**National University of Computer & Emerging Sciences, Karachi**
**Fall-2020 Department of Computer Science**
**Mid Term-2**
**24th November 2020, 10:30 AM – 12:00 PM**

| **Course Code:** CS302 | **Course Name:** Design and Analysis of Algorithm |
|---|---|
| **Instructor Name / Names:** Dr. Muhammad Atif Tahir, Dr. Fahad Sherwani, Zeshan Khan, Waqas Sheikh, Sohail Afzal | |
| **Student Roll No:** | **Section:** |

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **6 questions** on **3 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

**Time**: 90 minutes. **Max Marks: 17.5**

**Question # 1** **[3 marks]**

Consider the following instance of the 0/1 knapsack problem

| **items** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **values** | 3 | 4 | 5 | 6 |
| **weights** | 2 | 3 | 4 | 5 |

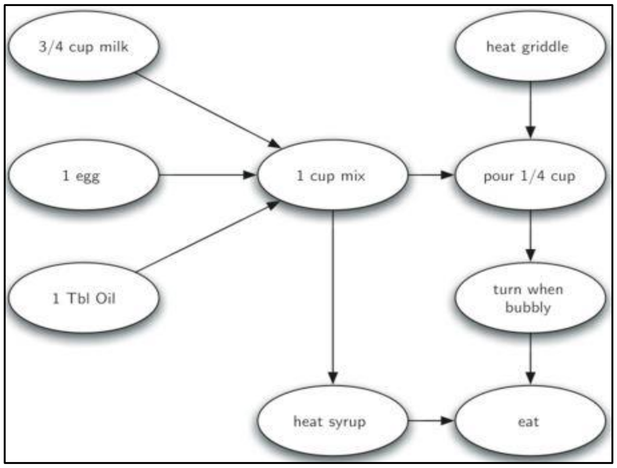
The maximum allowable total weight in the knapsack is W = 5.

Find an optimal solution for the above problem with the weights and values given above using Dynamic Programming. Be sure to state both the maximum value as well as the item(s) that will make maximum value in the given capacity W. Show all steps.

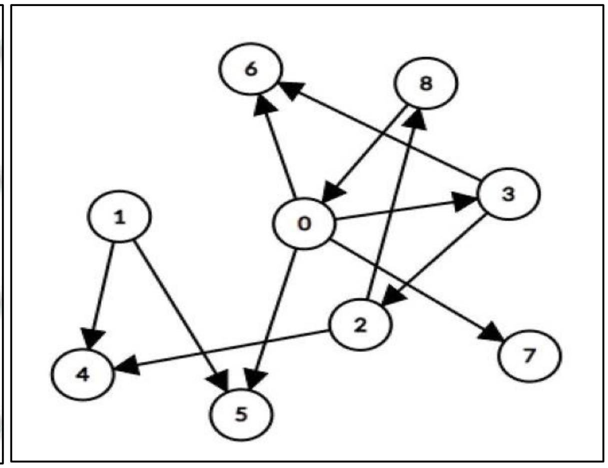
**Question # 2** **[1+0.5 = 1.5 marks]**

Given two directed graphs in part (a) and part (b), you need to write one topological ordering of both graphs if possible.

**Part (a)**




**Part (b)**




**Question # 3** **[1 + 1 = 2 marks]**

Compute the time complexity for both below mentioned algorithms. Show all steps.

