Important Note: There were two main changes this year. The first was that the exam was in Hebrew, due to popular demand and to the fact that my laptop now supports Hebrew. The second change was that only 50% of the test evaluated algorithmic thinking (questions 3,4). The other 50% evaluated knowledge (I consider question 5 as such since I did say in class, when suffix trees were defined, that I think you should try to figure out at home an example of a quadratic space uncompressed suffix trie over a binary alphabet). There is, of course, a difference in grading thinking questions and knowledge questions. In thinking questions you get the points on correct thinking, not on style or mathematical rigor. In the knowledge questions, I expected the answers to be formulated in a mathematically coherent way, therefore, even if it was clear that the person "knew" the answer, if it was written in an unintelligible or mathematically improper manner, points were deducted (albeit, not a lot of points).

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

Instructor: *Prof. Amihood Amir*
Length of Exam: 2 hours
Time: July 23, 2009, 12:00
NO OUTSIDE MATERIAL ALLOWED!!!

1. (20 points) What is the solution of the following Linear Program:
   Objective Function: $\min x_1 + x_2 - x_3$
   Constraints: $x_1 - x_2 = 0$
   $x_3 - x_1 = 0$
   $x_1 + x_2 \geq 10$

   Answer:

   The first constraint means that $x_1 = x_2$. The second constraint means that $x_1 = x_3$. By transitivity, $x_2 = x_3$, i.e. $x_1 = x_2 = x_3$. The third constraint means that x_1, x_2, and x_3 all have values at least 5. Because all variables have equal value, the objective function means that we just take $\min x_1$ which, by the third constraint is 5. Thus, for values $x_1, x_2, x_3 = 5$ the objective function is minimized and the minimum value is 5.

   Error Codes and Penalties:

   1. Value given for only one of the variables, or to none. (-4)

   2., 3. Insufficient explanation. (-2)

   4. The student seems to not understand the meaning of *objective function*. (-10)

   5. No optimal solution explicitly given for the objective function. (-4)

6. Either an answer with no justification at all or improper proof style. (-6)
7. Tried to construct dual form - incorrect. (-10)
8. Wrote in matrix form and tried to reduce matrices making mistakes en route. (-10)
9. Error in objective function computation. (-4)
10. Complicated process and incorrect answer. (-10)

2. (20 points) Define an online version of the Bin-Packing problem as follows: The objects are not given as an input all at once but rather arrive over time. The objects are to be packed so that when a new object arrives all previous objects are already in bins, and an object can not be taken out of a bin once it is packed there.

   (a) Is there a competitive algorithm for the online bin-Packing problem?
   (b) If so, describe your algorithm and analyse its competitive ratio.
   (c) If not, justify.

Answer:

Use the FF algorithm. It packs the elements one by one in order of arrival and no item is taken out of a bin once it is packed. Johnson, Demers, Ullman, Garey and Graham [1974] proved that the approximation ratio of FF is $\frac{17}{10}$. The approximation ratio is also the competitive ratio since it measures how close to the optimal offline solution the obtained result is.

Error Codes and Penalties:

The first four codes below are positive because they are all the elements needed for a correct answer (summing up to 20 points). any missing element of these caused reduction in the appropriate amount of points. The other codes are errors and their penalties are attached.

1. Identified that FF is a good solution. (+10)
2. Correctly observes why FF is online. (+4)
3. Correctly observes why the approximation ratio is the competitive ratio. (+4)
4. Knows that FF has a constant approximation ratio. (+2)
5. Supplies a non-competitive algorithm, dependent on the size or number of input items. (-20)
6. Describes an algorithm for a special case of the input. (-10)
7. Almost correct. (-3)
8. Correct algorithm but incorrect proof of competitiveness. (-10)
9. Provides an algorithm that is not online. (-20)
10. Defines competitiveness incorrectly. (-10)

3. (20 points) Define an *NE matching* of pattern $P = P_1 \cdots P_m$ at location i of text $T = T_1 \cdots T_n$ if $T_{i+j-1} \neq P_j, \quad j = 1, ..., m$.

   Let P and T be strings over alphabet $\Sigma = \{A, B\}$.

   Describe an algorithm that outputs all text locations where P NE-matches. What is the time complexity of your algorithm?

   Answer:

   Simply exchange in the pattern every A by a B and every B by an A and run KMP or the witness table algorithm. Every match means that all locations of the original P do not match. The time is $O(n)$.

   Error Codes and Penalties:

   This problem can be solved via different algorithms with different complexities. As always, the naive algorithm gives some points. In fact, one will generally get a higher grade writing the naive algorithm correctly, than by providing a complex algorithm and incorrectly analysing it. The first two codes are positive because they represent the naive algorithm, and the convolutions algorithm, that have complexity $O(nm)$ and $O(n \log m)$, respectively. Full points are given for a linear-time algorithm. The codes 3-10 are negative and represent errors.

