Tutorial 2

YASH PARAG BUTALA - 17CS30038

01-08-2019

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 pL and pR 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:

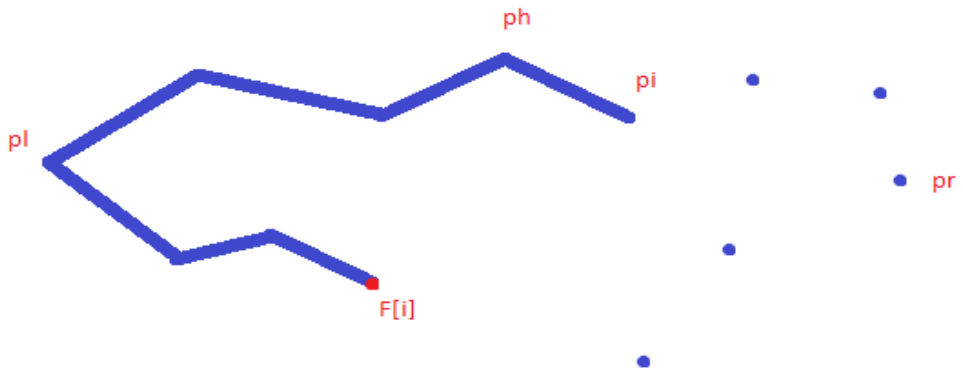
i) The upper vertex chain of Q is x-monotone (increasing) from pL top R.
ii) The lower vertex chain of Q is x-monotone (decreasing) from pR to pL.
iii) Perimeter of Q is minimum. You have to answer the following.

Provide necessary figures/- diagrams for explanations.

1. Develop the recurrences needed for DP, with clear arguments.
2. Design the algorithm and write its main steps.
3. Derive the time and space complexities of your algorithm.

2 Recurrences:

Lets assume we have points sorted with respect to x-coordinate as p1,p2,p3,... ,pn. The leftmost point is p1 and rightmost is pn. Hence every other point lies on one of the two monotones between p1 and pn.



From the picture we can see that there are two paths that both start from the leftmost point and each of the points on the left side of point i is in at least one of the two monotones. The minimum sum of edges of such a graph will be denoted by W[i].

We will need to store for each index i, the rightmost point before pi that is in the different monotone (this might be p1 even though p1 is always on the same path with pi). We will denote this point by F(i).

Let ph be the point preceding pi (in the same monotone)

Thus Recursive solution:

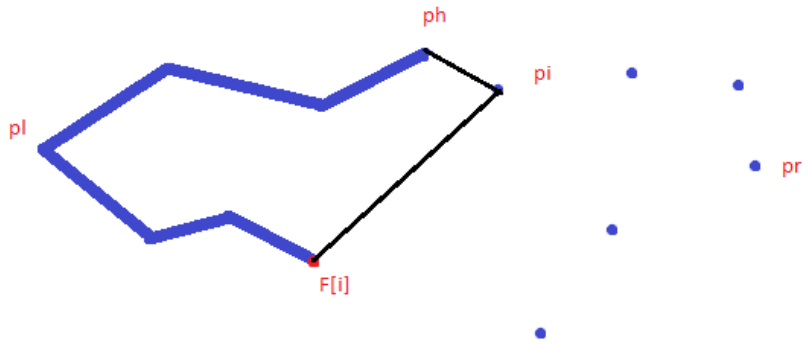
Base Case:

W[1]=0

Recursive Cases:

1) h = i-1:

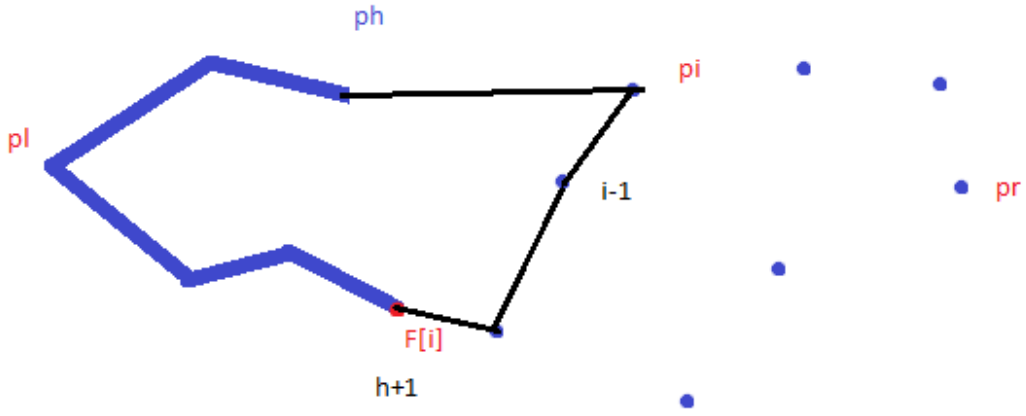
$W[i-1] + dist(pi-1, pi) + dist(pF[i-1], pi)$ and F[i]=F[i-1]



