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

Instructor: *Prof. Amihood Amir*
Length of Exam: 2 hours
Time: July 7th, 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). In addition, some notebooks were given besides the error a number of pluses (+) or minuses (-). This indicate either additional points that were added although some errors, or additional points being deducted due to some other issue which we did not find the need to list here.

1. (33 points) Let S be a string. Write an algorithm that finds all non-empty substrings of S that occur in S the maximum number of times (i.e. there does not exist a substring that occurs more times than any of those).
   Describe your algorithm's idea and prove its time complexity. The faster the algorithm (for a correct one) the more points given.

   Answer:

   The idea is to use the suffix tree. Note that the number of leaves in the substring rooted at node t is the number of times the substring from the root to t occurs in S. The substrings that occur a maximum number of times have to be children of the root (since any grand-child of the root means there was a split at the level above and thus the parent had more leaves in its subtree).

   So, construct a suffix tree and do a DFS to write for each node how many leaves there are in its subtree. Now consider the children of the root, and choose all those with the maximum number of occurrences (leaves).

   Don't forget: All prefixes along the edge from the root to each maximum node chosen above are also substrings that occur a maximum

number of times. In fact, any substring along the edge appears a maximum number of times. However, in order to remove possible extra work it is crucial to notice that the prefixes suffice (as the other substrings must be prefixes along some other edge which will be part of the output).

Time: The input-dependent time is dominated by the suffix tree construction, i.e., $O(|S|+tocc)$ for finite fixed alphabets, $O(|S| \log\log |\Sigma|+ tocc)$, for alphabets whose size is a polynomial in $|S|$, and $O(|S| \log |\Sigma|+ tocc)$, for general alphabets, where $tocc$ is the number of reported maximum-repeat substrings (output dependent).

Error Codes and Penalties:

1. The root cannot be a candidate (as it corresponds to the empty string). Not noticing this was deducted 2 points.
2. Using a parameter of $O(n^2)$ for the size of the output (which is in fact linear). 7 points.
3. Considered size of subtree, and not number of leaves. 5 points.
4. Suffix tree construction in $O(n \log n)$ or $O(n|\Sigma|)$. 5 points.
5. Major lack of details. 12 points.
6. An $O(n^2)$ solution (using tries, for example). 13 points.
7. An error in the time complexity. 5 points
8. The naive $O(n^3)$ solution. 18 points.
9. Missing $\log |\Sigma|$ in the time for constructing a suffix tree (note that this was emphasized in the recitation) . 5 points.
10. Traversing all of the substrings along an edge, and not just the prefixes. 2 points.
11. Traversing only one edge (and not all of the possible edges). 2 points.
12. In addition to 6, error in suffix tree construction complexity. 2 points.
13. Returning a single node. 3 points.
14. Finding only nodes, and not the appropriate substrings. 5 points.
15. Does not traverse the prefixes at all. 5 points.
16. Decision is based on number of children, and not on number of leaves. 20 points.

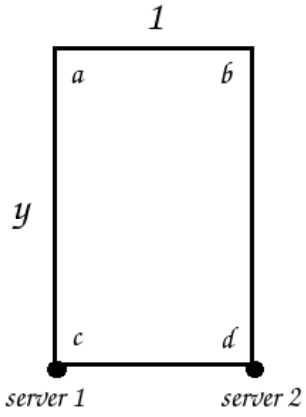
17. Returning the suffixes according to the leaves. 13 points.
18. A KMP based algorithm in $O(n)$ time. 20 points.

2. (33 points) Define a strategy **Balance1.175** for the two server problem as follows:
Given a service request of distance d_1 from server 1 and distance d_2 from server 2, and assuming that the distance travelled so far by server 1 is x_1 and the distance travelled so far by server 2 is x_2, the server that will handle this request is server i for which $x_i + 1.75d_i$ is smaller. Prove that the **Balance1.75** strategy is not 2-competitive.

Answer:

This was supposed to be a gift question since I assumed everyone would first try the example given in class for **Balance2**.

Consider the situation in the figure below, where servers 1 and 2 are at corners c and d of a a rectangle of height y and width 1. The service requests are at the corners of this rectangle and they appear in a cycle: $a, b, c, d, a, b, c, d, a, b, c, d....$



The optimum would be for one server to serve a and b, and the other server to serve c and d. However, for an appropriate choice of y, server 1 will serve a and c and server 2 will serve b and d. If y is greater than 2 this will mean that the algorithm is not 2-competitive.

What y should be chosen?
– The first service request will be handled by server 1. For the second to be handled by server 2 rather than server 1, we need to have:

$$y + \frac{7}{4} > \frac{7y}{4}$$

This happens when

$$\frac{7}{4} > \frac{3y}{4}$$
$$7 > 3y$$
$$\frac{7}{3} > y$$

Note that the above holds true also for the situation where, inductively, server 1 already traveled cy times and server 2 traveled $(c-1)y$ times, thus server 1 serves a, c and server 2 serves b, d.

