COMPLETE AD2 NOTES -

bit.ly/AD2ITER

ASSIGNMENT - 1

CSE4131: ALGORITHM DESIGN II

You are allowed to use only those concepts which are covered in the lecture classes, till date.

Submission deadline: 28^{th} May, 2022(Saturday). (Mandatory for Part A)

Submit the hard copies of the assignment, within the specified deadline.

NB: You can submit PART B on or before 11th June, 2022 (For details, contact to your respective AD2 class teacher).

PART A

1. Let us consider the product of three matrices namely A_1, A_2, and A_3 having dimensions p×q, q×r, and r×s respectively. If time taken to compute the product $(A_1A_2)A_3$ is more than that of the product $A_1(A_2A_3)$, then prove that 1/q+1/s>1/p+1/r.
2. Find an optimal parenthesization of a matrix chain multiplication whose sequence of dimension is {10,5,6,3,5}.
3. A 0/1 knapsack problem can be defined as: *given n items (cannot be splitted) of known weights* $\langle w_1, \cdots, w_n \rangle$ *and values/profits* $\langle v_1, \cdots, v_n \rangle$*, and a knapsack capacity W , then find the most valuable subset of items that fit into the knapsack assuming that* $w_i \in Z^+ (1 \le i \le n)$, $W \in Z^+$, *and* $v_i \in R^+ (1 \le i \le n)$.
   a. Discover the "overlapping sub problem(s)" property of the dynamic programming for the given knapsack problem. How many sub problems are to be solved to get the final answer when $\langle w_1, w_2, w_3, w_4 \rangle = \langle 7, 3, 4, 5 \rangle$, and $\langle v_1, v_2, v_3, v_4 \rangle = \langle \$42, \$12, \$40, \$25 \rangle$ with the knapsack capacity W = 10.
   b. Design a pseudocode that uses the concept of bottom-up dynamic programming to maximize the value/profit. Analyze the pseudocode to find its time and space complexity in the worst case scenario.
   c. Add a function **actual_knapsack_items ()** to the pseudocode to find out the subset of items selected to fit into the knapsack that maximizes the value. Why the time complexity to find the actual knapsack items is O (n + W)? Justify your answer.
4. Define the edit distance between two strings X and Y of length n and m, respectively, to be the number of edits that it takes to change X into Y. An edit consists of a character insertion, a character deletion, or a character replacement. For example, the strings **"algorithm"** and **"rhythm"** have edit distance 6. Design a $O(nm)-$ time algorithm for computing the edit distance between X and Y.
5. Describe the sub problem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?
6. In the dynamic programming below, assume the input consists of an integer S and a sequence $x_0, x_1, \ldots\ldots\ldots\ldots, x_{n-1}$ of integers 0 and S. Assume that each dynamic program uses sub problems (i, X) for $0 \le X \le S$ (just like Knapsack). Assume that the goal is to compute $DP(0, S)$, and that the base case is $DP(n, X) = 0$ for all X. Assume that the dynamic program is a memorized recursive algorithm, so that only needed

sub problems get computed. Circle the number next to the correct running time for each dynamic programming.

a. $$DP(i,X)=max\begin{cases} DP(i+1,X)+x_i, \\ DP(i+1,X-x_i)+x_i^2 if\ X\geq x_i \end{cases}$$

i. Exponential
ii. Polynomial
iii. Pseudo-polynomial
iv. Infinite

b. $$DP(i,X)=max\begin{cases} DP(i+1,S)+x_i, \\ DP(0,X-x_i)+x_i^2 if\ X\geq x_i \end{cases}$$

i. Exponential
ii. Polynomial
iii. Pseudo-polynomial
iv. Infinite

c. $$DP(i,X)=max\begin{cases} DP(i+1,0)+x_i, \\ DP(0,X-x_i)+x_i^2 if\ X\geq x_i \end{cases}$$

i. Exponential
ii. Polynomial
iii. Pseudo-polynomial
iv. Infinite

d. $$DP(i,X)=max\begin{cases} DP(i+1,X)+x_i, \\ DP(i+1,0)+x_i^2 if\ X\geq x_i \end{cases}$$

i. Exponential
ii. Polynomial
iii. Pseudo-polynomial
iv. Infinite

e. $$DP(i,X)=max\begin{cases} DP(i+1,X-\sum S)+(\sum S)^2, \\ for\ every \subset S\subseteq \{x_0,x_1,\ldots\ldots\ldots\ldots,x_{n-1}\} \end{cases}$$

i. Exponential
ii. Polynomial
iii. Pseudo-polynomial
iv. Infinite

7. In order to design a new joke for your standup comedy routine, You have collected n distinct measurements into an array $A[1,2,\ldots..,n]$, where $A[i]$ represents a measurements at time i. Your goal is to find the longest timespan $i\ldots.j, i.e.,$ maximize $j—i$ such that $A[i]<A[j]$. Note that the values in between A[i] and A[j] do not matter. As an example, consider the following array $A[1,2\ldots..,7]: A[1]=14, A[2]=6, A[3]=8, a[4]=1, A[5]=12, A[6]=7, A[7]=5.$ Your algorithm should return a span of 4 since $A[2]=6 \wedge A[6]=7.$ The next biggest span is A $[4]=1$ to $A[5]=5.$

a. Give an O(n) time algorithm to compute the minimums of the prefix $A[1,2,\ldots.,k]$ for each k, and store in $T[k]: T[k]=\min\limits_{1\leq i\leq k} A[i].$

b. Using the $T[i]$ computed above , give an $O(nlogn)$ time algorithm to maximize $j-i$ subject to $A[i]<A[j]$.

8. In dynamic programming, we derive a recurrence relation for the solution to one sub problem in terms of solutions to other sub problems. To turn this relation into a bottom up dynamic programming algorithm, we need an order to fill in the solution cells in a table, such that all needed sub problems are solved before solving a sub problem. For each of the following relations, give such a valid traversal order, or if no traversal order is possible for the given relation, briefly justify why.
   i. $A(i,j)=F(A(i,j-1),A(i-1,j-1),A(i-1,j+1))$
   ii. $A(i,j)=F(A(min(i,j)-1,min(i,j)-1),A(max(i,j)-1,max(i,j)-1))$
   iii. $A(i,j)=F(A(i-2,j-2),A(i+2,j+2))$

9. Fill in the following grid with the correct sub problem solutions for this sequence alignment problem with these weights: 0 for mutation, 1 for insertion or deletion, and 3 for a match (the goal is to maximize the sum of the weights). Here "ATC" is the starting sequence and "TCAG" is the ending sequence.

–	–	A	T	C
–	0			
T				
C				
A				
G				

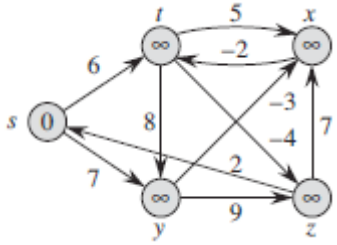
What is the optimal alignment?

10. Let $s_1=CCTACGTA$ and $s_2=ACTCGTAC$, what is the sequence alignment by considering the gap $\delta=-3$, match $M=+2$ and mismatch $MM=-2$, where the maximum formula is:

$$T(i,j)=\begin{cases} j\delta \text{ if } i=0, \\ i\delta \text{ if } j=0 \\ max(T(i-1,j-1)+\alpha_{ij}, T(i,j-1)+\delta, T(i-1,j)+\delta) \text{ other wise} \end{cases}$$

α_{ij} denotes match or mismatch value.

11. Run the Bellman-Ford algorithm on the directed graph of the following graph. Using vertex z as the source. In each pass, relax edges in the same order as in the figure. Now, change the weight of edge (z,x) to 4 and run the algorithm again, using s as the source.



[Change the given initial value according your requirement.]

PART B

A. Implement the following dynamic problems (free to use C/Java/Python)
    1. Matrix chain multiplication
    2. 0/1 Knapsack problem
    3. Bellman ford Algorithm
    4. Approximation string matching
    5. Weighted interval scheduling

ASSIGNMENT - 1

CSE 4131: Algorithm Design II

PART A

(1) Given A_1, A_2, A_3 are the matrices of order, $p \times q$, $q \times r$, $r \times s$ respectively.

Time taken to compute $(A_1 A_2) A_3$ > Time take to compute $A_1 (A_2 A_3)$

we need to prove $\frac{1}{q} + \frac{1}{s} > \frac{1}{p} + \frac{1}{r}$

$\Rightarrow$ $A_1 A_2$ takes pqr time to compute
$(A_1 A_2) A_3$ takes $pqr + prs$ time to compute,
similarly, $A_1 (A_2 A_3)$ takes $qrs + pqs$ time to compute.

Now, $pqr + prs > qrs + pqs$

$\Rightarrow pr(q+s) > qs(p+r)$

$\Rightarrow \frac{q+s}{qs} > \frac{p+r}{pr}$

$\Rightarrow \frac{q}{qs} + \frac{s}{qs} > \frac{p}{pr} + \frac{r}{pr}$

$\Rightarrow \frac{1}{q} + \frac{1}{s} > \frac{1}{p} + \frac{1}{r}$

Hence proved.

