15-750 — Graduate Algorithms — Spring 2008

Miller and Sinop and Wu

Assignment 2 Due date: Friday, February 22

1 Simple Paths and Convex Hull

[20 points]

Suppose that $P = \{p_1, \ldots, p_n\}$ is a set of points in the plane. We say the the sequences of distinct points $Path = (p_1, \ldots, p_k)$ is a simple path if the line segments $l_i = [p_i p_{i+1}]$ are disjoint except for $l_i \cap l_{i+1} = p_{i+1}$. We may also allow $p_1 = p_k$ and in this case $l_{k-1} \cap l_1 = p_k$.

In the following questions we shall investigate the relation between finding a simple path of a set of points and finding their convex hull.

1. Design an algorithm for finding a simple path using the points P. Make your algorithm as time efficient as possible.
2. In class we showed that computing the convex hull of n points in a comparison basedN - model requires $\Omega(n \log n)$ time. Show that given a simple path for these points one can find the convex hull in $O(n)$ time.

   HINT:

   The idea is to run a variant of incremental convex hull where we add the points in the order they appear on the path. Suppose we are give a simple path $Path = (p_1, \ldots, p_n)$ on n distinct points and for simplicity no three are collinear. We start by constructing the triangle from the first three points and storing it as a doubly linked list of edges and recording which vertex is connected to the remain points on the path.

   Let $I = \{i \mid p_i \in CH(p_1, \ldots, p_i)\}$ We will for each $i \in I$ incrementally compute the convex hull of $(p_1, \ldots, p_i)$. Make sure your algorithm handles the case when the point p_{i+1} is interior to $CH(p_1, \ldots, p_i)$.

   Use amortized analysis to show that your algorithm runs in $O(n)$ time.
3. Show that in general any comparison based algorithm that finds a simple path of the points in P requires $\Omega(n \log n)$ comparisons.

Solution:

1. Sort all the vertex by their X coordinate and if there is tie, sort them by Y coordinate. Connect the vertex by this order and it is easy to see this will form a simple path. The overall work is $O(n \log n)$.
2. We give following algorithm that is first found by Melkman. Suppose $CCW(v_1, v_2, v_3)$ is a function to judge whether v_1, v_2, v_3 is counter clockwise

Algorithm 1 Melkman's Algorithm

1: Start from v_1, v_2, v_3 to be counter clockwise and let double-end queue $d =< v_3, v_1, v_2, v_3 >$.
   $//d =< d_b..., d_t >$ in general, we can access elements from both top and bottom.
2: let i = 4
3: **while** $i \leq n$ **do**
4:   **if** CCW(d_{t-1}, d_t, v_i) AND CCW(d_b, d_{b+1}, v_i) **then**
5:     i = i+1 //this is the case that v_i is in previous vertices' convex hull.
6:   **else**
7:     **while** CCW(d_{t-1}, d_t, v_i) **do**
8:       d.top_pop(d_t)
9:     **end while**
10:     d.top_push(v_i)
11:     **while** CCW(v_i, d_b, d_{b+1}) **do**
12:       d.bottom_pop(d_b)
13:     **end while**
14:     d.bottom_push(v_i)
15:     i=i+1
16:   **end if**
17: **end while**

The timing analysis of this algorithm is very similar to the buildtent function in the class. Basically,when you push some element v_i at bottom or top, you put a credit on it. These credits will pay for the cost when v_i is poped out. And we know the push operation only runs O(n) time. So overall, this is an O(n) algorithm.

To see more of details of Melkman's algorithm, you can check the following web site:

http://cgm.cs.mcgill.ca/~athens/cs601/Melkman.html

3. Since there is a $\Omega(n \log n)$ lower bound for building convex hull. If we can improve the algorithm for finding simple path, it will contradict the lower bound we have for finding convex hull.