Conclude: The competitive ration is greater than 2 for:

$$2\frac{1}{3} > y > 2.$$

Note that although not correct, we decided to accept answers which only considered a constant number of requests. Recall that the definition of competitiveness requires that there exists a constant n_0 such that for any sequence of requests of size n where $n > n_0$, the cost of the algorithm on the sequence will be at most twice the cost of the optimal algorithm. Choosing an example with a constant number of requests does not suffice to negate this, as it is possible that the constant n_0 is much larger.

Also, if you demonstrated that you really know what competitive analysis is, and what is needed to disprove the claim, you received at least 18 points.

Error Codes and Penalties:

1. The solution works for a constant number of requests, but there was an error in an attempt to generalize it to a non-constant number of requests.
2. Was accepted as correct.

3. Ignored.
4. The solution was not understood. This does not mean that the solution was incorrect. If you had this error (as listed in the notebook), you are welcome to submit an appeal better explaining what you meant to do. But notice that your appeal will only be used to understand what is already written in the notebook, and will in no way be able to add additional information. If you feel that you deserve more points, then please appeal. 33 points
5. Algorithm is executed in correctly. 8 points.
6. An incorrect example that depends on y, without noticing the dependency on y. 5 points.
7. y is not computed, or computed incorrectly. 5 points.
8. Missing computation of costs. 15 points.

3. (34 points) We say that an integer *a fits at neighborhood 3* with integer b if $|a - b| \leq 3$.
If S_1, S_2 are strings of length n of integers then string S_1 *fits at neighborhood 3* with string S_2 if for each $i = 1, ..., n$ the i-th symbol of S_1 fits at neighborhood 3 with the i-th symbol of S_2.
Example: $3, 17, -1, 4$ fits at neighborhood 3 with $4, 19, -2, 2$.

The *neighborhood-3 matching problem (N3MP)* is defined as follows:
INPUT: Text T of length n, and pattern P of length m, both over the integers.
OUTPUT: All indices i in T for which the pattern fits at neighborhood 3 with the substring of length m of the text that starts at location i.

Write an algorithm that solves N3MP. Describe the algorithm's idea and prove its time complexity. The faster the algorithm (if correct) the more points given.

Answer:

There are a number of possible answers, all using Abrahamson-like ideas.

Notation: Denote by $S + c$ $(S - c)$ The string S' where $S'[i] = S[i] + c$ $(S[i] - c)$.

The first option is using the algorithm we saw in class for counting matches. Run that algorithm 7 times. Count the number of matches

of $P+c$ in T, for $c = -3, -2, -1, 0, 1, 2, 3$. Add them all up. For every location where the sum is m there is a neighborhood-3 match.

Time: $O(n\sqrt{m \log m})$ for each c, and this is done 7 times, for a total of $O(n\sqrt{m \log m})$.

The second option notes the following: For every text location t and pattern location p, $|t - p| \leq 3$ iff $t - p \leq 3$ and $p - t \leq 3$ iff $t - 3 \leq p$ and $p \leq t + 3$. This means that we have a neighborhood-3 match iff P is within a "band" of the text $+3$ and the text -3.

Therefore do two less-than matchings. Check if $P \leq T + 3$ and $P \geq T - 3$. Every location where both hold has a neighborhood-3 match.

Time: $O(n\sqrt{m \log m})$ for each less-than matching. This is done twice for a total of $O(n\sqrt{m \log m})$.

Note: The second option has an added advantage that it has the same time complexity for *neighborhood-d match* for every given d.

Error Codes and Penalties:

1. Using KMP or witness tables or incorrect convolutions, each 7 times. 24 points.
2. Counting all of the errors 7 times, but in $O(n \log m)$. 10 points.
3. Naive solution. 19 points.
4. Less than matching in $O(n)$ time, or without providing its time complexity. 10 points.
5. Using less than matching but incorrectly. 5 points
6. $O(n\sqrt{m} \log m)$. 3 points.
7. Expanding KMP or another algorithm to work on the difference between P and T (missing the lack of transitivity). 29 points.
8. Using Clifford and Clifford ideas (which don't work here, but this is slightly tricky). 5 points.
9. Less than matching in $O(n \log m)$ time. 8 points.
10. Variation of Clifford and Clifford which doesn't remotely work. 14 points.
11. Abrahamson-like Algorithm, with a major lack of details, or a major misunderstanding. 10 points.

4. (5 points) Basic training at the end of the Yom Kippur war was identified by:

   (a) The helmets were from 1898.

   (b) The rifles were from 1898.

   (c) The overcoats were from 1898.

   (d) All answers are true.

   The correct answer is (b).

GOOD LUCK