Build and Run

No need to build, just run:

```
time python3 main-opt.py examples_of_large_instances output.bin
```

The command line usage is

```
python3 main-opt.py inputfile outputfile
```

The Main Idea

The problem is to find a split to separate the given sequence of points into several segments, and minimize the total cost (and penalty).

The main idea is to use dynamic programming on the following subproblems and recurrence relations. The key idea of which are: 1) the computation is like in a DAG, from smaller index to larger index. 2) the computation of ε can be preprocessed, thus each set $\varepsilon(P)$ can be done in constant time. They together form the rough transition

$$dp_i = \min_j dp_j + \varepsilon(P_{j..i}) + C$$

DP and pseudo code

Subproblem

$$dp[j+1]$$

is the minimized cost (sum of errors) when splitting the points with index from 0 to j. with a vacant case

$$dp[0]$$

for sentinel.

Objective value to minimize

$$dp[n-1]$$

Base case

$$dp[0] = 0$$

Recurrence Relation

$$dp[i] = \min_j dp[j] + \varepsilon(P_{j..i}) + C\}, 0 \leq j < i$$

pseudo code

```
input is :
    n: the number of points
    C: the penalty of each new line
    pt: [0..n-1], an array of points
output is :
    k: the number of segments in the optimal plan
    LP: the last points array
    OPT: the optimal value

dp = [0..n] an array of numbers
bk = [0..n] an array of integers, recoding the backtracking of "last point"

dp[0] = 0
for i = 0 to n - 1
    dp[i + 1] = infinity
    for j = 1 to i
        if dp[j] + epsilon(j, i) + C < dp[i + 1]
            dp[i + 1], bk[i + 1] = dp[j] + epsilon(j, i) + C, j - 1

LP = the last pointer array
build LP by starting from n - 1:
   follow the bk array all the way down until we reach -1,
   also record the path length as k

return (k, LP, dp[n-1])
```

Proof of correctness

Prove by induction.

Base case The trivial case is that when think of the first 0 points, it contains 0 cost. so $dp[0] = 0$ is correct.

Induction Hypothesis We have now computed $dp[t]$ for all $0 \leq t \leq i$, and they are the optimal value for that t,

Induction Case for $dp[i + 1]$,

we consider the all possible choices to take the last full line, which are $P_k = \{p_j, p_{j+1}, ...p_i\}$ for all $0 \leq j < i$.

The new cost to add from $dp[j]$ is the $\varepsilon(P_k) + (Ck - C(k - 1)) = \varepsilon(P_k) + C$

Since each $dp[j]$ is optimal over all paths reach that $j - 1$ by induction,

$\{dp[j] + \varepsilon(p_{j..i}) + C\}$ will cover all possible minimum cost of all paths until i (inclusive), and the minimum of such set is also the minimum cost for $dp[i+1]$

Summary Thus the dp relation and base case are correct.

Time analysis

The preprocess of ε

Note the expression for ε contains many $\displaystyle\sum_{t=j}^{i} f(t)$ computations.

We will cache all $F(n) = F(n-1) + f(n-1), F(0) = 0$ in $O(n)$ time, thus

$$\sum_{t=j}^{i} f(t) = F(i+1) - F(j)$$

Thus the $\varepsilon(p_{j..i})$ can be done in $O(1)$ time,

while the $F(n)$ computed are $\sum x, \sum y, \sum xy, \sum x^2, \sum y^2$, and the preprocess takes $O(n)$.

The DP process

$O(n)$ subproblems, $O(n)$ choices each, $O(1)$ to compute each choice, in total $O(n^2)$.

Total

$O(n^2)$