COS 226 **Algorithms and Data Structures** **Spring 2023**

Final Solutions

1. **Initialization.**

   Don't forget to do this.

2. **Empirical running time.**

   16,000, $\Theta(V^3E)$

   When V doubles, the running time goes up by a factor of $8 = 2^3$; when E doubles, the running time goes up by a factor of 2. Thus, the order of growth of the running time is $\Theta(V^3E)$.

3. **Analysis of algorithms.**

   (a) $\Theta(V^2)$

   The familiar triangular sum: $0 + 1 + 2 + \ldots + V - 1$, which is $\Theta(V^2)$.

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

   The idiomatic code for iterating over the vertices and edges in a graph.

   (c) $\Theta(V)$

   The geometric sum: $V + V/2 + V/4 + \ldots + 1$, which is $\Theta(V)$.

4. **String sorts.**

   B *LSD radix sort (after 2 passes)*

   D *3-way radix quicksort (after the first partitioning step)*

   C *MSD radix sort (after the second call to key-indexed counting)*

   C *MSD radix sort (after the first call to key-indexed counting)*

   B *LSD radix sort (after 1 pass)*

5. **Depth-first search.**

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

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

   (c) no

   There is a directed cycle $3 \to 1 \to 8 \to 3$, so the digraph is not a DAG and has no topological order.

6. **Minimum spanning trees.**

   (a) 0 10 20 30 50 70 110 120

   (b) 10 20 0 30 50 70 120 110

7. **Shortest paths.**

   (a) 1 3 4 6

   Dijkstra's algorithm relaxes the vertices in increasing order of distances from s. So, by the time vertex 6 is relaxed, vertices 0 (0.0) and 5 (35.0) must have already been relaxed. Vertex 2 does not yet have a finite `distTo[]` *value, so it would not have been inserted onto the priority queue.*

   (b) 4

   When relaxing vertex 6, `distTo[3]` *will decrease to 42. But the* `distTo[4]` *is 41, so vertex 4 will be relaxed next.*

   (c) 7

   Since neither vertex 2 or 6 were relaxed in the diagram, `distTo[3]` *must have initialized to 43 when relaxing vertex 7. Since* `distTo[7]` *is 36 (and didn't change since it was relaxed), we can conclude that the weight of edge* $7 \rightarrow 3$ *is 43 - 36 = 7.*

8. **Maxflows and mincuts.**

   (a) 31

   The net flow entering t is $10 + 6 + 15 = 31$.

   (b) 63

   The capacity of the cut is $10 + 16 + 13 + 4 + 20 = 63$.

   (c) A B F G H

   These are the vertices reachable from s using either forward edges (that aren't full) or backward edges (that aren't empty).

   (d) $A \rightarrow G \rightarrow B \rightarrow C \rightarrow I \rightarrow D \rightarrow E \rightarrow J$
   $A \rightarrow G \rightarrow B \rightarrow C \rightarrow I \rightarrow J$

9. **Data structures.**

   (a) C D E A B
   C D E A B can arises (e.g., if the keys are inserted in the order A B C D E).
   A B C D E cannot arise (because no key is at its preferred table index).
   D B C E A cannot arise (because B is in Es spot, E is in As spot, and A is in Bs spot).

   (b) (7, 10) and (8, 14)
   The point (x, y) *must satisfy* $6 \le x \le 9$ *and* $9 \le y \le 15$.

