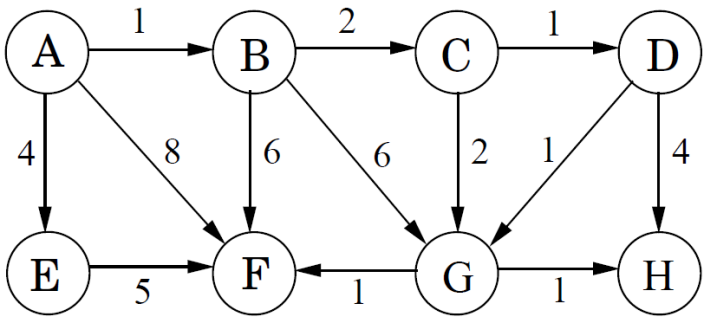
Algorithms & Data — Problem Set 3
(due 11:59pm, Friday February 26th)

Instructions:

- This assignment should be submitted via Blackboard. Late assignments will not be accepted.
- You are encouraged to solve the problems on your own. You are permitted to study with friends and discuss the problems; however, *you must write up your own solutions, in your own words.*
- If you do collaborate with any of the other students on any problem, you must list all the collaborators in your submission for each problem.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class (or the class staff) is strictly prohibited.
- We require that all homework submissions be neat, organized, and *typeset*. You may use plain text or a word processor like Microsoft Word or LaTeX for your submissions. If you need to draw any diagrams, however, you may draw them with your hand.

1. Dijsktra Algorithm Run (15 pts = 10+5)

Suppose Dijkstra's algorithm is run on the following graph, starting at node A.



(a) Draw a table showing the intermediate distance values of all the nodes at each iteration of the algorithm.

Solution: In the below table, the initial distance estimates are 0 for the starting vertex A and ∞ for all others (iteration/column 0). In each subsequent iteration (having its own column) we perform two steps

1. choose the vertex v with the smallest estimate; that estimate always comes from a parent in the tree being built; record the parent that provided the estimate

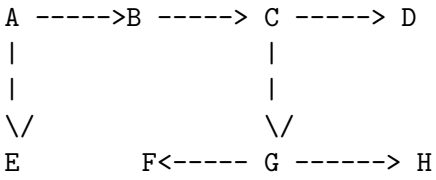
2. update estimates (if they are better) for all neighbors of v which have not been included yet in the tree being built

	0	1	2	3	4	5	6	7	8
A	0	0 A	0	0	0	0	0	0	0
B	∞	1	1 A	1	1	1	1	1	1
C	∞	∞	3	3 B	3	3	3	3	3
D	∞	∞	∞	4	4 C	4	4	4	4
E	∞	4	4	4	4	4 A	4	4	4
F	∞	8	7	7	7	7	6	6 G	6
G	∞	∞	7	5	5	5	5 C	5	5
H	∞	∞	∞	∞	8	8	6	6	6 G

The vertices with minimal estimates at each iteration have been marked in red together with the parents they come from. Hence the red entry in column 7 means that after 7th iteration vertex F has been included in the shortest path tree, F's parent in that tree is G, and the shortest path from A to F is 6.

(b) Show the final shortest-path tree.

Solution:



(Note that this tree can be deduced from the red entries in the above table.)

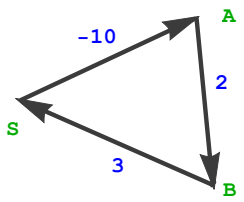
---

2. Negative Edges (10 pts)

Consider a directed graph in which the only negative edges are those that leave s; all other edges are positive. Can Dijkstra's algorithm, started at s,

fail on such a graph? Prove your answer.

Solution: Dijkstra algorithm may not return the correct shortest paths in this case. This will happen when the graph contains negative cycles and the shortest path between some vertices does not exist. Here is an example:



There are many paths $S \rightsquigarrow A$: (S, A) of length -10, (S, A, B, S, A) of length -15, and so on; cycling around the graph we can obtain paths of arbitrarily small lengths.

---

3. Skiing agency (10 pts)

A ski rental agency has n pairs of skis, where the height of the ith pair of skis is s_i. There are n skiers who wish to rent skis, where the height of the ith skier is h_i. Ideally, each skier should obtain a pair of skis whose height matches her/his own height as closely as possible. We would like to assign skis to skiers so that the sum of the absolute differences of the heights of each skier and her/his skis is minimized.

Design a greedy algorithm for the problem. Prove the correctness of your algorithm.

(*Hint:* Start with two skis and two skiers. How would you match them? Continue to three skis and three skiers, and identify a strategy.)

