CSE 373 A 19Wi: Final March 19 2019

Name Final Exam Answer Key

Student ID#

Instructions

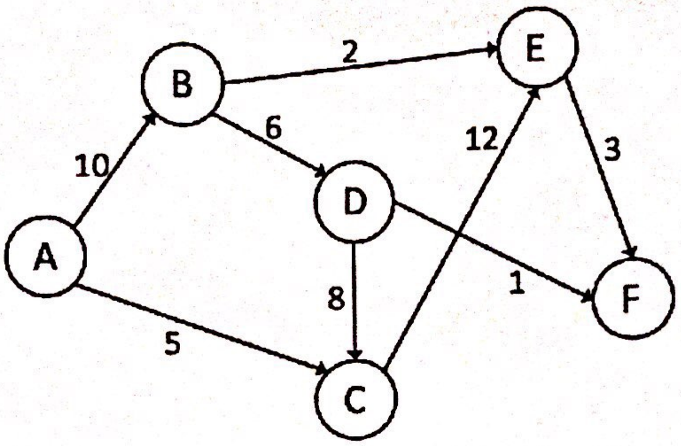
- This test is closed book, closed calculators, closed electronics.
- You are permitted a single sheet of 8.5" x 11" notes
- Do not start the exam until told to do so. You will receive a 10-point deduction if you work before the exam starts or keep working after the instructor calls for papers
- Please write your answers neatly in the provided space
- Be sure to **leave some room in the margins** as we will be scanning your answers and it's easy to miss markings too close to the edge of the paper
- Please note we will only be scanning the front of each page, so please **do not put any work you wish to be graded on the back of a page**
- If you have a question, please raise your hand to ask the course staff for clarification

Advice

- **Write down assumptions, thoughts and intermediate steps so you can get partial credit.** Clearly circle your final answer
- The questions are not necessarily in order of difficulty, skip around

1. Graph Algorithms

(A) Complete the table in *Figure 2* by running Dijkstra's algorithm on the graph in *Figure 1* using the vertex 'A' as the source.



$$\begin{tikzcd}
 & B \arrow[rr, "2"] \arrow[dr, "6"] & & E \arrow[dr, "3"] & \\
A \arrow[ur, "10"] \arrow[dr, "5"'] & & D \arrow[d, "8"] \arrow[rr, "1"] & & F \\
 & & C \arrow[uu, "12"'] & &
\end{tikzcd}$$

Figure 1

Vertex	Distance from A	Predecessor	
A	0	—	✓
B	~~∞~~ 10	A	✓
C	~~∞~~ 5	A	✓
D	~~∞~~ 16	B	✓
E	~~∞~~ ~~17~~ 12	~~C~~ B	✓
F	~~∞~~ 15	E	✓

Figure 2

(B) Given the graph in *Figure 3* construct a minimum spanning tree using Kruskal's and a disjoint set. **Make sure your implementation leverages both the path compression and union-by-rank optimizations.**

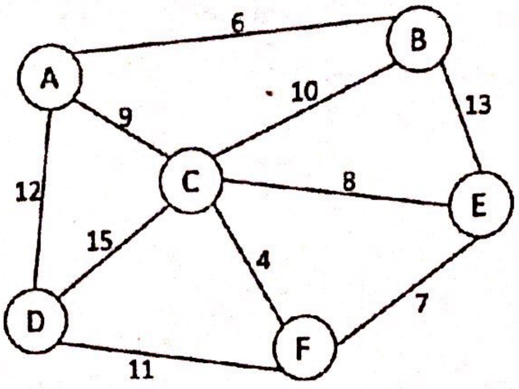
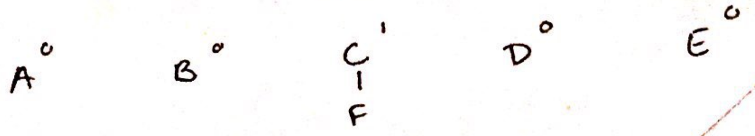


Figure 3

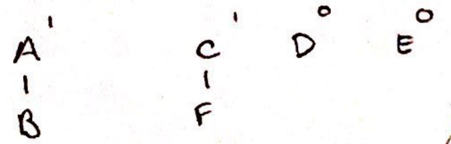
i. Draw the final internal state of the disjoint set in the form of nodes and pointers. To receive partial credit, show your work. Break ties by making the node that is alphabetically earlier the parent.