$$
\begin{array}{l}
void\ Algorithm1(A, int\ n)\{ \\
\quad for(int\ i = 0; i < \sqrt{n}; i++)\{ \\
\quad\quad for\ (int\ j = 0;\ j < \sqrt{n}; j++)\{ \\
\quad\quad\quad int\ temp = A[j][i]/A[i][i] \\
\quad\quad\quad for\ (int\ k = i + 1; k \leq n + i; k++)\{ \\
\quad\quad\quad\quad A[j][k-i] -= temp * A[i][k-i] \\
\quad\quad\quad \} \\
\quad\quad \} \\
\quad \} \\
\}
\end{array}
$$

$$
\begin{array}{l}
void\ Algorithm2(A, int\ n)\{ \\
\quad for(int\ i = 0; i < n/2; i++)\{ \\
\quad\quad for\ (int\ j = 1;\ j \leq n/2; j++)\{ \\
\quad\quad\quad for\ (int\ k = i + 1; k \leq i + 5; k++)\{ \\
\quad\quad\quad\quad for\ (int\ l = 1; l \leq n; l++)\{ \\
\quad\quad\quad\quad\quad sum += A[i + j][l] \\
\quad\quad\quad \} \\
\quad\quad \} \\
\quad \} \\
\}
\end{array}
$$

**Question # 4** **[3 marks]**

Let A and B be two sequences of *"n"* integers each, in the range $[1, n^2]$, from square series. Given an integer *x*, design an O(n)-time algorithm for determining if there is an integer "*a*" in A and an integer "*b*" in B such that $x = a + b$. (You may write algorithm in plain text)

**Question # 5** **[4 marks]**

Given a graph *G* and a minimum spanning tree *T*, suppose that we decrease the weight of one of the edges not in *T*. Give an algorithm for finding the minimum spanning tree in the modified graph. (You may write algorithm in plain text)

**Question # 6** **[0.5 + 0.5 + 3 = 4 marks]**

**Answer the following.**

a) How do we decide to split the matrix-chain and parenthesize for the optimization of multiplications operations?

b) Which of the following is the recurrence relation for the matrix-chain multiplication problem where mat[i-1] * mat[i] gives the dimension of the $i^{th}$ matrix?

1) M[i, j] = 1 if i=j
   $M[i, j] = \min_{i \le k < j} \{ M[i, k] + M[k+1, j]\}$

2) M [i, j] = 0 if i=j
   $M[i, j] = \min_{i \le k < j} \{M[i, k] + M[k+1, j]\}$

3) M [i, j] = 0 if i=j
   $M[i, j] = \max_{i \le k < j} \{M[i, k] + M[k+1, j]\} + mat[i-1]*mat[k]*mat[j]$

4) M [i, j] = 0 if i=j
   $M[i, j] = \min_{i \le k < j} \{M[i,k] + M[k+1,j]\} + mat[i-1] * mat[k] * mat[j]\}$

c) Design a **recursive algorithm** to compute the minimum number of scalar multiplications for the chain matrix product $A_{i...j}$. (You may write algorithm in plain text)

**BEST OF LUCK**

# National University of Computer & Emerging Sciences, Karachi
## Fall-2020 Department of Computer Science
## Mid Term-2
## 24th November 2020, 10:30 AM – 12:00 PM

| **Course Code:** CS302 | **Course Name:** Design and Analysis of Algorithm |
|---|---|
| **Instructor Name / Names:** Dr. Muhammad Atif Tahir, Dr. Fahad Sherwani, Zeshan Khan, Waqas Sheikh, Sohail Afzal | |
| **Student Roll No:** | **Section:** |

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **6 questions** on **3 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

**Time**: 90 minutes. **Max Marks: 17.5**

**Question # 1** **[3 marks]**

Consider the following instance of the 0/1 knapsack problem

| **items** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **values** | 3 | 4 | 5 | 6 |
| **weights** | 2 | 3 | 4 | 5 |

The maximum allowable total weight in the knapsack is W = 5.

Find an optimal solution for the above problem with the weights and values given above using Dynamic Programming. Be sure to state both the maximum value as well as the item(s) that will make maximum value in the given capacity W. Show all steps. [see appendix]

**Solution**

## Step-01:

- Draw a table say 'T' with (n+1) = 4 + 1 = 5 number of rows and (w+1) = 5 + 1 = 6 number of columns.
- Fill all the boxes of $0^{th}$ row and $0^{th}$ column with 0.

