

First of all, we can simplify this problem by reducing k to 1. It’s easy to infer that if we set city i as station, the value is



We can replace it with a more clear expression



Which prefix[i] means sum of [1,2,3,,,,i] and suffix[i] means sum of [i,i+1,,,,n].

So in this simplified vision, we can enumerate every city as station and calculate corresponding value in O(n).

We can find one conclusion that if we set city i as station, city in interval [1,i] will find closest station in [1, i], so we set array dp[k][i] express that when k-th station was set in i-th city, the minimize distance of cities in [1, i].

We can receive a simple formula like this.



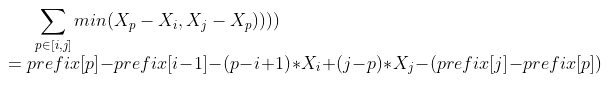
In this formula, we can konw last station in city i and the current k-th station in city j. The distance in [1, i] has no relationship with station j, so we focus on cities in interval [i, j].

For each city in interval [i, j], their closest station is i or j. So we can calculate it by function min.

The sum of state is nk, and for each state, we will transfer n times. In addition, we will enumerate n city in worst situation to calculate last part in formula. So the complexity of time is O(n^3k).

So we need to optimize this formula. We can find that for interval [i, j], we can find a index p satisfy xp <= (xi+xj)/2. So cities in interval [i, p] will find station I more closer and cities in interval [p+1,j] will more closer to station j.

Just like we infer in simplified version, we can replace last part of formula by



We can also find another conclusion that for i1<i2<i3, if p1<(i1+i2)/2 and p2 <(i2+i3)/2, p1 is bigger than p2. So we can enumerate i from j-1 to 1 and the value of p will always decrease. The time complexity of this step will cost n in each state nk. So we can find the final time complexity of formula is O(n^2k).

At last, we also need to calculate cities in interval [i, n]. These city’s closest station is k-th station obviously. We can calculate it by using suffix array.

Code

for (int i = 1; i <= n; i++)

{

dp[1][i] = i \* x[i] - prefix[i];

}

for (int kk = 2; kk <= k; kk++)

{

for (int j = kk; j <= n; j++)

{

dp[kk][j] = inf;

int mid = j;

for (int i = j - 1; i >= kk - 1; i--)

{

while (x[mid] - x[i] > x[j] - x[mid])

mid--;

int extra\_v = prefix[mid] - prefix[i - 1] - (mid - i + 1) \* x[i] + (j - mid) \* x[j] - (prefix[j] - prefix[mid]);

dp[kk][j] = min(dp[kk][j], dp[kk - 1][i] + extra\_v);

}

}

}

int ans = inf;

for (int i = k; i <= n; i++)

{

int tans = dp[k][i] + suffix[i + 1] - (n - 1 - i) \* x[i];

ans = min(ans, tans);

}