(2) Sequence of Dimension is $\{10, 5, 6, 3, 5\}$
Let A_1, A_2, A_3, A_4 are the matrices,

A_1	A_2	A_3	A_4
10×5	5×6	6×3	3×5

	1	2	3	4
1	0	300	$240^{k=1}$	$390^{k=3}$
2		0	90	$165^{k=2}$
3			0	90
4				0

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k \le j-1} \{ m[i,k] + m[k+1, j] + p_{i-1} p_k p_j \} & i < j \end{cases}$$

$m[1,3] \Rightarrow$ for $k=1 \Rightarrow 90+150 = 240$ (min)
for $k=2 \Rightarrow 300+180 = 480$

$m[2,4] \Rightarrow$ for $k=2 \Rightarrow 90+150 = 240$
for $k=3 \Rightarrow 90+75 = 165$ (min)

$m[1,4] \Rightarrow k=1 \Rightarrow 165+250$
$k=2 \Rightarrow 300+90+300$
$k=3 \Rightarrow 240+150 = 390$ (min)

∴ The optimal parenthesization is

$(A_1)(A_2\ A_3)(A_4)$

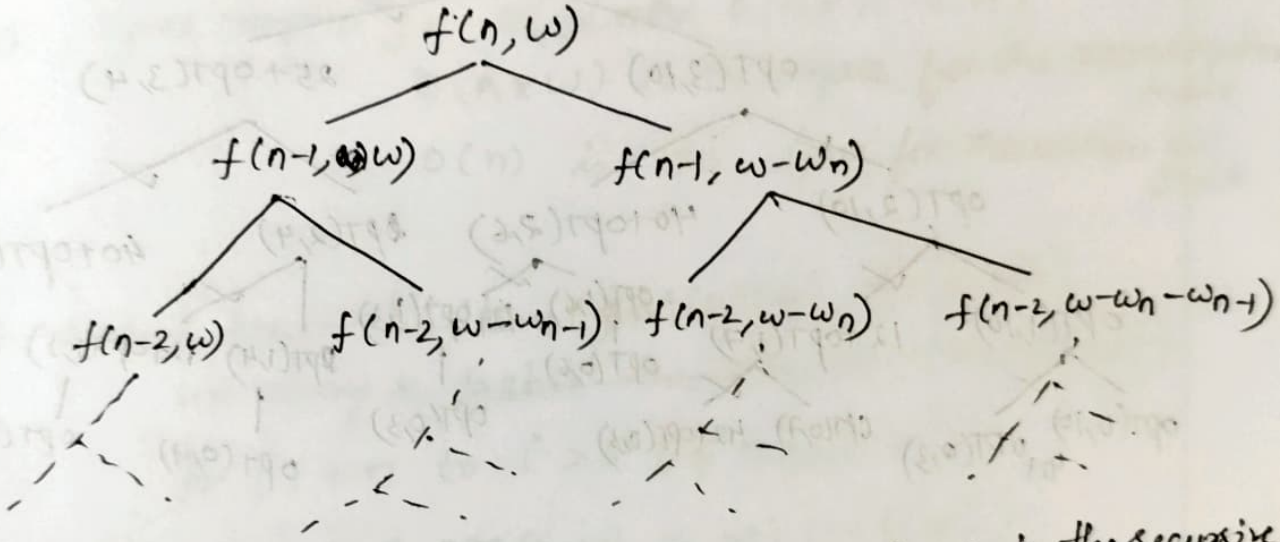
(3) (a) An overlapping subproblem is the same subproblem that will be visited again and again (i.e subproblem share subproblems).

In the given 0/1 knapsack problem,

the optimal solution is O

* if $n \notin O$, then $OPT(n, w) = OPT(n-1, w)$, since we can simply ignore item n.
* if $n \in O$, then $OPT(n, w) = w_n + OPT(n-1, w - w_n)$, since we now seek to use the remaining capacity of $w - w_n$ in an optimal way across items $1, 2, \ldots n-1$

Using the recursion approach, overlapping subproblem arises, due to which the time complexity of recursion approach becomes exponential.



In this recursion tree, $f(n, w)$ is the recursive call where n is the item we are currently looking at & w is the capacity.

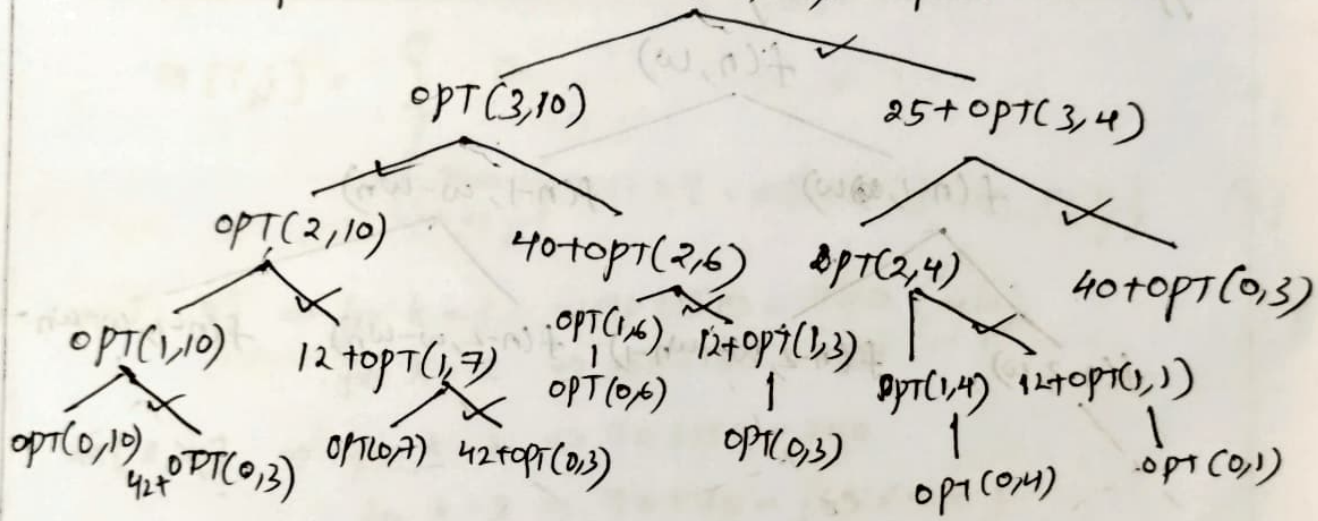
Given, $\langle w_1, w_2, w_3, w_4 \rangle = \langle 7, 3, 4, 5 \rangle$

$\langle v_1, v_2, v_3, v_4 \rangle = \langle \$42, \$12, \$40, \$25 \rangle$

$W = 10$.

v_i	w_i	i	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	7	1	0	0	0	0	0	0	0	42	42	42	42
12	3	2	0	0	0	12	12	12	12	42	42	42	54
40	4	3	0	0	0	12	40	40	40	52	52	52	54
25	5	4	0	0	0	12 / 40	40	40	40	52	52	65	65

The optimal solution $OPT(4, 10) = \$65$



(b) Pseudo code :-

Input : $n, W, w_1, \ldots, w_N, V_1, \ldots, V_N$

```
for w = 0 to W
    M[0, w] = 0
for i = 1 to n
    for w = 0 to W
        if w_i > W :
            M[i, w] = M[i-1, w]
        else :
            M[i, w] = max(M[i-1, w], v_i + M[i-1, w-w_i])
```

return M[n, W]

Time complexity, in worst case, the outer for loop runs for n times and the inner for loop runs for W times,

thus, $T(n) = O(nW)$.

Space complexity, An array M[i] ranging from $0 \le i < n$ is the array that stores the optimal solution.

$\therefore S(n) = O(n)$

(c) space complexity, $O(n * W + n)$

$O(n * w)$ is the space for the memoization table

$O(n)$ is the space for recursion call stack.

(c) actual_knapsack_items (M[n][W], opT(n, W))

~~int array k to store the result (k[n][W])~~

```
for (i = n to i > 0 && opT(n,w) > 0)
    if ( opT(n,w) = M[i-1][W])
        continue
    else
        print W[i-1]
        opT(n,w) = opT(n-w) - V[i-1]
        w = w - W[i-1]
```

(4) Algorithm to compute the edit distance between string X and string Y.

```
EditDistance (X, Y):
    For i=1, ..., m: A[i,0] = i * gap
    For j= 1, ..., n: A[0, j] = j x gap
```

```
For i=1,...,m:
    For j=1,...n:
        A[i,j] = min(
                    cost(a[i], b[j]) + A[i-1, j-1],
                    gap + A[i-1, j],
                    gap + A[i, j-1]
                )
    EndFor
EndFor
Return A[m,n]
```

Time complexity :- $O(m \times n)$

Space complexity :- $O(m \times n)$

(5) The vertices of the subproblem graph are the ordered pair v_{ij}, where $i \leq j$.

* if $i = j$, the vertex v_{ij} has no output edge
* if $i < j$, for each k, such that $i \leq k < j$, the subproblem graph contains edges (v_{ij}, v_{ik}) and $(v_{ij}, v_{k+1,j})$, and these edges indicate that to solve the subproblem of optimally parenthesizing the product $A_i \ldots A_j$ we need to solve subproblems of optimally parenthesizing the products $A_i \ldots A_k$ and $A_{k+1} \ldots A_j$.

Number of vertices is: $\sum_{i=1}^{n} \sum_{j=1}^{n} (1) = \sum_{i=1}^{n} n$

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

Number of edges is $\sum_{i=1}^{n} \sum_{j=1}^{n} (j-i) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} j - ni \right)$

$$= \sum_{i=1}^{n} \left(\frac{n(n+1)}{2} - ni \right)$$

$$= \frac{(n-1)\,n(n+1)}{6}$$

(6) (a) $DP(i, X) = \max \begin{cases} DP(i+1, X) + x_i \\ DP(i+1, X - x_i) + x_i^2, \text{ if } X \geq x_i \end{cases}$

~~i. Exponential~~
~~ii. Polynomial~~
~~iii.~~ Ans:- Pseudo Polynomial

(b) $DP(i, X) = \max \begin{cases} DP(i+1, S) + x_i, \\ DP(0, X - x_i) + x_i^2, \text{ if } X \geq x_i \end{cases}$

Ans:- Infinite.

(c) $DP(i, X) = \max \begin{cases} DP(i+1, 0) + x_i, \\ DP(0, X - x_i) + x_i^2 \text{ if } X \geq x_i \end{cases}$

Ans:- Pseudo polynomial or infinite

(d) $DP(i, X) = \max \begin{cases} DP(i+1, X) + x_i, \\ DP(i+1, 0) + x_i^2 \end{cases}$

Ans:- Polynomial

(e) $DP(i, X) = \max \begin{cases} DP(i+1, X - \sum S) + (\sum S)^2 \\ \text{for every subset } S \subseteq \{x_0, x_1, \ldots x_{n-1}\} \end{cases}$

Ans:- Exponential.

(7)(a) Input : A [1, 2, ..., k]
output : T[k]

MinimumOfPrefix(A)
for i = 0 to k
if i = 0
T[i] = A[i]
if i = 1
T[i] = ~~max~~ min (A[0], A[1])
T[i] = ~~max~~ min (T[i-1], A[i])

(b) Consider a single element A[j]. if we have T[1...j-1] we want to find an index i such that T[i] < A[j]
& T[i-1] ≥ A[j]
⇒ T[i] = A[i] is an unique element

Maximise j-i (A, T)
Here we do binary search function over T[1...j-1]
here low = 1, high = j-1

Binary_search (T, low, high)
mid = $\frac{low + high}{2}$
if T[mid] < A[j]
Binary_search (T, 1, mid-1)
if T[mid] > A[j]
Binary_search (T, mid+1, high)

Binary_search function is called until we have the right i. for this j

~~Binary search~~

This takes $O(\log n)$ time.

for every j it takes $O(n \log n)$

After we have the right i, j pairs, we pick the one that maximizes $j - i$.

8) (i) Solve $A(i, j)$

```
for (i from 0 to n)
    for (j from 0 to n)
```

(ii) Solve $A(k, k)$

```
for (k from 0 to n)
    solve rest in any order.
```

(iii) It cannot be solved since it is cyclic in nature.

