COS 226 Algorithms and Data Structures Spring 2018

Second Written Exam

This exam has 7 questions (including question 0) worth a total of 80 points. You have 80 minutes. This exam is preprocessed by a computer, so please **write darkly** and **write your answers inside the designated spaces. Whenever there is a multiple choice question, fill in the appropriate bubble(s) with your pencil.**

Policies. The exam is closed book, except that you are allowed to use a one page cheatsheet (8.5-by-11 paper, two sides, in your own handwriting). No electronic devices are permitted.

Discussing this exam. Discussing the contents of this exam before solutions have been posted is a violation of the Honor Code.

This exam. Do not remove this exam from this room. Write your name, NetID, and the room in which you are taking the exam in the space below. Mark your precept number. Also, write and sign the Honor Code pledge. You may fill in this information now.

Timing. When we announce that "time is up", you may finish the sentence you are writing. You should turn in the exam to one of the instructors outside the room.

Name:

NetID:

Exam room:

Precept:

P01	P02	P02A	P03	P04	P04A	P04B	P05	P05A	P05B
◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

"I pledge my honor that I will not violate the Honor Code during this examination."

Signature

0. **Initialization. (2 points)**

In the space provided on the front of the exam, write your name, NetID, and the room in which you are taking the exam; mark your precept number; and write and sign the Honor Code pledge.

1. **Short questions: Hash Tables (9 points)**

Suppose that the following keys have corresponding hash values:

A	B	C	D	E	F	G	H	I	J	K	L
5	4	5	2	1	4	3	2	5	0	3	1

Consider the following sequence of `put` operations into an empty hash table T:

```
T.put('A'); T.put('F'); T.put('I'); T.put('J'); T.put('C'); T.put('L');
```

a) Suppose that the hash table is implemented with *separate chaining*, with a table of size $m = 6$, how many elements does the longest chain have after the operations above?

○ 0 ○ 1 ○ 2 ● 3 ○ 4

b) Suppose that the hash table is implemented with *linear probing*, with a table of size $m = 6$ and **no resizing**. What will be the key corresponding to $keys[1]$ in the array of keys?

○ 'A' ○ 'F' ○ 'I' ● 'J' ○ 'C' ○ 'L' ○ null

c) Suppose that the hash table is implemented with *linear probing*, with a table of size $m = 20$ and **no resizing**. What will be the key corresponding to $keys[1]$ in the array of keys?

○ 'A' ○ 'F' ○ 'I' ○ 'J' ○ 'C' ● 'L' ○ null

2. **Short questions: graphs (12 points)**

a) What is the order of growth of the memory used to represent a graph with V vertices and E edges using the adjacency-lists representation and the adjacency-matrix representation?

Memory proportional to:	V	E	$E+V$	V^2	$E \cdot V$
adjacency-lists	○	○	●	○	○
adjacency-matrix	○	○	○	●	○

b) Suppose that an undirected graph G represented using adjacency-lists. Not counting initialization time, depth-first search marks all vertices connected to s in time proportional to:

○ The number of vertices in G.
○ The number of edges in G.
○ The number of vertices in the connected component containing s.
● The sum of the degrees of the vertices in the connected component containing s.

c) Suppose that a **digraph** G is represented using the *adjacency-lists* representation. What is the order of growth of the running time to find all vertices that point to a given vertex v?

○ indegree(v) ○ outdegree(v) ○ V ○ V+indegree(v) ● $V+E$

d) Suppose that during an execution of depth-first search in a digraph G, dfs(v) is called as a recursive subcall after dfs(w) is called, but before dfs(w) returns. Which of the following **must** be true of the graph G? (Select all that apply).

○ There exists a directed path from v to w.
● There exists a directed path from w to v.
○ There does not exist a directed path from v to w.
○ There exists a directed cycle containing both v and w.

3. **Short questions: Graph Algorithms. (21 points)**

a) Let $e = v \to w$ be an edge with weight 7. Suppose that during the generic shortest paths algorithm, distTo[v] = 16 and distTo[w] = 25. What will the pair (distTo[v], distTo[w]) be after calling relax(e)?

○ $(16, 16)$ ● $(16, 23)$ ○ $(16, 25)$ ○ $(18, 23)$ ○ $(18, 25)$ ○ $(25, 25)$

b) What is the order of growth of Dijkstra's algorithm running time if we don't store distances in an Index PQ, but just keep the array of distTo[v]? Assume the graph has V vertices and E edges, and there are no self-edges or parallel edges.

