Assignment 2 – Solutions

COMP3121/9101 22T1

Released March 3, due March 17

This document gives model solutions to the assignment problems. Note that alternative solutions may exist.

1. (20 points) You are playing the popular card game "Inscribings". In this game, your opponent places n monster cards onto the board, the ith of which has h_i health points. You in turn have $m \geq n$ hero cards in your hand, the jth of which deals d_j damage per turn. To begin the game, you will choose n heroes from your hand and assign each of them to a different enemy monster. Each turn, your heroes will deal damage equal to their damage power to the opposing enemy. If at any point an opponent's monster reaches 0 health or less, then it is destroyed.

   You are given a limited number of turns k to destroy all enemy monsters. Design an algorithm which runs in $O(m + n \log n)$ time and determines whether it is possible to assign your heroes in such a way as to destroy all enemy monsters in k turns or fewer. If it is possible, your algorithm must also return any such assignment.

   Up to 15 points will be awarded for an algorithm which runs in $\Theta(m \log m)$ time.

Solution

Sort the monsters in decreasing order of health points, and the heroes in decreasing order of damage power, using merge sort for each. Re-index both lists accordingly, so that monster i has the ith most health points, and hero j has the jth highest damage power.

Iterate through both lists simultaneously. If the ith hero is able to defeat the ith monster, i.e. if

$$\left\lceil \frac{h_i}{d_i} \right\rceil \leq k,$$

then assign this hero to this monster, and otherwise report that the task is impossible. If all n monsters have an assigned hero, report the full assignment.

From the construction above, any assignment produced by our algorithm is valid (i.e. each hero is assigned to a different monster) and correctly destroys all monsters. Therefore it only remains to confirm that our algorithm never fails to find such an assignment when it is indeed possible.

Suppose there is some sequence $a_1, \ldots, a_n$ where the a_jth best hero can successfully destroy the jth strongest monster. Now, monster j is no stronger than any monster

$i < j$ which was defeated by hero a_i, and therefore monster j can be defeated by the weakest of heroes $a_1, \dots, a_j$, i.e. hero $\max(a_1, \dots, a_j)$. But the maximum of j different positive integers is at least j, so monster j can be defeated by hero j. It follows that our algorithm correctly reports that all monsters can be destroyed.

Sorting the monsters takes $\Theta(n \log n)$ time, and sorting the heroes takes $\Theta(m \log m)$ time. The latter of these expressions can be improved to $\Theta(m + n \log n)$ by noting that only the n best heroes are required. First use median of medians quickselect with blocks of five in order to select the nth best hero, then filter out all but the n best heroes in one pass, and finally sort these n heroes. Finally, iterating through the lists until an answer is reported takes $\Theta(n)$ time. Therefore the overall time complexity is $O(m + n \log n)$ as required.

---

2. (20 points) Alice writes n distinct integers on a blackboard, and picks a positive integer K. She then allows Bob to make *moves*, each of which consist of the following steps.

   1. Identify two integers x and y on the blackboard which differ by at most K, i.e.

   $$|x - y| \leq K.$$

   2. Erase the smaller of the two chosen integers.

   Bob's task is to make moves in this way until he is no longer able to do so. Note that in some cases, Bob may be unable to make even a single move.

   Design an algorithm which runs in $O(n \log n)$ time and finds the longest sequence of moves. If there are several sequences of maximum length, you may find any of them.

Solution

Sort the numbers in increasing order using mergesort, and denote the sorted values by $A[1], A[2], \dots, A[n]$. Now, compare $A[1]$ and $A[2]$. If they differ by no more than K, erase the smaller of the two and record a move corresponding to this pair. Next, compare $A[2]$ with $A[3]$, and so on until the end of the array. The answer is the number of moves recorded, together with the list of moves.

Observe that the smallest number can be erased if and only if it is within K of the second smallest number. Furthermore, if the smallest number can be erased, there is no advantage in leaving it on the board, as it cannot facilitate the erasing of any other number. Therefore, Bob should erase the smallest number if possible, and otherwise ignore the smallest number, as in our algorithm.

