USACO MAR10 Problem 'balloc' Analysis

by Jaehyun Park

At first glance, the problem looks like a maximum flow problem, probably because of the word "capacity." However, this problem actually has a greedy solution, similar to many typical scheduling problems. First, sort the intervals by B_i in increasing order. If two or more intervals have the same B_i value, shorter intervals come before longer ones. Now, we greedily allocate the intervals from the beginning of this sorted list. That is, if the current interval can be allocated, we always allocate it.

The correctness of this algorithm can be proven using a simple contradiction argument with some cases. Assume this algorithm didn't give as many intervals as the optimal algorithm. Let P be the set of intervals chosen by the greedy algorithm, and Q be the optimal set of intervals. Sort the intervals in the same order as described above, and compare the intervals one by one. Find the first i such that P[i] is not equal to Q[i]. Then, there are several cases to consider.

(1) Q[i] ends before P[i].
This case is impossible because the greedy algorithm would have chosen Q[i] instead of P[i] if Q[i] ends before P[i].

(2-1) P[i] ends before Q[i], and they don't intersect.
This is contradictory, because the Q, which is supposed to be optimal, can become even larger by including this interval P[i].

(2-2) P[i] ends before Q[i], and they intersect.
We can make another optimal set Q' by throwing out Q[i] and selecting P[i] instead. Then, we will end up one of the cases above.

Note that all three cases lead to a contradiction, which means that the greedy algorithm is indeed optimal. The rest of the problem is that whenever we allocate an interval, we need to update the capacities quickly enough to yield an O(n log n) algorithm. This can be done by maintaining an interval tree, or binary indexed tree. Please refer to the solution below, coded by Jelle van den Hooff. The trick is to use lazy evaluation on intervals.

```
#include <cassert>
#include <cstdio>
#include <cstring>
#include <algorithm>
#define A first
#define B second
using namespace std;

FILE *fin = freopen ("balloc.in", "r", stdin);
FILE *fout = freopen ("balloc.out", "w", stdout);

const int MAXN = 1<<17, INF = 1<<30;
int n, m, c[MAXN*2], d[MAXN*2];
```

```
pair<int, int> r[MAXN];

void prop (int x, int l, int r);

int get (int x, int l, int r, int a, int b) {
  if (b <= l || r <= a) return INF;
  if (a <= l && r <= b) return c[x];
  prop(x, l, r);
  return min (get (x*2, l, (l + r) / 2, a, b), get (x*2+1, (l + r) / 2, r, a,
b));
}

void dec (int x, int l, int r, int a, int b, int y=1) {
  if (b <= l || r <= a) return;
  if (a <= l && r <= b) { c[x] -= y, d[x] += y; return; }
  prop (x, l, r);
  dec (x*2, l, (l + r) / 2, a, b), dec(x*2+1, (l + r) / 2, r, a, b);
  c[x] = min(c[x*2], c[x*2+1]);
}

void prop (int x, int l, int r) {
  dec(x*2, l, (l + r) / 2, l, r, d[x]), dec(x*2+1, (l + r) / 2, r, l, r,
d[x]);
  d[x] = 0;
}

int main () {
  scanf ("%d%d", &n, &m);
  memset (c, 0, sizeof(c));
  memset (d, 0, sizeof(d));
  for (int i = 0; i < n; i++) scanf("%d", &c[MAXN+i]);
  for (int i = MAXN - 1; i >= 1; i--) c[i] = min(c[i*2], c[i*2+1]);
  for (int i = 0; i < m; i++)
    scanf ("%d%d", &r[i].A, &r[i].B), r[i].A--, r[i].B--;
  sort (r, r + m);
  int ans = 0;
  for (int i = n - 1, j = m - 1; i >= 0; i--)
    for (; j >= 0 && r[j].A == i; j--)
      if (get(1, 0, MAXN, r[j].A, r[j].B + 1) > 0)
        ans++, dec(1, 0, MAXN, r[j].A, r[j].B + 1);
  printf ("%d\n", ans);
  return 0;
}
```