2 Smallest Enclosing Rectangle(Ying Liu)

[10 points] Suppose you are given a convex polygon of n points in two dimensional space, give an algorithm that returns the smallest(in terms of area) rectangle containing all the n points. Note that a rectangle's edge does not need to be vertical or horizontal. Show the correctness of your algorithm and analyze its time complexity. For full credit your algorithm should run in linear time.

HINT: You may use the following fact without proof. The boundary of the minimum enclosing rectangle must include an edge from the convex hull.

Solution: Here is the algorithm:

1. find the points with maximum and minimum x coordinate, let them be p_i and p_j

2. find the points with maximum and minimum y coordinate, let them be p_l and p_m

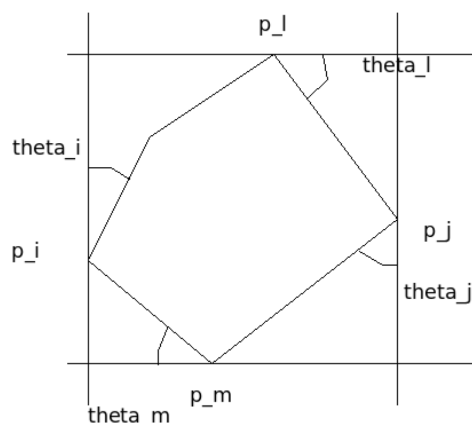

Figure 1:

3. construct two vertical lines of support through p_i and p_j, let them be L_i and L_j, respectively
4. construct two horizontal lines of support through p_l and p_k, let them be L_l and L_k, respectively
5. let *min_rec* be the area of the rectangle formed by the four lines
6. Let count to be the number of edges on four lines.
7. If (count == n), then program finish, return *min_rec*.
8. calculate the four angles formed by L_i and $\overline{p_i p_{i+1}}$, L_j and $\overline{p_j p_{j+1}} \ldots$, denote them as θ_i, θ_j, θ_l, and θ_k, respectively.
9. let $\theta = \min\{\theta_i, \theta_j, \theta_l, \theta_k\}$, turn each line of angle θ clockwise, let k be the number of edges that becomes on one of the lines and we recalculate by $count = count + k$.
10. calculate the area of the new rectangle and compare it with *min_rec*, if it is smaller than *min_rec*, then update *min_rec* with this new value.
11. go to step 7.

Correctness: We know the boundary of the minimum enclosing rectangle must include an edge from the convex polygon. This algorithm just enumerates every minimum enclosing rectangle satisfying this criterion and choose the smallest one.

Analysis: step 1-6 takes $O(n)$ time; each iteration of step 7-10 takes constant time, and there are at most n iterations(each time we check at least one new edge). So the total time is $O(n)$.

3 Making a Palindrome

[20 points]

Suppose we have two sets of code words V and W with n words each where the longest word has length l. Assume we are working over the binary alphabet.

3.1 Common Message

A common message of codes V and W is a message that can be formed from code words in V and from code words in W.

- Give (and prove) an upper bound on the length of the shortest common message of V and W. Here, length refers to number of bits, not number of words. (Try to make your bound as good as possible. You don't need to show a lower bound.)
- Give an $O(l^2n^2)$ time algorithm to determine if we can form a message from V that is also a message from W. You are allowed to uses the same word more than once.

3.2 Palindrome

Give an $O(l^2n^2)$ time algorithm to determine if we can form a palindrome using words in W. You are allowed to uses the same word more than once.

Recall: A palindrome is a string m which is equal to its reversal, e.g. "010010" and "00100".

Hint: We recommend you do the common message part first, if you haven't already.

