

# NATIONAL UNIVERSITY OF SINGAPORE

## CS2020 - Data Structures and Algorithms (Accelerated)

(Semester 2 AY2016/17)

Time Allowed: 2 Hour

### Instructions

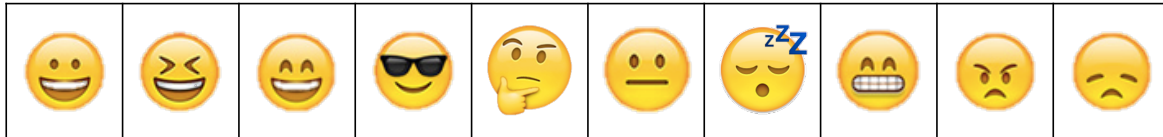
- Write your Student Number below, and on every page. Do not write your name.
- The assessment contains 7 multi-part problems. You have 120 minutes to earn 100 points.
- The assessment contains 28 pages, including the cover page and 5 pages of scratch paper.
- The assessment is closed book. You may bring two double-sided sheet of A4 paper to the assessment. You may not use a calculator, your mobile phone, or any other electronic device.
- Write your solutions in the space provided. If you need more space, please use the scratch paper.
- Show your work. Partial credit will be given. Please be neat.
- You may use (unchanged) any algorithm or data structure that we have studied in class, without restating it. If you want to modify the algorithm, you must explain exactly how your version works.
- Draw pictures frequently to illustrate your ideas.
- Good luck!

Student Number.: \_\_\_\_\_

Problem #	Name	Possible Points	Achieved Points
1	True/False	10	
2	Drawing Pictures	18	
3	Binary Search Redux	11	
4	The Last Trip	21	
5	Shortest Paths	20	
6	Levelling Up	20	
<b>Total:</b>		100	

**Problem 0. Before we begin.** [0 points]

Circle the image that best represents how you feel right now.



**Problem 1. True/False** [10 points]

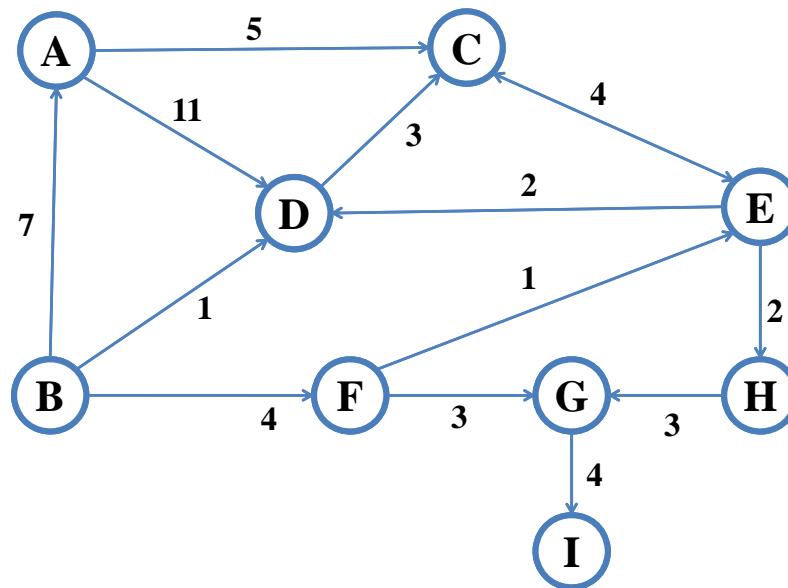
For each of the following statements, select **True** if the statement is always true, and select **False** otherwise. All graphs are connected and contain at least one node and at least one edge.

	True	False
<p><math>G</math> is a weighted, undirected graph, and <math>u</math> is a node in the graph. Edge <math>(u, v)</math> is the edge adjacent to <math>u</math> with the smallest weight. Then some minimum spanning tree of <math>G</math> contains edge <math>(u, v)</math>.</p>	<input type="checkbox"/>	<input type="checkbox"/>
<p>You are given a hash table that resolves collisions with chaining. The table is of size <math>m</math> and contains <math>n</math> elements. Assume that the hash function satisfies the Simple Uniform Hashing Assumption. Then the expected time for a search operation is <math>O(1)</math>.</p>	<input type="checkbox"/>	<input type="checkbox"/>
<p>We will call the inner loop of Bellman-Ford (where each edge in the graph is relaxed exactly once) an “iteration.” Assume you run Bellman-Ford on a graph <math>G</math> with no negative weight cycles. Then at the end of <math>i</math> iterations, every node within <math>i</math> hops of the source has an estimate equal to its correct distance from the source.</p>	<input type="checkbox"/>	<input type="checkbox"/>
<p>Given an array sorted from smallest to largest, we can build a min-heap (i.e., a heap where the smallest element is at the root) in time <math>\leq O(\log n)</math>.</p>	<input type="checkbox"/>	<input type="checkbox"/>

