Ice hockey

Divide the matches into two parts, each of size at most 20. Compute the sums of all (at most $2^{20} \approx 10^6$) subsets of the first part, and store them in a sorted array A. For each subset X of the second part, compute its sum s; the number n_X of ways this subset can be extended to a set of matches with cost at most M is the number of elements of the array A that are smaller or equal to $M - s$. This number can be determined by binary search in A. The result is obtained by taking the sum of n_X over all subsets X of the second part.

Nuclearia

Suppose that we already know the radiation levels for each cell. Then, we precompute the sum of radiation levels $r_{x,y}$ in each rectangle with corners $(1,1)$ and (x,y). The sum of radiation levels in a rectangle with corners (x_1,y_1) and (x_2,y_2) is $r_{x_2,y_2} - r_{x_1-1,y_2} - r_{x_2,y_1-1} + r_{x_1-1,y_1-1}$, and thus we can answer the queries in constant time.

So, we only need to compute the radiation levels. Consider the radiation from an explosion of one nuclear plant in a fixed row. Initially, the radiation is 0, then it grows linearly in some interval I_1, then it stays constant in an interval I_2, and then it decreases linearly in an interval I_3, eventually becoming 0. To deal with the linear intervals, we will actually compute two coefficients $c_{x,y}$ and $d_{x,y}$ for each cell (x,y), so that the radiation in the cell is equal to $c_{x,y} + d_{x,y} \cdot x$. So, we want to increase d in the interval I_1 by b and add a corresponding offset to c, decrease d in the interval I_3 and add a corresponding offset to c, and leave d unchanged and add a corresponding level of radiation to c in I_2. Note that we can perform these operations independently for each nuclear plant, as the levels of radiations are additive.

Doing the updates in each field separately would be too slow, though. So, we use a trick: we introduce auxiliary arrays c' and d' such that $c'_{x,y} = c_{x,y} - c_{x-1,y}$ and $d'_{x,y} = d_{x,y} - d_{x-1,y}$ for each (x,y). Then the values of c' and d' change only at the ends of intervals I_1, I_2, and I_3, and thus their values can be updated in a constant time in each row. At the end of the process, we can compute $c_{x,y} = c'_{x,y} + c_{x-1,y}$ and $d_{x,y} = d'_{x,y} + d_{x-1,y}$ in one pass over the arrays c and d.

This is still too slow, though, since for each nuclear plant, we may need to update values in H rows, and $N \cdot H$ may be too large (the solution as described above obtains 55 points). However, observe that for each plant, we only adjust the values of c' and d' on a constant number of intervals in columns or on diagonals, and the changes are always by adding some linear function in the y coordinate. Hence, we can repeat the same trick and introduce auxiliary arrays for keeping track of the changes of c' and d' in the columns and on the diagonals. After processing the plants, we then first summarize these to obtain the actual values of c' and d', and then obtain the values of c and d as described before.

The total time complexity of the algorithm is $O(N + WH + Q)$.

Calvinball, again

The task is to find a proper coloring of a graph by the smallest possible number of colors.

Tests 1–4: small tests, can be solved by brute force algorithms (somewhat sophisticated one is needed for the test 4), or by some heuristic.

Test 5: the graph contains no edge uv such that the difference of u and v would be divisible by 3. Consequently, assigning each vertex v the color $v \bmod 3$ gives a proper 3-coloring.

Tests 6 and 7: for each edge uv, we have $|u - v|$ bounded by a small constant (8). Hence, the best possible coloring can be found by dynamic programming. Let $V(a,b)$ denote the set of vertices $a, a+1, \ldots, b$. For each vertex v and for each coloring φ of 7 vertices, we compute the minimum possible number of colors in a proper coloring of the subgraph induced by $V(1,v)$ such that its restriction to $V(v-6,v)$ is equal to φ.

Tests 8 and 9: Graphs of maximum degree 4, and containing a clique on 4 vertices. Coloring by 4 colors can be found by taking a spanning tree and assigning colors greedily from leaves. This way, every vertex except possibly for the last one has a non-colored neighbor, and thus one color available. We could in theory run into a problem at the last vertex, but in practice this almost never happens (and if it does, choosing the colors for each vertex from the available ones at random fixes this with a good probability).

Test 10: the graph is bipartite and its coloring can be found by a standard algorithm.