CS 58000\_01 Algorithm Design Analysis & Implementation(3 cr.)

Final Exam (Assignment As05)

Student Name: Harsh Sharma

This final exam (assignment As05) is due at 12:00 noon, Tuesday, December 12, 2023. Please submit your assignment to Brightspace (purdue.brightspace.com). No late turn-in is accepted. Please write your name on the first page of your assignment. (Take 10 off without giving your proper name on the first page). Your file name should be the first character of your first name, followed by your last name, such as PNg.docx. Please number your problem-answer clearly, such as Problem (1.a), (1.b), (1.c), (1.d), (1.e), Problem (2.a), (2.b), …, (4.d). The answers to the problems must be arranged in order. Please answer your questions using only a Word file (.docx file). No pdf file will be accepted. Without using a Word file (.docx file), the submitted problems’ answers would not be graded or would take 10 points off. If you attach a pdf page of the solution to your Word file, please leave a few lines blank before your solution page; this allows me to place the cursor for writing comments on your Word file.

The total number of points for this Final Exam (Assignment As\_05) is 170.

Given a connected weighted directed graph G(V, E) with its adjacency matrix, which is as follows:

2

5

6

3

7

3

4

1

1. Its adjacency matrix, R(0)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | 0 | 1 | 0 | 0 | 0 |
| B | 0 | 0 | 1 | 1 | 1 |
| C | 1 | 0 | 0 | 0 | 0 |
| D | 1 | 0 | 1 | 0 | 0 |
| E | 0 | 0 | 0 | 1 | 0 |

Figure 1. A graph G(V, E) and its adjacency matrix, R(0).

**Problem 1:[50 points]**

Warshall’s algorithm constructs the transitive closure through a series of n x n boolean matrices: *Compute all the elements of each matrix R(k) from its immediate predecessor R(k-1) in series (1.1), with each intermediate vertex numbered not higher than k.*

R(0) , … , R(k-1) , R(k) , … , R(n) . (1.1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | 0 | 1 | 0 | 0 | 0 |
| B | 0 | 0 | 1 | 1 | 1 |
| C | 1 | **1** | 0 | 0 | 0 |
| D | 1 | **1** | 1 | 0 | 0 |
| E | 0 | 0 | 0 | 1 | 0 |

R(1) uses only A as the intermediate vertex.

C to A to B, then (C, B).

D to A to B, then (D, B)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | 1 | 1 | 1 | 1 | 1 |
| B | 1 | 1 | 1 | 1 | 1 |
| C | 1 | 1 | 1 | 1 | 1 |
| D | 1 | 1 | 1 | 1 | 1 |
| E | 1 | 1 | 1 | 1 | 1 |

1. Its transitive closure, R(n)

Given graph G(V, E) with its adjacency matrix R(0),as shown in Figure (1), the matrix provides the reachability information. If an entry has 1 in the ith row and jth column of the matrix R(0), then there is a directed edge as a directed path from an ith vertex to jth vertex. For example, there is 1 in row B and C column of the matrix R(0), then B reaches C in graph G. The path from B to C has no intermediate vertex. Since k = 0, no intermediate vertex will be served. According to Warshall’s Algorithm, the matrix R(k)  is the result computed from the immediate predecessor R(k-1). At most, k number of vertices can serve as the intermediate vertex for the reachability between any two vertices in the given graph G.

(1.a) For the given graph G(V, E), what are the elements (vertices) for computing R(k) for 0 < k ?

Answer: The elements (vertices) for computing R(k) for 0 < k will be the set of all the vertices present in the graph. So, the vertices are: **A, B, C, D, and E.**

(1.b) Given the graph G(V, E) with its adjacency matrix R(0), for k = 3, what are the vertices in G that could serve as the intermediate vertex for defining a directed path of any two vertices?

Answer: For k = 3, the vertices that could serve as intermediate vertices are A, B, and C. Vertex D cannot serve as an intermediate vertex when k = 3 since D is numbered higher than k=3. Same is the case with vertex E.

So, the **vertices that can act as intermediate vertices remain A, B, and C.**

(1.c) Given the graph G(V, E) with its adjacency matrix R(0), what is the largest value of n of the matrix R(n) that has to be computed for reachability?

Answer: The largest value of n for the matrix R^(n) that has to be computed is 5.

The largest value of n for the matrix R(n) that needs to be computed for reachability in Warshall's algorithm is equal to the number of vertices in the graph G(V,E).

So, the largest value of n for the matrix R(n) that needs to be computed for reachability would be equal to the total number of vertices, Five (5) in our this graph G(V,E).