	True	False
Assume you have a sorted array containing $n$ items. Ten arbitrary items from the array are removed and replaced with ten new items. Then running InsertionSort on the resulting array will take $O(n)$ time.	<input type="checkbox"/>	<input type="checkbox"/>
Assume you have a graph with no positive weight cycles. Then you can find the longest path in the graph by negating all the edge weights and running Dijkstra's algorithm.	<input type="checkbox"/>	<input type="checkbox"/>
Assume you have an AVL tree, and let $u$ be a node in the AVL tree of height $h$ . Then, after every operation completes, the subtree rooted at node $u$ contains at most $2^{h+1}$ nodes.	<input type="checkbox"/>	<input type="checkbox"/>
Every directed acyclic graph has <i>exactly</i> one valid topological ordering.	<input type="checkbox"/>	<input type="checkbox"/>
Given a set of 2D points $P$ , the boundary of the convex hull of $P$ always includes a point $p \in P$ with the maximum $x$ -coordinate.	<input type="checkbox"/>	<input type="checkbox"/>
Given a complete graph $K_n$ with $n$ nodes of 2D positions with $\binom{n}{2}$ edges, if the weight of each edge is the distance between the two endpoints, we can compute the minimal spanning tree of $K_n$ in $O(n \log n)$ expected time.	<input type="checkbox"/>	<input type="checkbox"/>

**Problem 2.** Drawing Pictures [18 points]

**Problem 2.a.** Find a topological ordering of the directed acyclic graph below using depth-first search (DFS). In the table below, for each node, fill in the order in which the DFS visits the node, and its position in the topological order. Assume the DFS begins at node  $A$ . *When there is more than one possible choice of which node to visit next, assume the DFS chooses the node with the earliest alphabetic id (e.g.,  $X$  before  $Y$  before  $Z$ ).*



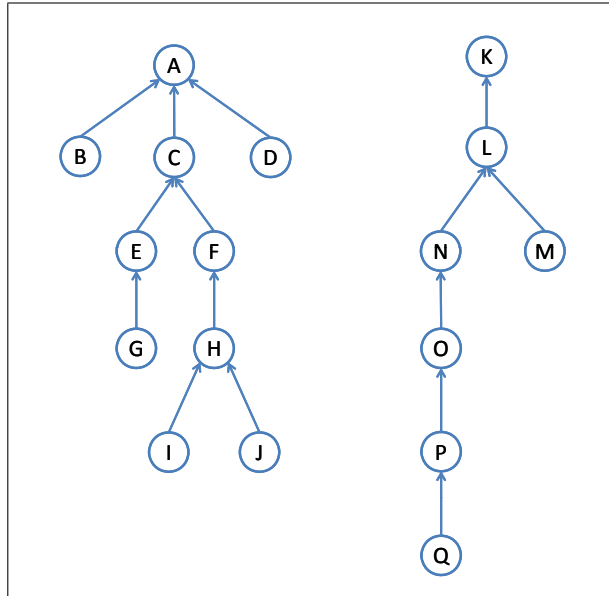
Visit order: (Please include every time a node is seen on the traversal, *not just the first time*.)

[illegible]

Topological order:

--	--	--	--	--	--	--	--	--

**Problem 2.b.** Below is a state of the Union-Find algorithm (for implementing the Disjoint Sets ADT), with 17 elements in two disjoint sets. Assume that Union-Find is implemented with Quick-Union and path compression. (The below state is actually impossible, but assume that it is really the state.)