Solution:

1. Define a *tail* in V to be a nonempty word in Σ^* such that there exists messages $v_1, \ldots, v_s$ and $w_1, \ldots, w_t$ for which the following hold.
   (a) $v_i \in V, w_i \in W$,
   (b) $v_1, \ldots, v_s = w_1, \ldots, w_t$, and
   (c) t is a suffix of v_s.
   - There are $O(nl)$ tails for V and W because we can form each one by choosing a word from V or W and a cutting it off at one of at most l places. If in the course of building the messages the same tail occurs twice, then the sections of each message between the repeated tails can be eliminated, yielding a shorter message and a contradiction. So, we may assume that each tail occurs at most once. Thus, the messages are built from $O(nl)$ tails, each of length $O(l)$. It follows that the shortest common message has length $O(nl^2)$.
   - We give an algorithm using dynamic programming. We create arrays $Vtails$ and $Wtails$ of $O(nl)$ suffixes of words in V and W respectively. The goal is to determine if each suffix is a tail for V (or W resp.). Each time we find a new tail, we push it on a stack.
     (a) Seed the table: For each pair $(v, w) \in V \times W$
       - IF $w = v$ THEN return TRUE. (Common Message exists)
       - IF $\exists t$ such that $v = tw$ then $Vtails[t] = 1$
       - IF $\exists t$ such that $vt = w$ then $Wtails[t] = 1$
     (b) While there is a tail t we haven't looked at yet.
       - POP t off the new tails stack.
       - IF $Vtails[t] = 1$ THEN
         * FOR ALL $w \in W$

  - $*$ IF $t = w$ THEN return TRUE.
  - $*$ IF $\exists s$ such that $v = tw$ then $Vtails[s] = 1$
  - $*$ END FOR
- – END IF
- – Repeat for the case for when $Wtails[t] = 1$

(c) return FALSE

Proof of Correctness

If the algorithm terminates during the seed phase, then there is a common word in W and V therefore there is trivially a common message.

If the algorithm terminates during the main loop, then it must have returned TRUE. We will show that there must be a common message.

Each time a tail t_i is added to $Vtails$ or $Wtails$ during the main loop of the algorithm, it happens while processing another tail t'_i and a word w_i. Therefore, at the completion of the algorithm, it is possible to take any tail that was considered and trace it back to one of the seed tails. In particular, if the algorithm returns TRUE then there is such a sequence that ends with $t'_i = w_i$. The sequence of w_i's gives a common message.

Suppose that there exists a common message $v_1 \dots v_r = w_1 \dots w_s$. We want to prove that our algorithm will return TRUE. If we construct the common messages one word at a time, so that no message is always longer than the other by some tail, we get a sequence T of tails. We observe that the first word in this must be a single word and thus is added to $Vtails$ or $Wtails$ in the seeding phase of the algorithm. We further observe that for any tail $t_i \in T$, if T is checked by the algorithm, then tail t_{i+1} will be added to $Vtails$ or $Wtails$. It follows by induction that every tail in T is checked by the algorithm. The last tail checked will be matched by a word in V or W and thus the algorithm will return $TRUE$.

Timing Analysis

(a) Seeding the table takes time $O(ln^2)$ because we check all pairs of words and each check takes time $O(l)$.

(b) There are $O(nl)$ tails. For each tail, we try appending all words from V or W to get a new tail. So, we do $O(nl)$ work per tail. Total, the running time is $O(n^2l^2)$.

2. Construct a set of codewords V such that each $v_i \in V$ is the reverse of a word $w_i \in W$. The algorithm to check if there is a palindrome is to simply check if there is a common message in V and W. We claim that there is a palindrome in W if and only if there is a common word in V and W.

   Proof of Correctness

   $Common - Message(V, W) \Rightarrow Palindrome(W)$
   Let $M = v_1 \dots v_r = w_1 \dots w_s$ be a common message in V and W. Say $\overline{M}$ is the message formed by reversing the letters in M. In particular $\overline{M} = \overline{v_r} \dots \overline{v_1}$. Since each $\overline{v_i}$ is a word in W, it follows that $\overline{M}$ is a message in W. Therefore $M\overline{M}$ is a message in W and it is a palindrome.

   $Palindrome(W) \Rightarrow Common - Message(V, W)$
   Let $M = w_1 \dots w_s$ be a palindrome in W. Then $M = \overline{M} = \overline{w_s} \dots \overline{w_1}$. Therefore, M is a message common to V and W.