Comp 251: Mid-term examination #2

Instructor: Jérôme Waldispühl

October 28th, 2020.

- Answer the questions on Crowdmark.
- You have 3 hours to complete this exam and 30 extra minutes to account for any technical issue. You must initiate your submission within 3 hours from the start. If you meet any difficulty while uploading your solution 15 minutes before the end of the submission timeframe (3h + 30min), email us at `cs251@cs.mcgill.ca` with you solution. We will NOT proceed to any manual upload or address any request beyond this time-point.
- You can update your submission on Crowdmark during the timeframe of the exam.
- The clarity and presentation of your answers is part of the grading. Answers poorly presented may not be graded. This includes the clarity of the writing or the quality of the image you uploaded. It is your responsibility to ensure that the image has the good resolution and contrast.
- Keep the size of any file you upload on Crowdmark as small as possible to avoid technical issues.
- Unless specified, all answers must be explained.
- Partial answers will receive credits.
- The conciseness of your answer is part of the grading. An answer that is unnecessarily long or poorly structured will be penalized.
- This is an open book examination.
- It is strictly forbidden to use any external help, including online tutoring systems, or to provide aid to someone else. You are not allowed to communicate to anyone during the exam.
- It is strictly forbidden to share or disseminate this exam or any information related to this exam.
- This exam contains a mandatory academic integrity statement that you should agree with and sign. *We will not grade the exam otherwise*.
- This exam contains 12 pages.

Question:	1	2	3	4	5	Total
Points:	10	20	30	15	25	100
Score:						

Statement of Academic Integrity

In submitting this exam, I confirm that my conduct during this exam adheres to the Code of conduct and academic integrity (`https://www.mcgill.ca/students/srr/academicrights`). I confirm that I did NOT act in such a way that would constitute cheating, misrepresentation, or unfairness, including but not limited to, using unauthorized aids and assistance, personating another person, and committing plagiarism. I will not share or disseminate this exam on any platform or through personal communication.

Write your name and date to sign this statement. *We will not grade your exam if you do not agree with and sign this statement.*

Short answers

1. True or False? Circle your answers. No justification. **Wrong answers will receive a penalty of -1**.

   (a) (2 points) When using open addressing, the load factor α cannot exceed 1.
   A. True B. False

   (b) (2 points) We run the depth-first search algorithm (DFS) on a graph G and identify a back edge. Thus, G has at least one cycle.
   A. True B. False

   (c) (2 points) Given a partition of the vertices of a weighted undirected graph, the cut has one and only one light edge.
   A. True **B. False**

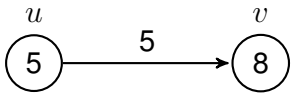
   (d) (2 points) We run the Dijkstra's algorithm on a graph with a negative-weight cycle. Then, the algorithm may not terminate.
   A. True **B. False**

   (e) (2 points) A bipartite graph has no cycle.
   A. True **B. False**

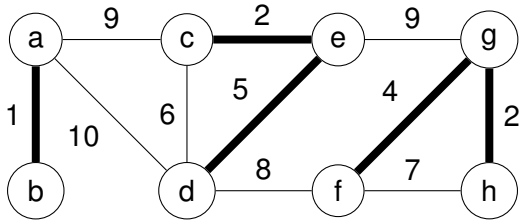
Multiple choice

2. Multiple choice questions. No justification. No penalty for wrong answers. A question has at least one correct answer but can also have multiple correct answers. Full credits is given if and only if you identify all correct answers and do not include wrong ones.

