Data Structure and Algorithm II

Homework #1

Due: 12pm, Tuesday, October 11, 2011

=== Homework submission instructions ===

- Submit the answers for writing problems (including your programming report) through the CEIBA system (electronic copy) or to the TA in R432 (hard copy). Please write down your name and school ID in the header of your documents. You also need to submit your programming assignment (problem 1) to the Judgegirl System(`http://katrina.csie.ntu.edu.tw/judgegirl/`).

- Each student may only choose to submit the homework in one way; either all as hard copies or all through CEIBA except the programming assignment. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted through CEIBA (the part that the TA chooses).

- If you choose to submit the answers of the writing problems through CEIBA, please combine the answers of all writing problems into only one file in the doc/docx or pdf format, with the file name in the format of "hw1_[student ID].{pdf,docx,doc}" (e.g. "hw1_b99902010.pdf"); otherwise, you might only get the score of one of the files (the one that the TA chooses).

Problem 1. (30%) Given a set of n ($0 \leq n \leq 10000$) points $P = \{p_1, p_2, \ldots, p_n\}$ in a 2-dimensional space, where each point p_i can we represented with a 2-dimensional coordiante (x_i, y_i) and $-15000 \leq x_i, y_i \leq 15000$, for $i = 1$ to n. Please derive and implement an algorithm based on the divide-and-conquer strategy to determine the closest pair of points. Submit your program to the judgegirl system.

Please also submit a report in which you give a clear description of your algorithm. Write down the recurrence which represents the running time of your algorithm and solve the recurrence in the report too. Out of the 30 points of this problem, 10 points are for the report and 20 points are for the correctness of the 10 test cases on the judgegirl system.

Input: $n+1$ lines. n, the number of points in P, is in the first line. In the next n lines, each has x_i y_i to represent the coordinate of point p_i, for $i = 1, 2, ..., n$. x_i and y_i are separated by a space character (' ').

Output: 1 line. Return $\|u-v\|^2$, the square of the distance between u and v, where (u,v) is the closest pair of points in P.

Example:

Input:

5

0 2

6 67

43 71

39 107

189 140

Output:

1312

Sol: We can use recursive divide-and-conquer approach to solve the problem in $O(n \log n)$.

1. Step 1: Sort all the points based on the x coordinate.
2. Step 2: Find the median point x_{mid} in x coordinate. Set a vertical line through the x_{mid}, then split all the point into two equal-sized part.
3. Step 3: Divide the problem into left part and right part of the original problem, then solve these problem recursively. We can get the d_{Lmin} and d_{Rmin} as the minimum distance of the left part and right part of the original problem.
4. Step 4: Find the minimum distance d_{LRmin} between the left part and right part, that one point is belong to the left part and the other is belong to the right part.
5. Step 5. The last step is find the minimum in the d_{Lmin}, d_{Rmin} and d_{LRmin}.

For the first step, we sort all the points based on x coordinate in $O(n \log n)$. Secondly, it took $O(1)$. Step 3 recursively find the d_{Lmin} and d_{Rmin} until the part left one point or two points. Step 4 need $O(n)$. During the process of finding the minimum between the left

