National University of Computer & Emerging Sciences, Karachi
Fall-2019 Department of Computer Science
Final Term
16th December 2019, 09:00 AM – 12:00 PM

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

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **09 questions** on **04 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

Time: 180 minutes. **Max Marks: 50**

Question # 1 **[(0. 25+0.25)*10=5 marks]**

Complete the following table.

Algorithm	**Worst Case**	**Write below whether the algorithm belongs to Dynamic Programming / Greedy Algorithms / None of them**
0/1 Knapsack using Dynamic Programming		
0/1 Knapsack using Brute Force		
Breadth First Search		
Depth First Search		
Kruskal's algorithm for finding MST		
Prim's algorithm for finding MST		
Shortest path by Dijkstra		
Shortest path by Bellman Ford		
Matrix Chain Multiplication using Dynamic Programming		
Matrix Chain Multiplication using Brute Force		

Solution:

Algorithm	**Worst Case**	**Write below whether the algorithm belongs to Dynamic Programming / Greedy Algorithms / None of them**
0/1 Knapsack using Dynamic Programming	O(nW)	Dynamic Programming
0/1 Knapsack using Brute Force	$O(2^n)$	None of them

Breadth First Search	O(V + E)	Greedy
Depth First Search	O(V + E)	Greedy
Kruskal's algorithm for finding MST	O(ElogV)	Greedy
Prim's algorithm for finding MST	O(ElogV)	Greedy
Shortest path by Dijkstra	O((V + E)logV)	Greedy
Shortest path by Bellman Ford	O(V * E) or O(V^3)	Dynamic Programming
Matrix Chain Multiplication using Dynamic Programming	$O(n^3)$	Dynamic Programming
Matrix Chain Multiplication using Brute Force	$O(4^n)$	None of them

Question # 2 **[3+1+2=6 marks]**

(a) What is meant by P and NP Problems? Explain P = NP.
(b) Why it is important to find approximate solutions for NP Complete Problems?
(c) Explain the difference between greedy algorithms and dynamic programming?

Solution:

A)

- P problems
  - (The original definition) Problems that can be solved by **deterministic Turing machine** in polynomial-time.
  - (A equivalent definition) Problems that are solvable in polynomial time.
- NP problems
  - (The original definition) Problems that can be solved by **non-deterministic Turing machine** in polynomial-time.
  - (A equivalent definition) Problems that are **verifiable** in polynomial time.
    - Given a solution, there is a polynomial-time algorithm to tell if this solution is correct.

P = NP means whether an NP problem can belong to class P problem. In other words, whether every problem whose solution can be verified by a computer in polynomial time can also be solved by a computer in polynomial tim

(b) If a problem is NP-complete, there is very likely no polynomial-time algorithm to find an optimal solution. The idea of approximation algorithms is to develop polynomial-time algorithms to find a near optimal solution

(c) Explain the difference between greedy algorithms and dynamic programming **[2 Points]**

Greedy Algorithms: A greedy algorithm is a mathematical process that looks for simple and easy-to-implement solutions to complex, multi-step problems by deciding which next step will provide the most obvious benefit.

Dynamic Programming: It is often used to solve optimization problems. Its is a tabular method in which we break down the problem into subproblems and place the solution to the subproblems in a matrix

Question # 3 **[5 marks]**

What is convex Hull? Design $O(N^3)$, $O(Nh)$ and $O(NlogN)$ algorithms to solve convex hull problem?

Solution:

Convex Hull
A set of points is convex if for any two points p and q in the set, the line segment pq is completely in the set.

$O(N^3)$ algorithms

Brute-force
For all pairs of points p and q in P
• compute ccw(p, q, x) for all other x in P
• p-q is on hull if all values positive

$O(Nh)$ algorithms

Package wrap
Start with point with smallest y-coordinate.
• Rotate sweep line around current point in ccw direction.
• First point hit is on the hull.
• Repeat

$O(NlogN)$ algorithms

Graham scan.
• Choose point p with smallest y-coordinate.
• Sort points by polar angle with p to get simple polygon.
• Consider points in order, and discard those that would create a clockwise turn.

Question # 4 **[0.5+5+0.5=6 marks]**

Floyd-Warshall can be used to determine whether or not a graph has *transitive closure*, i.e. whether or not there are paths between all vertices using the following procedure:

- Assign all edges in the graph to have weight = 1
- Run Floyd-Warshall
- Check if all $d_{ij} < n$ i.e. all values in matrix D is less than all values in matrix n.

