Algorithms II 89-322-01, 89-322-02, FINAL EXAM
MOED B

Instructor: *Prof. Amihood Amir*
Length of Exam: 2 hours
Time: August 28th, 08:30
NO OUTSIDE MATERIAL ALLOWED!!!

A general note about the grading process: Below you will find how many points were deducted for each error. Some answers incurred several errors, and so the amount of points deducted in such case was usually the sum of points deducted per each error, although some exceptions were made (deducting less points).

1. (33 points) Consider the following strategy for the two server problem: The server closest to the dirt handles it. The other server advances half way from its position toward the dirt. Is the algorithm competitive? Prove.

   Answer:

   In general, when the dirt occurs **exactly** in a location where there is a server, we assume that nothing is done (the server just cleans it) so there is an implicit assumption that dirt only happens in locations where there are no servers. However, since this was not explicitly stated we were lenient in the grading and gave full points to people who assumed that even if the server who cleans the dirt is right there, the other server still moves halfway towards the dirt. For completeness, we present solutions according to each of these assumptions.

   The Standard Assumption:

   Consider two points on a line, a and b with distance d between them. In the initial position, the two servers s_1 and s_2 are halfway between points a and b. The sequence of events is:
   $ababababab...ab$
   When the first a arrives, wlog s_1 serves it (moves distance $\frac{d}{2}$, and server s_2 moves halfway towards point a, i.e. distance $\frac{d}{4}$. When the next event occurs at point b, s_2 moves distance $\frac{3d}{4}$ to serve it, and s_1 moves distance $\frac{d}{2}$ in the direction of point b.

   The total distance travelled after these two events is thus $2d$. For every

further event, one server travels distance $\frac{d}{2}$ to serve the event, and the other server travels distance $\frac{d}{2}$ to the halfway point, for a total of d.

Thus if n events occur, our algorithm requires travel of distance nd.

However, the optimum algorithm places s_1 on point a and s_2 on point b for the cost of d and that suffices for any n. Therefore, there does not exists a constant c for which the time of our algorithm is less than cd, since for any c take $n > c$ and there is no c-competitivity.

The assumption where there is movement if the dirt is on the spot of one of the servers:

Consider two points on a line, a and b with distance d between them. In the initial position, s_1 is at point a and s_2 is at point b. The sequence of events is:
aaaaa...aa
Whenever an event occurs, s_1 serves it and s_2 moves half the distance toward point a. The total distance for n events is, therefore,

$$\sum_{i=1}^{n} \frac{d}{2^i}.$$

This is clearly at least $\frac{d}{2}$ but since the optimum is 0 distance, there can be no constant c for which $\frac{d}{2} \leq c \cdot 0$.

Error Codes and Penalties:

1. Bad error in distance computation. -16.
2. Proof not well written. -7.
3. Error in distance calculation. -3.
4. Did not calculate distance in optimal strategy. -7.
5. Did not calculate algorithm's distance. -7.
6. Major problems in example. -28.
7. Example not fully described. -7.

2. (33 points) The **Smallest Set of True Atoms (SSTA)** problem is defined as follows:
INPUT: A Boolean formula in conjunctive normal form (CNF) without negations and with exactly three literals in every conjunct, i.e.:

$$(\alpha_{1,1} \vee \alpha_{1,2} \vee \alpha_{1,3}) \wedge (\alpha_{2,1} \vee \alpha_{2,2} \vee \alpha_{2,3}) \wedge \cdots \wedge (\alpha_{k,1} \vee \alpha_{k,2} \vee \alpha_{k,3})$$

Where $\alpha_{i,j}$ are atoms (no negations).
OUTPUT: The smallest number of atoms that, if assigned truth value `TRUE`, will satisfy the formula.

Note that every such formula is satisfiable. We are interested in the *smallest* number of atoms that, if assigned value `TRUE`, will cause the entire formula to be satisfied.

Example:

$$\alpha = (A \vee B \vee C) \wedge (D \vee B \vee A) \wedge (E \vee A \vee C) \wedge (A \vee D \vee E) \wedge (E \vee B \vee D)$$

If A and B are assigned value `TRUE` the the formula is satisfied even if all other atoms are assigned value `FALSE`.

The SSTA problem is $\mathcal{NP}$-hard. Write an Integer Program in the slack form for the SSTA problem.

Answer:

Assume there are n different atoms in the formula. For each one we assign a variable that will get value 0 if its respective atom is assigned value `FALSE`, and 1 if its respective atom is assigned value `TRUE`.Assume our variables are $x_1, ..., x_n$. The IP is as follows:

Objective Function: $\max \sum_{i=1}^{n} -x_i$.

