University of Chicago Invitational Programming Contest 2012

Problem analysis

Artem Vasilev Vitaly Aksenov

ETH Zurich Training Week, April 2017

A. CosmoCraft

Problem statement

- An RTS-style game.
- Produce facilities, workers or army units.
- Survive all the attacks and build the largest army by turn t.

A. CosmoCraft

Solution

- A few observations:
  1. Always spend all money
  2. Useless to have an army unit sitting here for more than 2 rounds
  3. Only buy production when there's nothing else to buy
  4. Unless we need to buy army over two turns and there are more workers than production
- In each turn, do the following:
  1. Either the game in two turns, so create army units using all money
  2. If we can make enough army to survive the next attack, do it next day
  3. Otherwise, build the army over the 2 turns
- Use all remaining money to build workers and production

B. Covered Walkway

Problem statement

- Cover n points on the plane with some number of segments.
- The cost of segment (l, r) is $c + (r - l)^2$.
- Straightforward DP is $O(N^2)$ and is too slow.

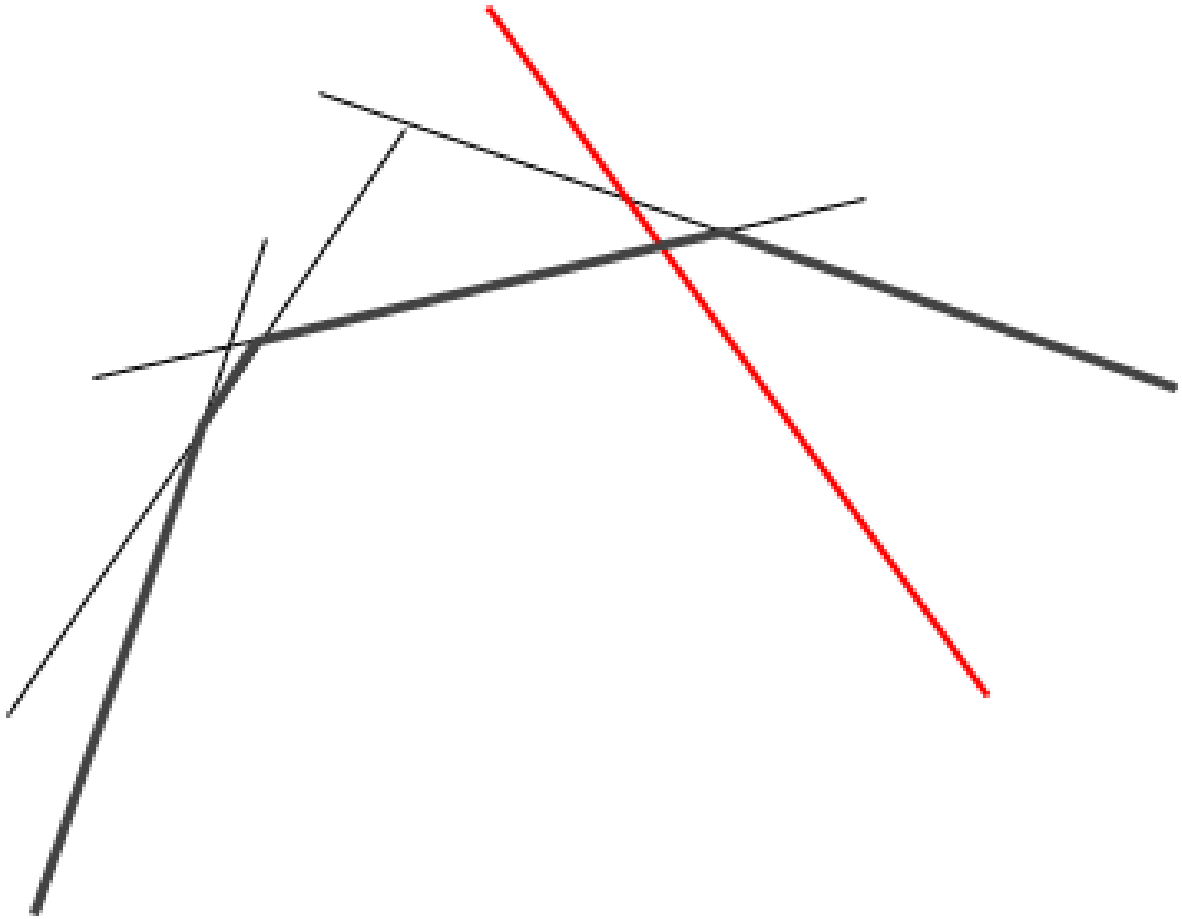
B. Covered Walkway

Convex Hull Optimization

- Consider a data structure that hold linear functions $y_i(x) = a_i x + b_i$
- Need to implement function $add(a_i, b_i)$ and $get(x) = \min_i y_i(x)$.
- Assume that lines are added in the order of *decreasing* a_i.
- Keep the lower hull of all the lines in a stack (just like in the convex hull algorithm).
- When adding a line, remove lines from the top of the stack until it's convex.