Use the above algorithm to determine whether the graph in ***Figure 1*** has *transitive closure* or not.

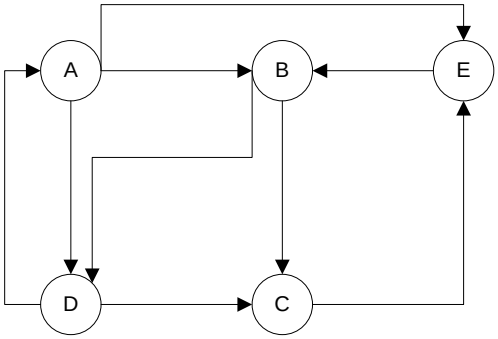
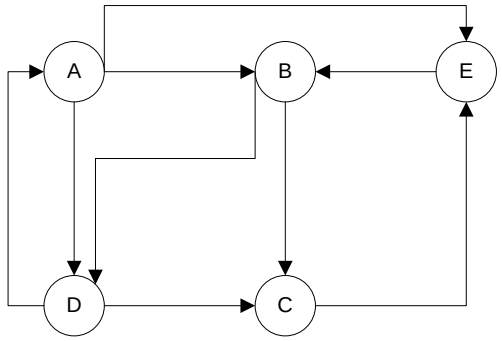
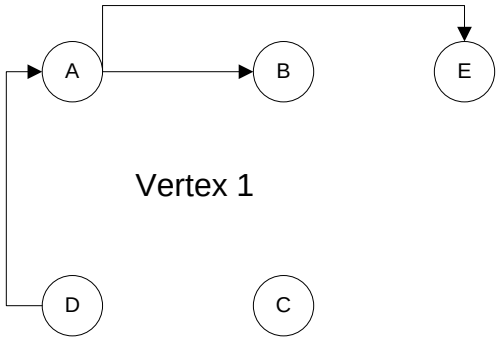
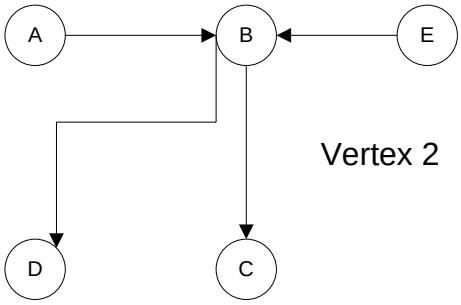


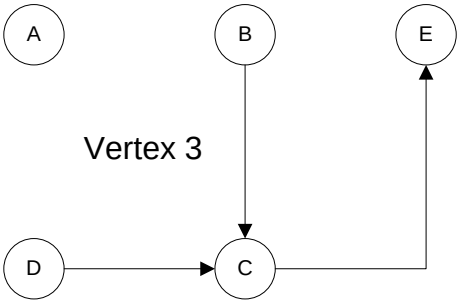
Figure 1

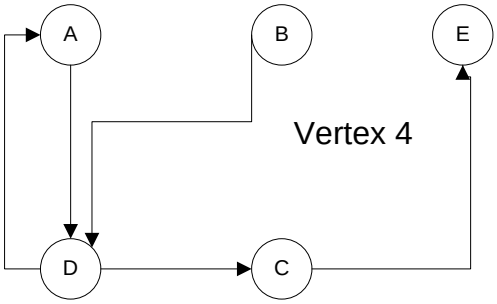
Solution:

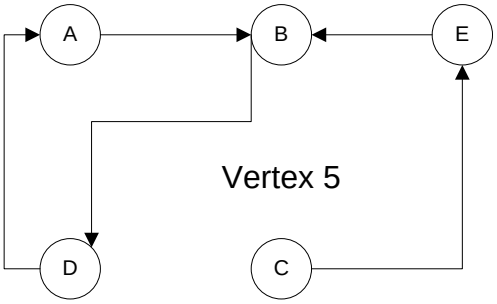












For iteration 0 (Matrix D)

D	1	2	3	4	5
1	0	1	Inf	1	1
2	inf	0	1	1	Inf
3	inf	inf	0	inf	1
4	1	inf	1	0	Inf
5	inf	1	inf	inf	0