2) h < i-1:

C[j]=min$\{W[h] + dist(ph, pi) + dist(pF[h], ph+1) + \sum_{a=h+1}^{i-1} dist(pa, pa+1)\}$ }

And F[i]=pi-1



Hence for a given i, taking minimum perimeter from all possible h, the optimal substructure is obtained. Thus:

W[i] = min$\{W[i-1] + dist(pi-1, pi) + dist(pF[i-1], pi)$,

$$\min_{h=1\ to\ i-2}\{W[h] + dist(ph, pi) + dist(pF[h], ph+1) + \sum_{a=h+1}^{i-1} dist(pa, pa+1)\}\}$$

3. Algorithm:

A Dynamic Programming Algorithm can be designed from the recurrence relation. From the recurrence relation, it is clear that W[1, . . . , n] will store the optimal values for first i elements. Similarly F(i) in F[1, . . . , n] stores the previous vertex of the other monotone.

F(i) can be computed as:

$$F(i) = \begin{cases} F(i-1) & \text{if edge, } (P[i], P[i-1]) \text{exists} \\ P[i-1] & \text{if edge, } (P[i-1], P[i]) \text{ does not exist} \end{cases}$$

The sum $\sum_{a=h+1}^{i-1} dist(pa, pa+1)$ will take O(n) time if we compute at each step. Hence, we store it in a array Dist such that Dist[i]= $\sum_{a=1}^{i} dist(pa, pa+1)$. This O(n) time pre-processing will save our time in each step to O(1).

Retrieving the polygon is again simple. We can save the vertices of one of the chains as we fill the array W. The other chain will be all the vertices that are not in the previous chain.

Finally, we have to find which monotone is upper and which one is lower. The first edge (edge from P[1]) will have a larger slope in as compared to the lower vertex chain.

Require: P[1, . . . , n]

Sort the points as per x coordinate.

Create arrays W[1, . . . , n], F[1, . . . , n], S[1, . . . , n] and Dist[1, . . . , n].

Fill the array D as per the rule, Dist[i] = S[i]= $\sum_{a=1}^{i} dist(pa, pa+1)$ as:

Dist[1] = 0

i ← 2

while i ≤ n **do**

    Dist[i] = Dist[i] + dist(P[i], P[i + 1])

end while

Computing the subproblems:

W[1] = 0

F[1] = 1

i ← 2

while i ≤ n **do**

    tmp1 = W[i − 1] + d(P[i − 1], P[i])

    tmp2 =

    $\min_{h=1\ to\ i-2}\{W[h] + dist(ph, pi) + dist(pF[h], ph+1) + Dist[i-1] - Dist[h+1]\}$

    tmp3== argmin(h) {W[i − 1] + dist(P[k], P[i]) + dist(P[F[k]], P[k + 1]) + Dist[i − 1] − Dist[k + 1]}

    if tmp1<tmp2 **then**

        F[i]=F[i-1]

W[i]=tmp1

S[i]=i

Else

F[i]=P[i-1]

W[i]=tmp2

S[i] =tmp3

end if

end while

Create lists W1 and W2 to keep the two vertex chains.

i ← n

while i > 1 **do**

Add S[i] in beginning of W1

i ← S[i]

end while

Form W2 = P − W1 − P[1] − P[n] (in same increasing order).

Decide on which of the two chains are upper and lower (assuming both of the chains are in increasing order).

slope1 = $\frac{W1[1].y-P[1].y}{W1[1].x-P[1].x}$

slope2 = $\frac{W2[1].y-P[1].y}{W2[1].x-P[1].x}$

if (slope1) > (slope2) **then**

(W1 is upper chain) Polygon Q = P[1] + W1 + P[n] + W2(in reverse order)

else

(W2 is upper chain) Polygon Q = P[1] + W2 + P[n] + W1(in reverse order)

end if

4. Time and Space Complexity:

A. Time Complexity :

Time taken to sort P by x-coordintes = O(nlogn)

Time taken to compute array D of summation = O(n)

Time taken to compute one element of W (we loop from h=1 to i-1)= O(n)

Time taken to compute W (while loop)= O(n) × O(n) = O(n^2)

Time taken to get W1 = O(n)

Time taken to get W2 = O(n), as the lists are sorted.

Total time complexity = O(n^2)

B. Space Complexity

All arrays formed are one dimensional with space required in O(n).

Total space complexity is O(n).