CS 473 ✧ Fall 2022

Homework 7

Due Tuesday, October 25, 2022 at 9pm

---

1. Suppose you are given a directed graph $G = (V, E)$, two vertices s and t, a capacity function $c : E \to \mathbb{R}^+$, and a second function $f : E \to \mathbb{R}$.

   (a) Describe and analyze an efficient algorithm to determine whether f is a maximum (s, t)-flow in G.

   (b) Describe and analyze an efficient algorithm to determine whether f is the *unique* maximum (s, t)-flow in G.

   Do not assume ***anything*** about the function f.

2. Suppose you are given a flow network G with ***integer*** edge capacities and an ***integer*** maximum flow f^* in G. Describe algorithms for the following operations:

   (a) INCREMENT(e): Increase the capacity of edge e by 1 and update the maximum flow.

   (b) DECREMENT(e): Decrease the capacity of edge e by 1 and update the maximum flow.

   Both algorithms should modify f^* so that it is still a maximum flow, but more quickly than recomputing a maximum flow from scratch.

3. Let G be a flow network with integer edge capacities. An edge in G is *upper-binding* if increasing its capacity by 1 also increases the value of the maximum flow in G. Similarly, an edge is *lower-binding* if decreasing its capacity by 1 also decreases the value of the maximum flow in G.

   (a) Does every network G have at least one upper-binding edge? Prove your answer is correct.

   (b) Does every network G have at least one lower-binding edge? Prove your answer is correct.

   (c) Describe an algorithm to find all upper-binding edges in G, given both G and a maximum flow in G as input, in $O(E)$ time.

   (d) Describe an algorithm to find all lower-binding edges in G, given both G and a maximum flow in G as input, in $O(VE)$ time.

Standard graph-reduction rubric. For problems solved by reduction to a standard graph algorithm covered either in class or in a prerequisite class (for example: shortest paths, topological sort, minimum spanning trees, maximum flows, bipartite maximum matching, vertex-disjoint paths, …).

Maximum 10 points =

- + 3 for constructing the correct graph.
  - + 1 for correct vertices
  - + 1 for correct edges
  - − ½ for forgetting "directed" if the graph is directed
  - + 1 for correct data associated with vertices and/or edges—for example, weights, lengths, capacities, costs, demands, and/or labels—if any
  - The vertices, edges, and associated data (if any) must be described as explicit functions of the input data.
- For most problems, the graph can be constructed in linear time by brute force; in this common case, no explicit description of the construction algorithm is required. If achieving the target running time requires a more complex algorithm, that algorithm will graded out of 5 points using the appropriate standard rubric, and all other points are cut in half.
- + 3 for explicitly relating the given problem to a specific **problem** involving the constructed graph. For example: "The minimum number of moves is equal to the length of the shortest path in G from $(0, 0, 0)$ to any vertex of the form $(k, \cdot, \cdot)$ or $(\cdot, k, \cdot)$ or $(\cdot, \cdot, k)$." or "Each path from s to t represents a valid choice of class, room, time, and proctor for one final exam; thus, we need to construct a path decomposition of a maximum (s, t)-flow in G."
  - – No points for just writing (for example) "shortest path" or "reachability" or "matching". Shortest path in which graph, from which vertex to which other vertex? How does that shortest path relate to the original problem?
  - – No points for only naming the algorithm, not the problem. "Breadth-first search" is not a problem!
- + 2 for correctly applying the correct black-box **algorithm** to solve the stated problem. (For example, "Perform a single breadth-first search in H from $(0, 0, 0)$ and then examine every target vertex." or "We compute the maximum flow using Ford-Fulkerson, and then decompose the flow into paths as described in the textbook.")
  - − 1 for using a slower or more specific algorithm than necessary, for example, breadth- or depth-first search instead of whatever-first search, or Dijkstra's algorithm instead of breadth-first search.
  - − 1 for *explaining* an algorithm from lecture or the textbook instead of just invoking it as a black box.
- + 2 for time analysis in terms of the *input* parameters (not just the number of vertices and edges of the constructed graph).

An extremely common mistake for this type of problem is to attempt to modify a standard algorithm and apply that modification to the input data, instead of modifying the input data and invoking a standard algorithm as a black box. This strategy can work in principle, but it is much harder to do it correctly, and it is terrible software engineering practice. ***Clearly correct*** solutions using this strategy will be given full credit, but partial credit will be given only sparingly.