○ $V + E$ ○ $E \log V$ ● V^2 ○ $V \cdot E$ ○ V^3 ○ $V^2 \cdot E$

c) Let N be a network, denote by F_{max} the value of the max st-flow in N. Let f be any st-flow in N. Let (A, B) be any st-cut. Let $C_{A,B}$ be the capacity of the cut (A, B). Let $F_{A,B}$ be the net flow in f across (A, B), and let F_f be the value of f. Which of the following must be true? (Select all that apply):

☑ $F_f \geq F_{A,B}$ ☑ $F_f \leq F_{A,B}$ ☑ $C_{A,B} \geq F_{max}$
☐ $C_{A,B} \leq F_{max}$ ☐ $F_f \geq F_{max}$ ☑ $F_f \leq F_{max}$

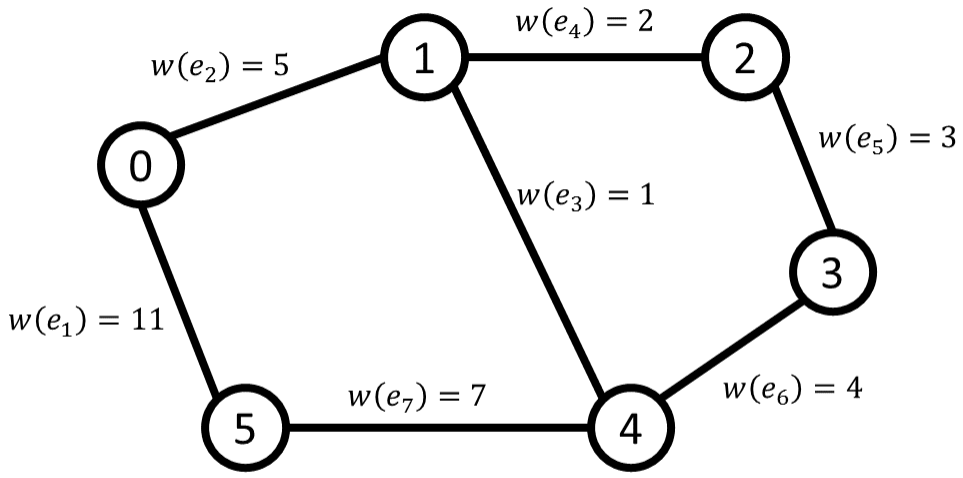
Note: $X = Y$ if and only if $X \geq Y$ and $X \leq Y$.

d) Consider a simple version of Bellman-Ford algorithm where we initialize distTo[s] to 0 and the rest of distTo[v] to $+\infty$. Then fix an order on all edges $e_1, \ldots, e_E$ and relax them in this order V times. If there are no negative cycles, we know this procedure will result in the array distTo[v] containing the distance from s to v for each v. Suppose we perform no actions in addition to the above, and the graph *does* have a negative cycle accessible from s. Which of the following must be true? (Select all that apply):

☐ At the end of the process, at least one of the distTo[v] values will be negative.
☐ At the end of the process, distTo[s] must be negative.
☐ The execution as described above will crash.
☑ None of the above.

If we continue the process indefinitely, one of the distTo[v]'s will eventually become negative, but this may take longer than V rounds.

The following three questions refer to the following weighted graph:



e) Suppose we run Kruskal's MST algorithm on the graph above. What will be the order in which edges are *added* to the MST?

◯ $(e_3, e_4, e_5, e_6, e_2, e_7)$ ◯ $(e_3, e_4, e_5, e_6, e_2)$ ● $(e_3, e_4, e_5, e_2, e_7)$
◯ $(e_2, e_3, e_4, e_5, e_7)$ ◯ $(e_2, e_3, e_4, e_5, e_7)$ ◯ None of the above

f) If we run Dijkstra's algorithm with $s = 0$, in which order will the vertices be deleted from the PQ?

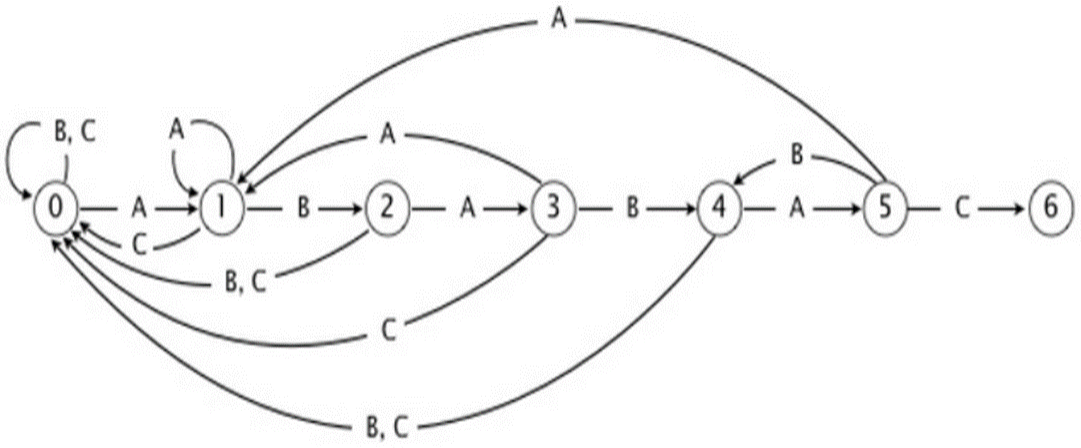
◯ $(0, 1, 2, 3, 4, 5)$ ◯ $(1, 4, 2, 3, 5)$ ● $(0, 1, 4, 2, 3, 5)$
◯ $(0, 1, 5, 4, 2, 3)$ ◯ $(1, 5, 4, 2, 3)$ ◯ None of the above

