Homework 4

CS 4104 (Spring 2016)

Assigned on Tuesday, March 1, 2016.
Submit PDF solutions on Canvas
by the beginning of class on Tuesday, March 15, 2016.

Instructions:

- You can pair up with another student to solve the homework. You are allowed to discuss possible algorithms and bounce ideas with your team-mate. **Do not discuss proofs of correctness or running time in detail with your team-mate. Do not send a written solution to your team-mate for any reason whatsoever.** Please form teams yourselves. Of course, you can ask me for help if you cannot find a team-mate. You may choose to work alone. *Each of you must write down your solution individually, and write down the name of the other member in your team. If you do not have a team-mate, please say so.* **If your solution is largely identical to that of your team-mate or another student, we will return it ungraded.**
- Apart from your team-mate, you are not allowed to consult any sources other than your textbook, the slides on the course web page, your own class notes, the TAs, and the instructor. In particular, do not use a search engine.
- Do not forget to typeset your solutions. *Every mathematical expression must be typeset as a mathematical expression, e.g., the square of n must appear as n^2 and not as "n^2".* Students can use the LaTeX version of the homework problems to start entering their solutions.
- Describe your algorithms as clearly as possible. The style used in the book is fine, as long as your description is not ambiguous. Explain your algorithm in words. A step-wise description is fine. *However, if you submit detailed pseudo-code without an explanation, we will not grade your solutions.*
- Do not make any assumptions not stated in the problem. If you do make any assumptions, state them clearly, and explain why the assumption does not decrease the generality of your solution.
- Do not describe your algorithms only for a specific example you may have worked out.
- You must also provide a clear proof that your solution is correct (or a counter-example, where applicable). Type out all the statements you need to complete your proof. *You must convince us that you can write out the complete proof. You will lose points if you work out some details of the proof in your head but do not type them out in your solution.*
- Describe an analysis of your algorithm and state and prove the running time. You will only get partial credit if your analysis is not tight, i.e., if the bound you prove for your algorithm is not the best upper bound possible.

Problem 1 (10 points) Solve the recurrence $T(n) = T(\lfloor\sqrt{n}\rfloor) + 1$. In words, the $T(n)$ is the running time of an algorithm that creates one sub-problem of the size equal to the square root of n, solves this sub-problem recursively, and spends one more unit of time. You can assume that $T(1) = T(2) = 1$ and that $n > 2$ in the recurrence relation. Remember to prove your solution by induction, even if you use the "unrolling" method to guess a solution.

Problem 2 (30 points) Let $G = (V, E)$ be an undirected connected graph and let $c : E \to \mathbb{R}^+$ be a function specifying the costs of the edges, i.e., every edge has a positive cost. Assume that no two edges have the same cost. Given a set $S \subset V$, where S contains at least one element and is not equal to V, let e_S denote the edge in E defined by applying the cut property to S, i.e.,

$$e_S = \arg\min_{e \in cut(S)} c_e.$$

In this definition, the function "arg min" is just like "min" but returns the argument (in this case the edge) that achieves the minimum. Let F be set of all such edges, i.e., $F = \{e_S, S \subset V, S \neq \emptyset\}$. In the definition of F, S ranges over *all* subsets of V other than the empty set and V itself. Answer the following questions, providing proofs for all but the first question.

(i) (5 points) How many distinct cuts does G have? We will use the same definition as in class: a cut is a set of edges whose removal disconnects G into two non-empty connected components. Two cuts are distinct if they do not contain exactly the same set of edges. For this question, just provide an upper bound.

(ii) (8 points) Consider the graph induced by the set of edges in F, i.e., the graph $G' = (V, F)$. Is G' connected?

(iii) (7 points) Does G' contain a cycle?

(iv) (5 points) How many edges does F contain?

(v) (5 points) What conclusion can you draw from your answers to the previous statements?

Problem 3 (15 points) Consider the version of Dijkstra's algorithm shown below written by someone with access to a priority queue data structure that supports *only* the INSERT and EXTRACTMIN operations. Due to this constraint, the difference between this version and the one discussed in class is that instead of the CHANGEKEY$(Q, x, d'(x))$ operation in Step , this version simply inserts the pair $(x, d'(x))$ into Q. The danger with this algorithm is that a node x may occur several times in Q with different values of $d'(x)$. Answer the following questions.

1. (5 points) What is the running time of this algorithm? Just state the bound in terms of the number of nodes n and the number of edges m in G.
2. (5 points) When the algorithm inserts a pair (x, d_1) into Q, suppose the pair (x, d_2) is already in Q. What is the relationship between d_1 and d_2?
3. (5 points) Using this relationship, how will you fix this algorithm? You just have to describe your correction in words, e.g., by saying "I will add the following command after Step X: ..." You do not have to prove the correctness of your algorithm.

Algorithm 1 DIJKSTRA'S ALGORITHM(G, l, s)

```
1: INSERT(Q, s, 0).
2: while S ≠ V do
3:    (v, d'(v)) = EXTRACTMIN(Q)
4:    Add v to S and set d(v) = d'(v)
5:    for every node x ∈ V − S such that (v, x) is an edge in G do
6:       if d(v) + l_(v,x) < d'(x) then
7:          d'(x) = d(v) + l_(v,x)
8:          INSERT(Q, x, d'(x))
9:       end if
10:   end for
11: end while
```

Problem 4 (35 points) Solve exercise 3 in Chapter 5 (pages 246–247) of your textbook. Let us call the equivalence class with more than $n/2$ cards the *important* class. It is enough for your algorithm to return a card that belongs to this class, if it exists, or no card at all. Note that the problem specifies that the only operation you can perform on a pair of cards is to decide if they are equivalent. You cannot perform any other operation, e.g., compare them in order to sort them. Your proof of correctness must clearly address why your algorithm will find a set of important class, if it exists. There are many things that can go wrong. For instance, there may be an important class for all n cards but the recursive calls don't find any important class (for the subset of cards they process). Alternatively, the recursive calls may find a *different* important class. It is also possible that there is no important class for all

n cards but your recursive calls find an important class. Your algorithm or your proof of correctness must consider all such bad eventualities and you must show that they cannot happen.

Problem 5 (10 points) Given a sorted array of distinct integers $A[1, \ldots n]$, you want to find out if there is an index i such that $A[i] = i$. Give a divide and conquer algorithm to solve this problem. Of course, you must prove its correctness and analyse its running time.