B. Covered Walkway

Convex Hull Optimization



B. Covered Walkway

Convex Hull Optimization

- Add a line $y = a_i + b$. `getX(a, b)` returns x coordinate of intersection of lines a and b.
- ```
  while (getX(prevLast, newLine) < getX(prevLast, last))
      remove(last)
  ```
- Get value for x: find an optimal line such that x lies between intersection with neighboring lines.
- Binary search for lines in stack in $O(\log n)$.
- If, in addition, queries x are always increasing, you can keep a pointer to the current best line.
- It only moves to the right, except when the best line was deleted, so it's $O(1)$ amortized.
  ```

B. Covered Walkway

Solution

- Use Convex Hull Optimization to calculate DP
- $dp_i = \min_{j<i}(dp_j + (x_i - x_{j+1})^2 + c) = x_i^2 + c + \min_{j<i}(dp_j + x_{j+1}^2 - 2x_i x_{j+1})$
- Set $a_j = -2x_{j+1}$, $b_j = dp_j + x_{j+1}^2$.
- Can use Convex Hull Optimization to calculate all dp_i values in $O(1)$ time.

C. Double Dealing

Problem statement

- Given a deck of n cards and a deck shuffling method.
- Give first card to first player, second — to second, ..., k-th to k-th, then loop.
- Then concatenate the players' decks.
- How many shuffles until the cards in the deck are in original order?

C. Double Dealing

Solution

- One shuffle is a permutation of initial order of cards, call it π.
- After k shuffles it becomes π^k (ordinary permutation multiplication).
- We need to find number n such that π^n is an identity permutation (called the *order* of the permutation).
- Order of permutation is the LCM of the lengths of its cycles.
- Find all cycles, calculate $\text{lcm}(a, b) = \frac{a}{\gcd(a,b)} \cdot b$ to avoid overflow.

D. The End of the World

Problem statement

- Given a Hanoi Tower puzzle configuration (a part of the optimal solution).
- Find the number of moves left.

D. The End of the World

Solution

- Write a recursive function $f(n, c)$ for calculating the cost of moving n smallest disks from the current configuration to peg c.
- If the largest disk is already on peg c, do nothing and call $f(n-1, c)$.
- If it's on peg $d \neq c$, then you'll have to move all disks smaller than it on the third peg $e \neq c$ or d, then move everything to c.
- Call $f(n-1, e)$ and add 2^{n-1} to the answer.

E. Estimation

Problem statement

- Given an array A, create an array B of the same length.
- B should have k contiguous sections with equal values.
- Minimize $\sum_i |A_i - B_i|$

E. Estimation

Solution

- The value that minimizes the sum $\sum_i |A_i - M|$ is the median of array A.
- Calculate medians and the minimum cost function for all subarrays of A.
- Keep the lower $\lceil \frac{n}{2} \rceil$ in the max-heap, and other $\lfloor \frac{n}{2} \rfloor$ elements in the min-heap.
- The median will be on top of the max-heap.
- You can add a number to this data structure and calculate $\sum_i |A_i - M|$ in $O(\log n)$ time.

E. Estimation

Solution

- Dynamic programming solution: $dp_{n,k}$ is the minimum cost for first n elements of A and k contiguous section.
- $dp_{n,k} = \min\limits_{j<i} dp_{j,k-1} + \text{cost}(j+1, i)$
- $O(nk)$ states, each transition is done in $O(n)$ time.
- Overall running time is $O(n^2 k + n^2 \log n)$.

F. Juggler

Problem statement

- Given a permutation π
- You can either shift it to the left, to the right, or remove the first element.
- Remove all the elements in the increasing order in minimal number of moves.

F. Juggler

Problem statement

- Since the order of removal is fixed, you need to determine the minimal circular distance to the next element.