A^0 B^0 C^0 D^0 E^0 F^0

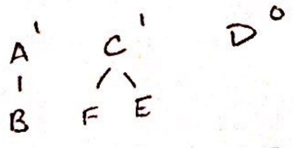
Select CF



Select AB

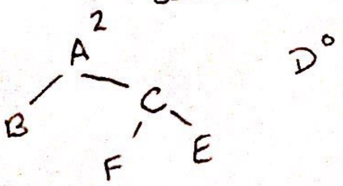


Select EF



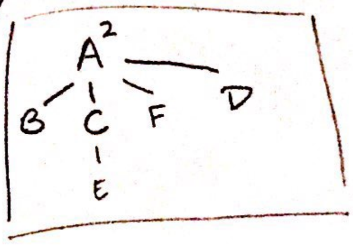
Ignore CE

Select AC



Ignore BC

Select DF → Path compression on F



ii. Based on the figure you drew in part i, fill in the array in *Figure 5* so that it represents the structure of the tree set as you did in your ArrayDisjointSet class from homework 7. Use the given hash dictionary in *Figure 4*, mapping, to establish the relationship between the index in pointers and the vertex it represents.

Vertex	A	B	C	D	E	F
Index	0	1	2	3	4	5

Figure 4 - mapping

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

Figure 5 – final internal state of disjoint set

iii. Finally, draw out the resulting MST based on your findings by adding in the missing edges to the given vertices in *Figure 6*.

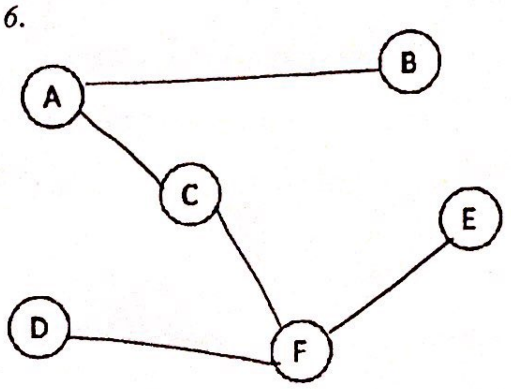


Figure 6 – final MST for graph in Figure 3

Scanned with CamScanner

2. Code Modeling

For this question consider the following main method:

```
public static void main(String[] args) {
   DoubleLinkedList<Edge> edges = m1(9);
   edges = m2(edges);
   ChainedHashMap<Vertex, ChainedHashSet<Edge>> graph = m3(edges);
}
```

(A) For the first part of this question consider the following implementation of "m1":

```
1    public static DoubleLinkedList<Edge> m1(int n) {
2       DoubleLinkedList<Edge> result = new DoubleLinkedList<Edge>();
3       int vertices = 0;
4       int edges = 0;
5       for (int i = 1; i <= 3; i++) {
6          ChainedHashSet<Integer> component = new ChainedHashSet<Integer>();
7          for (int j = i; j <= n; j += 3) {
8             component.add(j);
9             vertices++;
10         }
11         for (int v : component) {
12            for (int u : component) {
13               if (v != u) {
14                  Edge e = new Edge(new Vertex(v), new Vertex(u), (v + u) * i);
15                  edges++;
16                  result.add(e);
17               }
18            }
19         }
20      }
21      System.out.println("vertices count = " + vertices);
22      System.out.println("edges count = " + edges);
23      return result;
24   }
```

(i) What is the simplified tight O runtime in terms of n of line 8. Assume this action will never trigger resizing nor a collision.	$O(1)$	**(vi)** If given an initial n of 9, how many items will be added to "result" during the first iteration of the loop between 5 and 20?	3
(ii) What is the simplified tight O runtime in terms of n of the loop between lines 7 and 10?	$O(n)$	**(vii)** In each iteration of the loop between 5 and 20 how many items are added to result in terms of n?	$n/3$
(iii) What is the simplified tight O worst case runtime in terms of n of line 16?	$O(1)$	**(viii)** If given an initial n of 9, what would be printed on line 21?	9
(iv) What is the simplified tight O runtime in terms of n of the loop between lines 12 and 18? Assume creating a new Edge is constant runtime.	$O(n)$	**(ix)** If given an initial n of 9, what would be printed on line 22?	18
(v) What is the simplified tight O runtime in terms of n of the loop between lines 11 and 19?	$O(n^2)$	**(x)** What is the all up simplified tight O runtime of m1?	$O(n^2)$

(B) For the second part of this question consider the following implementation of "m2".

```
1    public static DoubleLinkedList<Edge> m2(DoubleLinkedList<Edge> edges) {
3        Node<Edge> sortedEnd = edges.front;
4        Node<Edge> current = edges.front.next;
5        while (current.next != null) {
6            if (current.data.getWeight() < sortedEnd.data.getWeight()) {
7                Node<Edge> sortedList = edges.front;
8                Node<Edge> itemToMove = current;
9                current = current.next;
10               current.prev = sortedEnd;
11               sortedEnd.next = current;
12               while (sortedList.data.getWeight() <= itemToMove.data.getWeight()) {
13                   sortedList = sortedList.next;
14               }
15               Node<Edge> prev = sortedList.prev;
16               itemToMove.next = sortedList;
17               itemToMove.prev = prev;
18               prev.next = itemToMove;
19               sortedList.prev = itemToMove;
20           } else {
21               sortedEnd = sortedEnd.next;
22               current = current.next;
23           }
24       }
25       return edges;
26   }
```

Assume the initial size of "edges" is n.

