USACO FEB13 Problem 'route' Analysis

by Mark Gordon

The 'no intersecting' condition is what made this problem interesting. Without loss of generality suppose that after taking two routes we end up further down (at larger indexes) the river. Then it should be clear that the sequence of sites we visit on each side of the river must be strictly increasing. If we went back to smaller indexes on the river we would necessarily intersect with the second to last route we made.

Therefore this problem can be transformed into finding the sequence of sites (l_1, r_1, ...) such that the sum of associated values is maximized, all adjacent sites are connected by routes, and l and r are strictly increasing.

One obvious dynamic programming approach would be to set our state as the last two sites we visited. Unfortunately with 40,000 sites on each side of the river this is too many states. To improve, notice that only 2R of the states are actually possible (in each orientation), therefore we can identify a state by a route index and orientation.

However, a little cleverness is still required to compute the state transitions quickly. To compute the maximum value attainable from our current state we need to find the maximum value attainable from any state departing from our current site to a site of larger index than our departing site. Therefore to compute the value of the state for route (x <-> y) we need to have already computed the value for all states with routes (a <-> b) where x <= a or y <= b (and they aren't the same route).

One natural way of ordering the states to achieve this is to sort the routes lexicographically (order small left indexes early, break ties with right index). Finally, to find the maximum value attainable from a child state we simply we simply query the maximum value produced from a route going out of each site that we've visited so far which we can calculate and update in an array as we process routes. In fact, there is no need at all to store the values associated with each route.

```
#define MAXN 40010
#define MAXR 100010

long long A[MAXN]; /* Values associated with left side of river. */
long long B[MAXN]; /* Values associated with right side of river. */
pair<int, int> E[MAXR]; /* Routes (first = left index, second = right
index). */

/* Maximum value attainable from each left index with the routes processed so
 * far. DPB is similar for right side of river. */
long long DPA[MAXN];
long long DPB[MAXN];

int main() {
  freopen("route.in", "r", stdin);
  freopen("route.out", "w", stdout);
```

```
  int N, M, R;
  scanf("%d%d%d", &N, &M, &R);
  for(int i = 0; i < N; i++) {
    scanf("%lld", A + i);
  }
  for(int i = 0; i < M; i++) {
    scanf("%lld", B + i);
  }
  for(int i = 0; i < R; i++) {
    scanf("%d%d", &E[i].first, &E[i].second);
  }
  sort(E, E + R);

  memcpy(DPB, B, sizeof(B));
  memcpy(DPA, A, sizeof(A));
  for(int i = R - 1; i >= 0; i--) {
    int u = E[i].first - 1;
    int v = E[i].second - 1;

    /* Compute the new maximum values attainable using this route. */
    long long vl = A[u] + DPB[v];
    long long vr = B[v] + DPA[u];
    DPA[u] = max(DPA[u], vl);
    DPB[v] = max(DPB[v], vr);
  }
  printf("%lld\n", max(*max_element(DPA, DPA + N),
                       *max_element(DPB, DPB + M)));
  return 0;
}
```

Note: Given the constraints it was possible for 32 bit signed arithmetic to overflow. However this was not the intention of the problem setter. With constraints that may have misled contestants to think overflow impossible we felt it was fairer to not include cases that required more than 32 bit signed arithmetic.