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | | | | | |
| 2 | 0 | | | | | |
| 3 | 0 | | | | | |
| 4 | 0 | | | | | |

## Step-02:

Start filling the table row wise top to bottom from left to right using the formula-

$$T\ (i\ ,\ j) = \max\ \{\ T\ (\ i\text{-}1\ ,\ j\ )\ ,\ value_i + T(\ i\text{-}1\ ,\ j - weight_i\ )\ \}$$

## Finding T(1,1)-

We have,

- i = 1
- j = 1
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,1) = max { T(1-1 , 1) , 3 + T(1-1 , 1-2) }

T(1,1) = max { T(0,1) , 3 + T(0,-1) }

T(1,1) = T(0,1)            { Ignore T(0,-1) }

T(1,1) = 0

## Finding T(1,2)-

We have,

- $i = 1$
- $j = 2$
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,2) = max { T(1-1 , 2) , 3 + T(1-1 , 2-2) }

T(1,2) = max { T(0,2) , 3 + T(0,0) }

T(1,2) = max {0 , 3+0}

T(1,2) = 3

## Finding T(1,3)-

We have,

- $i = 1$
- $j = 3$
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,3) = max { T(1-1 , 3) , 3 + T(1-1 , 3-2) }

T(1,3) = max { T(0,3) , 3 + T(0,1) }

T(1,3) = max {0 , 3+0}

T(1,3) = 3

## Finding T(1,4)-

We have,

- $i = 1$
- $j = 4$
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,4) = max { T(1-1 , 4) , 3 + T(1-1 , 4-2) }

T(1,4) = max { T(0,4) , 3 + T(0,2) }

T(1,4) = max {0 , 3+0}

T(1,4) = 3

## Finding T(1,5)-

We have,

- $i = 1$
- $j = 5$
- $(value)_i = (value)_1 = 3$
- $(weight)_i = (weight)_1 = 2$

Substituting the values, we get-

T(1,5) = max { T(1-1 , 5) , 3 + T(1-1 , 5-2) }

T(1,5) = max { T(0,5) , 3 + T(0,3) }

T(1,5) = max {0 , 3+0}

T(1,5) = 3

## Finding T(2,1)-

We have,

- $i = 2$
- $j = 1$
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,1) = max { T(2-1 , 1) , 4 + T(2-1 , 1-3) }

T(2,1) = max { T(1,1) , 4 + T(1,-2) }

T(2,1) = T(1,1)          { Ignore T(1,-2) }

T(2,1) = 0

## Finding T(2,2)-

We have,

- $i = 2$
- $j = 2$
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,2) = max { T(2-1 , 2) , 4 + T(2-1 , 2-3) }

T(2,2) = max { T(1,2) , 4 + T(1,-1) }

T(2,2) = T(1,2)          { Ignore T(1,-1) }

T(2,2) = 3

## Finding T(2,3)-

We have,

- $i = 2$
- $j = 3$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

T(2,3) = max { T(2-1 , 3) , 4 + T(2-1 , 3-3) }

T(2,3) = max { T(1,3) , 4 + T(1,0) }

T(2,3) = max { 3 , 4+0 }

T(2,3) = 4

## Finding T(2,4)-

We have,

- $i = 2$
- $j = 4$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

T(2,4) = max { T(2-1 , 4) , 4 + T(2-1 , 4-3) }

T(2,4) = max { T(1,4) , 4 + T(1,1) }

T(2,4) = max { 3 , 4+0 }

T(2,4) = 4

## Finding T(2,5)-

We have,

- $i = 2$
- $j = 5$
- $(value)_i = (value)_2 = 4$
- $(weight)_i = (weight)_2 = 3$

Substituting the values, we get-

T(2,5) = max { T(2-1 , 5) , 4 + T(2-1 , 5-3) }

T(2,5) = max { T(1,5) , 4 + T(1,2) }

T(2,5) = max { 3 , 4+3 }

T(2,5) = 7

Similarly, compute all the entries.