(i) If edges is initially in sorted order, how many times will lines 7 to 19 be executed in terms of n?	zero
(ii) If edges is initially in reverse sorted order, how many times will lines 7 to 19 be executed in terms of n?	n
(iii) During the very first iteration of the loop between lines 5 and 24, how many times will the loop between lines 12 and 14 run? (assume the test on line 6 is true)	zero
(iv) During the very last iteration of the loop between lines 5 and 24, what is the maximum number of times the loop between 12 and 14 can run? (assume the test on line 6 is true)	n - 1
(v) What is the worst-case runtime of the loop between lines 12 and 14 in terms of n?	$O(n)$
(vi) What is the all up simplified tight O runtime of m2?	$O(n^2)$
(ix) What sorting algorithm does this code most closely resemble?	Insertion Sort

(C) For the third part of this question consider the following implementation of "m3".

```
1   public static ChainedHashMap<Vertex, ChainedHashSet<Edge>> m3 (
                                                  DoubleLinkedList<Edge> edges) {
2     ChainedHashSet<Vertex> vertices = new ChainedHashSet<Vertex>();
3     for (Edge e : edges) {
4       int v1 = e.getVertex1().data;
5       if (!vertices.contains(v1)) {
6         vertices.add(v1);
7       }
8       int v2 = e.getVertex2().data;
9       if (!vertices.contains(v2)) {
10        vertices.add(v2);
11      }
12    }
13
14    ChainedHashMap<Vertex, ChainedHashSet<Edge>> graph = new
                                  ChainedHashMap<Vertex, ChainedHashSet<Edge>>();
15    for (int v : vertices) {
16      ChainedHashSet<Edge> outgoing = new ChainedHashSet<Edge>();
17      Node<Edge> back = edges.back;
18      while (back.prev != null) {
19        Edge e = back.data;
20        if (e.getVertex1().data == v) {
21          outgoing.add(e);
22        }
23        back = back.prev;
24      }
25      graph.put(new Vertex(v), outgoing);
26    }
27    return graph;
28  }
```

Assume the initial size of "edges" is n.

(i) What is the simplified tight O runtime of lines 6 and 10, assuming they do not trigger resizing nor collisions?	O(1)	**(v)** What is the simplified tight O runtime of line 21, assuming it does not trigger a resizing or collision?	O(1)
(ii) What is the simplified tight O runtime of lines 5 and 9?	O(1)	**(vi)** What is the simplified tight O runtime of the loop between lines 18 and 24? (Assume e.getVertex() runs in constant time)	O(n)
(iii) What is the simplified tight O runtime of the loop between lines 3 and 12 in terms of n? (Assume e.getVertex() runs in constant time)	O(n)	**(vii)** Assume that there are exactly n items added to the ChainedHashSet "vertices". What is the simplified tight O runtime of the loop between lines 15 and 26?	$O(n^2)$
(iv) What is the simplified tight O runtime of line 25, assuming it does not trigger a resizing or collision?	O(1)	**(viii)** What is the all up simplified tight O runtime of m3 in terms of V and E where E is the number of items contained within "edges" and V is the number of items contained within "vertices"?	$O(V \cdot E)$

3. Graph Modeling

(A) The University of Washington is working on its cold weather response procedures #snowpocalypse2019. UW has decided to invest in heated sidewalks, select pathways that will never accumulate snow or ice. However, as installing these are quite expensive UW has to be very selective with which pathways it upgrades. There is a set budget for how much UW can spend upgrading paths. The cost to upgrade a path is dependent on the length and the incline of the path. UW has also measured which paths have the most amount of foot traffic and intends to upgrade the most highly trafficked paths first.

i. Explain how you would model this scenario as a graph. Answer the following questions in bullet points with short sentences.

NOTE: Answers that did not exactly match these still had the potential for credit if they were well justified & factually correct

a. What are the vertices and the edges?