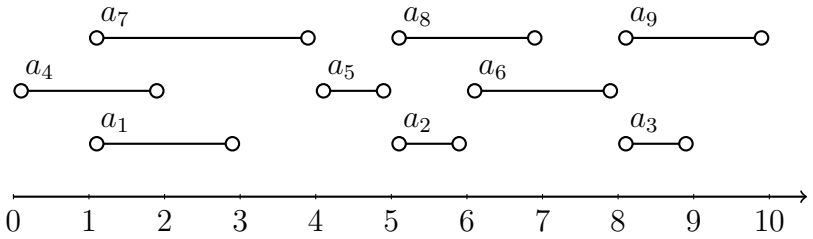
   (a) (4 points) We relax the following edge (u, v) during the execution of the Dijkstra algorithm. We show *shortest path estimate* δ before relaxation of the edge inside the vertices. What will be the value of the shortest path estimate $\delta(v)$ of v after relaxing this edge?
   A. 5 **B. 8** C. 10 D. 13



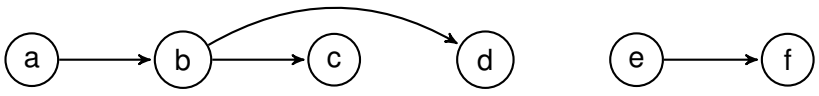
   (b) (4 points) We are in the middle of the execution of the Kruskal's algorithm for computing a minimal spanning tree (MST) of the graph below. The bold edges have already been selected to be in the solution found so far by the Kruskal algorithm, and the edge (d, e) has just been added to the MST (i.e. under construction). Which edge will be the next one to be added to the MST?
   A. (c, d) B. (a, b) **C.** (d, f) D. (f, h)



   (c) (4 points) What is the longest series of activities (i.e. number of activities) that will be returned by the greedy algorithm for solving the scheduling problem (i.e. finding the maximal number of compatible activities) as seen in class?
   A. $(a_7, a_5, a_2, a_6, a_3)$ B. $(a_1, a_5, a_2, a_6, a_9)$ C. $(a_4, a_5, a_8, a_3, a_9)$ **D.** $(a_4, a_5, a_2, a_6, a_3)$

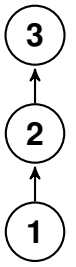


   (d) (4 points) Which of the following propositions is/are **NOT** valid topological orders of the vertices of the direct acyclic graph below.



   A. a, b, c, d, e, f B. a, b, c, e, f, d **C.** a, d, b, c, e, f D. e, f, a, b, d, c **E.** b, c, a, d, e, f

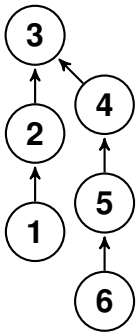
(e) (4 points) We use a tree representation to model disjoint sets, and implement unions as *union-by-height* with path compression. We have the two following trees T_1 and T_2 representing two distinct disjoint sets.



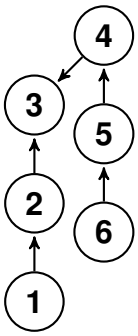
(A) T_1

(B) T_2

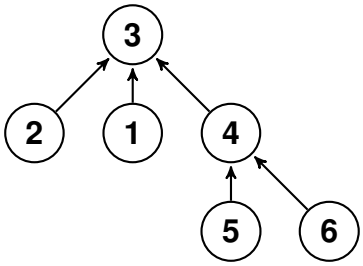
We perform the union of the set containing 6 with the set containing 1 (i.e. `union(6,1)`). Which of the options below is/are possible output(s)?