Matrix n	1	2	3	4	5
1	-	1	-	1	1
2	-	-	2	2	-
3	-	-	-	-	3
4	4	-	4	-	-
5	-	5	-	-	-

For iteration 1 through vertex 1 (Matrix D)

D	1	2	3	4	5
1	0	1	Inf	1	1
2	Inf	0	1	1	Inf
3	Inf	inf	0	inf	1

Matrix n	1	2	3	4	5
1	-	1	-	1	1
2	-	-	2	2	-
3	-	-	-	-	3

4	1	2	1	0	2
5	inf	1	inf	inf	0

4	4	1	4	-	1
5	-	5	-	-	-

For iteration 1 through vertex 2 (Matrix D)

D	1	2	3	4	5
1	0	1	2	1	1
2	Inf	0	1	1	Inf
3	Inf	inf	0	inf	1
4	1	2	1	0	2
5	inf	1	2	2	0

Matrix n	1	2	3	4	5
1	-	1	2	1	1
2	-	-	2	2	-
3	-	-	-	-	3
4	4	1	4	-	1
5	-	5	2	2	-

For iteration 1 through vertex 3 (Matrix D)

D	1	2	3	4	5
1	0	1	2	1	1
2	Inf	0	1	1	2
3	Inf	inf	0	inf	1
4	1	2	1	0	2
5	inf	1	2	2	0

Matrix n	1	2	3	4	5
1	-	1	2	1	1
2	-	-	2	2	3
3	-	-	-	-	3
4	4	1	4	-	1 or 3
5	-	5	2	2	-

For iteration 1 through vertex 4 (Matrix D)

D	1	2	3	4	5
1	0	1	2	1	1
2	2	0	1	1	2
3	Inf	inf	0	inf	1
4	1	2	1	0	2
5	inf	1	2	2	0

Matrix n	1	2	3	4	5
1	-	1	2	1	1
2	4	-	2	2	3
3	-	-	-	-	3
4	4	1	4	-	1 or 3
5	-	5	2	2	-

For iteration 1 through vertex 5 (Matrix D)

D	1	2	3	4	5
1	0	1	2	1	1
2	2	0	1	1	2
3	4	2	0	3	1
4	1	2	1	0	2
5	3	1	2	2	0

Matrix n	1	2	3	4	5
1	-	1	2	1	1
2	4	-	2	2	3
3	5	5	-	5	3
4	4	1	4	-	1 or 3
5	5	5	2	2	-

Question # 5 **[4 marks]**

Consider the following APPROX-VERTEX-COVER algorithm. Proof that this algorithm is 2-approximation method for VERTEX-COVER.

```
APPROX-VERTEX-COVER(G)
        C = ∅;
        E'=G.E;
        while(E' ≠ ∅ ){
                Randomly choose a edge (u,v) in E', put u and v into C;
                Remove all the edges that covered by u or v from E'
        }
        Return C;
```

Solution:

Proof:

- Assume a minimum vertex-cover is U*
- A vertex-cover produced by **APPROX-VERTEX-COVER**(G) is U
- The edges chosen in **APPROX-VERTEX-COVER**(G) is A
- A vertex in U* can only cover 1 edge in A
  - So |U*|>= |A|
- For each edge in A, there are 2 vertices in U
  - So |U|=2|A|
- So |U*|>= |U|/2
- So $\frac{|U|}{|U^*|} \leq 2$

Question # 6 **[4 marks]**

An Instance (X, F) of the set-covering problem consists of a finite set X and a family F of subset of X, such that every element of X belongs to at least one subset of F:

$$X = \bigcup_{S \in F} S$$

We say that a subset $S \in F$ covers all elements in X. Our goal is to find a minimum size subset $C \subseteq F$ whose members cover all of X.

$$X = \bigcup_{S \in C} S$$

Algorithm 1: GREEDY-SET-COVER (X, F)

```
1 U ← X
2 C ← ∅
3 While U ≠ 0
4    do select an S ∈ F that maximizes |S ∩ U|
5       U ← U − S
6       C ← C ∪ {S}
7 return C
```

Consider each of the following words as a set of letters: {arid, dash, drain, heard, lost, nose, shun, slate, snare, thread}. Show which set cover GREEDY-SET-COVER produces, when we break ties in favor of the word that appears first in the dictionary.

