CS31005 Algorithms – II, Autumn 2021–2022

Second Test: Evaluation Notes

---

1. No credit for proving correctness.

6 marks for giving a correct counterexample (no explanation needed). [Only one correct counterexample suffices.]

4 marks if only a correct explanation is given for incorrectness but without any counterexample.

2. **(a)** 2 marks for listing all the events (deduct partial marks for omissions), 2 marks for printing in the correct order.

(b) 1 or 2 mark penalty for listing extra events (depending on how many extra events are printed), 1 mark penalty for not listing all the events.

3. **(a)** 1 mark for listing all the events, 1 mark for listing in correct order [both binary].

(b) 2 (before) + 2 (after) [binary marking]

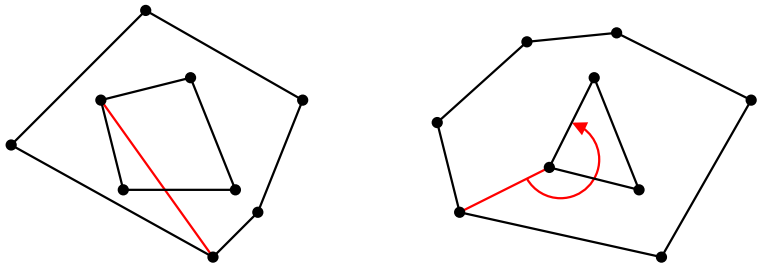
(c) binary marking

4. Let S be a set of n points (in general position) in the two-dimensional plane. We want to reorder ***all*** the points of S to a list $P_1, P_2, \ldots, P_n$ in such a way that

(1) no segment P_iP_{i+1} intersects with any other segment P_jP_{j+1} except perhaps at the endpoints, and
(2) $P_iP_{i+1}P_{i+2}$ is a right turn for all $i = 1, 2, \ldots, n-2$.

Propose an efficient algorithm to solve this problem. What is the time complexity of your algorithm? **(8 + 2)**

Many solutions are based on onion layers. No problem, but there is a necessity to join the layers. This is the crux of the solution. Very little credit for computing the onion layers only (which is already covered in the tutorials). Joining a layer to the leftmost point of the next inner layer is a bad idea, as illustrated by the figure below. On the left, there is an example of non-trivial intersection of the segments. On the right, there is an example of a wrong turn.



The correct idea is to draw a tangent from the last point listed to the next inner layer, and start listing the points of the inner layer from the point of tangency. Another possibility is to keep this last point during the construction of the next layer.

Running time: We have seen in the tutorials that Jarvis march achieves a running time of $O(n^2)$ which may be tight. Some students claimed that Graham scan achieves everything in $O(n \log n)$ time. This is wrong. The initial sorting phase takes $O(n \log n)$ time. After that, $\Theta(n)$ layers may be constructed, each taking $O(n)$ time (because the running time of Graham scan depends on the number of remaining points, not on the number of points on the layer). In each algorithm, there is a necessity to *delete* points already sent to outer layers. For a simple array-based compacting technique, this may also take $\Theta(n^2)$ time. In short, both Jarvis march and Graham scan take $O(n^2)$ time, not $O(n \log n)$. Whether this problem can be solved in $O(n \log n)$ time is not known to me, but Graham scan cannot do it.

5. A forest is inhabited by n prides of lions. The i-th pride lives in a den at location $P_i = (x_i, y_i)$, and requires a circular area of a fixed radius r around P_i for hunting. A pride is called safe if its hunting area does not intersect with the hunting area of any other pride. A survey gives the forest ranger the den locations x_i, y_i, and the hunting radius r (same for all prides). The ranger wants to figure out which prides are safe.

Let S be the set of the n den locations (assumed to be in general position). Propose an algorithm that, given $\mathrm{Vor}(S)$, solves the ranger's problem in $O(n)$ time. Assume that each segment (finite or semi-infinite) of $\mathrm{Vor}(S)$ stores the indices i, j such that the segment belongs to the perpendicular bisector of P_iP_j. Justify the correctness of your algorithm, and that its running time is $O(n)$. **(10)**

There are n dens, and so $\binom{n}{2} = \Theta(n^2)$ pairs of them. Looking at all these pairs leads to an $O(n^2)$-time algorithm. The VD (Voronoi diagram) of S contains only $\Theta(n)$ of the edges, that is, information about $\Theta(n^2)$ pairs is not stored in the VD. Despite that, the VD suffices in solving this problem because of a property of Voronoi diagrams: If Q is the den closest to P, then the Voronoi cells of P and Q share an edge. This is the reason why VD can be used to solve the *all-closest-neighbor problem*. Identifying and proving this property is what is asked in the question. Given what is covered in the tutorials, this is a few lines of proof, but must be there in the solution. Facts like two circles intersect if and only if the distance between their centers is less than $2r$ or that P is safe if a circle of radius r and center P resides completely within $\mathrm{VCell}(P)$ are obvious, and do not require proofs.

An argument is also needed why the running time is $O(n)$. The reason is that the size of $\mathrm{Vor}(S)$ is $\Theta(n)$. Facts like each VCell has $O(1)$ (or three) vertices or the algorithm spends $O(1)$ time for each den are wrong.

6. Let $\Phi(x_1, x_2, \ldots, x_n)$ be a Boolean formula in the conjunctive normal form (CNF). We say that Φ is all-but-one satisfiable if there is a truth assignment of the variables for which all except exactly one of the clauses of Φ evaluate to true. By AB1SAT, we denote the problem of deciding whether the given CNF formula Φ is all-but-one satisfiable.

(a) Prove that AB1SAT is in NP. **(4)**

(b) Prove that AB1SAT is NP-complete. (**Hint:** Use reduction from CNFSAT.) **(6)**

(a) The size of the input is the size s of the formula Φ. It need not be $O(n^k)$ for some constant k. If you throw out unused variables, you can also assume that all variables appear in Φ, so you can take the size as $O(s+n)$. This is also $O(s)$, but s need not be a polynomial in n.

(b) In order to prove the NP-hardness of AB1SAT, a reduction from CNFSAT may be done. In addition to the construction, it is needed to prove that the reduction works. That is, if the reduction converts Φ to $\hat{\Phi}$, then you need to present a *two-way* proof that Φ is satisfiable *if and only if* $\hat{\Phi}$ is all-but-one satisfiable.

Many proofs are based on the construction $\hat{\Phi} = \Phi \wedge y$ for a new variable y. This construction is faulty. If Φ is satisfiable, then $y = 0$ in tandem with a Φ-satisfying assignment all-but-one satisfies $\hat{\Phi}$. The converse is not true. Suppose that $\hat{\Phi}$ is all-but-one satisfiable. There is nothing to prevent all such AB1 assignments to have $y = 1$. For these assignments of the variables, Φ never evaluates to true. Moreover, if $\hat{\Phi}$ is only all-but-one satisfiable (but not satisfiable), then Φ is not satisfiable at all. For example, if $\Phi = x_1 \wedge \overline{x_1}$, the construction gives $\hat{\Phi} = x_1 \wedge \overline{x_1} \wedge y$. But then, Φ is not satisfiable (it is indeed always false), whereas $\hat{\Phi}$ is all-but-one satisfiable.

A few solutions took complements of clauses. Note that in general the complement of a clause (a sum of literals) is not a single clause (it is a product of literals).