Tutorial 2

Anshul Choudhary(17CS10005)

04-08-2019

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 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.

i) The upper vertex chain of Q is x-monotone (increasing) from pL to pR.
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

Let p1,p2, . . . ,pn denote the points by ordered by increasing x-coordinates, l(i,j) denote the length of a shortest path with endpoints p_i and p_j , i < j and dist(p_i,p_j) denote the distance between point pi and pj.

$$l(i,j) = \begin{cases} dist(p_{\mathrm{i}}, p_{\mathrm{j}}) & if\ i = 1\ and\ j = 2 \\ l(i, j-1) + dist(p_{\mathrm{j\text{-}1}}, p_{\mathrm{j}}) & if\ i < j-1 \\ min_{1 \le k < i}\ (l(k,i) + dist(p_{\mathrm{k}}, p_{\mathrm{j}})) & if\ j > 2\ \ and\ \ i = j-1 \end{cases}$$

3 Algorithm

N(i,j) denotes the neighbour of p_j in a shortest x-monotone path from p_i to p_j

```
1  n← |p|
2  Compute l(i,j) and N(i,j), for all 1 ≤ i < j < n.
3  for j = 2 to n do
4  |  for i = 1 to j-1 do
5  |  |  if i = 1 and j = 2 then
6  |  |  |  l[i,j] ⟵ dist(p[i],p[j])
7  |  |  |  N[i,j] ⟵ i
8  |  |  end
9  |  |  else if j > i + 1 then
10 |  |  |  l[i,j] ⟵ p[i,j − 1] + dist(p[j − 1],p[j])
11 |  |  |  N[i,j] ⟵ j − 1
12 |  |  end
13 |  |  else
14 |  |  |  l[i,j] ⟵ +∞
15 |  |  |  for k = 1 to i-1 do
16 |  |  |  |  q ⟵ l[k,i] + dist(p[k],p[j])
17 |  |  |  |  if q < l[i,j] then
18 |  |  |  |  |  l[i,j] ⟵ q
19 |  |  |  |  |  N[i,j] ⟵ k
20 |  |  |  |  end
21 |  |  |  end
22 |  |  end
23 |  end
24 end
25 Construct the path. Stacks S[1] and S[2] will be used to construct the
     two x-monotone parts of the path.
26 Let S be an array of two initially empty stacks S[1] and S[2].
27 k ⟵ 1
28 i = n − 1
29 j = n
30 while j > 1 do
31 |  PUSH(S[k],j)
32 |  j ⟵ N[i,j]
33 |  if j < i then
34 |  |  swap i ⟷ j
35 |  |  k ⟵ 3 − k
36 |  end
37 end
38 PUSH(S[1],1)
39 while S[2] is not empty do
40 |  PUSH(S[1],POP(S[2]))
41 end
42 for i=1 to n do
43 |  T[i] ⟵ POP(S[1])
44 end
```

Array T contains the points in the order they are connected consecutvely.

4 Observations

1. Points p_{n-1} and p_n are neighbours in any bitonic tour that visits points $p1, p2, ..., pn$.
2. Given a normal bitonic path P with endpoints p_i and p_j , $i < j$, let p_k be the neighbour of p_j on this path. Then the path P' obtained by removing p_j from P is a normal bitonic path with endpoints p_i and p_k. In particular, p_{j-1} p_i, p_k . If P is a shortest normal bitonic path with endpoints p_i and p_j , then P' is a shortest normal bitonic path with endpoints p_i and p_k.
3. Consider the neighbour p_k of p_j in a normal bitonic path P with endpoints p_i and p_j , $i < j$. If $i = j - 1$, we have$1 \leq k < i$. If $i < j - 1$, we have $k = j - 1$.

5 Time and space complexities

It is easy to see that lines 25-43 take linear time. Indeed, the loop in Lines 42-44 is executed n times and performs constant work per iteration. The loop in Lines 39-40 is executed $|S[2]|$ times, and each iteration takes constant time; hence, it suffices to prove that $|S[2]| \leq n$ after the execution of Lines 25-38. To prove this, it suffices to show that the loop in Lines 30-37 is executed at most n-1 times because every iteration pushes only one entry onto stack S[1] or S[2]. The loop in Lines 30-37 takes constant time per iteration. It is executed until j=1. However, initially j=n, and the computation performed inside the loop guarantees that j decreases by one in each iteration. Hence, the loop is executed at most n-1 times. To analyze the running time of Lines 1-24, we first observe that this part of the algorithm consists of two nested loops; the code in Lines 5-22 is executed $O(n^2)$ times because i runs from 2 to n and in every iteration of the outer loop, j runs from 1 to i-1. Now, we perform constant work inside the loop unless i=j-1. In the latter case, we execute the loop in Lines 15-21, (i-1)=O(n) times. Since there are only n-1 pairs (i,j) such that $1 \leq i = j - 1 < n - 1$, we spend linear time in only n-1 iterations of the two outer loops and constant time in all other iterations. Hence, the running time of this algorithm is,

$$TimeComplexity = O(n^2 \ . \ 1 + n \ . \ n) = O(n^2).$$

Since we are using a 2D matrix of size N x N ($N \Longrightarrow no.ofpoints$) in the first part. Rest of the extra space used is of the size of O(n). Hence the Space Complexity will be,

$$SpaceComplexity = O(n^2)$$