After all the entries are computed and filled in the table, we get the following table-

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 3 | 3 | 3 | 3 |
| 2 | 0 | 0 | 3 | 4 | 4 | 7 |
| 3 | 0 | 0 | 3 | 4 | 5 | 7 |
| 4 | 0 | 0 | 3 | 4 | 5 | 7 |

T-Table

- The last entry represents the maximum possible value that can be put into the knapsack.
- So, maximum possible value that can be put into the knapsack = 7.

## Identifying Items To Be Put Into Knapsack-

Following Step-04,

- We mark the rows labelled “1” and “2”.
- Thus, items that must be put into the knapsack to obtain the maximum value 7 are-

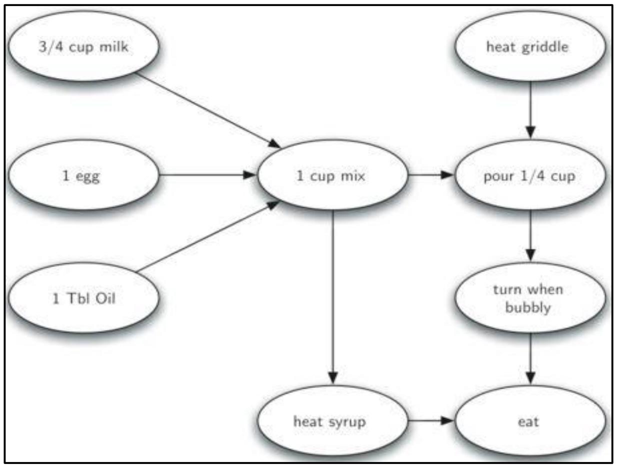
**Item-1 and Item-2**

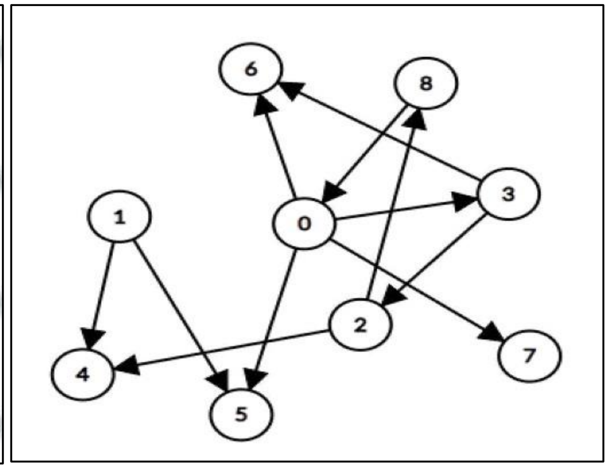
**KEEP matrix can be created to identify which item will make 7**

**Question # 2** **[1+0.5 = 1.5 marks]**

Given two directed graphs in part (a) and part (b), you need to write one possible topological ordering of both graphs.

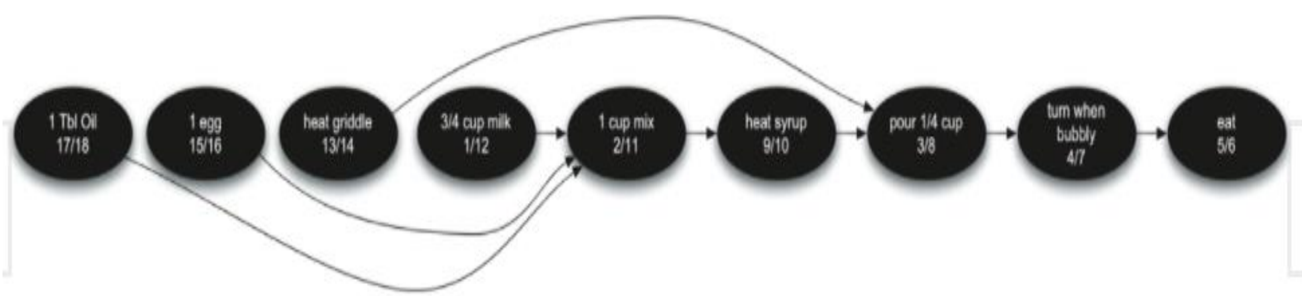
**Part (a)** **Part (b)**




**Solution**

Part (a):