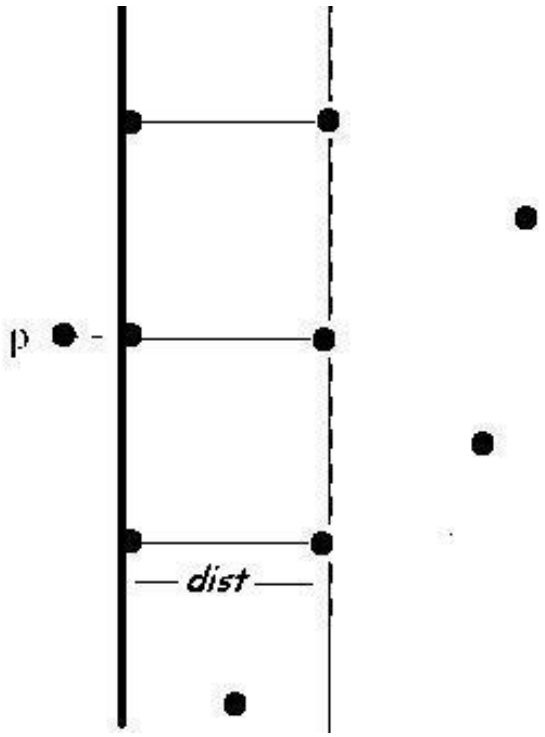


Figure 1: A point needs to compare with at most six points

part and the right, we didn't have to compare all the points. Set d the minimum of the d_{Lmin} and d_{Rmin}, and set the line $x = L$ that split all the points into two equal-sized part. Then we choose the points that lie within the two vertical lines $x = L - d$ and $x = L + d$ for comparison. There are at most six points lying in the rectangle of dimensions (d, 2*d) 1. Each point lying in the left part of this area needs to be compared with at most six points that lie in the rectangle. Therefore, step 4 and 5 need $O(cn)$ that c is a constant. Then the recurrence relation for this problem can be written as $T(n) = 2T(n/2) + O(n)$, which we can solve by using the master theorem to get $O(n \log n)$. You may find hw1 sample code from the url [1].

Problem 2. (20%) Assume that n is a positive integer which is a power of two. We have a chessboard which is of size n x n with a randomly selected single tile taken away (see Figure 2(a)). You are asked to cover the entire chessboard with the piece shown in Figure 2(b). For example, see Figure 2(c). Note that the piece can be rotated. All the tiles on the chessboard except the one taken way have to be covered, and no tiles can

[1] `http://www.csie.ntu.edu.tw/~r00922113/hw1.c`

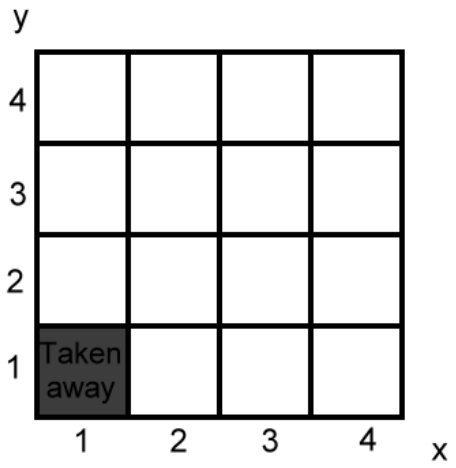
be left outside of the chessboard. In this problem, we ask you to design an algorithm to solve this problem with the divide-and-conquer strategy. Analyze the running time of your algorithm by using the recurrence.

Sol: We can divide the input square S into 4 half-length squares, $S = \{S_I, S_{II}, S_{III}, S_{IV}\}$, which locate in different quadrant. Firstly, we find out the quadrant S_t which taken-away tile locates in. Then, we put an L-shape piece on the center of S, and the quadrant of the missing tile of this piece is the same with S_t. After putting an L-shape piece, four half-length squares all contain a 1x1 taken-away tile. Thus, we can divide the original problem into four smaller problems. Algorithm 1 gives a pseudo code of this problem. Suppose n is the length of the chessboard. The recurrence form of the algorithm 1 is $T(n) = 4T(\frac{n}{2}) + \Theta(1)$. Thus, $T(n) = \Theta(n^2)$ by Master theorem.

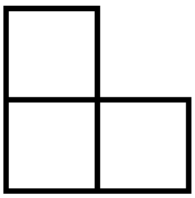
Algorithm 1 DecomposeSquare($centerPt$,$takenPt$,len)

```
takenQuadrant ← GetQuadrant(centerPt, takenPt)
AddLShape(centerPt, takenQuadrant)
if (len > 2) then
    for newQuadrant ← I to IV do
        newQuarant ← GetNewCenterPt(newQuadrant, centerPtr, len)
        if (newQuadrant == takenQuadrant) then
            newTakenPt ← takenPt
        else
            newTakenPt ← GetNewTakenPt(newQuadrant, centerPtr)
        end if
        DecomposeSquare(newCenterPt, newTakenPt, len/2)
    end for
end if
```

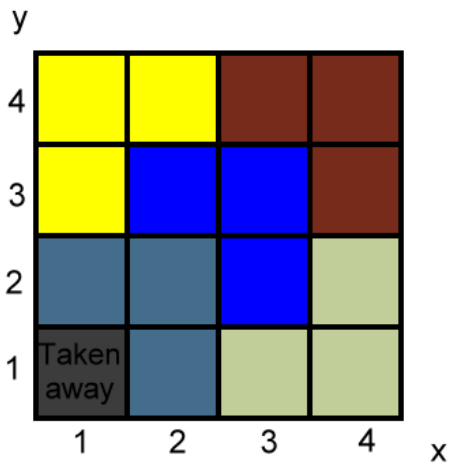
Problem 3. (16%) Consider the multiselection problem: given a set S of n elements and a set K of r ranks $k_1, k_2, ..., k_r$, find the k_1th, k_2th, ..., k_rth smallest elements. For example, if $K = \{2, 7, 9, 50\}$, the problem is to find the 2nd, 7th, 9th, and 50th smallest elements. This problem can be solved trivially in $\Theta(rn)$ by using the selection algorithm we talked



