COL 702

Homework III

Due on Sept. 19, 2019

Justify your answers with proper reasonings/proofs.

1. The second minimal spanning tree is one that is distinct from the minimal spanning tree (has to differ by at least one edge) and is an MST if the original tree is ignored (they may even have the same weight). Design an efficient algorithm to determine the second MST.

2. Let G be a connected graph and T and T' be two different spanning trees of G. We say that T and T' are *neighbours* if T contains exactly one edge that is not in T' (and vice versa). Now from the graph G, we construct a new graph $\mathcal{H}$ as follows: the nodes of $\mathcal{H}$ correspond to the spanning trees of G, and there is an edge between two two nodes of $\mathcal{H}$ if the corresponding spanning trees are neighbours. Is it true that for any connected graph G, $\mathcal{H}$ is connected ? Either give a proof or a counterexample.

3. Let C be a unit radius circle. An arc of C is given by a pair $[\theta_1, \theta_2]$, where $\theta_1 < \theta_2$ are angles between 0 and 360 degrees. You are given a set of n arcs in the circle and would like to select a subset of arcs of maximum cardinality so that no two of them overlap. Give an efficient algorithm to find an optimal solution.

4. You are given n jobs and m machines. Each job has a size p_j and needs to be scheduled on one of the machines without preemption. The goal is to minimize the average completion time of the jobs. Design a greedy algorithm and prove that it is optimal.

5. A multi-stack consists of an infinite series of stacks $S_0, S_1, \ldots$ where the ith stack S_i can hold upto 3^i elements. The user always pushes and pops elements from the smallest stack S_0. However, before any element can be pushed onto any full stack S_i, we first pop all the elements of S_i and push them into S_{i+1} (and if S_{i+1} gets full, we need to recurse). Similarly, before any element can be popped from any empty stack S_i, we first pop 3^i elements from S_{i+1} and push them into S_i. Again if S_{i+1} becomes empty during this process, we recurse. Assume push and pop operations for each stack takes $O(1)$ time. The pseudo-code is as follows:

```
MultiPush(x):

  i = 0;
  while (S_i is full)
     i = i+1;
  while (i > 0) {
     i = i-1;
     for j=1 to 3^i
```

```
        Push(S_{i+1}, Pop(S_i));
  }
  Push(S_0, x);


MultiPop():
   i = 0;
   while (S_i is empty)
      i = i+1;
   while (i > 0) {
      i = i-1;
      for j=1 to 3^i
          Push(S_i, Pop(S_{i+1});
   }
   return Pop(S_0);
```

- In the worst case, how long does it take to push one more element onto a multistack containing n elements?
- Prove that if the user never pops anything from the multistack, the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack during its lifetime.
- Prove that in any intermixed sequence of pushes and pops, each push or pop operation takes $O(\log n)$ amortized time, where n is the maximum number of elements in the multistack during its lifetime.

6. Consider the following process. At all times you have a single positive integer x, which is initially equal to 1. In each step, you can either increment x or double x. Your goal is to produce a target value n. For example, you can produce the integer 10 in four steps as follows:
$$1 \to 2 \to 4 \to 5 \to 10.$$
Obviously you can produce any integer n using exactly n 1 increments, but for almost all values of n, this is horribly inecient. Describe and analyze an algorithm to compute the minimum number of steps required to produce any given integer n.

7. Suppose you are faced with an infinite number of counters x_i, one for each integer i. Each counter stores an integer mod m, where m is a fixed global constant. All counters are initially zero. The following operation increments a single counter x_i; however, if x_i overflows (that is, wraps around from m to 0), the adjacent counters x_{i1} and x_{i+1} are incremented recursively. Here is the pseudocode:

```
Nudge_m(i)

  x_i = x_i + 1;
  while (x_i >= m) {
    x_i = x_i - m;
    Nudge_m(i+1);
    Nudge_m(i-1);
}
```

- Suppose we call $Nudge_3$ n times starting from the initial state when all counters 0. Note that each call can start from any of the counters. Show that the amortized time complexity is $O(1)$.
- What is the amortized time complexity in the above question if the global counter m is 2 instead of 3 ?