Part (b):

Not possible as there is cycle among nodes (8,0,3,2). But how computer will know that there is cycle. For this, there are two approaches. Either apply DFS(depth first search) or Kahn algorithm(choose node where indegree is zero and so on) for finding topological ordering. You will notice that both approaches will fail. Hence topological ordering not possible

**Question # 3** **[1 + 1 = 2 marks]**

Compute the time complexity for both below mentioned algorithms. Show all steps.

```
void Algorithm1(A, int n){
        for(int i = 0; i ≤ √n; i++){
                for (int j = 0; j ≤ √n; j++){
                        int temp = A[j][i]/A[i][i]
                        for (int k = i + 1; k ≤ n + i; k++){
                                A[j][k − i] −= temp ∗ A[i][k − i]
                        }
                }
        }
}
```

```
void Algorithm2(A, int n){
        for(int i = 0; i ≤ n/2; i++){
                for (int j = 1; j ≤ n/2; j++){
                        for (int k = i + 1; k ≤ i + 5; k++){
                            for (int l = 1; l ≤ n; l++){
                                        sum += A[i + j][l]
                        }
                }
        }
}
```

**Solution**

Algorithm 1:

3(a)

$$\sum_{i=0}^{\sqrt{n}} \sum_{j=0}^{\sqrt{n}} \left( \sum_{k=i+1}^{n+i} 1 \right)$$

$$\sum_{i=0}^{\sqrt{n}} \sum_{j=0}^{\sqrt{n}} \left( n+i-i-1+1 \right)$$

$$\sum_{i=0}^{\sqrt{n}} \sum_{j=0}^{\sqrt{n}} n$$

$$\sum_{i=0}^{\sqrt{n}} \left( n\left(\sqrt{n}-0+1\right) \right)$$

$$\sum_{i=0}^{\sqrt{n}} \left( n\sqrt{n}+n \right)$$

$$\left(\sqrt{n}-0+1\right)\left(n\sqrt{n}+n\right)$$

$$\left(\sqrt{n}+1\right)\left(n\sqrt{n}+n\right)$$

$$n^2+n\sqrt{n}+n\sqrt{n}+n$$

$$n^2+2n\sqrt{n}+n$$

$$\boxed{=O(n^2)}$$

Algorithms 2:

$$\sum_{i=0}^{n/2} \sum_{i=1}^{n/2} \sum_{k=i+1}^{i+5} \sum_{l=1}^{n} 1$$

$$\sum_{i=0}^{n/2} \sum_{i=1}^{n/2} \sum_{k=i+1}^{i+5} (n-1+1)$$

$$\sum_{i=0}^{n/2} \sum_{i=1}^{n/2} \sum_{k=i+1}^{i+5} (n)$$

$$\sum_{i=0}^{n/2} \sum_{i=1}^{n/2} \left[ (i+5-i-1+1) n \right]$$

$$\sum_{i=0}^{n/2} \sum_{i=1}^{n/2} 5n$$

$$\sum_{i=0}^{n/2} \left[ \left(\frac{n}{2}-1+1\right) 5n \right]$$

$$\sum_{i=0}^{n/2} \frac{5n^2}{2} \Rightarrow \left(\frac{n}{2}-0+1\right)\frac{5n^2}{2}$$

$$\left(\frac{n}{2}+1\right)\left(\frac{5n^2}{2}\right)$$

$$\frac{5n^3}{4}+\frac{5n^2}{2}$$

$$\boxed{=O(n^3)}$$

**Question # 4** **[3 marks]**

Let A and B be two sequences of *"n"* integers each, in the range $[1, n^2]$, from square series. Given an integer *x*, design an O(n)-time algorithm for determining if there is an integer "*a*" in A and an integer "*b*" in B such that *x* = *a* + *b*. (You may write algorithm in plain text)