Constraints:
$x_i + s_i = 1$ (Every variable is at most 1).

Let $(\alpha_{j,1} \vee \alpha_{j,2} \vee \alpha_{j,e})$ be a conjunct where the variables of its respective atoms are a, b, and c. Then add constraint:

$a + b + c - t_j = 1$ (At least one atom in the conjunct is satisfied).

$x_i \geq 0 \quad i = 1, ..., n.$

$s_i \geq 0 \quad i = 1, ..., n.$

$t_j \geq 0 \quad j = 1, ..., k.$

Error Codes and Penalties:

1. Slack notation not provided. -13.
2. Slack notation objective function should be max, not min. -4.

3. Did not provide constraint for variables to be between 0 and 1. -5.
4. Not every *disjunct* has a variable, every *atom* has a variable. -7.
5. At least one `TRUE` variable is necessary in every **conjunct**. -5.
6. Failed to provide slack form for variable constraint. -4.
7. Failed to provide slack form for conjunct constraint -5.
8. Slack variable should all be greater than or equal to 0. -7.
9. Error in constraint of the conjunct. -5.
10. Did not make sure that a different slack variable is necessary for every conjunct. -4.

3. (34 points) The **Longest Reverse Substring** Problem is defined as follows:
INPUT: String T over alphabet Σ.
OUTPUT: The longest substring S of T such that S^R is also a substring of T, where if $S = S[1], ..., S[m]$, then $S^R = S[m], ..., S[1]$.

Write an algorithm that solves the Longest Reverse Substring problem. Describe the idea of the algorithm and prove its time complexity. The more efficient the algorithm (provided it is correct) the more points will be given.

Answer:

The idea is as follows: Concatenate T and T^R (with a symbol not the alphabet between them), i.e. $T' = T\$_1 T^R \$_2$, where $\$_1, \$_2 \notin \Sigma$. Note that any substring that occurs in T' both *before* $\$_1$ and *after* $\$_1$ is a candidate for our output. A longest such substring is the output. Because $\$_1 \notin \Sigma$ it can not be the case that there are two such substrings which cross $\$_1$.

So we need to find a way to efficiently find such substrings. Obviously, a suffix tree of T' will provide all substrings. Any node that has a split where in one side there is a $\$_1$ and on the other side there is no $\$_1$, is a candidate.

We can figure out the length of the suffix leading to every node by DFS on the suffix tree, so a DFS on the tree can identify the longest substring that appears in both trees. The only remaining question is how to recognize the nodes that are potential substrings for the output.

This can also be done by a DFS on the tree. Note that any leaf representing a suffix starting at the first n positions of T' means a substring of T, paint it white. Any leaf representing a suffix starting after location $n+1$ means a substring of T^R, paint it blue. Going up the paths, whenever a node has a blue child and a white child, it is a substring that appears both in T and T^R. If all children have the same color, then the parent node get that color.

Time: Suffix tree construction is done in time $O(n \log \sigma)$, where $\sigma = \min(|\Sigma|, n)$, by the Weiner algorithm mentioned in class. The rest of the operations are DFS searches on the tree, so their time is linear.

Error Codes and Penalties:

0. Completely erroneous algorithm. -34.
1. Correct naive algorithm of time $O(n^3)$. 10.
2. Correct algorithm of time $O(n^2 \log n)$. 14.
3. Correct algorithm of time $O(n^2)$. 17.
4. Did not give time for unbounded alphabets. -3.
5. No explanation of correctness. -15.
6. Did not seek a leaf that has a child in T and a child in T^R. -10.
7. Error in finding a leaf that has a child in T and a child in T^R. -10.
8. Nonstandard suffix tree construction (e.g. generalized suffix tree. Since we did not mention it in class, this usage requires proof. -10.
9. Did LCA of all suffixes of T with all suffixes of T^R. There are n^2 such possibilities. -15.
10. Wrong suffix tree construction time. -5.
11. How do you know which leaf is T and which is T^R? -5.
12. Assumes the suffix tree of T' has some leaves ending in $\$_1$ and some leaves ending in $\$_2$. -5.
13. Constructs a trie but claims time is $O(n \log \sigma)$. -15.
14. Assumes that an LCA between all suffixes of T and all suffixes of T^R can be found in linear time - no explanation how. -10.

15. Uses an algorithm that constructs two suffix trees, one for T and one for T^R, then compares these trees and claims time is linear rather than $O(n^2)$. -15.
16. Uses LCA of a *single* node?!? -10

GOOD LUCK