9)

–	–	A	T	C
–	0 →	1 →	2 →	3
T	1	2	4 →	5
C	2 →	3	5	7
A	3	5 →	6	8
G	4	6	7	9

Sequence Alignment:-

S_1 : A T C – –

S_2 : – T C A G

2 match = $2 \times 3 = 6$

3 gap = $1 \times 3 = 3$

$\} = 9$ //

(10) $S_1 = CCTACGTA$

$S_2 = ACTCGTAC$

$\delta = -3$

Match $M = +2$

Mismatch $MM = -2$.

$$T(i,j) = \begin{cases} j\delta \text{ if } i = 0 \\ i\delta \text{ if } j = 0 \\ \max\{T(i-1,j-1) + \alpha_{ij},\ T(i,j-1) + \delta,\ T(i-1,j) + \delta\} \end{cases}$$

	—	C	C	T	A	C	G	T	A
—	0	-3	-6	-9	-12	-15	-18	-21	-24
A	-3	-2	-5	-8	-7	-10	-13	-16	-19
C	-6	-1	0	-3	-6	-5	-8	-11	-14
T	-9	-4	-3	+2	-1	-4	-7	-6	-9
C	-12	-7	-2	-1	0	+1	-2	-5	-8
G	-15	-10	-5	-4	-3	-2	+3	0	-3
T	-18	-13	-8	-3	-5	-5	0	5	2
A	-21	-16	-11	-6	1	-4	-3	2	7
C	-24	-19	-14	-9	-4	1	-2	-1	4

S_1: C C T A C G T A —

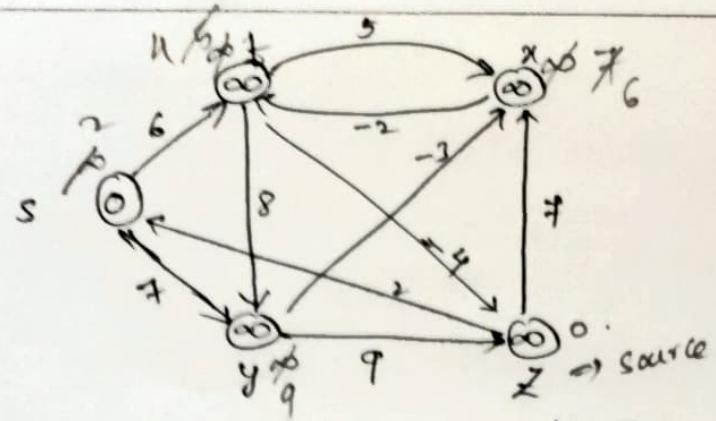
S_2: A C T — C G T A C

5 match

1 mismatch

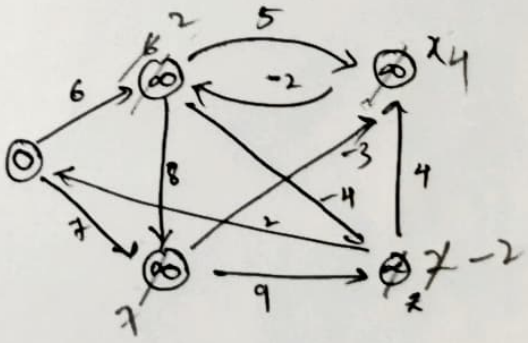
2 Gap.

(11)



if we change our source to z and ~~use the~~ relax the edges, the d values after successive iterations of relaxation are :-

s	t	x	y	z
∞	∞	∞	∞	0
2	∞	7	∞	0
2	5	7	9	0
2	5	6	9	0
2	4	6	9	0



s	t	x	y	z
0	∞	∞	∞	∞
0	6	∞	7	∞
0	6	4	7	2
0	2	4	7	2
0	2	4	7	-2

NAME - ABHISEK SAHOO
BRANCH - CSE
SECTION - Q
REG. NO - 2041019033
SUBJECT - Algorithm Design (PART B)

Q) Implementing the following dynamic problems.

1) Matrix Chain Multiplication

A)
```
Class MatrixChainMultiplication {
  static int MatrixChainOrder(int p[], int i, int j) {
    if (i==j)
      return 0;
    int min= Integer.MAX_VALUE;
    for (int k=i; k<j; k++) {
      int count = MatrixChainOrder(p,i,k)+MatrixChainOrder(p,k+1,j)
                  + p[i-1] * p[k] * p[j];

      if (count <min)
        min = count;
    }
    return min;
  }

  public static void main(String args[]) {
    int arr[] = new int [] {1, 2, 3, 4, 3};
    int n = arr.length;
    System.out.println("Minimum number of multiplication is "
                       + MatrixChainOrder(arr, 1, n-1));
  }
}
```

Output :-

Minimum number of multiplication is 30

2) 0/1 Knapsack problem

A)
```
Class Knapsack {
    Static int max (int a, int b) {
    return (a>b)? a: b;
    }
    Static int Knapsack (int W, int wt[], int val[], int n) {
     if (n==0 || W==0)
      return 0;
      if (wt[n-1]>W)
       return Knapsack (W, wt, val, n-1);
    else
      return max (val[n-1] + Knapsack (W-wt[n-1], wt,
                                       val, n-1),
                                  Knapsack (W, wt, val, n-1));
    }
    Public Static void main (String args[]) {
    int val[] = new int[] {60, 100, 120};
    int wt[] = new int[] {10, 20, 30};
    int W = 50;
    int n = val.length;
    System.out.println (Knapsack (W, wt, val, n));
}}
```

Output
220

3) Bellman Ford Algorithm

A)
```
Class Bellman {
    Static void BellmanFord (int graph[][], int V, int E, int src) {
    int [] dis = new int [V]
    for (int i=0; i<V; i++)
      dis[i] = Integer.MAX_VALUE;
      dis[src] = 0;
```

```
for (int i=0; i<V-1; i++) {
  for (int j=0; j<E; j++) {
    if (dis[graph[j][0]] != Integer.MAX_VALUE && dis[graph[j][0]]
        + graph[j][2] < dis[graph[j][1]])
      dis[graph[j][1]] = dis[graph[j][0]] + graph[j][2];
  }}
    for (int i=0; i<E; i++) {
      int x = graph[i][0];
      int y = graph[i][1];
      int weight = graph[i][2];
      if (dis[x] != Integer.MAX_VALUE && dis[x]+weight<dis[y])
    System.out.println("Graph contains negative :"+"weight cycle");
  }
  System.out.println("Vertex Distance from source");
  for (int i=0; i<V; i++)
      System.out.println(i+"\t\t"+dis[i]);
  }
    public static void main (String[] args) {
      int V=5;
      int E=8;
    int graph[][] = {{0,1,-1},{0,2,4},{1,2,3},{1,3,2},
                     {1,4,2},{3,2,5},{3,1,1},{4,3,-3}};

  BellmanFord(graph,V,E,0);
}}
```

O/P

Vertex Distance from Source

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

4) Approximate String Matching

A)

```
import java.util.*;
public class ASM {
 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  int sum=5;
  String source, pattern;
  System.out.println("Enter sequence:");
   pattern = sc.nextLine();
  System.out.println("Enter source:");
  source = sc.nextline();
 if (pattern.length()==source.length()&& pattern.equals(source))
 { System.out.println("Sequence = Source");
 }
  char[] x = pattern.toCharArray(), y=source.toCharArray();
  int i,j,s,d,last,m=x.length, n=y.length;
  int[] b= new int[65536];
for (i=0; i<b.length; i++) {
}    b[i]=0;
  s=1;
for(i=m-1; i>=0; i--) {
     b[x[i]]|=s;
      s<<=1;
}
  j=0;
while(j<=n-m) {
   i=m-1;
   last=m;
   d=-0;
 while(i>=0 && d!=0) {
    d&=b[y[j+i]];
    i--;
  if(d!=0){
        if(i>=0) {
          last=i+1;
  }else{
System.out.println("Sequence is Source starting at position:");
System.out.println(j);
System.out.println("Sequence");
```

```
        System.out.println(pattern);
        System.out.println("Source:");
        System.out.println(source.substring(j, j+m));
    }
    d <<= 1;
    j += last;
  }
}
```

5) Weighted Interval Scheduling

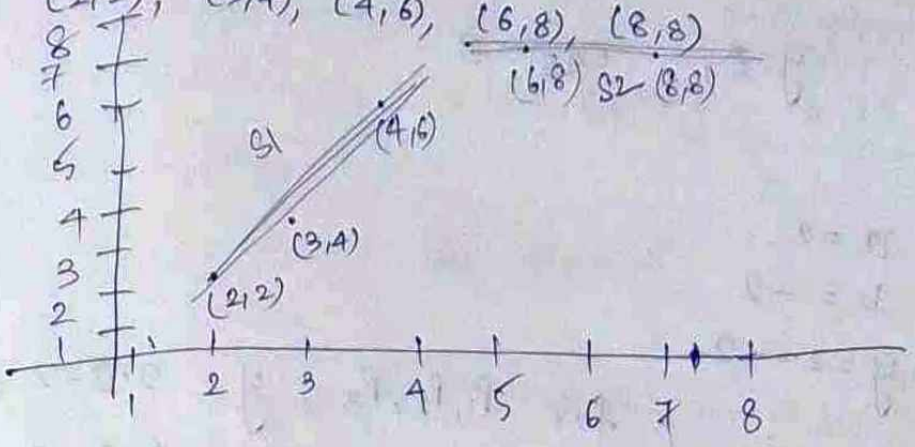
```
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

Class Interval {
    int start, finish, profit;
    Interval(int start, int finish, int profit) {
        this.start = start;
        this.finish = finish;
        this.profit = profit;
    }

Class Main {
    public static int findMaxProfit(List<Job> jobs) {
    Collections.sort(jobs, Comparator.comparingInt(x -> x.start));
    int n = jobs.size();
    if (n == 0)
        return 0
    }
    int [] maxProfit = new int[n];
    for (int i=0; i<n; i++)
    {
    maxProfit[i] = 0;
    for (int j=0; j<i; j++)
    {
```

```
if (Interval.get(j).finish <= Interval.get(i).start &&
        maxProfit[i] < maxProfit[j]) {
    maxProfit[i] = maxProfit[j];
}
}
maxProfit[i] += Interval.get(i).profit;
}
return Arrays.stream(maxProfit).max().getAsInt();
}
public static void main(String[] args){
List<Interval> Interval = Arrays.asList(
                new Interval(0, 6, 60),
                new Interval(5, 9, 50),
                new Interval(1, 4, 30),
                new Interval(5, 7, 30),
                new Interval(3, 5, 10),
                new Interval(7, 8, 10)
        );
    System.out.println("The maximum profit is "+findMaxProfit
                        (Interval));
}
}
```