(a) The chessboard with 1 tile taken away



(b) The piece to be filled in the chessboard



(c) The chessboard covered by the pieces

Figure 2: The chessboard tiling problem

about in the class (run the algorithm for r times, once for each rank k_j, $1 \leq j \leq r$. Give an $O(n \log r)$ time algorithm to solve this problem. Prove that your algorithm indeed has that running time.

Sol: Suppose K is sorted in ascending order such that $k_1 \leq k_2 \leq ... \leq k_r$. Before doing the selection, we build a *height-balanced binary search tree* on K, denoted by $BST(K)$, which needs $O(r \log r)$ time, with the root node having the rank value $k_{\lceil r/2 \rceil}$.

After $BST(K)$ is built, we traverse $BST(K)$ by breadth-first search starting from the root, so that we can apply the linear time selection algorithm more efficiently by restricting the size of the input array. Suppose the rank of the left child and the right child of the root is k_i and k_j, respectively. First we apply the selection algorithm to find the $k_{\lceil r/2 \rceil}$th smallest element $s_{\lceil r/2 \rceil}$ in K. Then we classify S into two sets A_1 and A_2 by comparing those elements in S to $s_{\lceil r/2 \rceil}$, that is, for each element $x \neq s_{\lceil r/2 \rceil}$ in S, if $x < s_{\lceil r/2 \rceil}$ then $x \in A_1$, otherwise $x \in A_2$. After the classification, we put sufficient numbers of $s_{\lceil r/2 \rceil}$ into A_1 and A_2 such that $|A_1| = k_{\lceil r/2 \rceil} - 1$ and $|A_2| = n - k_{\lceil r/2 \rceil}$. This can be done in time $O(n)$.

Similarly, we can recursively call the same procedure to find the k_ith and the k_jth elements in S by finding the k_ith smallest element in A_1 and the $(k_j - k_{\lceil r/2 \rceil})$th smallest element in A_2. At this level, we need $O(|A_1|) + O(|A_2|)$ time to find these two elements. Since $|A_1| + |A_2| < n$, it's easy to verify that $O(|A_1|) + O(|A_2|) = O(n)$; we can further conclude by induction that each level in $BST(K)$ costs $O(n)$ time to finish the above procedure. Therefore, the k_1th, k_2th, ..., k_rth smallest elements can be found in time $O(n \log r)$ by traversing all nodes in $BST(K)$.

Problem 4. Solve the following problems on the textbook:

1. (5%) 4.4-6 on p.93
2. (5%) 4.4-8 on p.93
3. (4%) Use the master theorem to solve the recurrence in 4-1 (c,d,e,f) on p.107.

Sol:

1. Since the highest leaf of the recursion tree is located at the $(\log_3 n)$-th level, and each level in the recursion tree costs cn, $T(n) \geq cn \log n = \Omega(n \lg n)$.

2. Observe that the cost of the ith level in the recursion tree is $c(n-ia)+T(a)$. Since the height of the recursion tree is bounded by $\lceil n/a \rceil$,

$$\begin{aligned} T(n) &= \sum_{i=0}^{\lceil n/a \rceil} (c(n-ia)+T(a)) \\ &= cn(\lceil n/a \rceil + 1) - ca \sum_{i=0}^{\lceil n/a \rceil} i + (\lceil n/a \rceil + 1)T(a) \\ &= \Theta(n^2). \end{aligned}$$

3. (c) $T(n) = 16T(n/4) + n^2$
   $\Rightarrow T(n) = \Theta(n^2 \log n)$
   (d) $T(n) = 7T(n/3) + n^2$
   $\Rightarrow T(n) = \Theta(n^2)$
   (e) $T(n) = 7T(n/2) + n^2$
   $\Rightarrow T(n) = \Theta(n^{\log_2 7})$
   (f) $T(n) = 2T(n/4) + \sqrt{n}$
   $\Rightarrow T(n) = \Theta(\sqrt{n} \log n)$

Problem 5. (20%) Solve Problem 4-6 on p.110-111 of the textbook.

