8 Some interesting algorithms and their analysis

8.1 The closest-pair problem

Reading: MAW 10.2.2

- Problem:

  Given n 2D points, find the two closest points.

  Remarks:

  - Input is $(x_1,y_1),\ldots,(x_n,y_n)$ and output is $(x_i,y_i),(x_j,y_j)$.
  - The distance between $p_1=(x_1,y_1)$ and $p_2=(x_2,y_2)$ is $|p_1p_2|=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$.
  - A brute-force algorithm: $\binom{n}{2}=O(n^2)$.

- A divide-and-conquer algorithm:

  Let X be point set P sorted by increasing x and Y be point set P sorted by increasing y.

  1. Divide: Use a vertical line l to bisect P into P_L and P_R. X and Y are thus correspondingly partitioned into X_L and X_R, Y_L and Y_R.
  2. Conquer: Use two recursive calls to find the closest pairs in P_L and P_R. Let δ_L (δ_R) be the distance between the two closest points in P_L (P_R). Define $\delta=\min\{\delta_L,\delta_R\}$.
  3. Merge: Find the closest pair in P. It can be the pair with distance δ found in the previous step, or a pair with one point in P_L and the other in P_R with a distance less than δ.

  See Figure 10.30 on page 413 of MAW.

- Time complexity: $T(n)=T_P(n)+T_{DAC}(n)$.

  - Preprocessing: Sort P twice to construct X and Y. $\Rightarrow O(n\log n)$
  - Divide: Use the median in X to create the partition of P into P_L and P_R. Construct the partition of X into X_L and X_R. (This is easy.) Construct the partition of Y into Y_L and Y_R. (This can be tricky.) All of the above must be done in linear time. To check whether you have the right partitions, are P_L, X_L, and Y_L the same point set, and are P_R, X_R, and Y_R the same point set? $\Rightarrow O(n)$
  - Conquer: Two recursive calls on point sets of size $\frac{n}{2}$. $\Rightarrow 2T_{DAC}(\frac{n}{2})$
  - Merge: Many technical details to fill in. We wish to spend only linear time for the merge. Can we achieve this goal? $\Rightarrow O(n\log n)$

  So $T_{DAC}(n)=2T_{DAC}(\frac{n}{2})+O(n)=O(n\log n)$. Overall, $T(n)=T_P(n)+T_{DAC}(n)=O(n\log n)+O(n\log n)=O(n\log n)$.

- Merge in linear time:

  Assume that

  - $P\Rightarrow P_L,P_R$, $X\Rightarrow X_L,X_R$, $Y\Rightarrow Y_L,Y_R$ by the vertical line l.
  - δ_L is the distance between the closest points in P_L and δ_R is the distance between the closest points in P_R.
  - $\delta=\min\{\delta_L,\delta_R\}$.

  Goal: Determine if there are two points, one in P_L and the other in P_R, with distance less than δ.

  An exhaustive search checks all pairs and may take $O(n^2)$. Can we just check $O(n)$ pairs and not miss any one with distance less than δ? Yes and here is why.

  Define a strip centered at l with width 2δ. Let P_S be the set of points in the strip and Y_S be P_S sorted by y. (Remember our linear time restriction: Can you create P_S and Y_S in linear time?)

  Claim: If there are points $p\in P_L$ and $q\in P_R$ with $|pq|<\delta$, then p and q must be in the strip.

Pause: Can we check out all pairs in P_S to determine the one with the smallest distance?

For each $p \in Y_S$, define a rectangle $R(p)$ of height δ and width 2δ, with the bottom edge of the rectangle passing p.

Claim: If there are p and q with $|pq| < \delta$ and $q.y \geq p.y$, q must be in $R(p)$.

Claim: There can be at most eight points in each $R(p)$.

Why? Divide $R(p)$ ($\delta \times 2\delta$) into eight $\frac{\delta}{2} \times \frac{\delta}{2}$ squares. In each square, if there are two or more points, say q_1 and q_2, then

$$|q_1 q_2| \leq \text{diagonal of the square} = \sqrt{2}\frac{\delta}{2} < \delta,$$

which is impossible since q_1 and q_2 are on the same side of the vertical line l. So there can be at most one point in each square, with a total of eight points in $R(p)$.

Claim: For any $p \in Y_S$, if there is q such that $|pq| < \delta$, then q must be one of the seven points following p in Y_S.