**Solution**

Sort both a and b by count sort altered version

```
countingSort(array):

  size = len(array)

  output = [0] * size

  count = [0] * 10

  for i in range(0, size):

        count[sqrt(array[i])] += 1//Modification

  for i in range(1, 10):

        count[i] += count[i - 1]

  i = size – 1

  while i >= 0:

        output[count[array[i]] - 1] = array[i]

        count[array[i]] -= 1

        i -= 1

  for i in range(0, size):
```

```
array[i] = (output[i]* output[i]) //Modification
```

Then apply the following linear algorithm
Start “i” from first index of A and “j” from last index of  B

```
if(A[i] + B[j] < x) -> update index i to be i + 1
if(A[i] + B[j] > x) -> update index j to be j - 1
if(A[a] + B[b] = x) -> success
```

**Question # 5** **[4 marks]**

Given a graph *G* and a minimum spanning tree *T*, suppose that we decrease the weight of one of the edges not in *T*. Give an algorithm for finding the minimum spanning tree in the modified graph. (You may write algorithm in plain text)

**Solution**

If we were to add in this newly decreased edge to the given tree, we would be creating a cycle. Then, if we were to remove any one of the edges along this cycle, we would still have a spanning tree. This means that we look at all the weights along this cycle formed by adding in the decreased edge, and remove the edge in the cycle of maximum weight. This does exactly what we want since we could only possibly want to add in the single decreased edge, and then, from there we change the graph back to a tree in the way that makes its total weight minimized.

**Question # 6** **[0.5 + 0.5 + 3 = 4 marks]**

**Answer the following.**

a) How do we decide to split the matrix-chain and parenthesize for the optimization of multiplications operations? (0.5 marks)

## Solution (a)

The matrix-chain problem can be solved recursively to determine the best value of k, we will consider all possible values of k, and pick the best of them to split the matrix-chain accordingly, providing optimal multiplication operations for each subproblem. Parenthesizing can be done in the form of (A[1…k])+(A[k+1…..n]).

b) Which of the following is the recurrence relation for the matrix-chain multiplication problem where mat[i-1] * mat[i] gives the dimension of the $i^{th}$ matrix? (0.5 marks)

1) M[i, j] = 1 if i=j

M[i, j] = min{ M[i, k] + M[k+1, j]}

2) $M[i, j] = 0$ if $i=j$

$M[i, j] = \min\{M[i, k] + M[k+1, j]\}$

3) $M[i, j] = 0$ if $i=j$

$M[i, j] = \max\{M[i, k] + M[k+1, j]\} + mat[i-1]*mat[k]*mat[j]$

4) $M[i, j] = 0$ if $i=j$

$M[i, j] = \min\{M[i, k] + M[k+1, j]\} + mat[i-1]*mat[k]*mat[j]$

**Solution (b)**

4) $M[i,j] = 0$ if $i=j$

$M[i,j] = \min\{M[i, k] + M[k+1, j]\} + mat[i-1]*mat[k]*mat[j]$

c) Design a **recursive algorithm** to compute the minimum number of scalar multiplications for the chain matrix product $A_{i...j}$ in a **top-down manner**. You can also write the algorithm in English. (3 marks)

**Solution (c)**

```
Rec-Matrix-Chain(array p, int i, int j) {
    if (i = = j) m[i, i] = 0;                    // basic case
    else {
            m[i, j] = infinity;                  // initialize
            for k = i to j − 1 do {              // try all possible splits
                    cost=Rec-Matrix-Chain(p, i, k) + Rec-Matrix-Chain(p, k + 1, j) + p[i − 1]*p[k]*p[j];
                    if (cost<m[i, j]) then
                            m[i, j]= cost;
                    }
            }                                    // update if better
    return m[i,j];                               // return final cost
}
```

**BEST OF LUCK**