Sol:

a. If an array A is Monge, by definition consider $k = i+1$ and $l = j+1$, then it is trivial that $A[i,j] + A[i+1,j+1] \leq A[i,j+1] + A[i+1,j]$. For the "if" part, suppose that $A[i,j] + A[i+1,j+1] \leq A[i,j+1] + A[i+1,j]$, for all $i = 1,2,...,m-1$ and $j = 1,2,...,n-1$. Then for all $i < k \leq m$ and $j < l \leq n$, $(A[i,j] + A[k,l]) - (A[i,l] + A[k,j]) = \sum_{r=i}^{k-1} \sum_{s=j}^{l-1} [(A[r,s] + A[r+1,s+1]) - (A[r,s+1] + A[r+1,s])] \leq 0 \Rightarrow A[i,j] + A[k,l] \leq A[i,l] + A[k,j]$.

b. Let $A[2,3] = 5$, then it's easy to verify that A is Monge using the property that $A[i,j] + A[i+1,j+1] \leq A[i,j+1] + A[i+1,j]$, for all $i = 1,2,3,4$ and $j = 1,2,3$.

c. Let $A : m \times n$ be a Monge array. Suppose on the contrary that there exists i,j such that $i < j$ and $f(i) > f(j)$. From $A[i,f(j)] > A[i,f(i)]$ and $A[j,f(i)] > A[j,f(j)]$ we get $A[i,f(j)] + A[j,f(i)] > A[i,f(i)] + A[j,f(j)]$, which contradicts to the fact that A

is Monge. Therefore, we can conclude that $f(1) \leq f(2) \leq \cdots \leq f(m)$ for any $m \times n$ Monge array.

d. Let $f(0) = 0$ and $f(m+1) = n$. Given that the leftmost minimum of the even-numbered rows is known, to compute $f(2k+1)$ for any odd-numbered row $2k+1$, $k = 0, \ldots, \lfloor (m-1)/2 \rfloor$, since for any $A : m \times n$ which is Monge, $f(0) \leq f(1) \leq f(2) \leq \cdots \leq f(m) \leq f(m+1)$, we only need to scan on $A[2k+1, f(2k)], A[2k+1, f(2k)+1], A[2k+1, f(2k)+2], \ldots, A[2k+1, f(2k+2)]$ to find the leftmost smallest elements lying in between. Therefore, there would be at most $\sum_{k=0}^{\lfloor (m-1)/2 \rfloor} f(2k+2) - f(2k) + 1 = f(2\lfloor (m-1)/2 \rfloor + 2) + \lfloor (m-1)/2 \rfloor = O(m+n)$ number of elements scanned.

e. Let $T(m,n)$ be the running time of the algorithm describes in part (d), then $T(m,n) = T(\lfloor m/2 \rfloor, n) + O(m+n)$, where $T(\lfloor m/2 \rfloor, n)$ is the time required for computing the leftmost minimum of the even-numbered rows, and $O(m+n)$ is the time required for computing the leftmost minimum of the odd-numbered rows.

We can use the substitution method to verify that $T(m,n) \leq d(m + n \lg m)$, for some constant $d > 0$. Assume that this bound holds for all positive $k < m$, i.e., $T(\lfloor m/2 \rfloor, n) \leq d(\lfloor m/2 \rfloor + n \lg(\lfloor m/2 \rfloor))$, then

$$\begin{aligned}
T(m,n) &= T(\lfloor m/2 \rfloor, n) + O(m+n) \\
&\leq T(\lfloor m/2 \rfloor, n) + c(m+n), \text{ for some constant } c > 0 \\
&\leq d(\lfloor m/2 \rfloor + n \lg(\lfloor m/2 \rfloor)) + c(m+n) \\
&\leq d(\frac{m}{2} + n \lg(\frac{m}{2})) + c(m+n) \\
&= d(\frac{m}{2} + n \lg m - n) + c(m+n) \\
&= (\frac{d}{2} + c)m + dn \lg m + (c-d)n \\
&\leq d(m + n \lg m),
\end{aligned}$$

as long as $d \geq 2c$. Therefore, $T(m,n) = O(m + n \log m)$.