Benelux
Algorithm
Programming
Contest
2016

Solutions

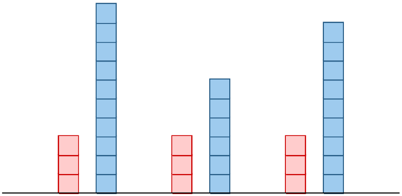
BAPC Preliminaries 2016

Delft University of Technology

September 24, 2016

A: Block Game

- Given stacks of height $a \geq b$, determine: can you win the game? There are three cases to consider.
  1. If $b \mid a$, you win by clearing the pile.
  2. If $b < a < 2b$, you have only one possible move, to $(b, a - b)$.
  3. If $a > 2b$, then you also always win. The position $(b, a\%b)$ must be winning or losing.
     - Losing: moving to $(b, a\%b)$ is a winning move.
     - Winning: moving to $(a\%b + b, b)$ is a winning move, because your opponent must move to $(b, a\%b)$.



- So simulate the game as long as you are in case 2.

B: Chess Tournament

- Is the set of reported chess matches inconsistent?
- In a graph:
  - Players for nodes;
  - Undirected edges for ties;
  - Directed edges for victories.
- Is there a cycle with at least one directed edge?
- Standard cycle detection algorithms only work on directed or undirected graphs, not mixed.
- Large input, so efficient solution is necessary!

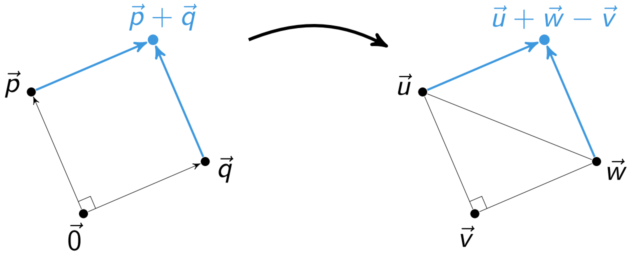
B: Chess Tournament

Benelux Algorithm Programming Contest 2016

- If two players are connected by a sequence of ties, they are of the same level.
- Collect all players into groups, based on who they tied with.
- Make a new graph with groups as nodes, and an edge from group A to group B if a player from A beat a player from B.
- Use flood fill algorithm. Complexity $\mathcal{O}(E)$.
- Look for cycles in this new graph. (Don't forget self-loops!)
- Use a standard topological sort. Complexity: $\mathcal{O}(E)$.

C: Completing the Square

- This was the easy problem.
- We are given an isosceles right triangle.
- It is not so hard to determine the location of the missing fourth corner once we know where the right angle is:



- How to find the right angle? Two options:
  - Look at the pairwise distances.
  - Look at the angles. (Two vectors p and q make a right angle at the origin if and only if the inner product $p \cdot q$ is zero.)

D: Hamming Ellipses (1)

- Task: Count the number of length-n strings over q symbols where $\mathrm{hammingdist}(p, f_1) + \mathrm{hammingdist}(p, f_2) = D$.

$f_1 = 0\ 1\ 2\ 0\ 1$, $f_2 = 2\ 1\ 2\ 1\ 0$, $p = 1\ 0\ 0\ 0\ 2$

- In positions where f_1 matches f_2, the symbol in p may
  - (k_1) match f_1 and f_2, or
  - (k_2) differ from both f_1 and f_2 in $(q-1)$ ways.
- In positions where f_1 differs from f_2, the symbol in p may
  - (k_3) differ from both f_1 and f_2 in $(q-2)$ ways, or
  - (k_4) differ from either f_1 or f_2 in 2 ways.
- Calculate $w = \mathrm{hammingdist}(f_1, f_2)$
- For all k_2, k_3, k_4 such that $k_2 \leq n - w$ and $k_3 + k_4 = w$ and $2k_2 + 2k_3 + k_4 = D$, count the number of points on the ellipse:

$$(q-1)^{k_2}\,(q-2)^{k_3}\,2^{k_4}\binom{n-w}{k_2}\binom{w}{k_3}$$

- Must be very careful to avoid overflow of `int64_t`

D: Hamming Ellipses (2)

Benelux Algorithm Programming Contest 2016

- Task: Count the number of length-n strings over q symbols where $\mathrm{hammingdist}(p, f_1) + \mathrm{hammingdist}(p, f_2) = D$.
- Alternative solution: dynamic programming over D and n.
- Construct a table $\mathrm{npoint}[k, d] =$ number of points at distance d, considering only the first k symbols of the strings.
- If f_1 and f_2 match at position k:
  $\mathrm{npoint}[k, d] = \mathrm{npoint}[k-1, d] + (q-1)\,\mathrm{npoin}[k-1, d-2]$
- If f_1 and f_2 differ at position k:
  $\mathrm{npoint}[k, d] = (q-2)\,\mathrm{npoint}[k-1, d-2] + 2\,\mathrm{npoint}[k-1, d-1]$
- Final answer is $\mathrm{npoint}[n, D]$
- Easier and safe against overflow.

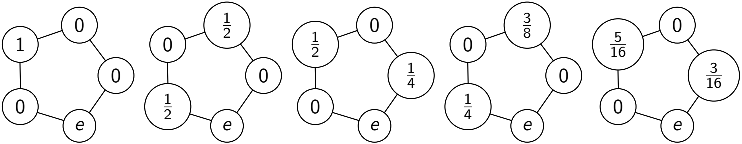
E: Lost in the Woods

Benelux Algorithm Programming Contest 2016

- What is the expected amount of time until your friend finds the exit?
- We can simulate the situation. Instead of simulating a single instance, we "simulate them all at once" as a Markov chain.
- Begin by putting probability weight 1 on the starting node, and 0 on all other nodes.
- At each step, redistribute the probability at each node to the nodes around it.
- Remove the weight at the exit of the woods, and update the expected time. Then repeat.
- Stop once the probability weight left in the woods is small enough.

