COS 226	Algorithms and Data Structures	Spring 2025

Final Exam Solutions

1. **Initialization.**

   Don't forgot to do this.

2. **Minimum spanning trees.**

   (a) 0 1 2 4 5 6 11

   (b) 6 4 1 2 0 5 11

3. **Depth-first search.**

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

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

   (c) yes

   The digraph is a DAG, so the DFS postorder is guaranteed to be a topological order.

4. **Shortest paths.**

   (a) 0 1 4 5 3 2

   (b) 0 5 4 3 1 2

   (c) 1 4

5. **Data structures.**

   (a) `T R H E S D A Y - - - - - - M O`

   (b) no

   (c) (10, 1), (11, 7)

   (d) $\Theta(E + V)$

   (e) $O(V)$, $O(V^2)$, $\Theta(V)$, $\Omega(1)$, $\Omega(V)$

6. **Maxflows and mincuts.**

   (a) 31

   (b) A B E F G

   (c) 34

   (d) 31

   (e) $A \to F \to B \to D \to C \to H$

   (f) 4

7. **Dynamic programming.**

   E C B I K M or E C B K I M

   It's also possible to substitute H for C.

8. **Karger's algorithm.**

   (a) A D E F

   (b) 4

9. **Multiplicative weights.**

   (a) $d_p = 0$

   (b) $v_p = 7$

   (c) $s_p = 0$

   (d) weight $= 25$

10. **Intractability.**

    ■ □ ■ □ ■ ■ ■ ■ ■ ■

11. **Threshold connectivity.**

Solution 1. The main idea is to use binary search to find the threshold weight w^*, maintaining an interval $[lo, hi]$ for which

- deleting all edges in G of weight greater than lo disconnects G
- deleting all edges in G of weight greater than hi does not disconnect G

For added efficiency, sort the edges by weight and do the binary search using only the actual edge weights (instead of all integers between 0 and U).

- Set $w_0 = 0$ an let $w_1, w_2, \ldots, w_E$ denote the edge weights in ascending order.
- Initialize $lo = 0$, $hi = E$.
- While $lo \neq hi - 1$:
  - set $mid = \frac{lo+hi}{2}$
  - create a graph G' that contains only the edges of weight $\leq w_{mid}$
  - if G' is connected, update $hi = mid$
  - otherwise, update $lo = mid$.
- Return $w^* = w_{hi}$

To sort the edges by weight, use mergesort. This takes $O(E \log E)$ time.

To determine whether G' is connected, use BFS or DFS. This takes $O(E)$ time. This calculation needs to be performed $O(\log E)$ times within the binary search. So, the overall running time of this phase is $O(E \log E)$.

Solution 2. Compute an MST of G using either Kruskal's algorithm or Prim's algorithm. Let w_{max} denote the heaviest edge in the MST. Then, $w^* = w_{max}$. To see why this works:

- $w^* \geq w_{max}$: If we remove all edges in G of weight $> w_{max}$, G remains connected because the MST connects all vertices in G and uses only edges of weight $\leq w_{max}$.
- $w^* \leq w_{max}$: When the edge of weight w_{max} was added to the MST by Kruskal or Prim, it was done so because it is min weight crossing edge in some cut. Removing all edges in G of weight $\geq w_{max}$ would remove all edges in this cut, thereby disconnecting G.

90% partial credit solution: Same as Solution 1, except binary search over the interval $[0, U]$ instead of the sorted array of weights. This takes $\Theta(E \log U)$ time in the worst case.

50% partial credit solution. Same as Solution 1, except sequentially search over the interval $[0, U]$ instead of binary search. This take $\Theta(EU)$ time in the worst case.

12. **Key-and-portal shortest path.**

(a) *The idea is to use familiar tricks to model each component:*

- *Model the requirement that a path must contain a key by using the graph copy trick.*
- *Model an undirected edge with two antiparallel edges.*
- *Model multiple destinations (portals) with an artificial sink vertex.*

Here's the formal construction:

- *Graph copy:* For each vertex v in G: create two vertices v' and v'' in G'.
- *Path must contain a key*: For each edge u–v of weight w in G that contains a key: create two edges $u' \to v''$ and $v' \to u''$ in G', of weight w. The only way to move from graph copy 1 to graph copy 2 is via one of these edges.
- *Undirected edges:* For each edge u–v in G of weight w: create two pairs of antiparallel edges $u' \to v'$, $v' \to u'$, $u'' \to v''$, and $v'' \to u''$, in G', all of weight w.
- *Multiple destinations*: Create an artificial sink vertex t'. For each portal vertex v in G, create an edge $v'' \to t'$ in G' of weight 0.
- *Source and sink*: Vertex s' is the source and vertex t' is the destination.

Key-and-portal paths in G correspond with directed paths from s' to t' in G', and they have the same weight.

(b) The lines with bidirectional arrows represent two antiparallel edges of the given weight.