Solution: We first establish the following claim.

Claim. Let A be a tallest skier let B be a longest ski. There is an optimal pairing in which A is paired with B.

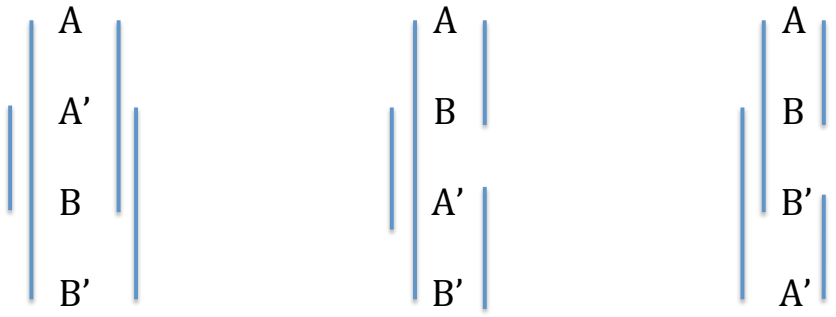
Proof: The proof is by contradiction. Consider an optimal pairing. Suppose ski A is paired with a ski B' shorter than B, and B with a skier A' shorter

than A. Let the heights of A, A', B, and B' be $h(A)$, $h(A')$, $h(B)$, and $h(B')$. We show that

$$|h(A) - h(B)| + |h(A') - h(B')| \leq |h(A) - h(B')| + |h(A') - h(B)|,$$

contradicting the assumption that the given pairing is optimal. We consider three cases, illustrated in the figure below.

1. $h(A) \geq h(A') \geq h(B) \geq h(B')$: In this case, both sides are equal to $h(A) - h(A') + 2(h(A') - h(B)) + h(B) - h(B')$.
2. $h(A) \geq h(B) \geq h(A') \geq h(B')$. In this case, the right-hand side exceeds the left-hand side by $2(h(B) - h(A'))$.
3. $h(A) \geq h(B) \geq h(B') \geq h(A')$. In this case, the right-hand side exceeds the left-hand side by $2(h(B) - h(B'))$.



Now, the algorithm is very simple.

1. Sort the n skiers and the n skis in decreasing order of their heights.
2. Pair the ith skier with the ith ski, in order.

Running time equals the time for sorting plus the time for pairing. Sorting can be done in $O(n \log n)$ time, and pairing in $O(n)$ time. So total time is $O(n \log n)$.

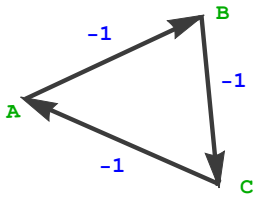
---

4. Trees (10 pts)

Argue that if all of the edge weights of a graph are positive, then any subset of edges that connects all of the vertices and has minimum total weight must be a tree. Give an example to show that the same conclusion does not follow if we allow some weights to be non-positive.

Solution: Proof by contradiction. Suppose that the minimal weight subset S of edges connecting all vertices in a graph G contains a cycle C. Removing any edge e from C leaves the set of C's vertices unchanged (because the remaining edges in C generate all C's vertices). Since no other vertices in G are affected by the removal of e, the set $S \setminus \{e\}$ spans the same set of vertices, i.e. the whole graph G. Hence $S \setminus \{e\}$ spans G but its weight is smaller than that of S — a contradiction.

To see that the minimum weight spanning set of edges does not have to be a tree when edge weights are allowed to be negative, consider simply a triangle:



Removal of any edge from $S = \{(A, B), (B, C), (C, A)\}$ increases S' weight, so S is minimal.

---

5. Maximum Edge Minimal (10 pts)

Describe an efficient algorithm that, given an undirected, connected graph G, determines a spanning tree of G whose largest edge weight is minimum over all spanning trees G.

Solution: We will show first that the minimum spanning three T for G has the required property. That is, a maximum weight edge $e \in T$ is no heavier than a maximum weight edge in any other spanning tree. To arrive at a contradiction, let's assume that this is not true — that there is a spanning three T' for G whose heaviest edge f is strictly lighter than e.

The edge e does not belong to T' because it is heavier than f and f is the heaviest edge in T'. Adding e to T' creates a cycle C in T'. e is the heaviest edge in C as all other edges of that cycle belong to T'. e is therefore strictly heaviest edge in some cycle of G. By the cycle property shown in class and in the textbook, e cannot belong to any spanning tree of G — a contradiction with the assumption that e belonged to a spanning three T.

This means that we can use Kruskals or Prim's algorithm to produce a tree whose heaviest edge is minimal.