Homework 6
Fundamental Algorithms, Fall 2001, Professor Yap

- DUE: Mon Dec 10, in class.
- ANNOUNCEMENT: This is the last graded homework. We will provide one more UNGRADED homework before the Final Exam. But solutions will be provided, so that you can check your own answers and discuss with us.
- Final Exam is likely to be Monday Dec 17, in class, but this awaits confirmation from department.

---

1. [**15 POINTS**] Problem 23.2-1, page 573. Kruskal's algorithm for MST.

2. [**15 POINTS**] Do a hand simulation of Dijkstra's algorithm on the undirected graph in figure 1. Please use vertex A is the source. Recall that this amounts to updating a single array $d[1..n]$ (or, $d[A, B, \ldots, F]$) indexed by the vertices.

   HINT: Try to follow conventions described in class. Display a matrix where the ith row is the value of $d[1..n]$ in stage i. When a node first enters the set S, write an underscore (or, circle) that value. You need not copy values to the next row if they are unchanged.

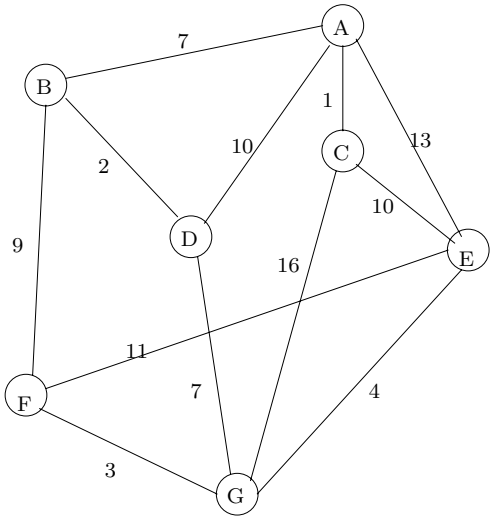


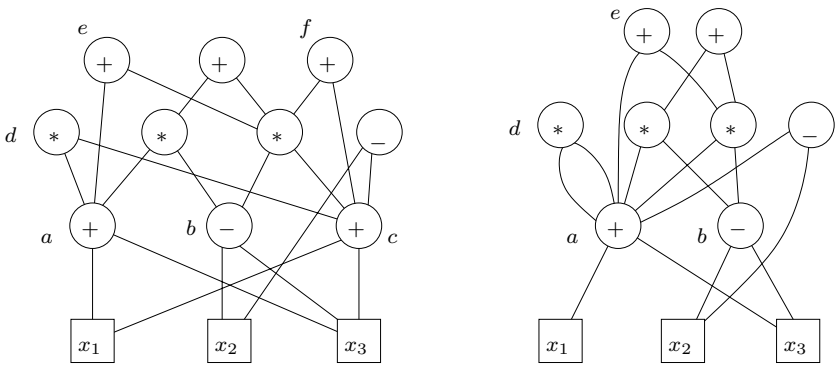
Figure 1: Graph with edge costs.

3. [**15 POINTS**] Problem 22.1-6, p.530. An $O(n)$ algorithm to test for universal sink in a digraph. HINT: use induction on the number of vertices. At each maintain at most one candidate for a universal sink.

4. [**15 POINTS**] Assume the path-compression heuristic but not the rank heuristic in this problem. Construct a sequence of 12 Link/Find operations on the nodes a, b, c, d, e such that the total cost is as large as possible. The cost of a Link is 1 and the cost of a Find is the number of nodes along a Find-path. You need not prove that your sequence is the best possible – so do this on a trial-and-error basis.

   NOTE: when performing $Link(x, y)$, you are allowed to make x or y the new root (in the absence of the rank heuristic).

5. [**15+15+20+15 POINTS**] Suppose you are given a directed acyclic graph ("DAG") G represented as an adjacency list. The nodes with indegree 0 are called sources, and the nodes with outdegree 0 are called sinks. Each sink of G is labeled with a distinct variable name (x_1, x_2, etc). Each non-sink node of G has outdegree 2 and are labelled with one of four arithmetic operations ($+, -, *, \div$). See figure 2, where the directions of edges are implicitly from top to bottom. Thus, every node of G represents an algebraic expression over the variables. E.g., node b represents the expression $x_2 - x_3$.



(a) G (b) G'

Figure 2: Two DAGs: G and G'

Furthermore, the two edges exiting from the $-$ and $\div$ nodes are distinguished (i.e., labeled as "Left" or "Right" edges) while the two edges exiting from $+$ or $*$ nodes are indistinguishable. This is because $x - y$ is different from $y - x$, and $x \div y$ is different from $y \div x$, while $x + y = y + x$ and $x * y = y * x$. Also, this is actually a multigraph because it is possible that there there are two edges from a node d to another node a (e.g., see the graph G' in figure 2(b). Also, the Left/Right order of edges are implicit in our figures.

We define two nodes to be **equivalent** if they are both internal nodes with the same operator label, and their "corresponding children" are identical or (recursively) equivalent. This is a recursive definition. By "corresponding children" we mean that the order (left or right) of the children should be taken into account in case of $-$ and $\div$, but ignored for $+$ and $*$ nodes. For instance, the nodes a and c in figure 2(a) are equivalent. Recursively, this makes e and f equivalent.

Our goal is to construct a **reduced DAG** G' from G in which each equivalence class of nodes in G are replaced by a single new node. For instance with G in figure 2(a) as input, the reduced graph is G' in figure 2(b).

(a) Describe an efficient algorithm to compute the height of each node of G. Height is defined as the length of longest path to a sink. E.g., height of node e in figure 2(a) is 3 (there are paths of lengths 2 and 3 from e to sinks).

(b) Briefly argue why your algorithm is correct and analyze its running time.

(c) Design an efficient algorithm to compute the reduced DAG for any input G. HINTS: To "merge" equivalent nodes, use Union-Find. Note that you only need to check for equivalence among nodes of the same height, and this must be done in a bottom-up manner (i.e., smaller height nodes are merged first). To avoid quadratic behavior, try sorting.

(d) Analyze the running time of your algorithm.

The following exercises are NOT to be handed in, but we encourage you to try to solve them.

1. Repeat problem 4, this time assuming the rank heuristic, but not path compression.
2. Problem 23.2-8, page 574. Divide and Conquer algorithm for MST?
3. Problem 24.3-2, page 600. Dijkstra's algorithm for negative weights?