COS 226 Algorithms and Data Structures Spring 2015

Final Exam Solutions

1. **Analysis of algorithms.**

   (a) $1160 = 128.9 \times 3^2$

   (b) $\sim 2104N$.
   Node class (2104 bytes): 16 (object overhead) + 8 (reference to `val`) + 8 (reference to `next[]`) + $24 + 8R$ (array of R pointers) $= 56 + 2048 = 2104$.

2. **Graph search.**

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

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

3. **Minimum spanning tree.**

   (a)
   - $x \leq 110$ because x selected before 110
   - $y \leq 80$ because if y were greater than 80, then the edge B-H would be in the MST instead of y.
   - $z \geq 90$; otherwise would be selected before A-F

   (b)
   - $x \leq 130$; otherwise G-H would be in MST
   - $y \leq 80$; otherwise B-H would be in MST
   - z could be any value

4. **Maximum flow.**

   (a) 32

   (b) A G B H I D J

   (c) 3

   (d) A B F G H I

   (e) $B \rightarrow C$, $I \rightarrow J$, $H \rightarrow D$
   $A \rightarrow G$, $G \rightarrow H$, and $D \rightarrow I$ are not edges that cross the mincut, so increasing their capacity will have no effect on the value of the maxflow.

5. **String sorting algorithms.**

   3. 3-way radix quicksort after the second partitioning step
   2. MSD radix sort after the first call to key-indexed counting
   3. 3-way radix quicksort after the first partitioning step
   1. LSD radix sort after 2 passes
   2. MSD radix sort after the second call to key-indexed counting
   1. LSD radix sort after 1 pass
   1. LSD radix sort after 3 passes

6. **Substring search search.**

   (a)

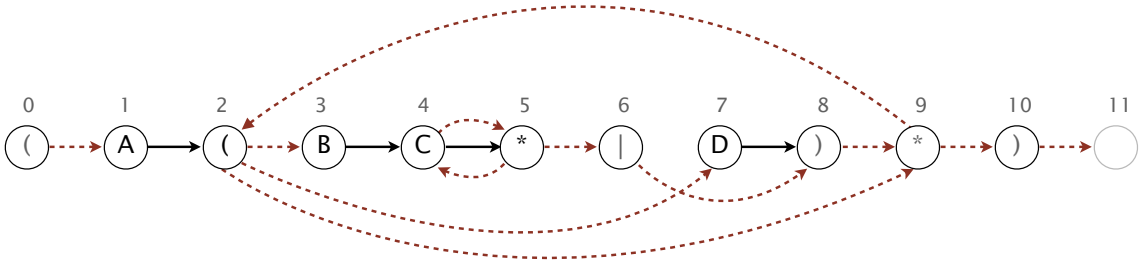
	0	1	2	3	4	5	6	7
A	0	0	3	0	0	3	7	0
B	0	0	0	0	0	0	0	8
C	1	2	2	4	5	6	2	4

   (b) 4

   $hash(269xyz77) = 115 \pmod{157}$
   $hash(10000000) = 42 \pmod{157}$
   $69xyz778 = (269xyz77 - 2 * 10000000) * 10 + 8$

$$\begin{aligned} hash(69xyz778) &= (269xyz77 - 2 * 10000000) * 10 + 8 \pmod{157} \\ &= (115 - 2 * 42) * 10 + 8 \pmod{157} \\ &= 4 \end{aligned}$$

7. **Regular expressions.**



   (a) $0 \to 1$, $2 \to 7$, $2 \to 9$, $4 \to 5$, $5 \to 4$, $5 \to 6$, $6 \to 8$, $9 \to 2$, $9 \to 10$

   (b) 2 3 4 5 6 7 8 9 10 11

8. **LZW compression.**

   (a) B B A C B B C A C B B C A C B

   (b)

i	codeword
81	BB
82	BA
83	AC
84	CB
85	BBC
86	CA
87	ACB
88	BBCA

9. **Burrows-Wheeler transform.**

   (a) 3
   B B D A C A B C

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

   (b) C D D B A A B B

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

10. **Properties of problems.**

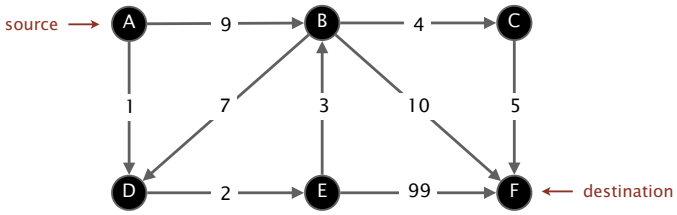
    (a) ii only
    (b) i, iii, v
    (c) iv