O/P

The maximum profit is 80

Q (2,2), (3,4), (4,6), (6,8), (8,8)



First we have to find e_{ij} for all pairs $i \le j$

e_{12} $a = 2$, $b = -2$

$\Rightarrow y = 2x - 2$

$error = (2-2)^2 + (4-4)^2 = 0$

actual y ↑ $(2-2)^2$; actual y ↑ $(4-4)^2$

$y = 2 \cdot 2 - 2 = 2$

$y = 2 \cdot 3 - 2 = 4$

$\Downarrow$

$\boxed{e_{12} = 0}$

using formulae $a = \dfrac{n \sum_i x_i y_i - \sum_i x_i \sum_i y_i}{n \sum_i x_i^2 - \left(\sum_i x_i\right)^2}$

$= \dfrac{2 \cdot 16 - 5 \cdot 6}{2 \cdot 13 - 25} = 2$

$b = \dfrac{\sum_i y_i - a \sum_i x_i}{n}$

$= \dfrac{6 - 2 \cdot 5}{2} = -2$

e_{23} $y = 2x - 2$, $\therefore e_{23} = 0$

e_{34} $y = x + 2$, $e_{34} = 0$

Q15 $y = 8, \quad e_{45} = 0$

Q13 $a = 2$

$b = -2$

$y = 2x - 2$

$e_{13} = 0$ [for P_1, P_2, P_3, $y = 2 \cdot 2 - 2 = 2 = a$

$y = 2 \cdot 3 - 2 = 4 = a$

$y = 2 \cdot 4 - 2 = 6 = a$

Q14 $a = 1.48 \ (52/35)$

$b = -0.57$

$\Rightarrow y = 1.48x - 0.57$

For P1, P2, P3, P4 :-

$y = 1.48 \times 2 - 0.57 = 2.39$; act. $y = 2$

$\Rightarrow$ difference $= 0.39$

$y = 1.48 \times 3 - 0.57 = 3.87$; act. $y = 4$

$\Rightarrow$ difference $= 0.13$

$y = 1.48 \times 4 - 0.57 = 5.35$; act $y = 6$

$\Rightarrow$ difference $= 0.65$

$y = 1.48 \times 6 - 0.57 = 8.31$; act $y = 8$

$\Rightarrow$ difference $= 0.31$

$e_{14} = \sum$ difference^2

$= 0.39^2 + 0.13^2 + 0.65^2 + 0.31^2$

$= 0.6876$

$a = 1, \; b = 1, \; y = x + 1$

e_{15}

$e_{15} = \sum (3-2)^2 + (4-4)^2 + (6-5)^2 + (8-7)^2 + (9-8)^2$

$= 4.$

e_{23} $\quad a = 2$, $b = -2$ $\Rightarrow y = 2x - 2$

$P_2 P_3$

e_{24} $\quad a = 1.28$, $b = 0.45$ $\Rightarrow \boxed{y = 1.28x + 0.45}$

for P_2, P_3, P_4

$e_{24} = (1.28 \times 3 + 0.45 - 4)^2 + (1.28 \times 4 + 0.45 - 6)^2$

$+ (1.28 \times 6 + 0.45 - 8)^2$

$\Rightarrow \boxed{e_{24} = 0.286}$

$e_{35} = 0.667$, $y = 0.5x + 4.33$

$e_{25} = 2.61$, $y = 0.78x + 2.41$

$e_{15} = 4$, $y = x + 1$

Now we have to calculate matrix M by using

$$M[j] = OPT(j) = \min_{1 \le i \le j} \{ e_{ij} + c + OPT(i-1) \}$$

For $j=1$ $\quad OPT(1) = \min_{1 \le i \le 1} \{ e_{i,1} + c + OPT(i-1) \}$

i has 1 value = 1

$= e_{1,1} + c + OPT(0)$

$\Rightarrow \boxed{M[1] = c}$ and $\boxed{M[0] = 0}$ from first line of subroutine

For $j=2$

$$OPT(2) = \min_{1 \le i \le 2} \{ e_{i,2} + c + OPT(i-1) \}$$

$\Rightarrow$ i has values 1 & 2 now

$$\Rightarrow \min \{ \underbrace{e_{1,2} + c + OPT(0)}_{i=1}, \underbrace{e_{2,2} + c + OPT(1)}_{i=2} \}$$

$$\Rightarrow \min \{ c, \; c+c \}$$

$$\Rightarrow \boxed{OPT(2) = c}$$

For $j=3$

$$OPT(3) = \min_{1 \le i \le 3} \{ e_{i,3} + c + OPT(i-1) \}$$

$\Rightarrow$ i has 3 values now : 1, 2, 3

$$\Rightarrow OPT(3) = \min \{ \cancel{e_{1,3}} + c + \cancel{OPT(0)}, \; \cancel{e_{2,3}} + c + \cancel{OPT(1)}, \; \cancel{e_{3,3}} + c + \cancel{OPT(2)} \}$$

$\Rightarrow \boxed{OPT(3) = c}$ for $i=1$.

For $j=4$ $OPT(4) = \min\{ c+0.6876,\ 2c+0.286,\ 2c,\ 2c \}$

Now, if $c=1$, $OPT(4) = \min\{ 1.6876,\ 2.286,\ 2,\ 2 \}$

$= 1.6876 \ (i=1)$

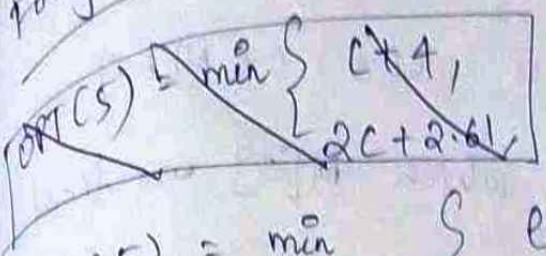
But if $c=0.15$, $OPT(4) = \min\{ 0.8376,\ 0.586,\ 0.3,\ 0.3 \}$

$= 0.3 \ (i=3 \text{ or } 4)$

(Choosing c determines the value of i for which $OPT(4)$ is finalized)

Lets choose $c=1$ for now $\Rightarrow$ $M[4] = c+0.6876$

For $j=5$



$$OPT(5) = \min_{1 \le i \le 5} \{ e_{i,5} + c + OPT(i-1) \}$$

$$= \min \{ e_{1,5} + c + OPT(0),\ e_{2,5} + c + OPT(1),$$
$$e_{3,5} + c + OPT(2),\ e_{4,5} + c + OPT(3),$$
$$e_{5,5} + c + OPT(4) \}$$

$$= \min \{ c+4,\ 2c+2.61,\ 2c+0.667,\ 2c,\ 2c+0.6876 \}$$

Substituting $c=1$,

$$OPT(5) = \min \{ 5,\ 4.61,\ 2.667,\ 2,\ 2.6876 \}$$

$= 2$ $(i=4)$.

Find Segments (5)

① Find i which minimizes $e_{i,5} + c + M[i-1]$

⇒ value of i for which $M[5]$ was finalized

⇒ $i=4$.

o/p $\{P_4, P_5\}$ with Find Segments (3)

② findSegments (3)

Find i which minimizes $e_{i,3} + C + M[i-1]$

⇒ value of i for which M[3] was first

⇒ i = 1.

o/p $\{P_1, P_2, P_3\}$ with findSegments (0)

③ findSegments (0)

o/p nothing

Final o/p

$\{P_1, P_2, P_3\}$, $\{P_4, P_5\}$

2 segments

ASSIGNMENT-02

CSE4131: ALGORITHM DESIGN II

You are allowed to use only those concepts which are covered in the lecture classes, till date.

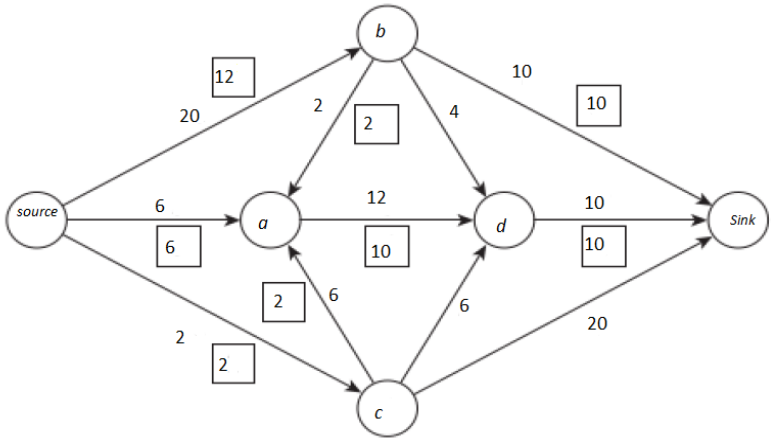
Submission deadline: 02^{nd} July, 2022(Saturday).

Submit the hard copies of the assignment, within the specified deadline.

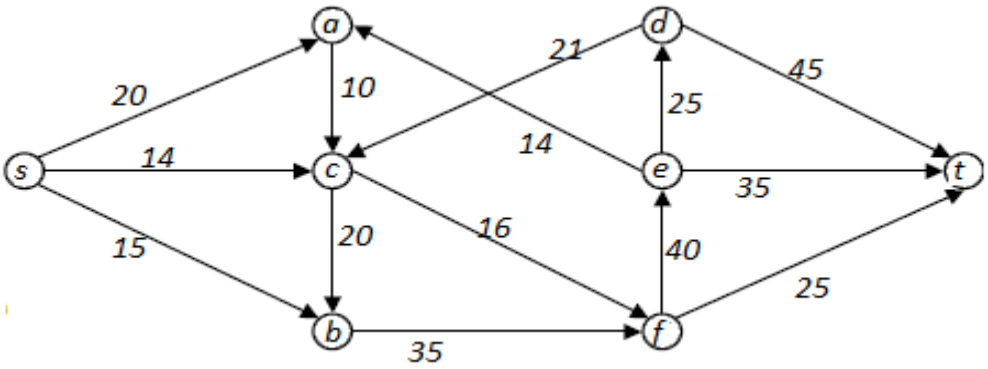
NB: Submit PART C after 02^{nd} July, 2022 (Saturday) (Contact to concern faculty).

PARTA: FLOW NETWORK AND BIPARTITE MATCHING

1. We have the below figure of a flow network $G\ (V, E)$ with source and sink node. The weight of each edge called capacity of the edge mentioned as a label next to the edge, and the values in small box means the amount of flow supplied on each edge. (No box means no flow being sent on that edge.)
   a. What is the value of this flow? Is this a maximum (s, t) flow in this graph?
   b. Find a minimum $s - t$ cut in the flow network and also mention its capacity.