(1.d) Using its adjacency matrix R(0) to compute matrix R(n), which column and row are considered?

Answer: To compute matrix R(n), **all columns and rows**, A through E, need to be considered.

In Warshall's algorithm, when computing the matrix R(n) from the initial matrix R(0), all columns and rows of the matrix are considered for each iteration k from 1 to n.

(1.e) What is the time efficiency for Warshall’s algorithm?

Answer: The time complexity of Warshall's algorithm is O(V3), where V is the number of vertices in the graph.

The algorithm involves three nested loops, iterating through all vertices and checking paths through each possible intermediate vertex. This triple nested loop runs V3 times.

**Therefore the time complexity for Warshall algorithm is O(V3) and O(53) in our case.**

**Problem 2: [40 points]**

For designing a dynamic programming algorithm for the knapsack problem, given n items of known weights wi with values vi,

(w1, v1) (w2, v2), …, (wn, vn)

a knapsack of capacity holding maximum weight W, the following formula is used to find the most valuable F(n, W) subset of the items that fit into the knapsack.

Max{ **F( i – 1, j),** vi + F( i -1, j - wi ) }, if j - wi ≥ 0.

F(i, j) =

F(i -1, j), if j - wi < 0. ….2.1

with the initial conditions which are as follows:

F(0, j) = 0 for j ≥ 0 and F(i , 0) = 0 for i ≥ 0. …. 2.2

Given an instance data ID:

|  |  |  |
| --- | --- | --- |
| **item** | **weight** | **value** |
| **1** | 2 | $ 12 |
| **2** | 1 | $ 10 |
| **3** | 3 | $ 20 |
| **4** | 2 | $ 15 |

Applying formulas (2.1) and (2.2), the following dynamic programming table is filled. The maximal value is F(4, 5) = 37.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| capacity jW=5 |  | F(i, 0) = 0 | **capacity j** | | | | |
| item i ----- weight j | i j | 0 | 1 | 2 | 3 | 4 | 5 |
| F(0, j) = 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **w1 = 2, v1 = 12** | 1 | 0 | 0 | 12. | 12. | 12 | 12 |
| **w2 = 1, v2 = 10** | 2 | 0 | 10 | 12 | 22 | 22 | 22 |
| **w3 = 3, v3 = 20** | 3 | 0 | 10 | 12 | 22 | 30 | 32 |
| **w4 = 2, v4 = 15** | 4 | 0 | 10 | 15 | 25 | 30 | 37 |

(2.a) For this table, how many entries are to be computed? In general, for n items and W Knapsack capacity, how many entries are to be computed?

Answer: For the given dynamic programming table, there are 6 rows (n+1 items) and 6 columns (W+1 capacities), so there are 6 \* 6 = 36 entries to be computed.

**In general**, for n items and a capacity W, there will be **(n+1) \* (W+1)** entries to compute in the table.

(2.b) When W is extremely large compared to n, this algorithm is worse than the brute-force algorithm (2n), which simply considers all subsets. Explain this.

Answer: When W is extremely large compared to n, this dynamic programming algorithm is worse than the brute-force algorithm O(2n). This is because the dynamic programming algorithm runs in O(n \* W) time, which grows linearly with W. So if W is very large, O(n \* W) can be much worse than O(2n). **The brute-force algorithm always takes 2n operations regardless of W.**

Background Description for the problem (2.c):

To improve the efficiency, the fact that it is unnecessary to determine the entries in the ith row for every w between 1 to W. For computing F(n, W), the only entries needed in the (n-1)st row are the ones F(n-1, W) and F(n-1, W- wn).

Continue to work backward from n to determine which entries in (n-1)th row are needed. The process stops when n = 1 or w 0.

After determining which entries are needed in the ith row, decide which entries are needed in the (i-1)st row using the fact that

F(i, w) is computed from F(i-1, w) and F(i-1, w-wi).

(2.c) For the given instance data ID, given the following dynamic programming table, determine which entries are needed during the process that reaches the goal (i.e., the maximal value is F(4, 5) = 37). Mark X on those entries that are needed to get the maximal value.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| capacity jW=5 |  | F(i, 0) = 0 | **capacity j** | | | | |
| item i ----- weight j | i j | 0 | 1 | 2 | 3 | 4 | 5 |
| F(0, j) = 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **w1 = 2, v1 = 12** | 1 | 0 |  |  |  |  |  |
| **w2 = 1, v2 = 10** | 2 | 0 |  |  |  |  |  |
| **w3 = 3, v3 = 20** | 3 | 0 |  |  |  |  |  |
| **w4 = 2, v4 = 15** | 4 | 0 |  |  |  |  |  |