11. **Properties of algorithms.**

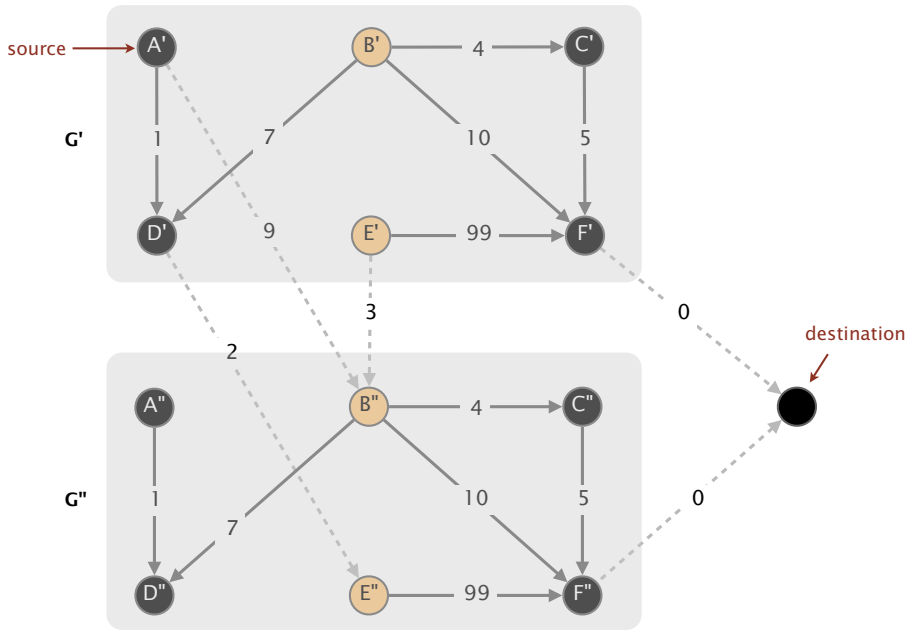
    (a) i, ii, iii, v
    (b) i, iii
    (c) i, iii, v

12. **Reductions.**

    (a) Color all vertices black, using the same edge-weighted digraph, source, and destination.



    (b) Create two copies of the digraph—G' and G'', replacing each edge $v \to w$ where w is an orange vertex with an edge $v' \to w''$. Also, include an artificial sink and connect t' and t'' to the artificial sink (with zero-weight edges). The artificial sink is necessary because the shortest Princeton path can use no orange vertices or the destination can be orange.



13. **Algorithm design.**

(a) Maintain a queue of integers, maintaining the total weight of all integers on the queue.

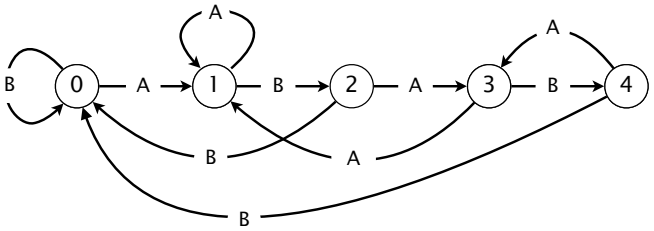
- *add*: enqueue the integer (and add the weight to the sum)
- *remove*: dequeue the integer (and subtract the weight from the sum)

To support all queue operations in constant time in the worst case, use a singly-linked list.

```
public class StreamingSum {
     private Node first, last;
     private long total;

     private static class Node {
          private int weight;
          private Node next;
     }
     ...
}
```

(b) Use a combination of KMP substring search (modified to find all matches) and a `StreamingSum` data type. To modify KMP to find all matches, we must include transitions from the "accept" state in the same manner as for the intermediate states, so that we can detect overlapping matches. Here is the modified KMP for `ABAB`.



When the KMP is in state i the `StreamSum` data type will store the weights of the last i characters.

- If the next character in the text *matches* the next character in the pattern, advance to state $i + 1$ and add the corresponding weight to the `StreamingSum` data type.
- If the next character in the text *does not match* the next character in the pattern (or if the DFA is in state M), retreat to state j, according to the KMP DFA. If $j = 0$, remove all of the integers from the `StreamingSum` data type; otherwise, remove $i - j + 1$ integers from the `StreamingSum` and add the corresponding weight to the `StreamingSum`.

The overall running time is proportional to $M + N$.

- Building the KMP DFA takes time proportional to M.
- The total number of calls to methods in `StreamingSum` is at most $2N$: the total number of calls to `add()` is at most N; the total number of calls to `remove()` is at most the number of calls to `add()`.