2. **Problem Statement:** Suppose there is a weighted graph $G(V, E)$ with a source vertex and destination vertex where $|V| = n$ and$|E| = m$. You have to find the maximum number of paths from the source vertex to destination vertex in which they do not share any common edges.
   a. How many such paths presents in the following graph?

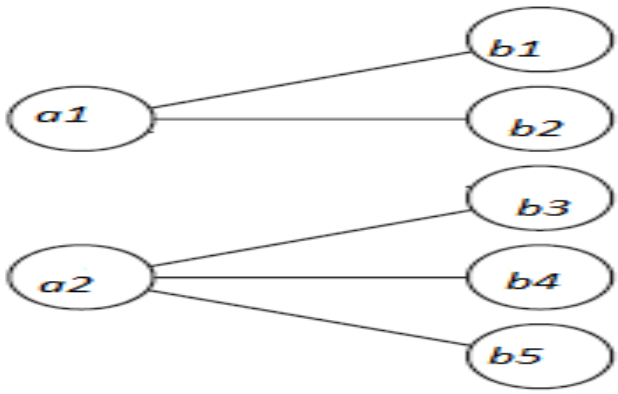


   b. Reduce the above problem into max flow network problem. (Hints: Values(weight) are not important, can be modify)
   c. Properly explain that your reduction problem is the correct to solve the above problem statement.

3. Let *G (V, E)* be a bipartite graph with vertex partition $V = L \cup R$, and let G' (V', E') be its corresponding flow network. If M is a matching in$G$, then prove that
   a. $|f| = |M| = Minimum\ Cut$, where f is an integer-valued flow inG'.
   b. Also explain with an appropriate example.

4. Suppose we design a network in the following way:
   Let $G(V, E, c)$ be a flow network with a source vertex s and a sink vertex t. The flow f of G be a real valued function $f: V \times V \to \mathbb{Z}^+$ and the capacity function $c: V \times V \to \mathbb{Z}^+$ such that$0 \leq f(u, v) \leq c(u, v)$. Also $G'(V, E, c')$ is a modified network of $G(V, E, c)$ such that $c(u, v) = c'(u, v)$ except $c'(u, v) = t$, here t is the non-negative integer values(the meaning of this t is: some weight of the edge (u, v) of G' are different from G). This should run in $O(l * (V + E))$ where $|c(u, v) - t| = l$. This algorithm will act differently because of the l value whether $l > c(u, v)$ or $l < c(u, v)$.
   a. State True or False and give the proper explanation of your answer of the following statement:
      i. If we increase the weight of an edge (u, v) by 10 units then the max flow also will be increased at most 10 units.
      ii. If we decrease the weight of an edge (u, v) by 10 units then the max flow also will be decreased at most 10 units.
   b. Suppose $l > c(u, v)$. Describe your algorithm in detail and analyses its time complexity in terms of V, E, l.
   c. If $l = c(u, v)$, then what will be the answer of the above designed problem. (Hints: need to check for the condition.)

5. Let $G = (V, E)$ where $V = S \cup T$, be a complete bipartite graph such that $|S| = |T| = m$. Prove that G has $m!$ distinct maximum matchings. (Complete bipartite graph is a bipartite graph where each vertex of left set to connected to every vertex of right set).
6. Consider the following problem. Suppose $G(V, E)$ is a given bipartite graph and M is it's matching $i.e$ any node x is covered by M if x is one end vertex of an edge in M. For a given positive integer k, we want to consider that if there is a matching M' in G such that $|M'| = |M| + k$,that means every vertex is covered by M is also covered by M' (M' is the extension of M).
   a. Give a polynomial time algorithm that there is an extension of M or if not, report there is no such extension.(Here M is given as an initial input)
      Let's take an example to understand the above problem:



      Suppose the matching M consists only one edge (a_1, b_2) and to solve the above problem $k = 1$. According to the condition $M' = \{(a_1, b_2), (a_2, b_4)\}$. Report: M' exists.
   b. Consider k_1 is the size of the matching M' so that every vertex is covered by M is also covered by M'. Consider k_2 is the size of the matching M'' so that every vertex is covered by M is also covered by M'', where M is also a matching.
      Then prove that k_1=k_2.

PART B: BACKTRACKING

NB: For this section, you have to take main skeleton of the backtracking implementation (Chapter 7; Page-232, **The Algorithm Design Manual by Steven S. Skiena, Springer Publication**). The three main functions $is\_a\_solution, construct\_candidates$ and $process\_solution$ are used in the skeleton.

1. Design the $construct\_candidates$ function to find the subset of a given set S whose sum is given K. Example: $S = \{30,35,25,31,29\}$ Output: $\{\{35,25\},\{31,29\}\}$.
2. A derangement is a permutation p of $\{1, 2, 3, \ldots\ldots, n\}$ such that no items in its proper position i.e $p_i \neq i$ for all $1 \leq i \leq n$. Design the $construct\_candidates$ with pruning that constructs all the derangements of n items.
3. Design the $construct\_candidates$ for printing all K −element subsets of n objects.
4. N −Queen Problem: Place N queen in a $N \times N$ chess board such that no two queen place in same row, same column and diagonally.
    a. Draw the state space tree for $N = 4$ using the above constraints. Assign each queen in different row. How many solutions are there? Also find the total number of backtrack are required to get the solution.
    b. How many nodes are in the state space tree of the N −queen problem with the constraint that no two queens place only same row and same column?
    c. How many leaf nodes are in the state space tree of the N −queen problem without any bound constraint (Assign each queen in different row)?

PART C: IMPLEMENTATION (Using NB of part B)

1. Implement PART B.Q1 in any language. Also show the output with an example.
2. Implement PART B.Q2 in any language. Also show the output with an example.
3. Implement PART B.Q3 in any language. Also show the output with an example.
4. Implement the backtracking problem **finding all subset**. Also show the output with an example.
5. Implement the backtracking problem **finding all permutation**. Also show the output with an example.
6. Implement the backtracking problem **finding all paths in a graph**. Also show the output with an example.

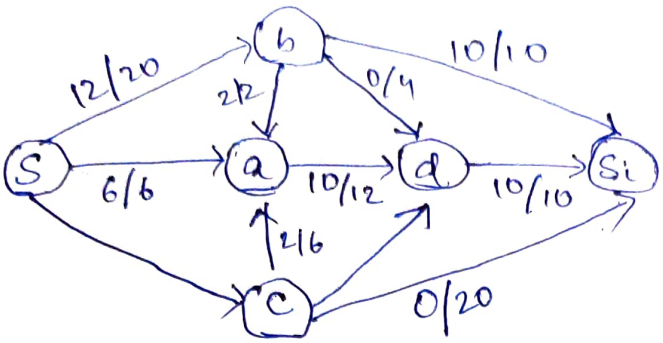
NAME → KAUSHIK LAKHANI
REG No → 2041012002
SECTION → CSIT - D
DATE → 02.07.2022

ASSIGNMENT-2

WHATSAPP GROUP - https://bit.ly/ITER2024

PART - A

(1)



S → source
Si → sink (destination)

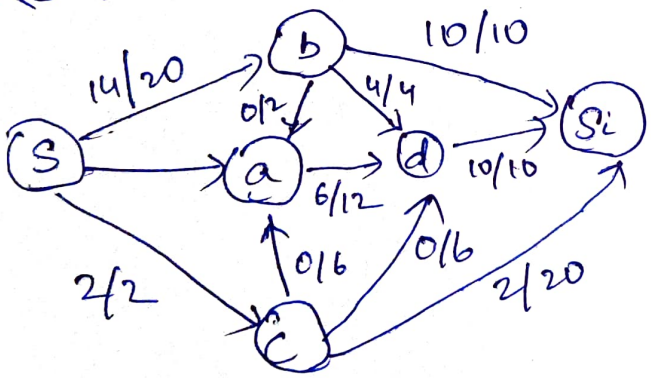
(a)

The value of flow = Total flow reaching the sink

S - b - Si - 10 = 20
S - b - a - d - Si - 2
S - a - d - Si - 6 ⇒ 20
S - c - d - Si - 2

The above flow may / may not be max flow so to check for the max flow, we have to use ford fulkerson algorithm

(b) Residual Graph:-



The maxm flow :-

S - b - Si - 10
S - a - d - Si - 6
S - b - d - Si - 4
S - c - Si - 2
22

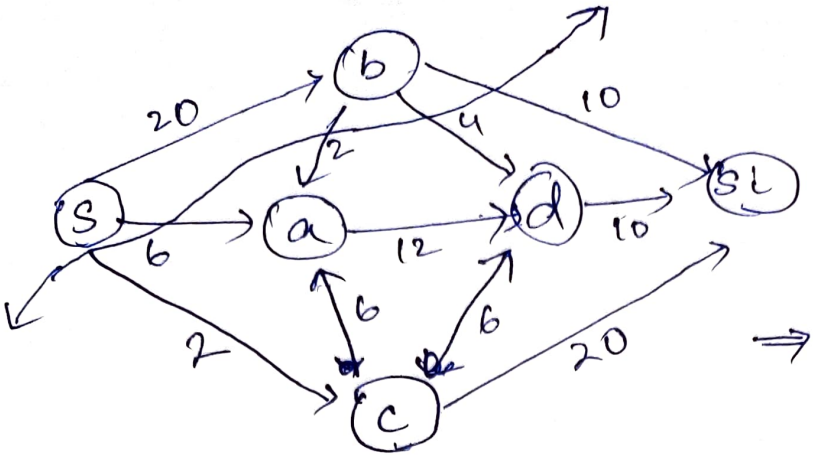
So, maxm flow = 22

(b)

Cut:

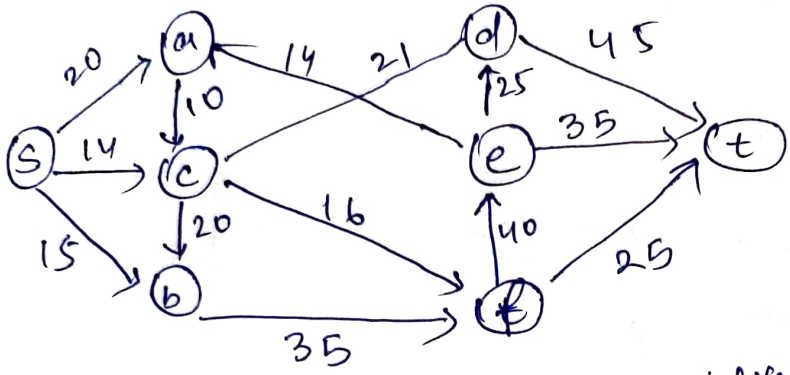
S = {S, b, d, a} T = {c, t}

Minm cut = c(S, c) + c(b, t) + c(d, t)
= 2 + 10 + 10
= 22



⇒ Minimum cut

(2)



There can be various possibilities (paths)

case 1 → s-a-c-f-t and s-b-f-e-t

case 2 → s-a-c-b-f-t and s-e-f-e-t

case 3 → s-b-f-e-d-t and s-c-b-f-t.

case 4 → s-a-c-b-f-e-d-t

For maxm flow

Augmented Path	Bottle Neck capacity
s-b-f-t	15
s-e-f-t	10
s-c-f-e-t	4
s-a-c-b-f-e-t	10

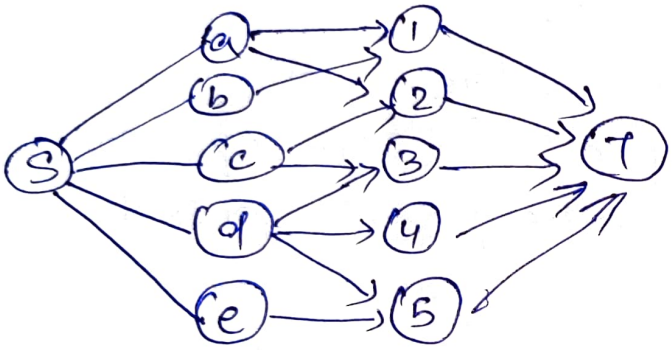
Total flow = 15 + 10 + 10 + 4 = 39

∴ Maxm flow through the graph is '39'.

