USACO DEC14 Problem 'marathon' (Gold) Analysis

by Allen Chen

Seeing that we have to update point coordinates and query the minimum travel time for a sub-route, we immediately see that this problem involves range queries and updates. A well known data structure that can be used to solve this problem is the segment tree, which allows us to handle such operations in logarithmic time.

First, let us take care of querying the minimum travel time, ignoring the fact that the cows can skip a checkpoint along the way. To do this, we treat each 2 consecutive checkpoints as one single node in a segment tree (let's call this tree "dist"). Then we just simply store the distance between each 2 consecutive points in each node.

Next, we need to handle the fact that cows can skip exactly one checkpoint along the way. Let's say we need to find the minimum travel time between checkpoints C_i and C_j and let's also say we skip a checkpoint C_k, where $i<k<j$. Let's also define $G(a,b)$, as the distance between checkpoint a and b. Therefore, the time taken without skipping a checkpoint is equal to:

$$D_1=G(C_i,C_{i+1})+G(C_{i+1},C_{i+2})+\ldots+G(C_{j-1},C_j).$$

while the time taken by skipping a checkpoint is equal to:

$$D_2=G(C_i,C_{i+1})+G(C_{i+1},C_{i+2})+\ldots+G(C_{k-2},C_{k-1})+G(C_{k-1},C_{k+1})+\ldots+G(C_{j-1},C_j).$$

If we take the difference:

$$D_1-D_2=G(C_{k-1},C_k)+G(C_k,C_{k+1})-G(C_{k-1},C_{k+1})$$

Notice that if we maximize the value D_1-D_2, then we will obtain a minimum D_2, since D_1 is a fixed value regardless of any checkpoint we skip. Thus we can use a second segment tree (call it Δ) and treat each 3 consecutive checkpoints as one node. In each node of Δ, we only need to store D_1-D_2, because we can obtain D_2 by taking: D_1 minus the best Δ value.

Thus, we can perform range maximum query on delta and a range sum query on dist to obtain our final answer while performing updates as necessary.

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<utility>

using namespace std;

typedef pair<int, int> pii;

const int maxn = 1 << 17;
int n, q;
pii mat[maxn];
int delta[2 * maxn];
int dist[2 * maxn];

int qa, qb;
int getd(int a, int b) {
        return abs(mat[a].first - mat[b].first) + abs(mat[a].second - mat[b].second);
}
```

```
void build(int rt, int a, int b) {
	if (a > b) return;
	if (a == b) {
		if (a < n - 1) delta[rt] = getd(a, a + 1) + getd(a + 1, a + 2) - getd(a, a
+ 2);
		else delta[rt] = 0;
		if (a < n) dist[rt] = getd(a, a + 1);
		else dist[rt] = 0;
		return;
	}
	int mid = (a + b) / 2;
	build(rt * 2, a, mid);
	build(rt * 2 + 1, mid + 1, b);
	delta[rt] = max(delta[rt * 2], delta[rt * 2 + 1]);
	dist[rt] = dist[rt * 2] + dist[rt * 2 + 1];
}

int query_dist(int rt, int a, int b) {
	if (a > qb || b < qa) return 0;
	if (qa <= a && b <= qb) return dist[rt];
	int mid = (a + b) / 2;
	return query_dist(rt * 2, a, mid) + query_dist(rt * 2 + 1, mid + 1, b);
}

int query_delta(int rt, int a, int b) {
	if (a > qb || b < qa) return 0;
	if (qa <= a && b <= qb) return delta[rt];
	int mid = (a + b) / 2;
	return max(query_delta(rt * 2, a, mid), query_delta(rt * 2 + 1, mid + 1, b));
}

void update_dist(int rt, int a, int b) {
	if (a > qb || b < qa) return;
	if (qa <= a && b <= qb) {
		if (a >= 1 && a < n) dist[rt] = getd(a, a + 1);
		else dist[rt] = 0;
		return;
	}
	int mid = (a + b) / 2;
	update_dist(rt * 2, a, mid);
	update_dist(rt * 2 + 1, mid + 1, b);
	dist[rt] = dist[rt * 2] + dist[rt * 2 + 1];
}

void update_delta(int rt, int a, int b) {
	if (a > qb || b < qa) return;
	if (qa <= a && b <= qb) {
		if (a >= 1 && a < n - 1) delta[rt] = getd(a, a + 1) + getd(a + 1, a + 2) -
getd(a, a + 2);
		else delta[rt] = 0;
		return;
	}
	int mid = (a + b) / 2;
	update_delta(rt * 2, a, mid);
	update_delta(rt * 2 + 1, mid + 1, b);
	delta[rt] = max(delta[rt * 2], delta[rt * 2 + 1]);
}

int main() {
	freopen("marathon.in", "r", stdin);
	freopen("marathon.out", "w", stdout);
	scanf("%d%d", &n, &q);
```

```
	for (int i = 1; i <= n; i++) {
		scanf("%d%d", &mat[i].first, &mat[i].second);
	}
	build(1, 1, n);
	for (int i = 0; i < q; i++) {
		char c[2];
		scanf("%s", c);
		if (c[0] == 'Q') {
			scanf("%d%d", &qa, &qb);
			qb--;
			int tot = query_dist(1, 1, n);
			qb--;
			int del = query_delta(1, 1, n);
			printf("%d\n", tot - del);
		}
		else {
			int ix, pa, pb;
			scanf("%d%d%d", &ix, &pa, &pb);
			mat[ix].first = pa;
			mat[ix].second = pb;
			for (int j = ix - 1; j <= ix; j++) {
				qa = j; qb = j;
				update_dist(1, 1, n);
			}
			for (int j = ix - 2; j <= ix; j++) {
				qa = j; qb = j;
				update_delta(1, 1, n);
			}
		}
	}
}
```