expected to be less than n

Exponential Time -

The time complexity of any alg^n is $O(a^n)$. In general exponential is worse than polynomial time.

Problems-

In Computer Science, many problems are solved where the objective is to maximize or minimize some values, whereas in other problems we try to find whether there is a solution available or not.

Problems are of two types → Optimization
→ Decision

Optimization problems are those for which the objective is to maximize or minimize some value,
Eg. Shortest path in graph.

Decision - Their are many problems for which the answer is yes or no. For eg. whether a given graph can be coloured by 4 colours, checking the graph has a Hamiltonian cycle or not.

Classification:

→ Easy → P (Polynomial)
→ Medium → NP (Non-deterministic polynomial)
→ Hard → NPC (NP complete)
→ Hardest → NPH (NP hard)

Class P Problem

The set of decision problems are said to be in class P, if their exists a deterministic algo^m which is solvable in Polynomial time.

Eg. Linear Search, Matrix Chain Multiplication, BFS, DFS, Merge Sort etc