For reduction, we implement the augmenting path using DFS. We add the bottle neck capacity after finding out the augmented path, if the flow value exceeds the path capacity then we subtract bottle-neck capacity from it. At last all the paths got blocked after getting flow value (when it reaches capacity) to give us the maxm flow. Therefore it is the correct way of reduction.

(3) (a) Let's solve by taking an example,

V = LUR → Here, m = 1.



The maxm flow = 5

S - a - 2 - t → 1

S - b - 1 - t → 1

S - c - 3 - t → 1

S - d - 4 - t → 1

S - e - 5 - t → 1

= 5

If M is a matching in G, then there is an integer valued flow F in G' with value |F| = |m|. conversely, if F is an integer-valued flow in G', then there is a matching M in G with cardinality |m| = |F|.

This is because, the edges incident to S & t ensures :-

→ each node in the L has in-degree 1.

→ each node in the R has out-degree 1.

→ so, each node in the bipartite graph can be involved once in the flow.

(4) Flow network - $G(V, E, C)$

Residual flow network - $G'(V, E, C)$

$|f(u, v) - t)| = l \rightarrow$ Residual flow

(a) (i) TRUE. This is because if the weight of edge is increased by 10 units, then it will be dependent on the flow of network on the previous edge. If there would be surplus flow in its previous edge, then the max^m flow also will be increased by atmost 10 units. For instance, taking $G(V, E, C)$, if $c_1 > c_3$ & we increase c_3 by 10 units, then flow can be increased from c_3 to $c_3 + 10 < c_1$.

(ii) True, → If the weight of the edge is decreased by 10 units, then depending upon the flow of network on the subsequent edges, it might be decreased by atmost 10 units.

(b) for each edge $e \in$ Augmenting Path (P)

```
IF (e ∈ E)
    l ← l + δ
ELSE
    l ← l - δ
RETURN
```

So, when $l > C(u, v)$, we have to apply $l \leftarrow l - \delta$.

where δ - bottleneck capacity.

Time complexity of this is $O(E)$.

(c) If $l = c(u, v)$

This means $t = 0$, which implies the flow capacity of edges connects to sink t will also be zero. Hence, there will be no flow/zero flow from source to sink whatsoever, be the capacity of the other edges.

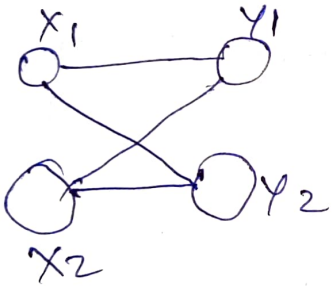
(5) Let $x = \{x_1, x_2, x_3, \ldots x_n\}$ &
$y = \{y_1, y_2, y_3 \ldots y_n\}$

x_1 y_1 So, x has m different choices
x_2 y_2 $(y_1, \ldots y_m)$.
x_3 y_3 After the matching of x_1 is completed, x_2 has m-1 different choices ... &
x_n y_n So, x_m will have 1 choice.

⇒ x_1 will have m no. of choices
x_2 will have m-1 no. of choices
x_m will have 1 no. of choices.

So, total matching
$= m \times (m-1) \times (m-2) \times \ldots 1$
$= m!$

⇒ Total no. of matching = m!

ex 

Matching = 2! = 2
$= \{(x_1, y_1), (x_2, y_2)\}$
$\{(x_1, y_2), (x_2, y_1)\}$
} → 2 matching

(6) Algorithm

```
boolean checking M'C int adj[][], int m){
int match = new int [adj[i].length ? 1];
while ((u,v) ∈ E & there is path from u to v){
  Cuv = 1;
} int min_capacity;
while (# augmenting path P from s to in Gf of FN){
min capacity = 1;
for (# edge (u,v) in path P){
 if((u,v) ∈ E){
```

```
f(u,v) = f(u,v) + 1;
match[u] = v; }
else
f(v,u) = f(v,u) - 1;
}}
if (match.size() > m) {
return true;
} else
return false;
}
```

Here, match stores the value of maxm matching vertices. It is indicate denotes the vertex at X (u ∈ X) & value denote V (v ∈ Y).

(b) $|M'| = k_1$ & $|M''| = k_2$ → given

$k_1 = |m| + k$

$\pm k_2 = \pm |m| \pm k$

$k_1 - k_2 = 0$ ⇒ $k_1 = k_2$ → Proved

PART → B

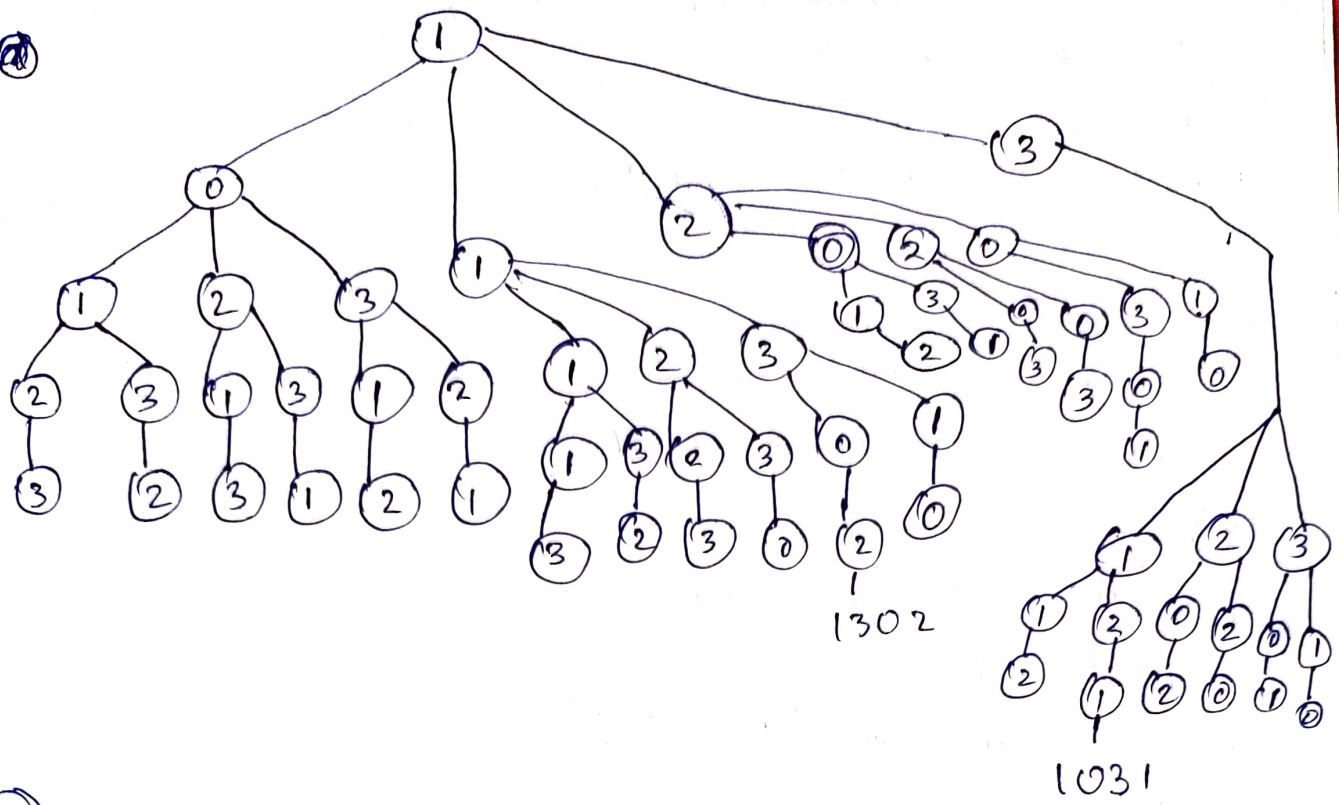
(1) S = {30, 25, 35, 31, 29}

```
construct_candidates (A[], k, i, s, List<Integer> nm); {
if (i == A.length) {
if (s == k) {
print (nm); }
return;
} nm.add (A[i]);
s += A[i];
construct_candidate (A, k, i+1, s, nm);
s -= nm.get (nm.size() - 1);
nm.remove (nm.size() - 1);
construct_candidate ( A, k, i+1, s, m);
}
```

(2)
```
construct_candidates (A[], i, j) {
if (i == A.length) {
int f;
for(f=0; f<A.length; f++)
if (A[f] == f+1)
break; }
if (f == A.length) {
print the element present in A
}}
for (int p=i; p<=j; p++) {
swap(A, i, p);
construct_candidates (A, i+1, j);
swap (A, i, p);
}} swap (A[], i, j) {
int d = A[i];
A[i] = A[j];
A[j] = d;
}}
```

(3)
```
construct_candidate (A[], i, kk, List<Integer> nm) {
if (i == A.length) {
if (kk == nm.size())
print(kk);
return;
} nm.add(A[i]);
construct_candidate (A, i+1, kk, nm);
nm.remove(nm.size()-1);
construct_candidate (A, i+1, kk, nm);
}
```

(4) (a)



(a) There are 2 solutions.

(b) There are 24 nodes in the spaces and of N-Queen problem width the constraints that no two queen place only same row and same column.