   (c) 1, 226, 36, 56, 646, 822, 8225, 826, 869

10. **Data compression.**

- E
  A worst-case family of inputs is an alternating sequence of 0s and 1s, e.g. 010101010... In this case, run-length coding (with 8-bit counts) needs 8 bits to encode each 0 and 8 bits to encode each 1. So, the compression ratio is ~ 8.
- A
  In the worst-case, each character appears with equal frequency. In this case, the extended ASCII encoding (8 bits per character) is an optimal prefix-free code. Thus, each 8-bit character requires 8 bits to encode. So, the compression ratio is ~ 1.
- B
  In the worst-case, LZW compression (with 12-bit codewords) requires 12 bits to encode each 8-bit character. In this case, the compression is $\sim 12/8$.
- A
  The Huffman coding part of the Burrows–Wheeler compression algorithm will ensure that the compression ratio is at most 1 (because the extended ASCII encoding (8 bits per character) is a prefix-free code and the move-to-front and Burrows–Wheeler transform portions of the algorithm do not affect the number of characters in the input (other than adding the integer index from the Burrows–Wheeler transform). We also know that no data compression algorithm can achieve a worst-case compression ratio of better than 1.

11. **Burrows–Wheeler transform.**

   (a) 2 L N A A B L E A

   (b) A A N A B E L L
   Two strings have the same Burrows–Wheeler core if and only if they are circular shifts of one another.

12. **DFS postorder.**
   C I H B E

13. **Shortest tiger path.**

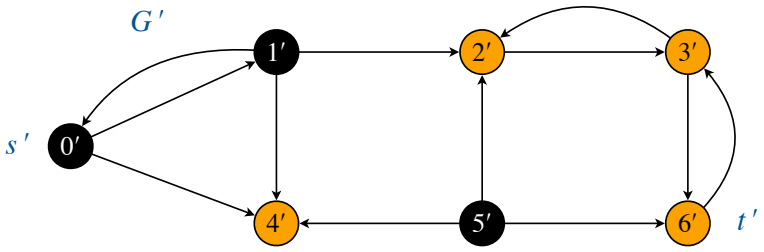
It is important that you provide a *reduction* to the classic (unweighted) shortest path problem in *directed* graphs, as opposed to creating an algorithm from scratch or a reduction to a different problem.

Solution 1. *The idea is to replace each undirected edge in G with two antiparallel edges in G', but don't include an edge $v' \to w'$ if v is orange and w is black. Since s is black and t is orange, any directed path from s' to t' will use exactly one edge whose endpoints have opposite colors.*

Here's the formal construction:

- For each vertex v in G, create one vertex v' in G'.
- For each edge v–w in G with both endpoints the same color, create two antiparallel edges $v' \to w'$ and $w' \to v'$ in G'.
- For each edge v–w in G with v black and w orange, create an edges $v' \to w'$ in G'.

Tiger paths in G are in one-on-one correspondence with directed paths in G' from s' to t'.



Solution 2. *The idea is to to create two copies of the original graph, with one copy of graph to model the beginning part of the tiger path (that uses only black vertices) and the second copy of the graph to model the ending part of the tiger path (that uses only orange vertices). The edges that go from the first copy to the second copy model the single edge in the tiger path whose endpoints have opposite colors.*

Here's the formal construction:

- For each vertex v in G, create two vertices v' and v'' in G'.
- For each edge v–w in G with both endpoints black, create two antiparallel edges $v' \to w'$ and $w' \to v'$ in G'.
- For each edge v–w in G with both endpoints orange, create two antiparallel edges $v'' \to w''$ and $w'' \to v''$ in G'.
- For each edge v–w in G with v black and w orange, create an edges $v' \to w''$ in G'.

Tiger paths in G are in one-on-one correspondence with directed paths in G' from s' to t''.

14. **Necklaces.**

The key idea is to treat the necklaces as binary strings and insert the reverses of those strings into a binary trie. The binary trie must be modified to keep track of the subtrie counts, i.e., how many strings start with a given prefix (or, equivalently, how many necklaces end with a given sequence).

Preprocessing:

- Treat each necklace as a binary string, with 0 for orange and 1 for black.
- Create a *binary trie* with an extra integer field in each node (for the subtrie counts).
- Insert the *reverse* of each string into a binary trie, incrementing the subtrie count of each node touched.

Query: To find the most popular ending sequence of length k, do an inorder traversal of the binary trie and identify the maximum subtrie count of any node at depth k. Return the (reverse of) the string corresponding to that node.