CSCE: 629 | ANALYSIS OF ALGORITHMS

PROJECT-1

UIN: 232009024
ASHUTOSH CHAUHAN

Q.> Multi-Line Fitting Problem

- Input: 1) A set of n points in a 2-dimensional plane

$$P = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

with $x_1 < x_2 < \dots < x_n$.

2.> A real number $C > 0$ i.e ~~pa~~ penalty for addition of an extra line.

⇒ Main idea of Algorithm:

In the given problem we need to find the ~~minimum~~ lines which can fit all points such that the overall cost is minimum.

we know, $\boxed{\text{cost } (c) = \text{error} + N * C}$

where, error: sum of all squared errors in each segment

N : No. of lines

C : Penalty of adding an extra line

If we are trying to find the lines that best fits n points, it would be easy if we knew the best solutions for n-1 points. We will break the complete problem into smaller subsequence problems and then use those results to compute bigger problem.

So for calculating an optimal cost of lets say j starting points, we will first calculate optimal cost of first point, best fit for 2 points, 3 points and so on.

So let us call optimal cost of point $P_1, P_2 \dots P_j$ i.e first j points as $\hat{C}(j)$,

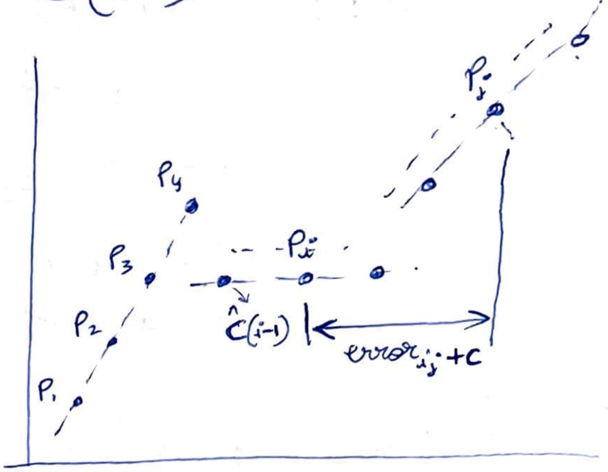
$$\hat{C}(j) = error_{ij} + C + \hat{C}(i-1)$$

for any i between 1 and j.

where, $error_{ij}$: min sum of error square for points $P_i \dots P_j$

C : penalty of adding an extra line between $i \to j$

$\hat{C}(i-1)$: optimal cost of points $P_1, P_2 \dots P_{i-1}$



We will calculate this for every value of i between 1 and j and then take the minimum of those costs as optimal cost for j points i.e.

$$\boxed{\hat{C}(j) = \min_{1 \leq i \leq j} \{ error_{ij} + C + \hat{C}(i-1) \}}$$

optimal cost of fitting first j points is the minimum of $error_{ij}$, line penalty(c) and $\hat{C}(i-1)$ across all of values of i that are smaller than j. This show us how we can solve the sub problem of size j as a function of optimal solution of smaller sub-problem of size $i-1$. Likewise we can find optimal cost for ~~all~~ fitting all n points.

⇒ Correctness of Algorithm :-

we will prove the correctness using mathematical induction. Let us assume that for a given value of n, the algorithm correctly computes the minimum error for fitting first n points using min number of line segments. We will show that the algorithm will also correctly compute the minimum error for fitting the first n+1 points using min number of line segments.

We calulate $\hat{C}(1)$ i.e ~~cost~~ optimal cost of first two points. Since a line can exactly pass two points so.

$$\hat{C}(1) = 0 + C = C.$$

Let assume we compute $\hat{C}(1) \rightarrow \hat{C}(K)$ correctly for some $K < n$. We will show that we can compute $\hat{C}(K+1)$ correctly using this information.

To calculate $\hat{C}(K+1)$ we will take all possible cases that minimize the cost. This we did by iterating over all possible starting points i between 1 & K+1.

$$\hat{C}(K+1) = \min_{1 \leq i \leq K+1} \{ error_{i,K+1} + C + \hat{C}(i-1) \}.$$

The above formula computes the minimum error for fitting the first (K+1) points using a min number of line segments.

By inductive hypothesis, we know that $\hat{C}(1), \hat{C}(2) \ldots \hat{C}(K)$ have been computed correctly. So min error for fitting the line is also correctly computed for all $i \leq K+1$

And since we are taking minimum of all the cost value for every possible i we are definitely achieving the minimum error for fitting first $(K+1)$ points.

Therefore, by mathematical induction we have proved that algorithm correctly fits the n points such that overall cost is minimum.

$\Rightarrow$ Pseudo Code & Time Complexity :-

```
def Multiline_parameters(x, y):                              ------- O(n²)
    create a matrix of size (len(x) x len(x))
    for i in range(len(x)-2):                                ------- O(n)
        for j in range(i+2, len(x)):                         ---- O(n)
            n = j-i+1
```

• calculate $a = \dfrac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$ -- O(1)

• and $b = \dfrac{\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i}{n}$. --- O(1)

using prefix sum array of Σx, Σy, Σx^2, Σy^2 and Σxy.

• use a, b values to compute error --- O(1).

error[i][j] = $(y - ax - b)^2$.

```
    return error

def Multiline_fitting(x, y, c):                              ------- O(n²)
    error = Multiline_parameters(x, y)                       --- O(n²)
    n = len(error[0])                                        --- O(1)
    optimal_cost = [0]*n                                     --- O(1)
    line_si_ind = [0]*n                                      --- O(1)
    for j in range(n):                                       --- O(n)
        optimal_temp = [0]*(j+1)                             --- O(1)
        optimal_temp[0] = error[0][j] + C                    --- O(1)
```

```
line[si_ind[j]] ...
for i in range(1, j+1):                                          - O(n)
    optimal_temp[i] = optimal_temp[i-1] + C + error[i][j]        - O(1)

optimal_cost[j] = min(optimal_temp)                              - O(n)
line_si_ind[j] = index of min value in                           - O(1).
                 optimal temp

                                                                 - O(1)
line_ei_ind = []
# Backtracking to store end points of every
  line segment.
  ~~while~~
  j = n - 1                                                      - O(1)
  while j >= 0:                                                  - O(n)
      line_ei_ind.append(j)
      j = int(line_si_ind[j] - 1)
  line_ei_ind.reverse()                                          - O(1).
  return line_ei_ind, optimal_cost[-1]

                                                                 — O(n^2).
def main:-
  read instances using pickle
  share x, y, c values of every instances                       - O(n)
  in loop -
  last_points, opt = Multiline_fitting(x, y, c)                  - O(n^2)
  Append output in solution dictionary.                          - O(n)
```

From above pseudo code and corresponding time analysis, we can observe that $O(n^2)$ time is taken for calculate errors of every i, j pairs. Apart from this we are traversing all points from $0 \to n$ and for each point we are considering all values before that point. This again is an $O(n^2)$.

$$\therefore \text{Overall time complexity} = \underbrace{O(n^2)}_{\text{error matrix calculation}} + \underbrace{O(n^2)}_{\text{traversing of points and cal min case}} + O(1)$$

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

⇒ How to run code:

1. Go to main function.
2. ~~change~~ Add the test instances file address ~~name~~ to "name_of_source_file" variable
3. change the O/P file address ~~name~~ you want by changing "name_of_solution_file" variable
4. $\boxed{\text{Note}}$: if ~~the~~ test instances file is in same folder and just add filenames in above variable.
5. Run code and it will automatically generate solution file. (You need to parse solution file using pickle for reading).