Answer:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| capacity jW=5 |  | F(i, 0) = 0 | **capacity j** | | | | |
| item i ----- weight j | i j | 0 | 1 | 2 | 3 | 4 | 5 |
| F(0, j) = 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **w1 = 2, v1 = 12** | 1 | 0 | X |  |  |  |  |
| **w2 = 1, v2 = 10** | 2 | 0 |  | X |  |  |  |
| **w3 = 3, v3 = 20** | 3 | 0 |  |  |  |  |  |
| **w4 = 2, v4 = 15** | 4 | 0 |  |  |  |  |  |

(2.d) How many needed entries containing nontrivial values (i.e., not those in row 0 or column 0) must be computed?

Answer:

The "X" marked entries that are needed for computing the maximal value F(4, 5) = 37 are:

F(4, 5)

F(4, 3)

F(3, 3)

F(3, 0)

These entries contain nontrivial values. Therefore, a total of four needed entries containing nontrivial values must be computed to determine the maximal value F(4, 5) = 37 in this specific instance of the knapsack problem.

**Problem 3: [40 points]**

Consider the **single-source shortest-paths problem:**

Given a vertex s (called the **source)** in a weighted connected graph G = (V, E),

The application of Dijkstra’s Algorithm finds the **shortest paths** to all its other vertices.

The single-source shortest-paths problem looks for a family of paths; each path leads from the source to a different vertex in the graph, and some paths may have edges in common.

The algorithm is ***not*** interested in *a single shortest path* that starts at the source and visits all the other vertices.

Given a connected weighted directed graph G(V, E) in Figure 1.

(3.a) Construct its weighted matrix.

Answer: From the figure, we get the following weighted matrix.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | 0 | 2 | 0 | **0** | **0** |
| B | **0** | 0 | 6 | 3 | 5 |
| C | 3 | 0 | 0 | 0 | 0 |
| D | 7 | 0 | 1 | 0 | 0 |
| E | 0 | 0 | 0 | 4 | 0 |

1. Its weight matrix

Description of the problem:

In Dijkstra’s Algorithm, it states

**for** every vertex u in V - VT that is adjacent to u\* **do {**

**if** du\* + w(u\*, u) < du

{ du ← du\* + w(u\*, u);

pu ← u\*

… } //end if

} //end for

(3.b) Complete the following two blanks.

Let the vertex A be the source.

du\* + w(u\*, u) < du

|  |  |  |
| --- | --- | --- |
| Tree vertices | Remaining vertices | Illustration |

VT = {}

V- VT = {A, B, C, D, E}

A(-, 0) min{B(A, 2)} VT = {A}

V- VT = {B, C, D, E}

B(A, 2) min{D(B, 3+2), E(B, 5+2), C(B, 6+2)} VT = {A, B}

V- VT = {C, D, E}

D(B, 5) min{C(D, 1+5), E(B, 7+5)} VT = {A, B, D}

V- VT = {C, E}

C(D, 6) min{E(B, 7)} VT = {A, B, D, C}

V- VT = {E}

E(B, 7) VT = {A, B, D, C, E}

V- VT = {}

(3.c) From the solution of (3.b), How many shortest paths from the single source, vertex A, do you obtain? List the shortest paths to all its other vertices from a given source, vertex A, with their weights.

Answer: Shortest paths from vertex A to other vertices:

Shortest path from A to B: A → B with weight 2

Shortest path from A to C: A → B → D → C with weight 6

Shortest path from A to D: A → B → D with weight 5

Shortest path from A to E: A → B → E with weight 7

(3.d) Consider the statement du\* + w(u\*, u) < du. Choose one of the Items for each of the blanks.

Items = { du\*, du , u\*, u, w(u\*, u) }.

(i) \_\_\_\_\_\_\_ u \_\_\_ A vertex in V- VT and is currently considered.

(ii) \_\_\_\_\_ u\*\_\_\_\_\_ A vertex in VT and is previously selected.

(iii) \_\_\_\_\_\_ w(u\*, u) \_\_\_\_ the weight for edges between u\* and u

(iv) \_\_\_\_\_ du\*\_\_\_\_\_ the weight of a path from a vertex in VT to the distinctive

source.

(v) \_\_\_\_\_\_ du \_\_\_\_\_ the weight of a path from a vertex in V- VT to the

distinctive source.

**Problem 4: [40 points]**

Given a connected weighted directed graph G(V, E) in Figure 1, the application of Prim’s Algorithm constructs a minimum spanning tree through a sequence of expanding subtrees.

