CS 61B Spring 2018 — Final Exam Guerrilla Section

May 5, 2018

Instructions

Form a small group. Start on the first problem. Check off with a helper or discuss your *solution process* with another group once everyone understands *how to solve* the first problem and then repeat for the second problem ...

You may not move to the next problem until you check off or discuss with another group and *everyone understands why the solution is what it is.* You may use any course resources at your disposal: the purpose of this review session is to have everyone learning together as a group.

1 The Tortoise or the Hare

1.1 Given an undirected graph $G = (V, E)$ and an edge $e = (s, t)$ in G, describe an $O(|V| + |E|)$ time algorithm to determine whether G has a cycle containing e.

1.2 Given a connected, undirected, and weighted graph, describe an algorithm to construct a set with as few edges as possible such that if those edges were removed, there would be no cycles in the remaining graph. Additionally, choose edges such that the sum of the weights of the edges you remove is minimized. This algorithm must be as fast as possible.

1.3 Given an undirected, positively weighted graph $G = (V, E)$, a set of start vertices S, and a set of end vertices T, describe an efficient algorithm that returns the shortest path starting from any one vertex in S and ending at any one vertex in T.

Hint: Consider adding dummy nodes to the graph to reduce this problem into something more familiar.

2 One Traversal to Rule Them All

2.1 Given a directed, acyclic graph, G, implement `onePath` which determines if G contains a path that visits every vertex exactly once. Briefly justify why the algorithm is correct and state the runtime.

Assume that the graph is implemented with the following API, where nodes are represented by integers.

```
1 public class Graph {
2         /** Returns true if this graph has an edge from u to v. */
3         public boolean hasEdge(int u, int v);
4
5         /** Returns a list of vertices, in topological order. */
6         public List<Integer> topologicalOrder();
7 }
```

```
public boolean onePath(Graph g) {









}
```

3 Comparison Sorting

3.1 For each of the following scenarios, choose the best sorting algorithm to use and explain your reasoning.

(a) Given a list that was created by taking a sorted list and swapping N pairs of adjacent elements.

(b) Given that the list must be sorted in-place on a machine where swapping two elements is much more costly than comparing two elements.

(c) Given the list is so large that not all of the data will fit into memory at once. As is, at any given time most of the list must be stored in external memory (on disk), where accessing it is extremely slow.

(d) Given a list of emails ordered by send time, sort the list such the emails are ordered by the sender's name first while secondarily maintaining the time-ordering.

(e) Given a randomly shuffled list of Java integers.

(f) Suppose you're designing a secure system that needs to defend against *adversarial inputs*. An attacker can give you any list they choose and understand exactly how your sorting algorithms are implemented.

3.2 Which sorting algorithms do each of the following illustrate? Your options are merge sort, insertion sort, selection sort, heap sort, and quick sort.

Algorithms illustrated may not conform exactly to those presented in discussion and in lecture. Please note that each of these are snapshots as the algorithm runs, not all iterations of its running.

(a)
```
5103 9914 0608 3715 6035 2261 9797 7188 1163 4411
0608 1163 5103 3715 6035 2261 9797 7188 9914 4411
0608 1163 2261 3715 6035 5103 9797 7188 9914 4411
```

(b)
```
5103 9797 0608 3715 6035 2261 9914 7188 1163 4411
0608 3715 2261 1163 4411 5103 9797 6035 9914 7188
0608 3715 2261 1163 4411 5103 6035 7188 9797 9914
```

(c)
```
dze ccf hwy pjk bkw xce aux qtr
ccf dze hwy pjk bkw xce aux qtr
ccf dze hwy pjk aux bkw qtr xce
aux bkw ccf dze hwy pjk qtr xce
```

(d)
```
dze ccf bkw hwy pjk xce aux qtr xpa atm
dze ccf bkw hwy pjk xce aux qtr atm xpa
dze ccf bkw hwy pjk xce aux atm qtr xpa
dze ccf bkw hwy pjk xce atm aux qtr xpa
dze ccf bkw hwy pjk atm aux qtr xce xpa
dze ccf bkw hwy atm aux pjk qtr xce xpa
dze ccf bkw atm aux hwy pjk qtr xce xpa
dze ccf atm aux bkw hwy pjk qtr xce xpa
dze atm aux bkw ccf hwy pjk qtr xce xpa
atm aux bkw ccf dze hwy pjk qtr xce xpa
```

4 Potpourri

4.1 For each of the following, give a data structure or algorithm that you could use to solve the problem or write **impossible** if it is impossible to meet the running time given in the question. For each question, we have one or more right answers in mind, all of which are among the data structures and algorithms listed below. Provide a brief description of how each algorithm or data structure can be used to solve the problem and what, if any, modifications are necessary.

- DFS
- BFS
- Dijkstra's algorithm
- Topological sort
- Kruskal's algorithm
- Linked lists
- Balanced search trees
- Heaps
- Hash Tables
- Tries
- Insertion sort
- Selection sort
- Quicksort
- Merge sort
- Radix sort

(a) Given a list of N words with K characters each, find for each word its longest prefix that is the prefix of some other word in the list in worst-case $O(NK)$ character comparisons.

(b) Given an undirected, weighted, and connected graph $G = (V, E)$, find the heaviest edge that can be removed without disconnecting the graph in worst-case $O(|E| \log |E|)$ time.

(c) We would like to build a data structure that supports the following operations: `add`, `remove`, and `find` the k^{th} largest element for any k in worst-case $O(\log N)$ comparisons for each operation (where N is the number of elements currently in the data structure).

(d) Given an unordered list of N `Comparable` objects, construct a binary search tree containing all of them in $O(N)$ comparisons.

4.2 Given a list of N input words all of length at most K, and M query words, we would like to find, for each query word, the number of input words that match the first $\frac{K}{2}$ letters of the query word. Describe an algorithm that accomplishes this and give its running time as a function of N, K, and M.