COS 226	Algorithms and Data Structures	Spring 2024

Final Solutions

1. **Initialization.**

   Don't forget to do this.

2. **Graph search algorithms.**

   (a) 0 2 6 8 3 5 1 4 7 9

   (b) 0 2 8 3 1 5 9 7 4 6

   (c) 5 7 9 1 6 4 3 8 2 0

   (d) no
   The digraph is not a DAG. For example, $3 \to 1 \to 9$ *is a directed cycle.*

3. **Minimum spanning trees.**

   (a) 10 20 30 40 50 90 120

   (b) 90 10 50 20 40 30 120

4. **Shortest paths.**

   (a) 0 4 5 3 1 2

   (b) 0 80 100 70 30 40

   (c) 0 1 3 4 5

5. **Maxflows and mincuts.**

   (a) $36 = 16 + 2 + 25 - 7$

   (b) $107 = 28 + 10 + 30 + 39$

   (c) $A \to F \to G \to B \to H \to D \to I \to J$

   (d) $71 = 28 + 7 + 36$

   (e) A F G

6. **Data structures.**

   (a) T T F F
   Insert each option to an empty table.

(b) (20, 4), (17, 8)
The constraints of the 2d-tree imply that, for any point (x, y) in T, we must have both $x \geq 12$ *and* $3 \leq y \leq 10$.

7. **Properties of graph algorithms.**
   T F T F T

8. **Dynamic programming.**
   A K D E I

```
int opt[] = new int[n + 1];
for (int i = 0; i <= n; i++) {
    opt[i] = values[i];
    for (int j = 1; j < i; j++)
        opt[i] = Math.max(opt[i], values[j] + opt[i-j] - 1);
}
return opt[n];
```

9. **Randomness.**
   T T F F T

10. **Multiplicative weights.**
    F F F T T

11. **Intractability.**
    T F T T T

12. **Design: shortest paths through a landmark.**

    (a) **Constructor:** Run Dijkstra's algorithm twice: once using s as the source vertex, and once using x as the source vertex. Store an integer variable `sToX` containing the distance from s to x (which is contained in `distTo[x]` of the Dijkstra's run that used s as the source) and also the `distTo` array of the Dijsktra's run that used x as a source, both as instance variables.
    pathLen: Report the sum of the distance from s to x and the distance from x to v, i.e. `sToX + distTo[v]`.

    (b) **Constructor:** First, run Dijsktra's in G using x as the source vertex and store the `distTo` array as an instance variable.
    Compute the reverse graph of G (obtained by reversing each edge in G and keeping the same vertices), call it G'. Run Dijkstra's in G' using x as the source and store the `distTo` array as an instance variable called `distToReverse`. The value of `distToReverse[u]` corresponds to the shortest path from u to x for any u.
    pathLen: Report the sum `distTo[v] + distToReverse[s]`.

13. **Design: shortest path with a reverse edge.**

    (a) No, the simplest example is a graph with two vertices s and t and one directed edge from s to t.

    (b) Construct a new graph G'. Create two copies of G and add them to G', call the first copy G_0 and the second copy G_1. For every edge $(u,v) \in G$, add an edge from v_0 to u_1 in G', i.e. an edge from the copy of vertex v in G_0 to the copy of vertex u in G_1. To find the shortest almost-path, run a BFS from s_0 and report the distance to t_1 (so from the copy of vertex s in G_0 to the copy of vertex t in G_1).

    The key idea of this solution is that vertices in G_0 correspond to paths only taking edges in the normal direction, and vertices in G_1 correspond paths that have taken exactly one edge in the opposite direction (and that's why why add an egde from v_0 to u_1 for each edge (u,v), note how the vertices are switched).

    (c) Alter the full solution to add a "super sink", i.e. a new vertex t' and edges from t_0 and t_1 to t'. Run BFS to find shortest path from s_0 to t' and report the result minus 1.

    Alternate solution: Run a BFS on the graph G from s to t, and report the minimum between this and the result found by the solution in part b.