Verticies = locations/intersections on campus

Edges - sidewalks

b. What information do you store for each vertex and edge?

vertex = anything appropriate for vertex choice

Edges = length of path, incline, foot traffic

c. Is this a weighted or unweighted graph? Give a 1 sentence explanation of your response.

Weighted. Holds length & incline to determine cost. Different sidewalks have different costs

d. Is this a directed or undirected graph? Give a 1 sentence explanation of your response.

Undirected. If you upgrade a path in one direction, you're also upgrading that path in the other direction.

ii. UW needs a set of pathways that it will upgrade. This set should respect the UW's budget, so the sum of the cost to upgrade these paths should not exceed the budget. In addition, UW wants to upgrade the paths that have the highest amount of foot traffic first. The set of paths you give UW does not need to connect every single location on campus.

e. What graph algorithm might you modify to produce this set of edges for the UW?

Kruskal's

f. How would you modify this algorithm to solve this problem?
How will you take into account UW's priority to upgrade the most heavily trafficked paths?

Use traffic level to judge "cost" of each sidewalk, when ordering which edge to consider next

How will you take into account UW's budget?

Don't add an edge if it costs more than UW's remaining budget. Keep a running total of how much budget is left.

(B) Congratulations, you are a high school senior who just got accepted to the University of Washington! It's time for you to find a bit of independence from your parents, so you've decided you won't let them follow you on Instagram. Unfortunately, that is not enough distance. You also need to prevent any of their friends from following you on Instagram as well. Instagram relationships are not bi-directional; thus, you know someone is your parents' friend if both your parents and the friend all mutually follow one another.

i. Explain how you would model this scenario as a graph. Answer the following questions in bullet points with short sentences.

a. What are the vertices and the edges?

Verticies = People; instagram accounts

Edges = Following status

b. What information do you store for each vertex and edge?

Verticies = Reasonable information for choice in (a)

Edges = Follower, person followed

c. Is this a weighted or unweighted graph? Give a 1 sentence explanation of your response.

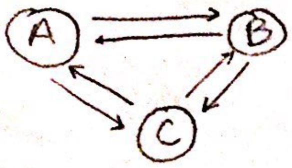
unweighted, following is binary (is / is not a follower)

d. Is this a directed or undirected graph? Give a 1 sentence explanation of your response.

Directed, you can follow someone without them following back

ii. Each time you receive a new follow request you need to first determine if they are "safe" to allow, meaning they are NOT one of your parents' friends.

a. Based on your answers to part i, draw a subset of a graph representing your two parents, "A" and "B", and an individual who is their friend "C". Your graph should include any pieces of information you specified.


$$\begin{tikzcd} A \arrow[rr, shift left] \arrow[dr, shift left] & & B \arrow[ll, shift left] \arrow[dl, shift left] \\ & C \arrow[ul, shift left] \arrow[ur, shift left] & \end{tikzcd}$$

b. Imagine the graph you described in part a is implemented with an Adjacency List, which requires a Dictionary called "graph" that maps from the vertices to the corresponding collection of outgoing edges.
Write a possible declaration of the dictionary "graph". This can use existing data types or ones of your own design. Give a short explanation of your response.

```
Dictionary<User, IList<User>> graph = new HashDictionary();
```

How would you use "graph" to tell if someone, "C" is your parents' friend (ie a follow request you will reject)? Describe in a sentence or two or use pseudo code using "A" and "B" as your parents.

```
If graph.get(A).contains(C) & graph.get(B).contains(C)
  & graph.get(C).contains(A) & graph.get(C).contains(B)
  then do not follow C ; otherwise, it's fine
```

4. Design Decisions

(A) Given each of the following scenarios choose the appropriate sorting algorithm, justify your choice with a short sentence. **Use each exactly once.**

insertion sort, selection sort, merge sort, quick sort

i. You are writing a program that sorts applicants for a prestigious fellowship. Each application is given a numerical score to provide an ordered ranking of applications. If two applications have the same score, the application that was received first will be selected, so the order in which the applications were submitted should be maintained.

Merge. - Stable
- Relatably fast

ii. You are a triage nurse and you are in charge of the queue that determines who sees the doctor first. You keep a single sorted list in a spreadsheet, and as new patients arrive you insert them into the queue based on the severity of their injury. If two patients have the same severity the patient that arrived first should get to see the doctor first.

Insertion - stable
- "Almost sorted" add case has O(n) runtime

iii. You are writing a program that sorts test scores for a class' final. The finals will be received in the order they are finished which generally means that higher scores will be received before lower scores, but not perfectly. The sorting will be used to determine the exam statistics, so the ordering of equivalent scores does not matter.