Algorithm for merge:

```
m = |Ys|
mindist = |Ys[0]Ys[1]|
p = Ys[0]
q = Ys[1]
for i = 1 to m-1
  k = min {i + 7, m}
  for j = i + 1 to k
    dist = |Ys[i]Ys[j]|
    if dist < mindist
      mindist = dist
      p = Ys[i]
      q = Ys[j]
If mindist < delta
  return p and q as the closest points
else return the closest points found by
     the recursive calls
```

8.2 The selection problem

Reading: MAW 7.7.6 and 10.2.3

- Given a list of n numbers, find the kth smallest number among them.
- First try: Sort the list in increasing order ($\Theta(n \log n)$) and locate the kth element ($\Theta(1)$).
- Second try: Similar to Quick Sort.

```
Select(L,k)
  if |L|<50
    then sort L and return the kth
    else choose any p from L as a pivot
         L1={a in L, where a<p}
         L2={a in L, where a=p}
         L3={a in L, where a>p}
         if |L1|>k then return Select(L1,k)
         else if |L1|+|L2|>=k
              then return p
              else return Select(L3,k-|L1|-|L2|)
```

Like Quick Sort, the time complexity of this algorithm heavily depends on the selection of the pivot in each recursion. If every time p happens to partition L evenly, then the time complexity is $O(n)$. However, if the partition is extremely uneven, the time complexity degrades to $O(n^2)$. Therefore, the worst-case time of the algorithm is $O(n^2)$, although the average-case time is $O(n)$.

- Third try: Choose the pivot cleverly. Replace "choose any p from L as a pivot" in the above algorithm by the following code:

```
divide L into |L| div 5 sublists of (up to) $5$ elements each
sort each sublist into increasing order
let M be the list of medians of all sublists
p=Select(M, |M| div 2)
```

From the illustration shown in Figure 10.36 on page 418, there will be at most $\frac{3}{4}|L|$ elements in $L1$ and at most $\frac{3}{4}|L|$ elements in $L3$. Therefore, $T(n) \leq O(1)$ for $n \leq 49$ and $T(n) \leq T(\frac{n}{5}) + T(\frac{3n}{4}) + O(n)$ for $n \geq 50$. By induction, $T(n) = O(n)$. Since $\Omega(n)$ is obvious a lower bound, this D&C algorithm is optimal.

- Theorem: Let $T(n) \leq cn$ for $n \leq 49$ and $T(n) \leq T(\frac{n}{5}) + T(\frac{3n}{4}) + cn$ for $n \geq 50$. Show that $T(n) \leq 20cn$.

  Proof Induct on n. When $n \leq 49$, $T(n) \leq cn \leq 20cn$. Assume that $T(i) \leq 20ci$ for $i \leq n-1$. Now consider $T(n)$.

$$\begin{aligned} T(n) &\leq T(\frac{n}{5}) + T(\frac{3n}{4}) + cn \\ &\leq 20c\frac{n}{5} + 20c\frac{3n}{4} + cn \\ &= 4cn + 15cn + cn \\ &= 20cn \end{aligned}$$

8.3 The minimum spanning tree problem

Reading: MAW 9.5

- Given an undirected graph with non-negative edge weights, $G = (V, E, w)$. Assume $|V| = n$ and $|E| = m$. A spanning tree is a connected subgraph containing all nodes in V such that there is no cycles in the subgraph. A minimum spanning tree (MST) is a spanning tree with the minimum total edge weight.
- Some observations:
  - A spanning tree of G always has n nodes and $n-1$ edges.
  - A spanning tree exists if and only if the graph is connected.
  - Adding an edge to a spanning tree creates a cycle and removing an edge from a spanning tree disconnects the subgraph.
  - Applications in telecommunication: What is the most cost-efficient way to set up a telephone network among n locations?
- Prim's algorithm:

  Build the minimum spanning tree by adding nodes one by one to an initially empty tree. The next node to be added is always the one reachable by some node in the current tree via the smallest weighted edge.

  See Figure 9.48 and Figure 9.49 on page 357 for an example.

```
Prim(G)
  initialize T to be an empty tree
  choose any node and add it to T
  do n-1 times
    let v be a node outside T such that there is u in T with
```

```
    w(u,v)=min{w(x,v) for all x in T}
  add node v and edge (u,v) to T
return T
```

Time complexity: $O(n^2)$ or $O(m \log n)$ if a heap is used (why?).

- Kruskal's algorithm:

  Add edges one by one in the order of non-decreasing weights into an initially empty tree. An edge is discarded if adding the edge creates a cycle in the tree.

  See Figure 9.57 on page 360 for an example.

```
Kruskal(G)
  initialize T to be an empty tree
  sort edges in E by non-decreasing weights
  while |T|<n-1
    if adding the next edge e in E to T does not create any cycle
    then e is added to T
    else e is discarded and not considered anymore
  return T
```

  Time Complexity: $O(m \log m)$ (why?).