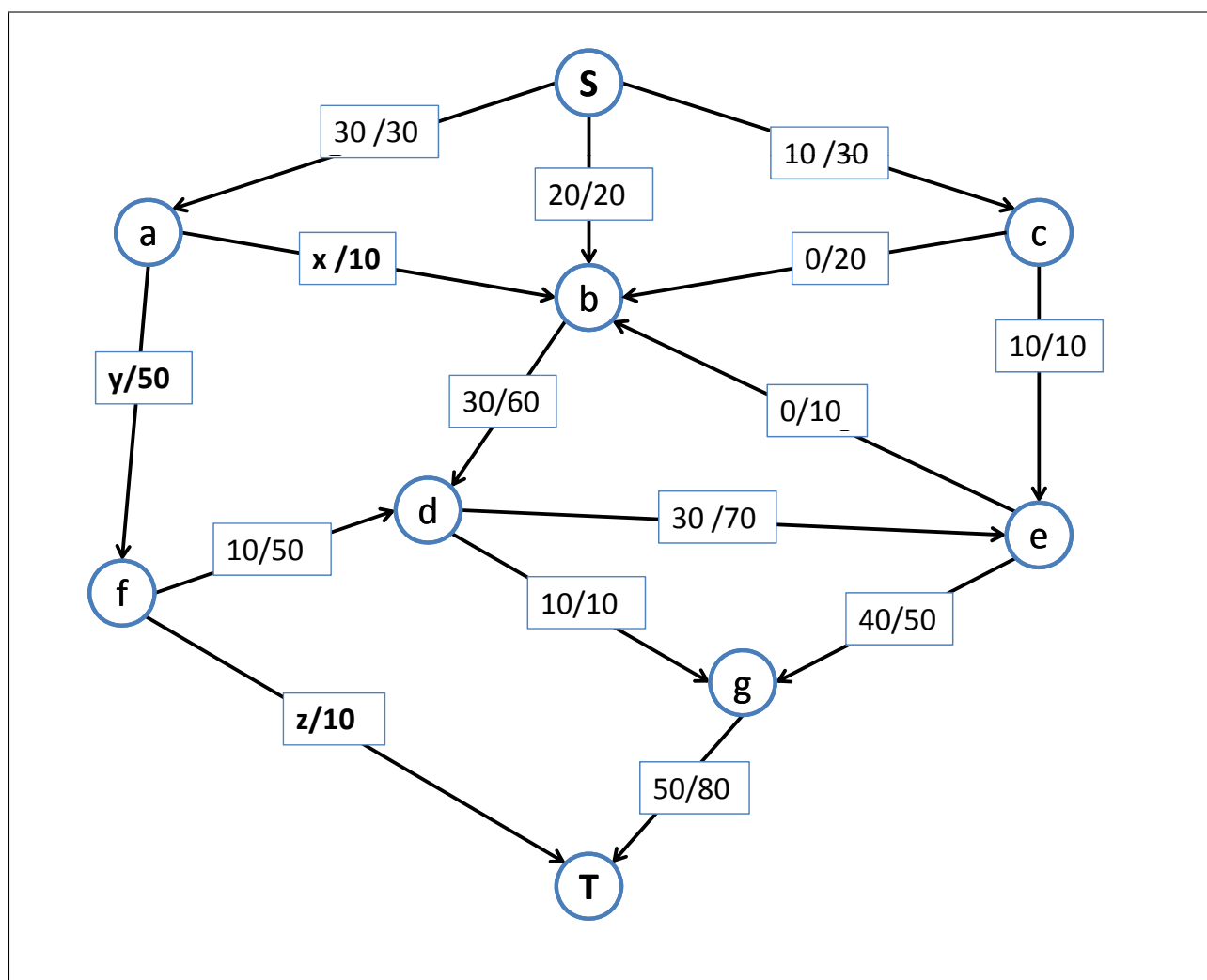


The program executes a `connected(K,H)` operation that determines whether  $K$  and  $H$  are connected using the Union-Find algorithm. (In this case, it should return the boolean **false**.) Draw (neatly) the state of the algorithm after the operation completes.



**Problem 2.c.** Below is given a flow network with a source  $S$  and a target  $T$ . Each edge has a capacity, and each edge is assigned a non-negative flow. The label  $a/b$  for an edge indicates that the edge has flow  $a$  and capacity  $b$ . (For three of the edges, the flow is not given and is shown only as a variable  $x$ ,  $y$ , or  $z$ .) The given flow respects all the requirements for a legal  $st$ -flow.

