COS 226 **Algorithms and Data Structures** **Fall 2014**

Final Exam Solutions

1. **Digraph traversal.**

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

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

2. **Analysis of algorithms.**

   (a) N

   (b) $\log N$

   (c) $N \log N$

   (d) RN

   (e) $(N + R) \log N$
   For a given value of i, the j loop is executed $\sim (N + R)/i$ times. Thus, the total is $(N + R)(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N}) \sim (N + R) \ln N$.

3. **String sorting algorithms.**

   0 *initial order*

   2 *MSD radix sort after the first call to key-indexed counting*

   3 *3-way radix quicksort after the first partitioning step*

   3 *3-way radix quicksort after the second partitioning step*

   1 *LSD radix sort after 3 passes*

   1 *LSD radix sort after 2 passes*

   1 *LSD radix sort after 1 pass*

   2 *MSD radix sort after the second call to key-indexed counting*

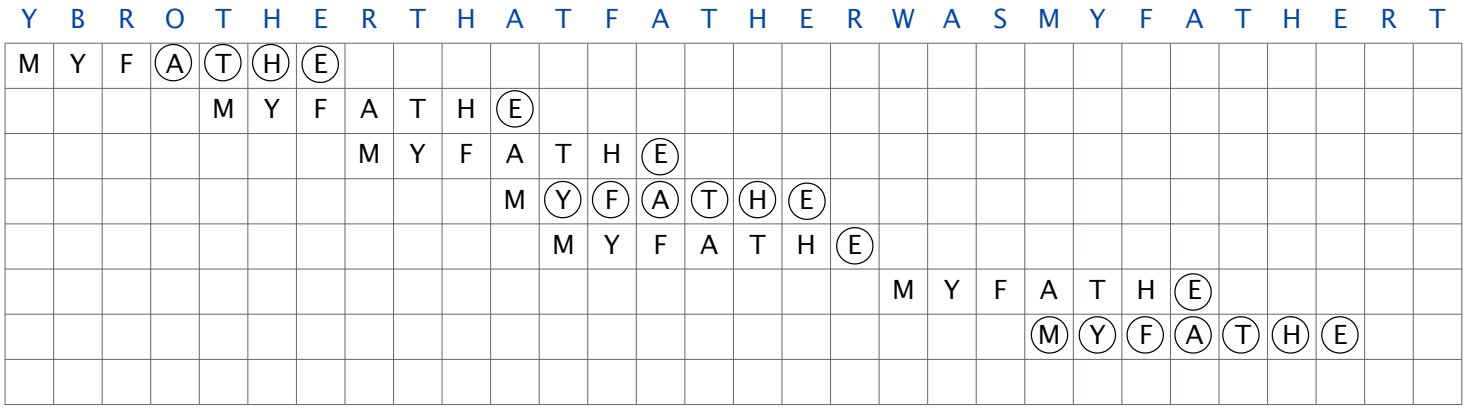
   4 *sorted order*

4. **Substring Search.**

(a)

	0	1	2	3	4	5	6	7
A	0	2	0	4	0	4	0	2
B	0	0	0	0	0	6	0	8
C	1	1	3	1	5	1	7	1

(b)



5. **Minimum spanning tree.**

(a) Kruskal only

(b) Prim only

(c) Prim only

(d) Neither

(e) Neither

(f) Both

6. **Maximum flow.**

(a) 3
To satisfy flow conservation at C, we need the flow on edge $C \to D$ to be 3.

(b) 15 *The net flow arriving at t is equal to the net flow leaving s. This implies that the flow on edge $D \to J$ is 2.*

(c) $A \to G \to B \to C \to I \to J$

(d) 18

(e) $\{A, B, C, F, G\}$.

7. **Properties of algorithms.**

   (a) **Shortest paths.**

   F *Consider an edge-weighted digraph with these edges and weights: $s \to v$ (1), $v \to w$ (2), $w \to t$ (3), $s \to t$ (4). Then, the shortest path is $s \to t$, which excludes the lightest edge.*

   F *In the example above, the shortest path excludes the second lightest edge.*

   F *In the example above, the shortest path includes the heaviest edge.*

   F *Consider an edge-weighted digraph with these edges and weights: $s \to v$ (1), $v \to t$ (2), $s \to t$ (3). Then there are two shortest paths: $s \to v \to t$ and $s \to t$.*