g) Suppose we treat the graph above as a digraph with a directed edge in both directions (so, for example, there is a directed edge of weight 7 from $4 \to 5$ and from $5 \to 4$. Suppose we run Bellman Ford from $s = 0$ by repeatedly relaxing the edges $(e_1, e_2, e_3, e_4, e_5, e_6, e_7)$ (by relaxing e_7 we mean relaxing both $4 \to 5$ and $5 \to 4$ in some order). How many times will distTo[5] get updated?

◯ 0 ● 1 ◯ 2 ◯ 3 ◯ 4

4. **DFAs (4 points)**

a) What state is the DFA below in after simulating it on the input string ABC AAB ABA BAB (the spaces are for readability only)?



○ 0 ○ 1 ○ 2 ○ 3 ● 4 ○ 5 ○ 6

b) Given below is a **partially** completed DFA, over the alphabet $\{A, B\}$, generated by Knuth-Morris-Pratt string search algorithm for a pattern of length 5. What pattern could be represented by this DFA?

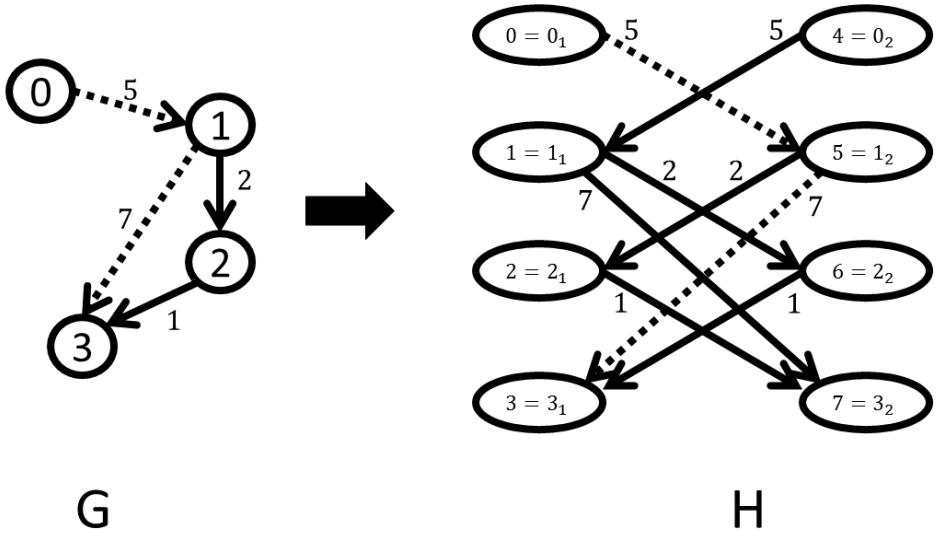
	0	1	2	3	4
A			3		
B				2	5

○ A B A B A ○ A A B B A ○ B A B A A ● A B A A B
○ none of the above is compatible with the table
○ more than one of the above is compatible with the table

5. **Even/odd paths. (20 points)**

The goal of this problem is to calculate the length of the shortest path with *an even number of links* from a node s in a digraph. One way of doing this is by creating a graph H with two copies of the graph G, so that each vertex a becomes two vertices a_1 and a_2, and cross-linking them so that the edge $a \to b$ of weight w becomes edges $a_1 \to b_2$ and $a_2 \to b_1$, each of weight w. Then a path from a_1 to b_1 in H corresponds to a path from a to b in G with an even number of links.

In the picture below, the shortest path from 0 to 3 in H is of weight 12, corresponding to the shortest path from 0 to 3 in G with an even number of links. Even-though there is a shorter odd-length path in G, it corresponds to a path from 0 to 7 in H and not from 0 to 3.