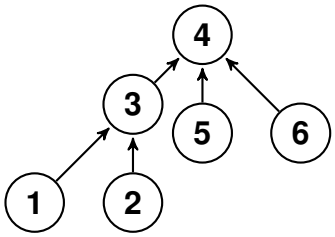
A. T_A



B. T_B



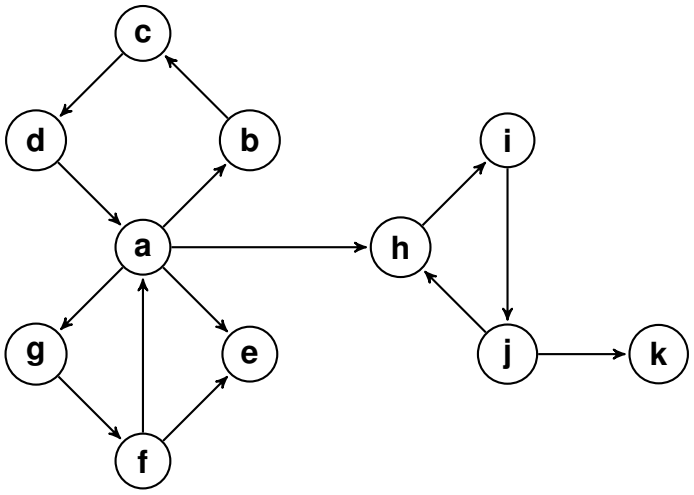
C. T_C



D. T_D

Strongly Connected Components

3. Consider the un-weighted directed graph $G = (V, E)$ below.



(a) (5 points) Show the strongly connected components (SCC) of G. Give your answer as the lists of vertices that belong to the same SCC. Each vertex must belong to one and only one SCC. Your answer must have the minimum number of SCC. No justification needed.

Solution:
$C_1 = \{a, b, c, d, f, g\}$
$C_2 = \{e\}$
$C_3 = \{h, i, j\}$
$C_4 = \{k\}$

(b) (5 points) Build the component graph $G^{SCC} = (V^{SCC}, E^{SCC})$ of G. You can draw it or give its set of vertices V^{SCC} and edges E^{SCC}. No justification needed.

Solution:
$V^{SCC} = \{C_1, C_2, C_3, C_4\}$
$E^{SCC} = \{(C_1, C_2), (C_1, C_3), (C_3, C_4)\}$

(c) (5 points) Describe an algorithm that computes the component graph. Your algorithm must be as efficient as possible.

Solution:
Use the method with two DFS as seen in class to compute the SCC's.
To find all the components of a graph, loop through its vertices, starting a new BFS or DFS whenever the loop reaches a vertex that has not already been included in a previously found component.

(d) (5 points) Give the complexity of your algorithm. The complexity must be represented using an asymptotic notation (i.e. big O) and it must be a tight bound of the worst case running time of your proposed algorithm. No justification needed.

Solution: $O(E + V)$

(e) (10 points) We will show that during a Depth-First Search (DFS), all vertices that belong to the same strongly connected component are in the same Depth-First Search Tree (i.e. the tree representing the edges used by the DFS algorithms to explore the graph).
Let u be the first vertex to be visited in a DFS search of a given SCC, and let v be another vertex of the same SCC that is not a *descendant* of u (i.e. in the same DFS tree). Show that it will lead to a contradiction (Hint: you can use a theorem introduced in class).

Solution: Since u and v are in the same SCC, it exists a path $u \to v$. u is the first vertex of this SCC to be visited. At this time, u just become gray and all other vertices in the SCC, including v, are white. There is thus white path from u to v, and by the white path theorem v is a descendant of u. Contradiction because a descendant cannot be in a different DFS tree.

Single-source shortest paths

4. Suppose that we are given a weighted, directed graph G = (V, E) in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. We also assume that the graph has no self-loop.
   In this question we will argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph (Hint: remember the argument using positive weights in the proof of the Dijkstra's algorithm). To do that you will solve the following questions:

   (a) (5 points) Recall the proof of the Dijkstra's algorithm for computing the single source shortest path we have seen in class (Lecture 14). Indicate which part of this proof rely on the assumption that the graph does not have negative weight cycle.

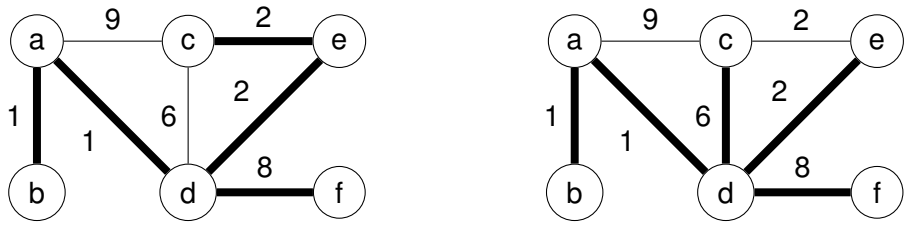
   Solution: The proof states that "y is on shortest path $p(s \rightarrow u)$, and all edge weights are nonnegative. $\Rightarrow \delta(s, y) \leq \delta(s, u)$."