Let Y(X, w) be the term “X reaches Y with weight w.” That means “there is a directed edge from X to Y with weight w.”

(4.a) Complete the following blank with answers written in the form Y(X, w), where X is in VT and Y is in V- VT.

Answer:

|  |  |  |
| --- | --- | --- |
| Tree vertices | Remaining vertices | Illustration |

VT = {}

V- VT = {A, B, C, D, E}

A(-, -) **B(A, 2),** C(-, ∞), D(-, ∞), VT = {}

E(-, ∞) V- VT = {B, C, D, E}

B(A, 2) **D(B, 3),** E(B, 5), C(B, 6) VT = {A, B}

V- VT = {C, D, E}

D(B, 3) **C(D,1) C(B,6) E(B,5)**  VT = {A, B, D}

V- VT = {C, E}

C(D, 1) E(B, 5) VT = {A, B, D, C}

V- VT = {E}

E(B, 5) VT = {A, B, D, C, E}

V- VT = {}

Explanation:

**Starting Point**:  
•Choose vertex A as the starting point.  
•VT={A}  
•V− VT ={B,C,D,E}  
•Candidate edges to add: B(A, 2), C(-, ∞), D(-, ∞), E(-, ∞)

**First Iteration**:  
•Add the smallest edge B(A, 2) to   
•VT ={A,B}  
•V− VT ={C,D,E}  
•Candidate edges to add: D(B, 3), E(B, 5), C(B, 6)

**Second Iteration**:  
•Add the smallest edge D(B, 3) to   
•VT ={A,B,D}  
•V− VT ={C,E}  
•Candidate edges to add from the new VT to V− VT:  
oC(D, 1) from D to C  
oE(B, 5) from B to E  
oC(B, 6) from B to C (not the minimum)

**Third Iteration**:  
•Add the smallest edge C(D, 1) to VT  
•VT ={A,B,D,C}  
•V− VT ={E}  
•Candidate edges to add from the new VT to V− VT:  
oE(B, 5) from B to E

**Fourth Iteration**:  
•Add the last edge E(B, 5) to VT  
•VT ={A,B,D,C,E}  
•V− VT ={}  
•All vertices are now included in VT, and the minimum spanning tree (directed) is   
complete.

(4.b) What is the minimum spanning tree of weighted, directed graph G(V, E) given in Figure 1? What is the total weight of this minimum spanning tree?

Answer:

The minimum spanning tree edges:

A → B with weight 2

B → D with weight 3

D → C with weight 1

D → E with weight 5

These edges form the minimum spanning tree for the given weighted, directed graph.

Total weight of the minimum spanning tree = 2 + 3 + 1 + 5 = 11

**Therefore, the minimum spanning tree of the weighted directed graph G(V, E) given in Figure 1 comprises the listed edges, and its total weight is 11.**

(4.c) Give why Prim’s Algorithm does not have to check whether to form a cycle by adding a newly found edge Y(X, w) to the immediate spanning tree.

**Answer**: Prim's algorithm maintains a single growing spanning tree. By only ever adding an edge from a vertex in the spanning tree to a vertex not in the spanning tree, it guarantees that no cycles will be formed.

The key aspect contributing to this is that Prim's Algorithm builds the minimum spanning tree iteratively by greedily selecting the minimum weight edge that connects the current tree to an adjacent vertex not yet included in the tree.

Here's why Prim's Algorithm inherently avoids forming cycles:

**Greedy Strategy**: Prim's Algorithm follows a greedy strategy by choosing the minimum weight edge at each step to expand the tree. By always selecting the minimum weight edge that connects the tree to a new vertex, it avoids forming cycles.

**Maintaining a Single Tree**: At any given step, the algorithm maintains a single tree, not a forest of disjoint trees. This ensures that it cannot form a cycle by adding an edge because it always connects a new vertex to the existing tree, preventing disconnected components.

**Maintaining Connectivity**: The algorithm guarantees that the tree remains connected by adding edges that extend the current tree without creating cycles. This property ensures that there's no way for a newly added edge to form a cycle.

Prim's Algorithm's careful selection of edges ensures that the tree remains acyclic and connected throughout the process, making explicit cycle checks unnecessary. As it systematically grows the minimum spanning tree, it inherently avoids forming cycles due to its greedy selection of edges based on minimum weights and the maintenance of a single connected tree.

(4.d) Does Prim’s algorithm always yield a minimum spanning tree?

**Answer**: Yes, Prim's algorithm always yields a minimum spanning tree (MST) when applied to a connected, weighted graph. This algorithm guarantees the creation of an MST, and it does so by consistently selecting the minimum weight edge that connects the existing tree to a new vertex.