(c) There are 2 leaf nodes, which are in the state space tree of the N-Queen problem without any bound constraint.

PART - C

```
1) import java.util.*;
   Public class A
{
Public Static void main (String [] args)
{
   int a[] = { 30, 25, 35, 31, 29 };
    int K = 60;
   List<Integer> nm = new ArrayList <> ();
   task (a, 0, nm, 0, K);
}
Static void task (int a[], int i, List <Integer> nm,
                  int s, int K)
{
   if (i == a.length)
   {
      if (s == K)
      {
         System.out.println (nm);
      }
      return;
   }
   nm.add (a[i]);
   s += a[i];
   task (a, i+1, nm, s, K);
   s -= nm.get (nm.size () - 1);
   nm.remove (nm.size () - 1);
   task (a, i+1, nm, s, K);
}
}
```

30	25	35	31	29

K = 60

[25, 35]
[31, 29]

Output.

2)
```
import java.util.*;
Public class A
{
  public static void main (String [] args)
  {
  List <Integer> nm = new ArrayList <>();
    nm.add (1);
    nm.add (2);
    nm.add (3);
    nm.add (4);
   task (nm, 0, nm.size () - 1);
}
static void task (List <Integer> nm, int i, int j)
{
  if (i == nm.size ())
    { int f;
   for (f=0; f<nm.size (); f++)
     {
      if (nm.get (f) == f+1)
        {
          break;
        }
     }
   if(f == nm.size ())
    System.out.println (nm);
  }
 for (int p=i ; p<=j ; p++)
   {
    swap (nm, i, p);
    task (nm, i+1, j);
    swap (nm, i, p);
   }
 }
static void swap (List <Integer> nm, int i, int j)
  {
    int d = nm.get (i);
```

```
        nm.add(i, nm.get(j));
        nm.remove(i+1);
        nm.add(j, d);
        nm.remove(j+1);
    }
}
```

(5)
```
import java.util.*;
class A
{
public static void main(String[] args)
{
List<Integer> nm = new ArrayList<>();
List<Integer> kk = new ArrayList<>();
  nm.add(1);
  nm.add(2);
  nm.add(3);
  nm.add(4);
 task(nm, 0, 3, kk);
}
static void task(List<Integer> nm, int i, int k,
                 List<Integer> kk)
{
 if(i == nm.size())
   {
    if(k == kk.size())
      System.out.println(kk);
    return;
   }
  kk.add(nm.get(i));
  task(nm, i+1, k, kk);
  kk.remove(kk.size()-1);
  task(nm, i+1, k, kk);
}
```

1)
```
import java.util.*;
Public class A
{
 public static void main (String [] args)
 {
  Scanner sc = new Scanner (System.in);
  int n = sc.nextInt();
 List <Integer> nm = new ArrayList <>();
 for (int i=0; i<n ; i++)
    {nm.add (sc.nextInt());
    }
 List <Integer> kk = new ArrayList <>();
  task (nm, 0, kk);
 }
static void task (List <Integer> nm, int i, List <Integer>
                                    kk)
 {
  if (i == nm.size()){
    {
     System.out.println (kk);
     return ;
    }
   kk.add (nm.get (i));
   task (nm, i+1, kk);
   kk.remove (kk.size()-1);
   task (nm, i+1, kk);
  }
}
```

```
5) import java.util.*;
   class A
{
public static void main (String [] args)
{
Scanner sc = new Scanner (System.in);
List <Integer> nm = new ArrayList <> ();
int n = sc.nextInt ();
for (int i = 0; i<n; i++)
  {
  nm.add (sc.nextInt ());
  }
  task (nm, i, n-1);
}
static void task (List<Integer> nm, int i, int j)
  {
    if (i == nm.size ())
      {
      System.out.println (nm);
      return;
      }
  for (int p = i; p<=j; p++)
    {
    swap (nm, i, p);
    task (nm, i+1, j);
    swap (nm, i, p);
    }
static void swap (List<Integer> nm, int i, int j)
    {
      int d = nm.get (i);
      nm.add (i, nm.get (j));
       nm.remove (i+1);
       nm.add (j, d);
       nm.remove (j+1);
    }
}
```

6)
```
import java.util.*;
class A
{
public static void main (String [] args);
{
        List<List<Integer>> nm = new ArrayList<>();
        int n = 6;
for (int i = 0; i<n ; i++)
{
nm.add (new ArrayList<>());
}
nm.get(0).add(1);
nm.get(0).add(2);
nm.get(0).add(3);
nm.get(1).add(0);
nm.get(1).add(2);
nm.get(1).add(5);
nm.get(2).add(0);
nm.get(2).add(1);
nm.get(2).add(3);
nm.get(2).add(4);
nm.get(2).add(5);
nm.get(3).add(0);
nm.get(3).add(2);
nm.get(4).add(5);
nm.get(4).add(2);
nm.get(5).add(1);
nm.get(5).add(2);
nm.get(5).add(4);
boolean v[] = new boolean [6];
List<Integer> k = new ArrayList<>();
task(nm, v, 0, k, n-1);
}
```

```
static void task (List<List<Integer>> nm,
                  boolean k[], int i, List<Integer> kk,
                  int n)
{
  if(i==n)
  {
    kk.add(0);
    System.out.println(kk);
    kk.remove(kk.size()-1);
    return;
  }
  k[i]=true;
  kk.add(i);
  for(int p:nm.get(i))
  {
    if(!k[p])
      task(nm,k,p,kk,n);
  }
  kk.remove(kk.size()-1);
  k[i]=false;
}}
```

ASSIGNMENT

CSE4131: ALGORITHM DESIGN II

You are allowed to use only those concepts which are covered in the lecture classes, till date.

Submission deadline: 12^{th} July, 2022(Tuesday).

Submit the hard copies of the assignment, within the specified deadline.

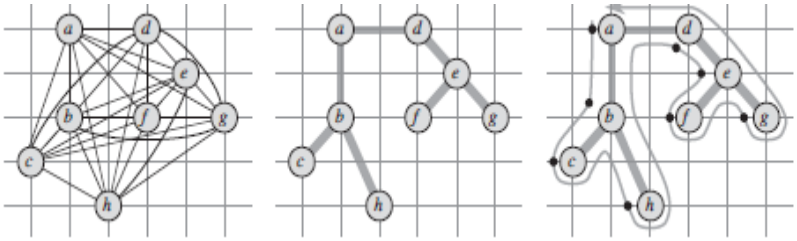
---

1. Suppose you are given a set of positive integers A= $\{a_1, a_2, \ldots\ldots, a_n\}$ and a positive integer B. A subset $S \subseteq A$ is called feasible if the sum of the numbers in S does not exceed B; $\sum_{a_i \in S} a_i \leq B$ the sum of the numbers in S will be called the total sum of S. You would like to select a feasible subset S of A whose total sum is as large as possible.
   a. Here is an algorithm for this problem:
      i. Initially $S \neq \emptyset$
      ii. Define T=0
      iii. $For\ i = 1,2, \ldots . n$
      iv. If $T + a_i \leq B$
      v. $S \leftarrow S \cup \{a_i\}$
      vi. $T = T + a_i$
      vii. End if
      viii. End for

Give an instances in which the total sum of the set S returned by this algorithm is less than half the total sum of some other feasible subset of A.

   b. Give a polynomial-time approximation algorithm for this problem with the following guarantee. It returns a feasible set $S \subseteq A$ whose sum is at least half as large the maximum total sum of any feasible set$S' \subseteq A$. Your algorithm should have a running time of at most $O(nlogn)$.

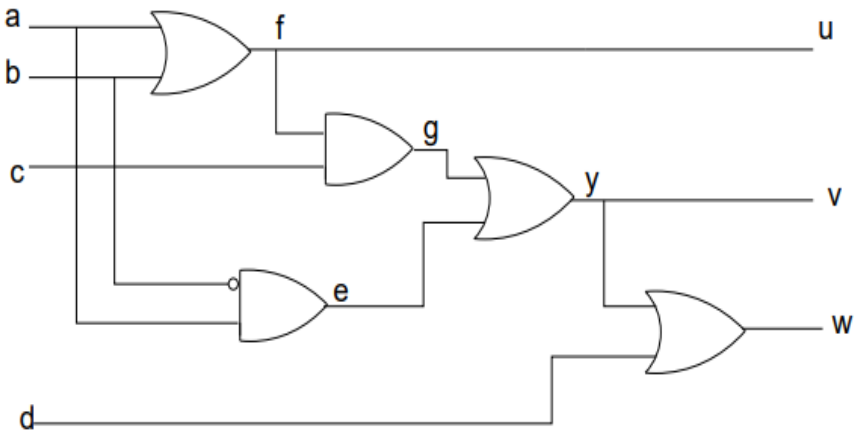
2. We are given a complete undirected graph $G = (V, E)$ that has a nonnegative integer cost $c(u, v)$ associated with each edge$(u, v) \in E$, and we must find a Hamiltonian cycle (a tour) of G with minimum cost. The triangle inequality is, for all $u, v, w \in V$, $c(u, w) \leq c(u, v) + c(v, w)$. We can find the solution using the following algorithm.
   APPROX-TSP-TOUR$(\boldsymbol{G}, \boldsymbol{c})$
   i. select a vertex $r \in V$ to be a "root" vertex
   ii. compute a minimum spanning tree T for G from root r
   Using MST-PRIM (G, c, r) [MST-PRIM: minimum spanning tree using PRIMS algorithm]
   iii. let H be a list of vertices, ordered according to when they are first visited
   In a pre-order tree walk of T
   iv. return the Hamiltonian cycle H



a. Suppose that the vertices for an instance of the traveling-salesman problem are points in the plane and that the cost $c(u, v)$ is the Euclidean distance between point u and v. Show that an optimal tour never crosses itself.

3. Give an efficient greedy algorithm that finds an optimal vertex cover for a tree in linear time.
4. Prove that 2 SAT problem is a P problem.
5. In the following, which functions are the satisfiable function and also give the proper explanation:
   i. f(x, y, z)=(x + y + z)(x + -y + z)(x + y + -z)(x + -y + -z)
   ii. f(x, y, z, w)=(x + -y + z)(-x)(-w + -x + y + z)
   iii. f(x, y)=(x + y) (-x + -y)
   iv. f(p, q, r)=(((p → q) ∧ (q → r)) ∧ p) ∧ (¬r))

6.



Write the circuit function of the above circuit in terms of Boolean function. Then check the satisfiability.