Please answer all the questions on the next page regarding this flow network.



**Question regarding the flow network on the previous page:**

Notice that for three of the edges  $((a, b), (a, f), (f, t))$  the flow is unspecified. Use the rest of the provided information to deduce the values of the flow on these edges:

---

$$(a, b) : \quad x \quad = \quad \boxed{\phantom{0000}}$$

$$(a, f) : \quad y \quad = \quad \boxed{\phantom{0000}}$$

$$(f, T) \quad z \quad = \quad \boxed{\phantom{0000}}$$

---

Give an  $st$ -augmenting path for the above flow network:

S									
---	--	--	--	--	--	--	--	--	--

What is the bottleneck capacity of the given augmenting path?

--

**Problem 3.** Binary Search Redux [11 points]

Software developer Espy Dittup has an idea for how to make binary search run even faster. A traditional binary search queries the middle element in an array, and then either recurses on the left or right half of the array. Espy wonders what will happen if we divide the array into *more* than two pieces. Espy proposes the following algorithm that divides the array into  $k$  pieces and searches for a value  $v$ :

- If the array has fewer than  $k$  elements, then do a linear search to look for the value  $v$ .
- Otherwise, choose  $k$  elements spaced evenly apart in the array. Call these elements set  $S$ .
- If any of the items in set  $S$  is equal to the value  $v$  you are looking for, then return true.
- Otherwise, let  $i$  be the index of the largest item in  $S$  that is  $< v$ , and let  $j$  be the index of the smallest item in  $S$  that is  $> v$ .
- Recursively search in the subarray  $[i + 1, j - 1]$ .

We will refer to this new algorithm as a  $k$ -ary Search.

**Problem continued on next page.**

Espy implements the  $k$ -ary search as follows:

```
1. static boolean karySearch(int v, int[] A, int begin, int end, int k){
2.     int size = (end - begin + 1);
3.     if (size < k) {
4.         for (int i=begin; i<=end; i++) {
5.             if (A[i] == v) return true;
6.         }
7.         return false;
8.     }
9.     int lastProbe = 0;
10.    // Loop through probe points
11.    for (int i=1; i<k; i++) {
12.        int currentProbe = begin + (size*i)/k;
13.        if (A[currentProbe] == v) return true;
14.        else if (A[currentProbe] > v) {
15.            return karySearch(v, A, lastProbe, currentProbe, k);
16.        }
17.        lastProbe = currentProbe;
18.    }
19.
20.    return karySearch(v, A, lastProbe, end, k);
21. }
```

The code is designed to sort an array of integers, and it is designed to be used in the following manner, e.g., to search for the value 17 (where  $k = 5$ ):

```
int[] A = new int[1000];
for (int i=0; i<1000; i++) A[i] = 2*i;
karySearch(17, A, 0, A.length-1, 5);
```

**Problem 3.a.** Unfortunately there is a bug in the code that will prevent it from working correctly for at least one value of  $k$ . Which of the following is a problem with this code?

- ☐ There is a compilation error.
- ☐ The program crashes (e.g., due to an invalid array access).
- ☐ The program does not terminate.
- ☐ The program returns an incorrect answer.

**Problem 3.b.**

Explain (briefly) what the bug is and how to fix it.

**Problem 3.c.** Let  $T(n, k)$  be the running time of the  $k$ -ary search on an array of  $n$  elements. What is the recurrence that describes the performance of this algorithm?

$$T(n, k) =$$

**Problem 3.d.** What is the solution to the recurrence in the previous part? (Give your answer in terms of  $n$  and  $k$ .)

$$T(n) =$$

**Problem 4.** The Last Trip [21 points]

**Problem 4.a.** Explain briefly (in one to two sentences) the differences between the `Iterator` interface and the `Iterable` interface. When do you use one and when do you use the other?

**Problem 4.b.** Explain briefly (in one to two sentences) the differences between the `HashMap` class and the `TreeMap` class. When do you use one and when do you use the other? What are the trade-offs?