In the space below, complete the method for constructing H from G by filling in the blanks (we omit input validation for succinctness). **Fill in the blanks in the following code:**

```
public static EdgeWeightedDigraph doubleGraph(EdgeWeightedDigraph G)
{
 H = new EdgeWeightedDigraph (2*G.V());
for (v=0; v<G.V(); v++){
for (DirectedEdge e : G.adj(v)) {

     H.addEdge(new DirectedEdge( e.from()       , e.to()+G.V() , e.weight() ));

     H.addEdge(new DirectedEdge( e.from()+G.V() , e.to()       , e.weight() ));

} }
return H; }
```

The `DirectedEdge` and `DijkstraSP` APIs are listed for your reference below. You may use the remainder of the next page as scrap paper (it won't be graded).

```
//Initializes a directed edge from vertex v to vertex w with the given weight
public DirectedEdge(int v, int w, double weight);
public int from(); //returns source
public int to(); //returns destination
public double weight(); //returns weight
```

```
//Computes a shortest-paths tree from the source vertex s to every other
//vertex in the edge-weighted digraph G
public DijkstraSP(EdgeWeightedDigraph G, int s);
public double distTo(int v); // length of a shortest path from s to v
public boolean hasPathTo(int v); // true if there is a path from s to v
public Iterable<DirectedEdge> pathTo(int v); // Returns a shortest path from s to v
```

In the box below, show how you would implement
`double ShortestEvenDistance(EdgeWeightedDigraph G, int s, int t)`
which returns the length of the shortest path with an even number of links from s to t in an edge-weighted digraph G. You may assume that G has no negative-weight edges.

Use pseudocode (and **not** prose) to answer this part. For full marks, your algorithm should run in time proportional to $E \log V$ on a G with E edges and V vertices.

```
EdgeWeightedDigraph H = doubleGraph(G);

DijkstrSP spH = new DijkstraSP (H, s);

if (! spH.hasPathTo(t) )
     return Double.POSITIVE_INFINITY;

return spH.distTo(t);
```

In the box below, show how you would implement
`boolean ShortestPathEven(EdgeWeightedDigraph G, int s, int t)`
which returns `true` if there is a shortest path from s to t in an edge-weighted digraph G that has an even number of links, and `false` otherwise. You may assume that G has no negative-weight edges.

Use pseudocode (and **not** prose) to answer this part. For full marks, your algorithm should run in time proportional to $E \log V$ on a G with E edges and V vertices.

The hardest part of the question was to parse it correctly. The question asks to return true if (1) there is a shortest path from s to t in G; (2) among the (possibly multiple) shortest paths from s to t in G one is even.

```
EdgeWeightedDigraph H = doubleGraph(G);

DijkstrSP spH = new DijkstraSP (H, s);

if (! spH.hasPathTo(t) )
     return false; // no even path from s to t

if (! spH.hasPathTo(t+G.V()) )
     return true; // there is even path but no odd path from s to t

return ( spH.distTo(t) <= spH.distTo(t+G.V()) );
// returns true if the even shortest path is shorter or equal to
// the odd shortest path
```

6. **Substring lookup (12 points)**

Given a string txt of length N design a data structure that allows one to quickly test whether a given string s of length $m \ll N$ is a substring of txt. The main goal (which you need to accomplish for full marks) is to be able to make queries in time proportional to m. The length m of s is unknown in advance, and the data structure should support queries of strings with variable lengths.

Your answer will be graded on clarity and efficiency. The primary objective is to support queries in $O(m)$ time, but the time to construct the data structure should also be as efficient as possible.

a) Design a data structure and explain how to construct it from txt.

Use pseudocode or concise, but precise, English prose, or a combination of both.

A data structure where lookup time is the chief concern (at the expense of space), is typically a hash table, or (in case of strings) a trie. Both types of solutions are possible, although in this case a trie is more natural, since one would need to be quite careful to construct a hash table in time proportional to N^2 and not N^3.

Construct an R-way trie T.
Insert the N suffixes $txt[i..N]$ of txt into T.
Note that a substring of txt must be a prefix of some suffix of txt.
Specifically, the substring $txt[i..j]$ is a prefix of the suffix $txt[i..N]$.

b) What's the order of growth (as a function of N) of the amount of time and space that your data structure requires?

It is ok to drop the dependence on R since it has been implicitly assumed to be a constant.

Time: $O(N^2R)$ **Space:** $O(N^2R)$

c) Given a string s of length m, how does the lookup procedure to test whether m appears as a substring of txt work?

Use pseudocode or concise, but precise, English prose, or a combination of both.

Search for s in the trie T from part (a) by following its symbols (at most m steps).

If we hit a `null` node before s is over, it means that s is not a substring of txt, and we return `false`.

Otherwise, s is a prefix of some suffix of txt and we return `true`.