- Keep the current shift and calculate the distance to the next element.
- The distance between two positions is the number of non-deleted elements between them, use segment tree or Fenwick tree (binary indexed tree) to calculate it in $O(\log n)$ time.

G. Red/Blue Spanning Tree

Problem statement

- Given a graph with red and blue edges.
- Determine if there's a spanning tree with exactly k blue edges.

G. Red/Blue Spanning Tree

Solution

- Find the minimum and maximum possible number of blue edges in a spanning tree.
- Every number between minimum and maximum is also possible.
- Minimal amount of blue edges: set red edges' costs to 0 and blue edges' to 1, find MST.
- Similarly for the maximum amount.

H. The Red Gem

Problem statement

- Given some circles on a plane and one special circle.
- Calculate the proportion of points on the large circumference for which the view of the special circle is not obstructed by other circles.

H. The Red Gem

Solution

- Let's look at just one circle.
- All points, for which the view is not obstructed by this circle, is an intersection of two half-planes.
- These half-planes are formed by two *inner tangents* of these two circles.
- If we consider all circles, there will be $2n$ half-planes to intersect.
- Each half-plane, when intersected with a circumference, leaves a valid segment.
- Use events and sorting to calculate the length of the intersection.

I. Science!

Problem statement

- Given a bipartite graph.
- Divide its edges into as many perfect matchings as possible.

I. Science!

Solution

- A k-regular bipartite graph can always be decomposed into k perfect matchings (can be proven using Hall's Theorem).
- Find the max k such that you can leave some edges to get a k-regular graph.
- Can be done with binary search + max flow algorithm, or just a simple path augmention.
- Then find the maximum matching k times and remove it from the graph.

J. The Worm in the Apple

Problem statement

- Given a set of points in 3D.
- Answer some queries: what is the distance from some point inside the convex hull to its face?

J. The Worm in the Apple

2 Easy Ways To Construct A 3D Convex Hull

- First, you'll have to construct a 3D convex hull.
- The first method is to do it iteratively: start with first 4 points and 4 faces.
- Add each point one by one and remove the faces that are seen from 'outside'.
- Then create the new faces with the new point added.
- The number of faces is linear, so each iteration works in $O(N)$ time, so the total time is $O(N^2)$.

J. The Worm in the Apple

2 Easy Ways To Construct A 3D Convex Hull

- Second way is similar to *gift-wrapping* algorithm.
- Write a recursive function that traverses all the edges of the convex hull.
- Find any edge of the convex hull, start from it.
- For each oriented edge, find the third point of that face in linear time, recurse from two other edges.
- Number of edges is still linear, each call of `dfs` works in linear time.

J. The Worm in the Apple

The Second Way

Code courtesy of **Lewin** at Codeforces:

```
// dfs(i,j), where i, j is any edge on convex hull, will visit all faces
void dfs(int i, int j) {
    if (vis[i][j]) return;
    vis[i][j] = true;

    int k = 0;
    while (k == i || k == j) k++;
    for (int l = 0; l < n; l++) {
        // side returns which side pts[l] lies on plane defined by pts[i, j, k]
        if (l != i && l != j && side(pts[i], pts[j], pts[k], pts[l]) > 0)
            k = l;
    }
    // points i,j,k form a face on convex hull.
    ans = min(ans, getVolume(i, j, k));
    dfs(k, j);
    dfs(i, k);
}
```