E: Lost in the Woods

- Put weight 1 on starting node, 0 elsewhere.
- At each step: redistribute, update expected time.



$E = 0$ $\quad E = 0$ $\quad E = \frac{2}{4}$ $\quad E = \frac{7}{8}$ $\quad E = \frac{11}{8}$

F: Memory Match

- Simulate the previous actions in the game and build a partial list of known card pictures.
  Mark pairs that are already matched.
- Build a Map from picture name to card position.
- Each card is now in one of four states:
  - (a) Already matched.
  - (b) Picture known, location of matching card known.
  - (c) Picture known, location of matching card unknown.
  - (d) Picture unknown.
- Every two cards of type (b) can be matched.
- If there is an equal number of cards of types (c) and (d), every unknown card can be matched with a known card.
- Otherwise, if there are exactly two cards of type (d), they can be matched together.

G: Millionaire Madness

- Given a rectangular grid of heights, find the least $k \geq 0$ such that there is a path from one corner to another using a ladder at most k.
- There can be up to 10^6 points in the grid – an efficient algorithm is necessary!
- Use a variant of Dijkstra's algorithm with the priority queue sorting on required ladder length (shortest first).
- Alternatively, use binary search and repeated flood fills (BFS) to find the least k for which you can traverse the grid.

H: Presidential Elections

- The problem is a variation on the classical 0–1 knapsack problem, which can be solved using dynamic programming.
- For each state i let A_i denote the number of *additional* votes required to win this state:

$$A_i = \max\left(\underbrace{\left\lfloor \frac{C_i + F_i + U_i}{2} \right\rfloor + 1}_{\text{the absolute majority}} - C_i\ ,\ 0\right).$$

  If we have $A_i > U_i$, then there is no way to win this state.
- Take as knapsack items all states satisfying $A_i \leq U_i$. All other states are discarded. The i-th state has price A_i and value D_i.
- Find cheapest way to fill strictly more than half of your knapsack with these items (standard 0–1 knapsack algorithm).
- Time complexity: $\mathcal{O}(S \cdot D_{tot})$, where D_{tot} denotes the total number of delegates, all states combined.

I: Rock Band

- Need to draw a vertical line such that each song only occurs on one side of the line:

4	5	2	1	6	8	3	7
5	2	4	8	6	1	3	7
2	5	4	8	1	6	3	7

- Find leftmost such line.
- Can be solved greedily in $\mathcal{O}(MS)$ time. For instance:
  - Precompute for each song its worst ranking.
  - Start with a vertical line after the first column.
  - Process all columns lying left of the line. If we encounter a song here whose worst ranking is right of the line, move the line further to the right, just beyond this worst ranking.
  - Stop once we have a stable set (all columns left of the line have been processed).
- Other similar greedy solutions will also work.

I: Rock Band

- Alternative solution: create a directed graph of songs where an arrow $X \to Y$ means

  > "If we play song X, then we should also play song Y."

- For each band member we add a path of $S - 1$ edges:

  $$4 \leftarrow 5 \leftarrow 2 \leftarrow 1 \leftarrow 6 \leftarrow 8 \leftarrow 3 \leftarrow 7$$
  $$5 \leftarrow 2 \leftarrow 4 \leftarrow 8 \leftarrow 6 \leftarrow 1 \leftarrow 3 \leftarrow 7$$
  $$2 \leftarrow 5 \leftarrow 4 \leftarrow 8 \leftarrow 1 \leftarrow 6 \leftarrow 3 \leftarrow 7$$

  (Image is a little misleading; we have one vertex per song.)

- To find the minimum length set list:
  - Pick one song X that we know has to be played. Any song ranked first by one of the band members suffices.
  - Find the set of all songs reachable from X.
  - This always gives the unique minimum length set list.
- Use BFS/DFS on a graph with S vertices and $M(S-1)$ edges. Time complexity: $\mathcal{O}(MS)$.

J: Target Practice

- Given a set of points, find out if two lines cover them.
- Ways to find at least one of the lines (if two covering lines exist):
  - Of any five points, three must be collinear. This gives one of the lines.
  - Of any three points, two must lie on one of the lines.
  - By repeatedly randomly picking two points, you are almost guaranteed to get two points on the same line.
- Once you have a candidate for one of the lines, it is easy to check if all remaining points lie on a line.

K: Translators' Dinner

- Let languages be nodes and translators be edges.
- Given a connected graph, give a matching of the edges, or report that no such matching exists.
- Theorem: a matching exists if and only if the number of edges is even.
- A proof of this theorem often leads to an algorithm, or vice versa!

K: Translators' Dinner

Benelux Algorithm Programming Contest 2016

- One solution uses an *almost spanning tree*, or AST.
- An AST on a graph G is a subtree of G which contains all vertices, except possibly some vertices of degree 1, which connect directly to the tree.
- Any spanning tree is also an AST.
- Any graph with an AST is connected.

K: Translators' Dinner

- Construct an AST T on the graph (by making a spanning tree).
- For a leaf $l \in T$, if there are at least two edges incident to l which are *not* in T; match them and remove them from the graph.
- Repeat until there are zero or one such edges left.
  - One: match that edge with the edge that connects the leaf to the tree and remove both of them from the graph.
  - Zero: remove the leaf from T (but not the graph).
- Repeat with a new leaf until T (and thus the graph) is empty.
- Because T is always an AST, this works.