Sorting the numbers takes $O(n \log n)$, and each of $n - 1$ comparisons takes constant time, so the time complexity is $O(n \log n)$ as required.

3. (20 points) You are given a weighted, undirected graph $G = (V, E)$ which is guaranteed to be connected. Design an algorithm which runs in $O(VE + V^2 \log V)$ time and determines which of the edges appear in *all* minimum spanning trees of G.

   Up to 15 points will be awarded for an algorithm which runs in $\Theta(VE \log V)$ time.

   Up to 10 points will be awarded for an algorithm which runs in $\Theta(E^2 \log V)$ time.

Solution

Define an edge as *essential* if it appears in every minimum spanning tree.

An edge e is essential if and only if deleting e either disconnects the graph or increases the total weight of the minimum spanning tree. Therefore we can first run Kruskal's algorithm using the union-find data structure to find the weight of the minimum spanning tree T of the original graph G. We then test individual edges e by running Kruskal's algorithm again, this time on $G - e$. Only those edges in the original MST T must be tested; all other edges are obviously not essential as they do not appear in T.

Sorting the edges takes $O(E \log E)$ time. Note that we only need to sort the edges once; subsequent applications of Kruskal's algorithm can re-use the same sorted edge list with one omission. Then, there are $V - 1$ edges of T to test. In each test, we perform $2E$ `find` operations each in constant time and $V - 1$ `union` operations in $O(V \log V)$ total time. Therefore, the overall time complexity is

$$\begin{aligned} O(E \log E + V(E + V \log V)) &= O(E \log V + VE + V^2 \log V) \text{ since } E < V^2 \\ &= O(VE + V^2 \log V), \end{aligned}$$

noting that $E \log V$ is dominated by both VE (since $\log V < V$) and $V^2 \log V$ (since $E < V^2$).

4. (20 points) There is an upcoming football tournament, and the n participating teams are labelled from 1 to n. Each pair of teams will play against each other exactly once. Thus, a total of $\frac{n(n-1)}{2}$ matches will be held, and each team will compete in $n-1$ of these matches.

   There are only two possible outcomes of a match:

   1. The match ends in a draw, in which case both teams will get 1 point.
   2. One team wins the match, in which case the winning team gets 3 points and the losing team gets 0 points.

   Design an algorithm which runs in $O(n^2)$ time and provides a list of results in all $\frac{n(n-1)}{2}$ matches which:

   (a) ensures that all n teams finish with the same points total, and

   (b) includes the fewest drawn matches among all lists satisfying (a).

Solution

If n is odd, then we can solve the problem without any draws: each team should win exactly $\lfloor n/2 \rfloor$ matches and lose the same number of matches. We then produce a list of results using the following construction: place all teams in order in a circle, and let the ith team win against the next $\lfloor n/2 \rfloor$ teams following it in the circle, and lose to all other teams.

Unfortunately, if n is even, we need to use some draws since the total sum of scores over all teams is exactly $3n(n-1)/2$ when there are no draws, and this number is not divisible by n when n is even. Indeed, letting $n = 2k$, we have

$$\begin{aligned}\frac{3n(n-1)}{2} &= \frac{3(2k)(2k-1)}{2} \\ &= 3k(2k-1) \\ &= 6k^2 - 3k \\ &= 2k(3k-2) + k.\end{aligned}$$

Each tie reduces the total sum by 1, so at least k draws are required to make $2k(3k-2) + k$ divisible by $2k$. So, if we find a list of results with exactly $n/2$ draws, it is optimal. We use a similar construction to the previous case: once again, place all teams in order in a circle, and now make the ith team win against the next $(n-2)/2$ teams, lose against the previous $(n-2)/2$ teams, and tie with the team directly opposite.

In either case, we can report each of the $O(n^2)$ desired results in constant time.