   (b) **Minimum spanning trees.**

   T *Apply the cut property to either endpoint of the lightest edge.*

   T *Let $e = v$-w be the lightest edge and $f = x$-y be the second lightest edge. Apply the cut property to either the vertex x or y (pick x or y so that it is different from both v and w).*

   F *Consider an edge-weighted graph with these edges and weights: v-w (1), w-x (2), x-v (3), x-y (4). Then, the unique MST contains the heaviest edge x-y.*

   T *As asserted in lecture/textbook, if the edge weights are distinct, then the MST is unique.*

   (c) **Burrows-Wheeler transform.**

   F *No string s has $x = 0$ AB as its Burrows-Wheeler transform.*

   T *Otherwise, we wouldn't be able to uniquely decode the message.*

   F *There are two valid Burrows-Wheeler transforms of $s = AA$: $x = 0$ AA and $y = 1$ AA. Thus, the inverse transforms of x and y are equal even though $x \neq y$.*

   F *Computing the Burrows-Wheeler transform is slower because it involves a circular suffix array computation, whereas the Burrows-Wheeler transform is a compact loop.*

8. **Huffman trees.**

   First, construct a frequency table of the characters.

frequency	*characters*
4	A T
3	R S -
2	U D
1	C E G H I L M N O

   Then, for each tree, weight the frequency of each letter by the depth in the tree, and sum up.

   O *114 bits*

   O *114 bits*

   N *115 bits*

   N *115 bits*

9. **LZW compression.**

   `42 81 82 43 41 81 43 83 80`

10. **Burrows-Wheeler transform.**

    (a) `1 C A D D B B D A`

    Below are the sorted circular suffixes:

```
  i  suffixes[i]   t
 ------------------
  0  A A D D B D B C
* 1  A D D B D B C A
  2  B C A A D D B D
  3  B D B C A A D D
  4  C A A D D B D B
  5  D B C A A D D B
  6  D B D B C A A D
  7  D D B D B C A A
```

    (b) `B C D C A A C D`

    Below are the sorted cyclic suffixes and the `next[]` *array:*

```
  i  suffixes[i]   t   next[i]
 ----------------------------
  0  A ? ? ? ? ? ? C     1
  1  A ? ? ? ? ? ? A     4
* 2  B ? ? ? ? ? ? D     5
  3  C ? ? ? ? ? ? D     0
  4  C ? ? ? ? ? ? A     6
  5  C ? ? ? ? ? ? B     7
  6  D ? ? ? ? ? ? C     2
  7  D ? ? ? ? ? ? C     3
```

11. **Algorithm design.**

We use a 4-way trie, but extend it so that each node also stores the number of strings that have been added to the trie that start with the corresponding prefix. In addition to the integer instance variable for the count, we need four node references (either an array of length four or four separate variables).

(a)
```
private Node root;

private static class Node {
    private int count;
    private Node a, c, t, g;
}
```

(b) *Insert the fragment into the trie as usual, but increment the count field in each node along the insertion path (but no special code is needed to deal with duplicate keys).*

```
public void add(String fragment) {
    root = add(root, fragment, 0);
}

private Node add(Node x, String fragment, int i) {
    if (i == fragment.length()) return x;
    if (x == null) x = new Node();
    x.count++;
    char c = fragment.charAt(i);
    if      (c == 'A') x.a = add(x.a, fragment, i+1);
    else if (c == 'C') x.c = add(x.c, fragment, i+1);
    else if (c == 'G') x.g = add(x.g, fragment, i+1);
    else if (c == 'T') x.t = add(x.t, fragment, i+1);
    return x;
}
```

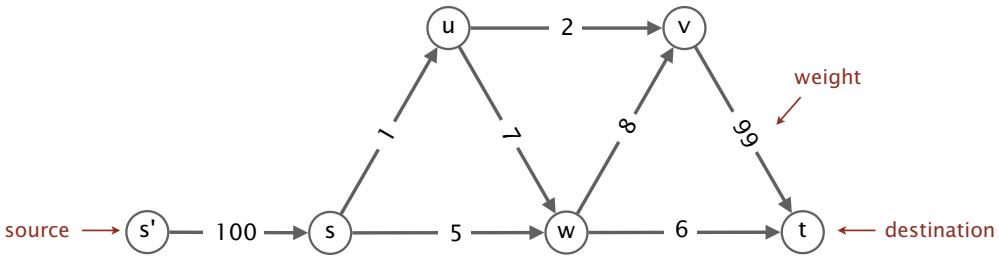
(c) *Find the node corresponding to the given prefix and return the count in that node.*

```
public int prefixCount(String prefix) {
    Node x = root;
    for (int i = 0; i < prefix.length() && x != null; i++) {
        char c = prefix.charAt(i);
        if      (c == 'A') x = x.a;
        else if (c == 'C') x = x.c;
        else if (c == 'G') x = x.g;
        else if (c == 'T') x = x.t;
    }
    if (x == null) return 0;
    else return x.count;
}
```

