COS 226 **Algorithms and Data Structures** **Fall 2018**

Final Exam Solutions

1. **Empirical running time.**

   (a) 32

   (b) $n^{\log_3 2}$

2. **Mathematical running time.**

   E E D C C

3. **String sorts.**

   A C D C B B E

4. **Depth-first search.**

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

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

5. **Breadth-first search.**

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

   (b) – 2 3 8 8 0 0 6 7 8

6. **Minimum spanning tree.**

   (a) A B F G H
   The answer is unique.

   (b) B–H, D–I, H–I
   The easiest way to determine this is to run Kruskal or Prim from scratch.

7. **Maximum flow.**

(a) 55

(b) 83 = 10 + 26 + 15 + 13 + 19.

(c) 55 = 10 + 26 + 13 + 19 – 12 – 1.
The net flow across any cut equals the value of the flow, so no need to do the arithmetic.

(d) A B F G H
All vertices reachable from the source via forward edges that aren't full or backward edges that aren't empty.

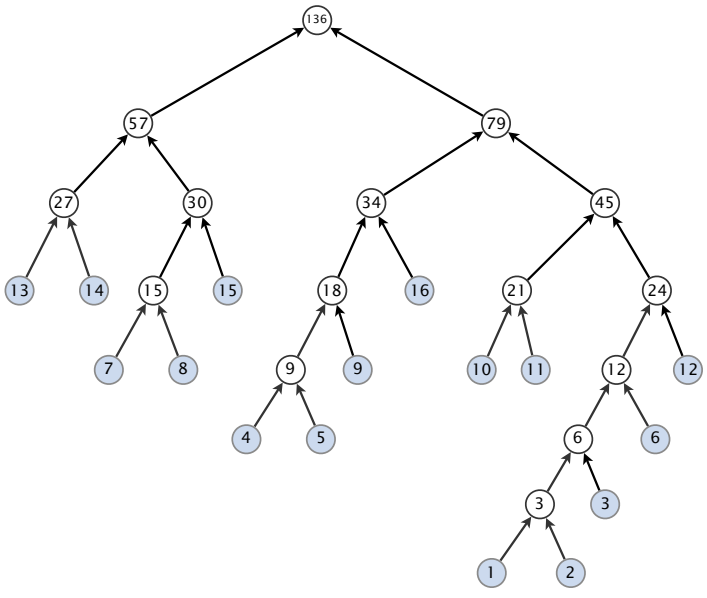
(e) B–C, H–I
You need to consider only those edges that are forward edges in the mincut from (d) because increasing the capacity of any other edge cannot change the mincut (or maxflow).

8. **Huffman compression.**
The length of the longest codeword equals the height of the Huffman trie. In many cases, you can deduce that the height of the Huffman trie is a simple function of the number of characters n in the alphabet.

(a) 4
If the n frequencies are equal, then the Huffman trie is a complete binary trie and the height will be $\lceil \log_2 n \rceil$.

(b) 15
If the n frequencies are successive powers of 2, then the height is $n - 1$.

(c) 7
No shortcut here. Build the trie



(d) 15
If the n frequencies are successive Fibonacci numbers, then the height is $n - 1$.

9. **LZW compression.**

   (a) 41 41 42 43 83 82 42 83 85 43 80

   (b) AA, AB, ABB, BB, BC, BCA, BCAC, BCB, CB

10. **Knuth–Morris–Pratt substring search. (6 points)**

    There is no need to construct the DFA or simulate it. Recall that DFA state is the length of the longest prefix of the pattern that is a suffix of the text.

    (a) 1

    (b) 5

    (c) 7

11. **Properties of shortest paths.**

    A C C A B C

    In the best case, Bellman–Ford is $E+V$ (when the vertices happen to be relaxed in increasing order of distance from s. In the best case, Dijkstra's algorithm is also $E+V$ (when the priority queue never has more than a constant number of vertices, so that all priority queue operations take constant time).