Solution:

Since all of the words have no repeated letters, the first word selected will be the one that appears earliest on among those with the most letters, this is "thread". Now, we look among the words that are left, seeing how many letters that aren't already covered that they contain. Since "lost" has four letters that have not been mentioned yet,and it is first among those that do, that is the next one we select. The next one we pick is "drain" because it is has two unmentioned letters. This only leave "shun" having any unmentioned letters, so we pick that, completing our set. So, the final set of words in our cover is $\{thread, lost, drain, shun\}$.

1 point for each explanation
1 point for correct order

Question # 7 **[4+3=7 marks]**

Analyze the time complexity of the following algorithms and write in the most appropriate asymptotic notations:

A)

```
double Algo(Point points[], int i=0, int j=n)
{
   if (j < i+2)
      return 0;
   double res = MAX;
   for (int k=i+1; k<j; k++)
        res = min(res, (Algo(points, i, k) + Algo(points, k, j) +
cost(points, i, k, j)));
   return  res;
}
```

Note: There are n points (i.e. points []) and the values of i and j are 0 and n-1 respectively.

B)

```
MatrixChainOrder(int dims[])
    n = dims.length - 1;
    for (i = 1; i <= n; i++)
        m[i, i] = 0;
    for (len = 2; len <= n; len++)
        for (i = 1; i <= n - len + 1; i++)
            j = i + len - 1;
            m[i, j] = MAXINT;
            for (k = i; k <= j - 1; k++)
                cost = m[i, k] + m[k+1, j] + dims[i-
1]*dims[k]*dims[j];
                if (cost < m[i, j])
                    m[i, j] = cost;
                    s[i, j] = k;
```

Solution:

A) $T(n) = O(n^n)$
B) $T(n) = O(n^3)$

Question # 8 **[3*3=9 marks]**

Which algorithm will you prefer to use in given scenarios, support your answer with some brief discussion?

A) A maximum bottleneck path, between a source *s* and a target *t* that allows the most flow to go through it. The amount of flow that can go through the bottleneck is equivalent to the weight of the minimum weight edge on that path. Because this edge effectively limits the amount of flow that can go from *s* to *t* in any path in the graph. Which algorithm will you apply to find the widest path and bottleneck edges?

B) You need to determine the total area of rainfall by analyzing a subset of all sensors that send the signal of the waterfall in a specific region to your computer, which algorithm you will apply to find out the total area of rainfall.

C) Suppose you are given a graph $G = (V, E)$ with edge weights $w(e)$ and a minimum spanning tree T of G. Now, suppose a new edge $\{u, v\}$ is added to G. Describe (in words) a method for determining if T is still a minimum spanning tree for G.

Solution:

A) A possibly more clear way of computing widest paths is by modifying the greedy choice of **Dijkstra's**. Dijkstra's greedy choice. Widest paths and bottleneck edges can also be computed very efficiently using slightly modification of Kruskal's or Prim's to compute the maximum spanning tree of a graph in O(mlogn) time and then simply take the single path in that maximum spanning tree that connects s and t as the widest path.

B) Use of **convex hull** algorithm by weather professionals to determine the area of rainfall.

C) Examine the path in T from u to v. If any vertex on this path has weight larger than that of the new edge, then T is no longer an **MST**. We can modify T to obtain a new MST by removing the max weight edge on this path and replacing it with the new edge. Using the graph for T, we can do a recursive tree traversal in T, starting at vertex u. Once the traversal reaches v, we "unwind" the recursion, and as we do so, we look for the max weight edge along the u,v path.

Question # 9 **[4 marks]**

Suppose that you've computed an MST T* for a graph $G = (V, E)$. Afterwards, you add a new edge $(u, v) \notin E$ to the graph with cost c(u, v). Depending on the shape of the graph, the initial edge weights, and the weight of (u, v), your old MST T* might not necessarily be an MST of the new graph $G' = (V, E\{(\cup u, v)\})$.
Assume that all edge weights in G' are distinct (which means that the edge weights in G are also distinct). Design an O(n)-time algorithm (pseudo code) that determines whether T* is an MST of G'.

Solution:

```
Is_MST(T*,c):
    edges = T*.DFS(c.u to c.v)//Linear (V=E)
    maxe=max(edges)//Linear
    return if(maxe<c)//constant
```

BEST OF LUCK