**Problem 4.c.** Explain briefly (in one to two sentences) what is the `Comparable` interface and why would you use it?

```
1.  public class LastTrip AAA ITravel {
2.      static int count = 0;
3.      LastTrip nextTrip = null; // Next trip that we will all take together!
4.      String name;
5.      ArrayList<BBB> listOfStuff;
6.
7.      LastTrip(String n, BBB x, BBB y, LastTrip next){
8.          listOfStuff = new CCC; // create new ArrayList
9.          listOfStuff.add(x); listOfStuff.add(y);
10.         nextTrip = next;
11.         name = n;
12.     }
13.
14.     @Override // findDistance is part of the ITravel interface
15.     DDD int findDistance(){
16.         count++;
17.         int distance = 0;
18.         for (int i=0; i<listOfStuff.size(); i++) {
19.             distance += listOfStuff.get(i);
20.         }
21.         return distance;
22.     }
23.
24.     public void printTripReport(){
25.         if (nextTrip != null) {
26.             nextTrip.printTripReport();
27.         }
28.         System.out.println(findDistance());
29.     }
30.
31.     EEE LastTrip updateTrip(LastTrip trip, LastTrip newNext) {
32.         LastTrip temp = trip;
33.         trip = new LastTrip("empty", 0, 0, null);
34.         temp.nextTrip = newNext;
35.         return temp;
36.     }
37.
38.     public static void main(String[] args){
39.         LastTrip a = new LastTrip("a", 7, 10, null);
40.         LastTrip b = new LastTrip("b", 12, 15, a);
41.         LastTrip c = new LastTrip("c", 22, 32, a);
42.
43.         LastTrip d = updateTrip(a, c);
44.         c.nextTrip = b;
45.         d.nextTrip = null;
46.
47.         c.printTripReport();
48.     }
49. }
```

**Problem 4.d.** In the `LastTrip` class on the previous page, important parts of the code have been replaced with the strings AAA, BBB, CCC, DDD, and EEE. Fill in below proper Java code to replace these placeholders.

Instructions	Line(s)	Placeholder	Answer
<code>ITravel</code> is an interface.	1	AAA =	<input type="text"/>
	5, 7	BBB =	<input type="text"/>
	8	CCC =	<input type="text"/>
Use some combination of: <code>public</code> , <code>private</code> , and <code>static</code>	15	DDD =	<input type="text"/>
Use either <code>public</code> or <code>public static</code>	31	EEE =	<input type="text"/>

**Problem 4.e.** On executing the main method, at each of the indicated lines of the program, indicate the **name** of the trip and the name of the **nextTrip** (or null, if there is no nextTrip).

Line	Variable	.name	.nextTrip.name
line 42	a	<input type="text"/>	<input type="text"/>
line 42	b	<input type="text"/>	<input type="text"/>
line 42	c	<input type="text"/>	<input type="text"/>
line 46	a	<input type="text"/>	<input type="text"/>
line 46	b	<input type="text"/>	<input type="text"/>
line 46	c	<input type="text"/>	<input type="text"/>
line 46	d	<input type="text"/>	<input type="text"/>

**Problem 4.f.** What is printed out when the program completes? (You may not need all the lines provided.)

**Output line 1 :**

**Output line 2 :**

**Output line 3 :**

**Output line 4 :**

**Output line 5 :**

**Problem 5.** Shortest Paths [20 points]

You are given a weighted, connected, and undirected graph  $G = (V, E)$  and a source  $S$ . For each edge  $(u, v)$ , let  $w(u, v)$  be the weight of the edge (i.e., the distance from  $u$  to  $v$ ). Assume that all the edge weights are  $> 0$ . You can find some examples at the page after next.

**Problem 5.a.** Each node  $u$  in the graph is labelled with a value  $d_u$ . The graph expert, Prof. D. S. Tance, claims that these values represent the shortest distance from  $S$  to the node. For example, if node  $u$  has label  $d_u$ , then the shortest path from  $S$  to  $u$  had length  $d_u$ . You want to decide whether Tance is correct.