12. **Why did we do that?**

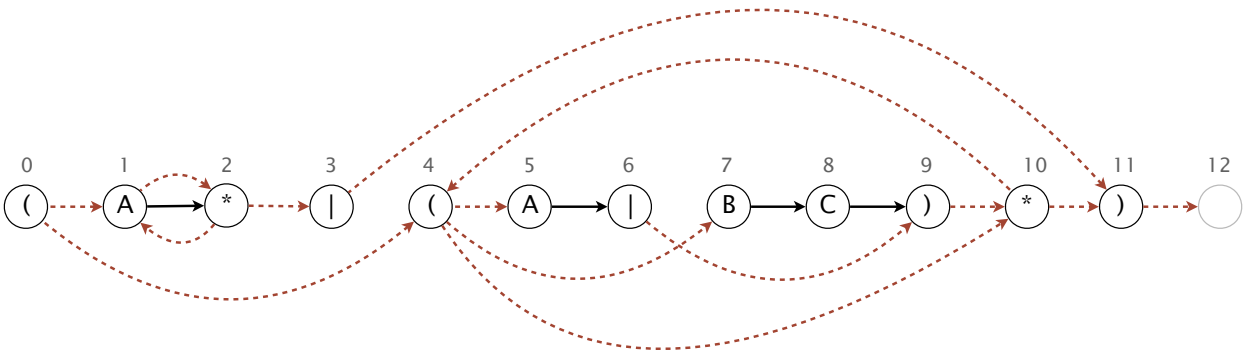
    A A A B A C C A

    Note that for suffix sorting, using an array of `String` *objects makes both the memory usage and running time quadratic (even in the best case). But, it doesn't actually change the worst-case running time (all As).*

13. **Regular expressions.**

    (a) 0→1, 0→4, 1→2, 2→1, 3→11, 4→7, 4→10, 6→9, 9→10, 10→4

    (b) 1 2 3 4 5 6 7 9 10 11 12



14. **Prefix count data structure.**

The main idea is to use an R-way trie, except add an integer `count` *field to each* `Node` *that counts the number of strings inserted in its subtrie (similar to the subtree count field used to implement rank and selection in BSTs).*

(a)
```
public class PrefixCount {
    private final static int R = 128;  // e.g., for ASCII
    private Node root;                 // root of trie

    private static class Node {
        private int count;             // number of strings in subtrie
        private Node[] next = new Node[R];
    }
}
```

(b) Do a standard insert into an R-way trie, except

- No `value` field.
- No special case for duplicate string keys.
- Increment the `count` instance variable for each `Node` referenced during the search.

(c) Do a standard R-way trie search for the given prefix, except

- If you reach a `null` link (because no inserted string starts with the given prefix), return 0.
- Return the `count` field in the terminal `Node`.

A TST (using a count field) almost works except prefix-count queries will take more than L time.

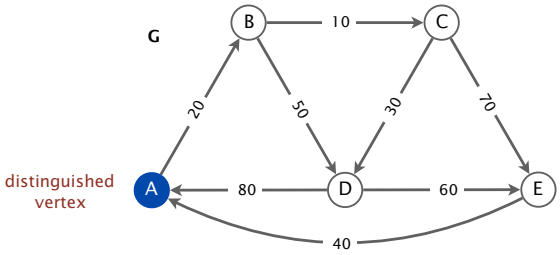
15. **Shortest directed cycle containing a given vertex.**

The key observation is that a shortest directed cycle containing s is a shortest directed path from s to v plus an edge from v to s for some vertex v.

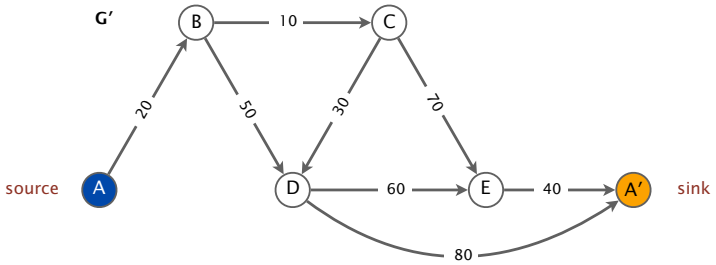
The edge-weighted digraph G' is the same as G except

- Add a new vertex s' to G'.
- Replace each edge in G pointing to s with an edge in G' pointing to s' (and give it the same weight).

Now, directed paths from s to s' in G' are in 1-to-1 correspondence with directed cycles containing s in G, and they have the same length. So, to find the shortest directed cycle containing s in G, compute the shortest directed path from s to s' in G'.



The shortest directed cycle in G containing A is A–B–C–D–A.



The shortest directed path in G' from A to A' is A–B–C–D–A'.

It's also fine to make s' an exact copy of s (i.e, G' contains edges pointing to s and leaving s'), as these edges will not participate in a shortest path from s to s'.