

15-750-S08:**Midterm 1**

February 27, 2008

Name:

Email:

Problem 1

Describe how to test if a given vertex of P is visible to q .

The point p_i is visible to q if $E(i-1,i)$ or $E(i,i+1)$ is visible.

Let $\text{visible}(p_i)$ be this visibility test.

Another useful test is whether a small movement along $E(i,i+1)$ starting from p_i gets us closer to q . This is just a dot product. Say $\text{closer}(i,i+1)$.

Suppose you are given a point p_i on P that is visible to q . Given an $O(\log n)$ time algorithm to find the closest point on P to q .

Here we will do a binary search.

WLOG we may assume that $\text{Closer}(i,i+1)$. Everything is modulo n .

Set $p_j = p_{\{i-1\}}$

Procedure $\text{Search}(p_i, p_j)$

```

While  $j-i > 1$  do
  set  $p_k = p_{\{(j-i)/2\}}$ 
  if  $\text{Visible}(p_k)$  and  $\text{Closer}(p_k, p_{\{k+1\}})$  then  $\text{Search}(p_k, p_j)$ 
  else  $\text{Search}(p_i, p_k)$ 

```

Finally we find closest point on the edge $E(i,j)$

Given an $O(\log n)$ time algorithm to find a visible point on P to q .

If p_1 or p_n is visible we are done.

Set $p_i = p_1$ and $p_j = p_n$

Procedure FindVisible(p_i, p_j)

 set $p_k = p_{\lfloor (j-i)/2 \rfloor}$

 if Visible(p_k) return p_k

 else if p_k above line (p_i, p_j) then FindVisible(p_i, p_k)

 else FindVisible(p_k, p_j)

Give an $O(n)$ algorithm to find the closest point on P to q .

Here we just find the closet point on each edge and return the closet one over all the edges.

Given an explanation why any algorithm to find the closest point on P to q must take $\Omega(n)$ time. (Hint: Consider adversarial argument where the location of the points of P are set only when a test that involves the point is run. Describe how an adversary could force any closest point algorithm to inspect a constant fraction of the points before it would be able to determine the closest point.)

Here we start with a regular n -gon centered at the origin, say G_n . We also set q to the origin.

As an adversary we move the the last point queried by the algorithm so that it is the closest point to q .

Thus the closest point is not even determined until the algorithm queries $n-1$ points.

Problem 2

Give a description of the subproblems you are solving.

Let $D(i, j)$ be the minimum cost of copy-editing $x[1..i]$ to $y[1..j]$.

Write the recurrence relation for your subproblems including the base cases.

Base case is $D(1, 1) = 0$.

We have

$$D(i, j) = \min \begin{cases} D(i, j-1) + C, & \text{if } x[i] = y[j-1] \text{ and } j \geq 2 \\ D(i, j-1) + I, & \text{if } j \geq 1 \\ D(i-1, j) + S, & \text{if } i \geq 1 \end{cases}$$

Explain why your recurrences are correct.

We will argue that $D(m, n+1)$ gives the minimum cost of copy-editing $X[1..m]$ into $Z[1..n]$ by strong induction.

For $i = j = 1$, the minimum cost of copy-editing an empty string into another empty string is 0.

Assume that for pair (i, j) , $D(i, j)$ is the minimum cost. At this point, there were three possibilities for the last operation performed. If $x[i] = y[j - 1]$ and $j \geq 2$, then the last operation could have been 'copy', in which case the cost would be $D(i, j - 1) + C$. If $j \geq 1$, then this operation could have been 'skip' with total cost $D(i - 1, j) + S$. Lastly we could have inserted a new character, with cost $D(i, j - 1) + I$. By inductive hypothesis, we know that the costs $D(i, j - 1)$ and $D(i - 1, j)$ are optimal, hence $D(i, j)$ is optimal as well.

A useful observation here is to view the entire computation as a graph, where table entries (i, j) correspond to nodes and different operations (copy, skip, insert) correspond to edges with corresponding costs as weights. Then the whole problem becomes searching for a shortest path from node $(1, 1)$ to the node $(i, n + 1)$ with minimum distance.

Describe an algorithm to find the cyclic edit distance in polynomial time. A runtime of $O(nm^2)$ will suffice for this part.

A naive idea will be to construct the string `cyclic_shift(X, l)` for every possible l , compute the copy-edit distance in $O(nm)$ time and output the one with lowest cost. Since there are m possible values for l , the whole operation will take $O(nmm) = O(nm^2)$ time.

Describe how we might view all the m optimal copy-edits, one for each cyclic shift of X in a single table of size n by $2m$.

Consider constructing the above graph for strings Y and $X \oplus X$. Now we can view a copy-edit with shift l as the weight of shortest path from node $(l, 1)$ to $(l + k, n + 1)$ for $m \geq k \geq 0$ which gives the minimum cost possible.

Describe why we may pick the m optimal copy-edits to be non-crossing.

In the graph view, if two optimal copy-edits are crossing, then this means that these paths intersect at one point and diverge at a later point (ie. after their intersection, they don't lie on top of each other).

Assume two paths, which represent optimal copy-edits for different shifting amounts l and l' , were crossing. Let the cost of first path be $d_1 + d'_1$, the second one be $d_2 + d'_2$, where $d_{1,2}$ are the costs up to the crossing point, and $d'_{1,2}$ are the costs after that point. Let $d'_1 \leq d'_2$ without loss of generality. Then we can modify the second path so that it uses the edges first path after intersection, without increasing its cost. Therefore we can always make paths non-crossing.

Give an algorithm to find all these m non-crossing copy-edits in $O(nm \log m)$ time.

With the above observation, we can first find the shortest path for shift $m/2$. Because of the above property, we know that the solution for any shift smaller than $m/2$ can't cross this shortest path, therefore we can break the table into two pieces. Each piece will have size at most $(n + 1)(3m/4)$. For each piece, we can again find the shortest path for the middle shift, which will cut the size of resulting tables by at least a fraction of $3/4$. Then the total running time will be

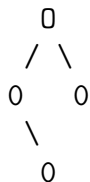
$$T(n, m) \leq T(n, 3m/4) + O(nm)$$

whose solution is $T(n, m) = O(nm \log m)$ as expected.

Problem 3

Problem 1: The Treap Problem (30 pts)

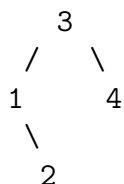
1. Suppose that 4 elements are stored in a treap. What is the probability that the tree has following structure:



Explain your answer.

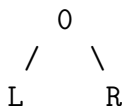
2. Let T be a binary tree on n nodes, and L and R be its left and right subtrees, respectively. Let $Prob(T)$ be the probability that we generate T as a treap on $|T|$ nodes. Write a recurrence for $Prob(T)$ in terms of $Prob(L)$ and $Prob(R)$. Give an explanation why your recurrence is correct.

1. Suppose the keys of the tree are 1,2,3,4. We know its shape is



Among all the 24 possible permutation of the keys, only 3124, 3412, 3142 will generate the tree above. Therefore, the probability is $3/24 = 1/8$

2. We know that to have the tree of the shape



The $(L+1)$ th key must be on the top. It is of probability $1/|T|$ to assign it the top priority. Given the $(L+1)$ th key is assigned the top priority, first $|L|$ keys will always be at left subtree and last $|R|$ nodes will be at the right subtree. Then the probability of the left subtree to have its shape L is $Prob(L)$ and the right subtree to have its shape R is $Prob(R)$. Notice all these events are also independent.

Overall, we have $Prob(T) = 1/T * Prob(L) * Prob(R)$