   (b) (10 points) Explain why the original proof still holds true in the particular case described above. *We will grade this part if and only if the previous answer is correct.*

   Solution: In the original proof, the key fact was that $\delta(s, y) \leq \delta(s, u)$. It is claimed that this holds because there are no negative edge weights, but it is stronger than needed. This always holds if y occurs on a shortest path from s to u and $y \neq s$ because all edges on the path from y to u have nonnegative weights. If any had negative weight, this would imply that we would use another edge incident with s, which implies that a cycle is involved in the path, and would only be the case if it were a negative-weight cycle. However, these are still forbidden.

Minimum Spanning Tree

5. Let $G = (V, E, w)$ be a weighted connected undirected graph, where $w(u, v)$ is the weight of an edge (u, v). We define $U \subseteq V$ a sub-set of vertices of G. Consider the set S_U all the spanning trees of G such that all vertices in U are leaves of the spanning tree (Note: the trees may also have leaves that are not in U). We want to compute the minimum spanning tree in S_U.
   We illustrate below this problem with an example. The two graphs show a solution of the problems on the same graph but with two different sets U. The bold edges show the edges of the (constrained) MST. On the left, we set $U = \emptyset$, which is the unconstrained problem and thus the regular MST. On the right, we set $U = \{e\}$, which forces the vertex e to be a leaf of the MST.



   (a) (5 points) Using the connected weighted graph showed above. Propose a set of vertex U of minimal cardinality (i.e. with the minimum number of vertices) for which the constrained minimum spanning tree does not exists.

   Solution: $U = \{a\}$ or $U = \{d\}$.

   (b) (4 points) Consider the Prim's algorithm as seen in class. Explain when and how this algorithm would incorrectly insert a new edge during the execution of an instance of the constrain MST problem (i.e. an edge that would violate the property that a vertex in U must be a leaf).

   Solution:
   Let $u \in U$ and assume an edge (x, u) has been already inserted in $\mathcal{A}$ (i.e. the partial solution being built in Prim's). Assume the next light edge is an edge (u, v). But since $u \in U$ it should not be added because u would no longer be a leaf.

(c) (4 points) To solve this problem, we propose a variant of the Prim's algorithm as seen in class, such that once a vertex u of U is inserted in the partial solution $\mathcal{A}$ (i.e. the "growing" MST in the Prim's algorithm), we do not update the priority queue Q with other edges connected to u (i.e. we do not consider the other edges connected to u).
Let $u \in U$ be a vertex of U connected in $\mathcal{A}$ with an edge (x, u) ($x \notin U$). Assume it exists another edge (y, u) ($y \notin U$) such that $w(y, u) < w(x, u)$. When could this happen?

Solution:
If (x, u) was a light edge of $\mathcal{A}$ before y gets added to $\mathcal{A}$.

(d) (8 points) Consider the graph $G' = (V \backslash U, E \backslash \{(u, v) | u \in U\})$ (i.e. the graph that contains only vertices that are not in U. The operator "\" is a set subtraction). Propose a method using this graph G' and the Prim's algorithm to solve the constrained minimum spanning tree problem (i.e. such that the vertices in U are leaves of the minimum spanning tree). Your solution must also states when and how you detect that the problem has no solution (i.e. it is not feasible). Your solution should be as efficient as possible.

Solution:
Use Prim's to compute the MST $\mathcal{A}$ of G'. If Prim's cannot find a MST for G', then this instance is not feasible.
Then, consider all the edges (u, v) such that $u \in U$ and $v \in V \backslash U$. Sort the edge by increasing weight. Use a Disjoint set structure to add the vertices of U in the solution. At the beginning each vertex of U is in its own component, and all vertices of $\mathcal{A}$ are in a single separate component. Scan the edges in increasing order of weight and add this edge if the vertex in U has not already been added to $\mathcal{A}$. Make Union if that was not the case. Ends when you have a single component.

(e) (4 points) Give the worst case running time complexity of your algorithm. Justify your answer. If needed, you can use the running time complexity results of an algorithm seen in class.

Solution:
Prim's is $O(E \log V)$ for computing the MST of G'. Sorting the edges is $O(E \log U)$. The sequence of union-find is $O(E\alpha(U))$. Thus, the total is $O(E \log V)$.