Name :- Shaikh Jamil Ahemad
Reg No :- 20410010134
Sec :- CSE 'M'

AD-2 Assignment-3

Q-1 :- a) Let $A = \{1, 2, 3, 4, 5\}$
$B = 16$

A/Q, $S \leq B$

If $S = \{1, 2, 3, 4, 5\}$
then $S \subseteq A$

$$\sum_{a_i \in S} a_i \leq B$$

Now,

If $S_1 = \{1, 2\}$
$S_1 \leq B$
and $\sum a_i \leq B$
$a_i \in S_1$

If $S_2 = \{3, 4, 5\}$
$S_2 \subseteq A$

$$\sum_{a_i \in S_2} a_i \subseteq B$$

Now,

$$\sum_{a_i \in S_1} a_i = 3 \quad \text{and} \quad \sum_{a_i \in S_2} = 12$$

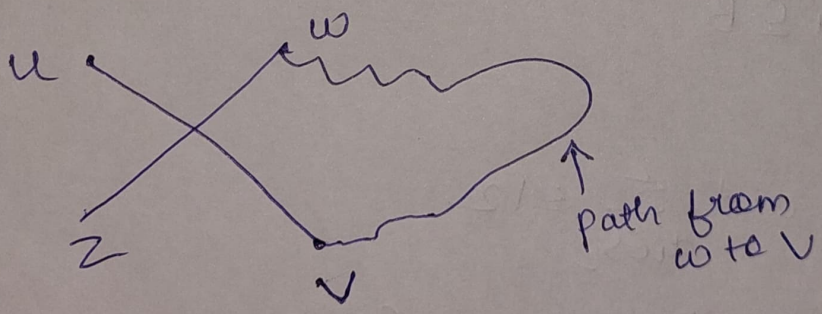
Therefore, $\sum_{a_i \in S_1} a_i < \frac{1}{2} \sum_{a_i \in S_2} a_i$ $\left(\because 3 < \frac{1}{2} \times 12 = 3 < 6\right)$

b) The idea of algorithm is given as: We are adding integers a_i to set S, one by one in descending order until an unfeasible set is obtained by adding an integer.

Algorithm is as follows :-

```
Subsetfeasible(A, B)
Sorting items in A in descending order
    S ← { }
    sum = 0
  for a_j from a1 to an in descending order
       if a_j > B then
          continue
       else if a_j + sum ≤ B
          S ← S ∪ {a_j}
          sum ← sum + a_j
       else
          break
    end if
    end for
    return S.
```

Q-2 :- Suppose the tour crosses itself then it would look like this :-



Explicitly, there must be vertices u, v, w, and z s.t. (u, v) crosses (w, z). In the path from z to w we find some unspecified path w to v, then edge from v to u.

Let p denote path obtained by original tour upto z, from z to v. But w to v in reverse preserving the original cost.

Now, we can imagine a point c which corresponds to point of intersection of edges. Let $c(x,y)$ be dist. between x and y. Since, the edges are Euclidean distances, we have,

$$c(u,v) + c(z,w) = c(z,c) + c(c,w) + c(u,c) + c(c,v)$$

By triangle law of inequality,

$$c(z,c) + c(c,v) \geq c(z,v) \text{ and } c(u,c) + c(c,w) \geq c(u,w)$$

Combining these with equality above gives:

$$c(u,v) + c(z,w) \geq c(z,v) + c(u,w)$$

Thus, the path P incurs a smaller total cost than original path. But original path was assumed to be optimal. This is a contradiction, so, it must be case that no optimal path can have crossing edges.

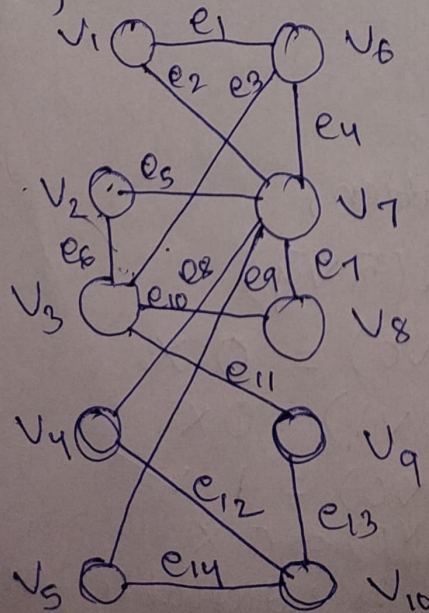
Q-3: For finding optimal vertex cover, we can establish relationship between vertex cover and independant set which would give us either of the solutions when required.

$V = V_I + V_S$

V_I: Independant set

V_S: Vertex cover.

Suppose,



Now,

$V_I = \{V_1, V_2, V_8, V_4, V_5, V_9\}$

Now,

$V_S = V - V_I$

$= \{V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}\} -$

$\{V_1, V_2, V_8, V_4, V_5, V_9\}$

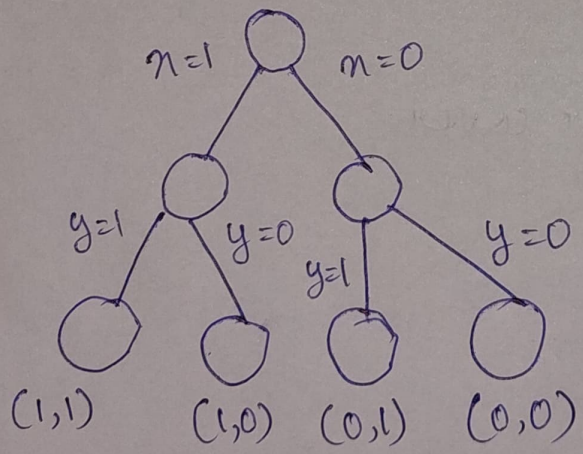
$= \{V_6, V_7, V_3, V_{10}\}$

Hence, this relation gives us optimal vertex cover. So, from independant set using greedy approach we can find out optimal vertex cover.

The time complexity is O(E). We can say, this is linear as no. of edges increases linearly on increasing the no. of vertices.

Q-4: 2-SAT ⟶ 2 literals

Drawing a state space tree for 2 SAT



Suppose we have a boolean function,

$f(x,y) = (x \vee y) \wedge (\bar{x} \vee y) \wedge (\bar{x} \vee \bar{y})$

$f(1,1) = (1 \vee 1) \wedge (0 \vee 1) \wedge (0 \vee 0) = 0$ ✗

$f(1,0) = (1 \vee 0) \wedge (0 \vee 0) \wedge (0 \vee 1) = 0$ ✗

$f(0,1) = (0 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 0) = 1$ ✓

$f(0,0) = (0 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) = 0$ ✗

Hence, the boolean function is satisfiable for $x=1, y=1$

Here, we implement DFS to traverse through the tree for finding the values of literals. So, the time taken by 2-SAT = Time taken by DFS. DFS takes polynomial time. Hence, 2 SAT is a P problem.

Q-5:-

(i) $f(x,y,z) = (x+y+z)(x+-y+z)(x+y+-z)(x+-y+-z)$

As, we have 3 literals, so, we have $2^3 = 8$ combinations for inputs.

Now,

$f(0,0,0) = 0 \cdot 1 \cdot 1 \cdot 1 = 0$

$f(0,0,1) = 1 \cdot 1 \cdot 0 \cdot 1 = 0$

$f(0,1,0) = 1 \cdot 0 \cdot 1 \cdot 1 = 0$

$f(0,1,1) = 1 \cdot 1 \cdot 1 \cdot 0 = 0$

$f(1,0,0) = 1 \cdot 1 \cdot 1 \cdot 1 = 1$ ✓

$f(1,0,1) = 1 \cdot 1 \cdot 1 \cdot 1 = 1$ ✓

$f(1,1,0) = 1 \cdot 1 \cdot 1 \cdot 1 = 1$ ✓

$f(1,1,1) = 1 \cdot 1 \cdot 1 \cdot 1 = 1$ ✓

Hence, the function is satisfiable for (1,0,0), (1,0,1), (1,1,0) and (1,1,1)

(ii) $f(x,y,z,w) = (x+-y+z)(-x)(-w+-x+y+z)$

Here, we have 4 literals, so, we have $2^4 = 16$ combinations.

But here $\nexists f(x,y,z,w) = 0$ for every $x = 1$. Therefore, we will avoid the condition having $x = 1$.

Now,

$f(0,1,1,1) = 1 \cdot 1 \cdot 1 = 1$ ✓

$f(0,1,1,0) = 1 \cdot 1 \cdot 1 = 1$ ✓

$f(0,1,0,1) = 0$

$f(0,1,0,0) = 0$

$f(0,0,1,1) = 1 \cdot 1 \cdot 1 = 1$ ✓

$f(0,0,1,0) = 1 \cdot 1 \cdot 1 = 1$ ✓

$f(0,0,0,1) = 1 \cdot 1 \cdot 1 = 1$ ✓

$f(0,0,0,0) = 1 \cdot 1 \cdot 1 = 1$ ✓

Here, the function is satisfiable for (0,1,1,1), (0,1,1,0), (0,0,1,1), (0,0,1,0), (0,0,0,1) and (0,0,0,0).

(iii) $f(x,y) = (x+y)(-x+-y)$

Here, we have 2 literals. So, there would be $2^2 = 4$ combinations.

Now,

$f(1,1) = 1 \cdot 0 = 0$

$f(1,0) = 1 \cdot 1 = 1$ ✓

$f(0,1) = 1 \cdot 1 = 1$ ✓

$f(0,0) = 0 \cdot 1 = 0$

Hence, the function is satisfiable for (1,0) and (0,1).

iv) $f(p,q,r) = (((p \to q) \wedge (q \to r)) \wedge p) \wedge (-r))$

Here, we have 3 literals. So, there are $2^3 = 8$ possible combinations.

But here the last literals is $-r$. So, we have to avoid the conditions where $r = 1$ which would give the total result false.

Now, taking the other inputs,

$f(1,1,0) = 0$ ×

$f(1,0,0) = 0$ ×

$f(0,1,0) = 0$ ×

$f(0,0,0) = 0$ ×

Hence, ~~there~~ ∄ any condition for which the total value of this function is true.

Therefore, this function is unsatisfiable.

Q-6:- Here, the boolean function will be

$w(a,b,c,d) = (a+b)\cdot c + b'a + d$

Here, we see that $b'a$ will be 0 if

$a=0, b=1$

$a=0, b=0$

$a=1, b=1$

So, we have to avoid this condition to get a total result as false.

Now,

$w(1,0,1,1) = 1\cdot1+1+1 = 1$ ✓

$w(1,0,1,0) = 1\cdot1+1+0 = 1$ ✓

$w(1,0,0,1) = 1\cdot0+1+1 = 1$ ✓

$w(1,0,0,0) = 1\cdot0+1+0 = 1$ ✓

Therefore, this boolean function is satisfiable for (1,0,1,1), (1,0,1,0), (1,0,0,1) and (1,0,0,0)