Tutorial 2

Deepank Agrawal (17CS30011)

28-07-2019

1 Problem Statement

$P[1..n]$ is an input list of n points on xy-plane. Assume that all n points have distinct x-coordinates and distinct y-coordinates. Let p_L and p_R denote the leftmost and the rightmost points of P , respectively. The task is to find the polygon Q with P as its vertex set such that the following conditions are satisfied.

1. The upper vertex chain of Q is x-monotone (increasing) from p_L to p_R.
2. The lower vertex chain of Q is x-monotone (decreasing) from p_L to p_R.
3. Perimeter of Q is minimum.

2 Recurrences

To solve the problem, we can design a Dynamic Programming(DP) algorithm. The formulation of the DP is:
Suppose that we have the x-monotone sorted list S of the point set P. Now, consider that for any $i,\, j \,\epsilon\, \{1, .., n\}$, we have already constructed the vertex chain with $S_i, S_{j-1} \,\epsilon\, \{S_1, .., S_n\}$, as the terminal vertices. Let's consider $i \leq j$ as $B[i][j] = B[j][i]$. Now, for S_j, following two cases arise for construction of the open polygon Q_{ij}(say), with S_i being one terminal vertex:

1. $i < j - 1$:
   Then the chain with terminal S_j must also include S_{j-1}. The new terminal vertices will be S_i and S_j for the vertex chains.
2. $i = j - 1$ or $i = j$:
   Then the optimal solution has one of the vertex chains ending in S_j joined to some $S_k \,\epsilon\, \{S_1, .., S_{j-1}\}$. The other terminal vertex will be S_i.

Let $p(Q_{ij})$ denote the optimal perimeter with S_i, S_j as the terminal vertices of the chains. Then, the k for which $p(Q_{ik}) + dist(k, j)$ is minimum will be selected for case 2.

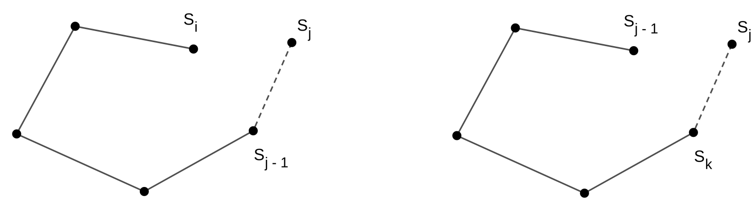

Figure 1: Cases 1 and 2

Let's define $M[1..n][1..n]$, where $M[i][j]$ stores the minimal value of $p(Q_{ij})$ with S_i and S_j as terminal vertices of the vertex chains. Now the recurrence relation for the DP, such that S_j is the vertex to be added and $\forall\, i,\, j \,\epsilon\, \{1, .., n\},\, i \leq\, j$, can be defined as:

$$M[i][j] = \begin{cases} M[i][j-1] \,+\, dist(j-1,\, j), & i < j-1 \\ \min\limits_{1 \,\leq\, k \,<\, j} \{M[i][k] \,+\, dist(k,\, j)\}, & i \,=\, j-1 \text{ or } i \,=\, j \\ -1, & else \end{cases} \tag{1}$$

So, the perimeter of the optimal polygon Q will be, $p(Q_{nn}) \;=\; M[n][n]$.

3 Algorithm

Now that Optimal Substructure has been defined, let's design the algorithm. Before that, we can see there are **overlapping sub-problems** (e.g. $M[3][1]$ will be required for $M[3][3]$). So, **memoization** will be used.

To solve the problem, we store the minimal value of the perimeter $p(Q_{ij})$, $\forall\, S_i,\, S_j \,\epsilon\, \{S_1, .., S_n\}$ using equation (1). For this we construct a matrix $M[0..n][0..n]$, where $M[i][j]$ stores the minimal value of the perimeter $p(Q)$ of the polygon Q_{ij} with S_i and S_j as terminal vertices of the vertex chains. To store the polygon Q, we construct a $T[0..n][0..n]$, where $T[i][j]$ stores the pair of terminal vertices of the open chains just before adding S_j vertex. Bottom-up approach will be used to fill matrix M.

Main Steps:

1. Sort the point set P in increasing x-coordinate and store in list S.

2. Initialize the matrix M with -1 values. $M[i][j] = -1$ denotes that open polygon Q_{ij} with S_i and S_j as terminal vertices of the vertex chains is not to be considered. Set $M[1][1] \,=\, 0$, as the base case.

3. Consider any open polygon Q_{ij} and calculate the minimum perimeter $p(Q_{ij})$ using equation (1). Accordingly, update $T[i][j]$.

4. $M[n][n]$ is the minimum perimeter $p(Q_{nn})$ for the polygon Q.
5. To get the polygon Q, we trace back from $T[n][n]$ up to $T[1][1]$ and store the two chains.

4 Time and space complexities

4.1 Time Complexity

Let's refer to the above mentioned steps for calculation of time complexity.

1. Sorting n points $\rightarrow O(n \log(n))$ time.
2. In equation (1), the 1^{st} condition takes $O(n^2)$ time and the 2^{nd} condition takes $O(n)$ time. So, for Step 3 $\rightarrow O(n^2)$ time.

So, overall Time complexity $= O(n^2)$.

4.2 Space Complexity

As a matrix $M[0..n][0..n]$ is constructed, the space complexity of the algorithm will also be $O(n^2)$.