   1. Naive algorithm. (+6)
   2. Convolutions algorithm. (+12)
   3. Running KMP on "don't care"s. (-15)
   4. Checking for a **single** mismatch, rather than **all** not equal. (-15)
   5. Using suffix trees for a case that has no transitivity. (-15)
   6. Providing an $O(nm)$ time algorithm and claiming it is linear. (-15)
   7. Doing Boolean operations on arrays in constant time. (-15)
   8. Figured out the inverse idea but then invented a wrong exact matching algorithm. (-8)
   9. Did a straight convolution on the input strings without translating to the appropriate characteristic functions. (-15)
   10. Correct algorithm, not justifications. (-10)
   11. Correct algorithm, wrong time analysis. (-10)

4. (30 points) Let Σ be an alphabet of k symbols. Describe an algorithm that outputs all text locations where P NE-matches. What is the time complexity of your algorithm?

Answer: For every symbol $\sigma \in \Sigma$ compute $\chi_\sigma(T) \times \chi_\sigma(P)^R$. This will compute, for every text location, the number of times that a σ in the pattern matches a σ in the text. Add up all the result vectors and every location that is a 0 has no match, i.e. has an NE-match.

Error Codes and Penalties:

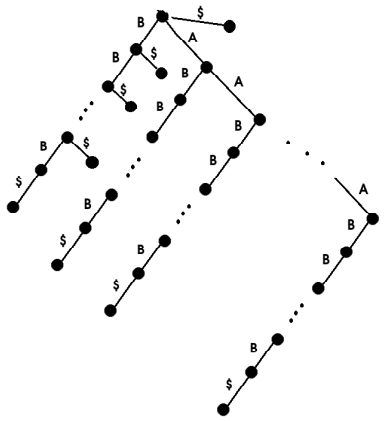
Again, the naive algorithm got partial credit, thus the second code is positive, this is the number of point that you got for the naive algorithm. However, if someone wrote the naive algorithm and then got smart and provided a wrong algorithm, I deducted 5 points (for making me read nonsense at 2 in the morning :-).

1. Straight representation of a general alphabet by $\log n$ bits. (-20)
2. Naive algorithm (+15)
3. Adding an incorrect algorithm to the naive response. (-5)
4. Running KMP on "don't care"s. (-22)
5. Checking for a **single** mismatch, rather than **all** not equal. (-22)
6. Providing an $O(nm)$ time algorithm and claiming it is linear. (-18)
7. Using witness tables for a non transitive matching relation. (-22)
8. Using Suffix trees for a non-transitive matching relation. (-22)
9. Incorrect reduction to binary alphabets. (-18)
10. Totally incoherent answer. (-25)
11. Glimmer of idea, no details, no complexity analysis. (-12)
12. Representation of general alphabet by $\log n$ bits, but an unsuccessful attempt to adjust solution. (-12)
13. Incorrect usage of $\leq$-matching. (-22)
14. Parameterized matching without adjusting answer. (-20)
15. Exponential algorithm, with correct time analysis. (-15)
16. Same as [12.].
17. Minor errors. (-5)
18. Right idea, errors in proof and analysis. (-10)

5. (10 points) Give an example of a string $P = P_1 \cdots P_m$ over alphabet $\Sigma = \{A, B\}$ whose uncompressed suffix tree has size $\Theta(m^2)$.

Answer:

Take $P = A^{m/2}B^{m/2}\$$. The suffix tree has $\frac{m}{2}$ nodes where the edge leading to them is labeled A and that have an outgoing path of $\frac{m}{2}$ B's, for a total of at least $\frac{m^2}{4}$ nodes.



Error Codes and Penalties:

The idea presented above is easy to prove. Many unbelievably complex and creative ideas were suggested without proof or with a bogus proof. I tried to compute some of them and realized that some are indeed not linear, yet also not quadratic. Then I gave up trying to analyse all of them (3 in the morning) and deducted points if the proofs were not correct (as was the general case for thee complex solutions).

1. A^n is linear. (-10)
2. Single finite string. How do you know it is quadratic? Maybe it just has a large multiplicative constant in the big "O"? (-10)
3. Le σ be a given finite string. $\sigma^{n/|\sigma|}$ gives a tree of size $O(n|\sigma|)$, which is linear for a fixed σ. (-7)
4. $ABA^2B^2 \cdots A^{\sqrt{m}}B^{\sqrt{m}}$ with wrong proof. (-5)
5. An example with no proof. (-5)
6. Incomplete proof. (-3)
7. Encoding in binary the infinite alphabet example. Missing the fact that there are many common prefixes now. (-7)
8. An example on a non-binary alphabet. (-10)
9. Bizarre examples with no proofs. (-8)
10. $ABA^2B^2A^3B^3 \cdots A^{\sqrt{n}}B^{\sqrt{n}}$ with bad proof. (-5)

GOOD LUCK