Give the most efficient algorithm to decide whether the labels are correct, i.e., return **true** if and only if the labels represent the lengths of the shortest paths from  $S$ :

Assume the graph has  $n$  nodes and  $m$  edges. What is the running time of your algorithm:

Time :

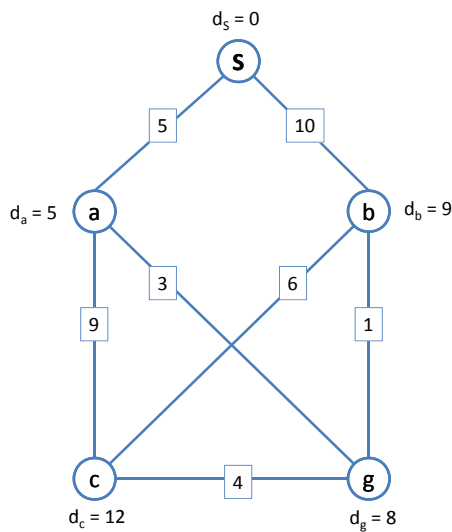
**Problem 5.b.** Prof. D. S. Tance loves finding shortest paths, and has now given each node  $u$  a label representing the shortest distance from *every* node in the graph. Each node  $u$  in the graph has a label  $d_u[v]$  representing the shortest path from  $v$  to  $u$ , for every node  $v$ . Assume that these labels are correct. (You may assume that Tance calculated these labels using an All-Pairs-Shortest-Path algorithm.)

Some of the edges in the graph are useless, in that they do not contribute to *any* shortest path. For every edge  $(u, v)$ , we say that it is *useless* if it is not part of a shortest path from some node  $x$  to another node  $y$ . For simplicity, assume that for every pair of nodes  $x, y$  there is a single unique shortest path between  $x$  and  $y$ .

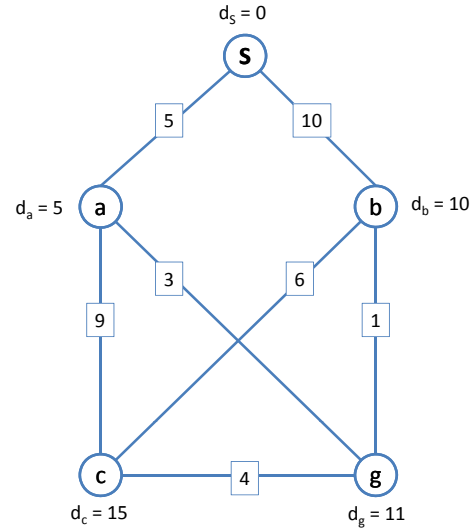
Give the most efficient algorithm you can to find *all* the useless edges in the graph:

Assume the graph has  $n$  nodes and  $m$  edges. What is the running time of your algorithm:

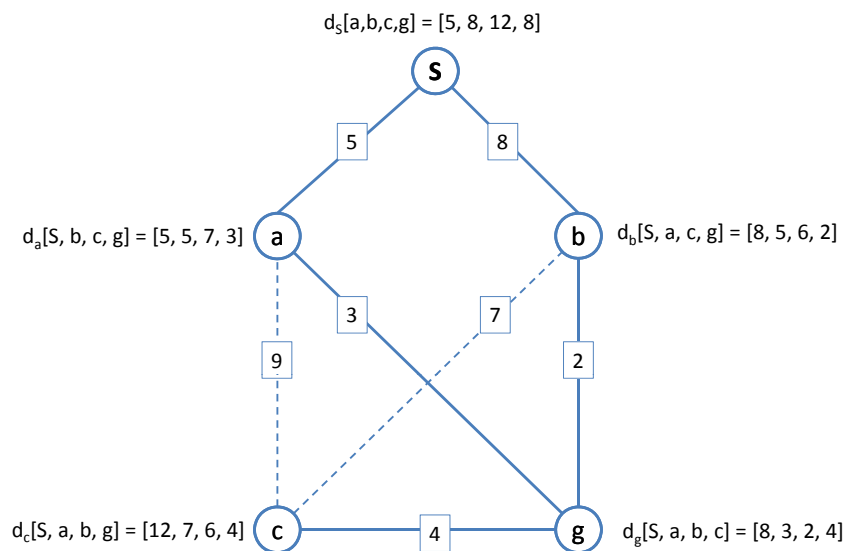
Time :

**Examples for Problem 6:**

Part a: Each of the distances from the source  $S$  is correctly labelled. For example, the distance from  $S$  to node  $b$  is 9 (via the path  $(S, a, g, b)$ ). For part (a), the algorithm should return true.