Quick sort - Not stable
- Generally fast.
- Knowing a rough ordering can help pick pivots

iv. You are a teacher and you release your students for lunch. The students know they are supposed to let younger students go first, but in their excitement, they forget and line up all out of order. Students don't like letting younger ones cut but are willing to swap line positions with a younger student if you ask them. There is not space in the cafeteria for an extra line, so you have to direct the students within the same line.

Selection sort - In place
- Swaps values

(B) Given each set of the following scenarios choose the appropriate ADT, justify your choice with a short sentence. **Use each exactly once.**

List, Stack, Queue, Dictionary, Heap

i. You are writing a program to manage a todo list with a very specific approach to tasks. This program will order tasks for someone to tackle so that the **most recent** task is addressed first. How would you store the transactions in appropriate order?

Stack. Most recent task would be the first popped off

ii. You are writing a study support program that creates flash cards where each flash card has two pieces of crucial information: the question to be asked and the correct answer. How would you store these flash card objects?

Dictionary Key = question Value = answer Use question to look up answer

iii. You are writing a program to store the history of transactions for a small business. You need to maintain the order in which the transactions occurred and be able to access entries based on the order in which they were received.

List Best at maintaining order, can get any element without manipulating data

iv. You are writing a program to surface candidates for a job. As candidates are met and evaluated, they are added to a collection from which hiring managers can request the next best applicant whenever they have an open job position. How would you store the candidates and their evaluations for hiring managers to be able to quickly get the next best applicant?

(Max) Heap where priority = goodness of candidate will return the best applicants quickly

v. You are writing a program to schedule jobs sent to a laser printer. The laser printer should process these jobs in the order in which the requests were received. How would you store the jobs in appropriate order?

Queue Processes jobs in order received - first job requested should be first job printed, which uses FIFO properties.

(C) Given each set of the following scenarios choose the appropriate dictionary implementation, justify your choice with a short sentence. **Use each exactly once.**

Array Dictionary, LinkedList Dictionary, AVL Dictionary, Hash Dictionary

i. You are writing a program to store incidents in a software service you maintain. The keys will be time stamps, so you know they will be unique. You will be adding incidents as they occur and removing them as they are resolved. You are more likely to need to access and remove incidents that have recently been added to the collection.

LinkedList. Add to front, won't need to traverse for a "put"

For your given answer, what is the average runtime to add a new incident to the collection?

O(1)

ii. You are writing a program that implements a company directory. The directory will store employee ids mapped to contact information. You will also implement auto complete, so that as someone types in a user id you will suggest for them possible entries. Your directory will need to grow and shrink as people join and leave the company.

AVL Auto complete uses all options in a subtree. Stays balanced when size changes

For your given answer, what is the average runtime to add a new employee to the directory?

O(log(n))

iii. You are writing a program to store an extremely large dictionary of English vocabulary. You will know how many entries you will store at the onset of the program and will not add any more after the initial processing. You want to optimize for quick look ups of definitions.

Hash Won't resize, very fast lookup

For your given answer, what is the average runtime to find a definition for a given word?

O(1)

iv. You are writing a program to store a rather small dictionary that maps exam questions to the average score for that question across all students. The questions are numbered sequentially starting at 0. Often you will want to read the entire set of scores in the order of the test.

Array Can quickly iterate over values

For your given answer, what is the average runtime to add a new exam question to the collection?

O(n)
↑
Resizing case

5. Multiple Choice

Clearly circle your answer

(A) If a problem is in the "NP-Complete" complexity class which of the following can we assume is also true?

a. It can be reduced to 3SAT

b. A solution to the problem could be applied to all other problems in the NP-Complete complexity class

c. A solution to the problem could be applied to all other problems in the NP complexity class

[✓ d.] All of the above

(B) Which of the following is true about the 3SAT problem?

a. It is in the P complexity class

b. It is in the NP complexity class

c. It is in the NP Complete complexity class

[✓ d.] Both b and c

(C) Which of the following best describes temporal locality?

a. The rings of memory surrounding the CPU that increase in size and decrease in speed

b. Data retained in memory based on its physical proximity to recently accessed data

[✓ c.] Data retained in memory based on how recently in time it was accessed

d. The operating systems mechanism of optimizing memory usage by punting data from inner layers of memory out to more long-term storage

(D) Which of the following best describes spatial locality?

a. The rings of memory surrounding the CPU that increase in size and decrease in speed

[✓ b.] Data retained in memory based on its physical proximity to recently accessed data

c. Data retained in memory based on how recently in time it was accessed

d. The operating systems mechanism of optimizing memory usage by punting data from inner layers of memory out to more long-term storage