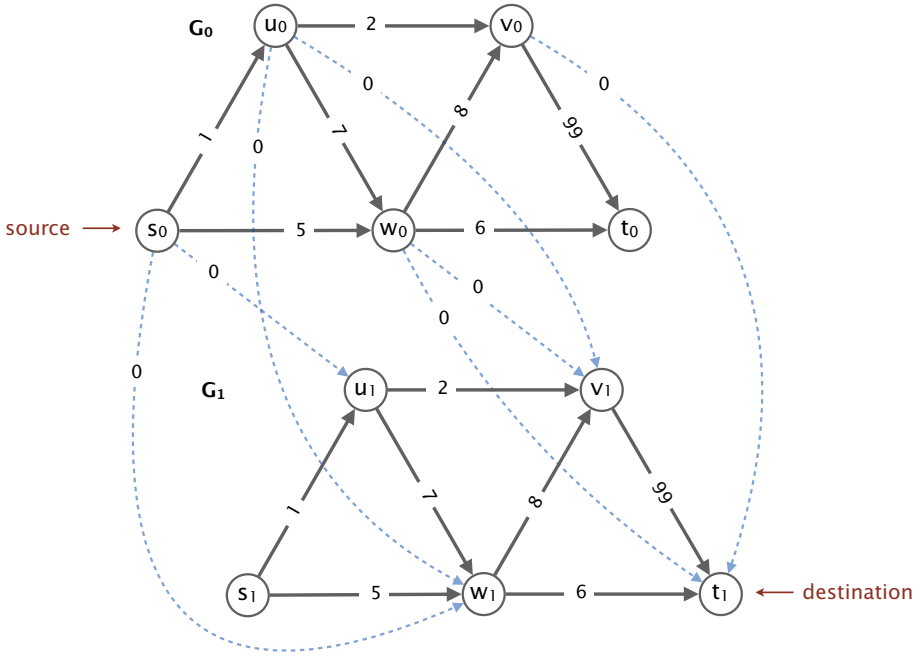
(d) The `prefixCount()` operation takes time proportional to W in the worst case.

12. **Reductions.**

(a) Create a new graph G' that is identical to G. Then add a source s' and an edge from s' to s with weight $= 1 + \max_e w_e$. We claim that finding a shortest teleport path from s' to t in G' yields a shortest path from s to t in G. To see why, observe that every shortest teleport path in G' must begin with the edge $s \to s'$ (because it is the only edge leaving s') and it must teleport across it (because it is the heaviest edge in G').



(b) Create a new graph G' that contains two copies of G, which we will call G_0 and G_1. For each edge $v \to w$ in G, add an edge from v_0 to w_1 with weight 0. We claim that finding a solution to SHORTEST-PATH in G' yields a solution to SHORTEST-TELEPORT-PATH in G. More precisely, the shortest path from s_0 to t_1 in G' provides the shortest teleport path. Any path from s_0 to t_1 must follow an edge from a vertex v_0 in G_0 to a vertex w_1 in G_1, which corresponds to teleporting across edge $v \to w$.



(c) T T T T

13. **Problem identification.**

P *You can use DFS to find a (simple) directed cycle.*

P *You can solve using BFS. For each edge $v \to w$ that has weight 2, add a vertex x and replace the edge by the two edges $v \to x$ and $x \to w$.*

P *This is equivalent to finding the shortest path from s to t using weights = -log of the original weights. Since the digraph is a DAG, this can be done in linear time.*

O *Using the same log trick, this is equivalent to finding an MST. There is no known linear-time algorithm of the problem.*

O *Using the same log trick, this is equivalent to finding the longest path in a graph, which is NP-complete.*

O *There is no known linear-time reduction from maxflow to mincut.*

P *Radix sort the strings; then look at consecutive entries in the sorted array.*

P *Radix sort the 64-bit integers; maintain two scanning pointers, one that indexes negative integers and goes from left to right and one that indexes positive integers and goes from right to left, checking for integers and their negations.*

P *Treat each integer as a string of length 2 over an alphabet of size R. Sort the resulting strings using LSD radix sort.*