Part a: The distances from the source  $S$  are not correctly labelled. For example, the distance from  $S$  to node  $b$  is 9 (via the path  $(S, a, g, b)$ ), not 10 (as labelled). For part (a), the algorithm should return false.



Part b: The solid edges are useful, while the dotted edges are useless. For example, the edge  $(S, a)$  is useful, because it is on the shortest path from  $S$  to  $a$ . The edge  $(a, c)$  is not useful because it is not part of any shortest path. For part (b), the algorithm should return the edges  $(a, c)$  and  $(b, c)$ .

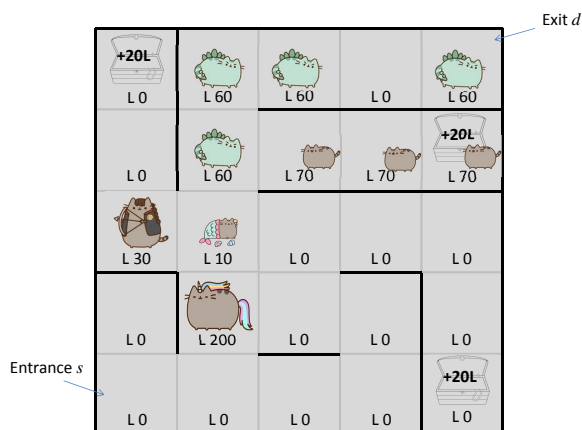
**Problem 6.** Levelling Up [20 points]

The *CS2020: The Last Voyage* is my favorite game. In this game, we have a maze consisting of rooms. We start at the entrance to the maze and the goal is to reach the exit. However, not everyone can enter each room in the maze. Each player has a *level*, and each room has a minimum level requirement: you cannot enter the room unless you have reached the minimum level.

Assume that the maze is given to you in the form of an undirected, connected graph  $G = (V, E)$  where the nodes represent rooms and the edges represent doors between the rooms. Assume there are  $n$  nodes and  $m$  edges. For each node  $u$ , you are given  $level(u)$ , i.e., the minimum required level for that room. There is a designated entrance  $s$  and a designated exit  $d$ . Assume the graph is provided as an adjacency list.

Inside the maze, there are a few special relics that will raise your level for the remainder of the maze. When you use a relic, it will add  $R$  levels to your current level. This will enable you to enter rooms that are many levels higher than would otherwise have been possible. Assume that there are at most  $k$  such relics and all of them raise your level by  $R$ . (Do not assume that  $k$  is necessarily small.) While inside the maze, you can only use one relic during the entire game because once you have found and used one relic, the others will be vaporized.

Your goal in this problem is to give an algorithm to find the minimum level a player needs on entering the maze in order to reach the exit. The running time of your algorithm should depend only on  $n$ ,  $m$ , and  $k$ . It should *not* depend on  $R$  or the maximum level number.



In this example,  $R = 20$ . You need to have initial level 40 in order to exit the maze. This is sufficient to reach either the relic in the bottom right corner or the top left corner, which boosts you to level 60, enabling you to exit the maze.

**Problem 6.a.** Explain, succinctly in **two to three sentences**, the main idea of how you plan to solve the problem.

**Problem 6.b.** Draw a picture that illustrates your idea.

**Problem 6.c.** Assume that the graph has  $n$  nodes,  $m$  edges, and  $k$  relics. What is the time complexity of your algorithm? Use asymptotic (big-O) notation.

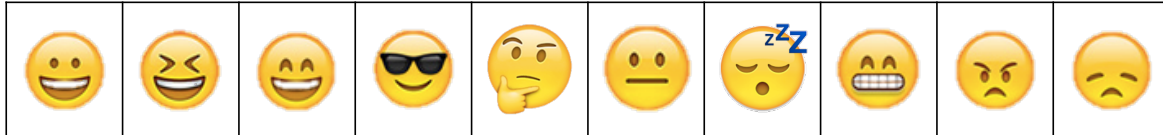
---

Time complexity :

**Problem 6.d.** Give the complete details of your algorithm. Explain why you achieve the performance claimed in the previous part.

**Problem 7.** After you are done. [0 points]

Circle the image that best represents how you feel right now.



## **Scratch Paper**

## **Scratch Paper**

## **Scratch Paper**

## **Scratch Paper**

**Scratch Paper**

**End of Paper**