(ETH Zurich)+ Contest

Problem analysis

Artem Vasilev Pavel Krotkov

Peking University Camp, April 2016

A. Arithmetic

Problem statement

- Given a, b, prime p, and a number k, find any x that $a^x + b^x$ is divisible by p^k.

A. Arithmetic

Edge cases

- Both a and b are divisible by p: can set $x = k$
- Exactly one of a and b is divisible by p: output -1.

A. Arithmetic

Solution

- Let $m = p^{k-1}(p-1)$. By Euler's theorem, we have $c^m \equiv 1 \pmod{p}^k$
- Let's incrementally solve $a^{x_i} + b^{x_i} \equiv 0 \pmod{p}^i$ for all i from 1 to k. For $i = 1$, if x_1 exists, it exists in range $[1, p-1]$, so it can be brute-forced.
- By the same reasoning, solution for $i = 2$ lies in $[1, p(p-1)]$, using the additional information that $x_2 \equiv x_1 \pmod{p-1}$, so it can be found in $O(p)$ time.
- We can implement this solution recursively, trying all the solutions to x_i we can get, fast enough to get OK.

A. Arithmetic

Alternative solution

- Multiplying the equality by $b^{-1} \pmod{p}^k$, we get $c^x \equiv -1 \pmod{p}^k$, where $c = ab^{-1}$.
- Since p is odd, let $p - 1 = 2^d s$, where s is odd. Acting similarly to Miller-Rabin primality test, we try $x = \phi(p^k), \frac{\phi(p^k)}{2}, \ldots, \frac{\phi(p^k)}{2^d}$ as the possible solution, and if none are correct, output -1.
- No formal proof (can you prove it?).

B. Canton Courier

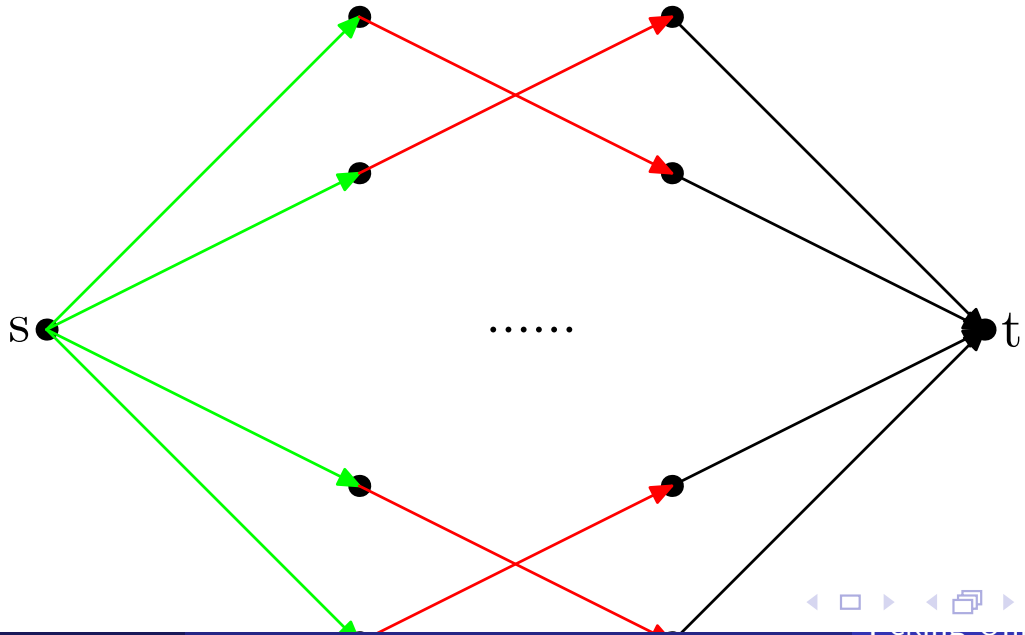
Problem statement

- n jobs, m zones
- Each job has p_i reward
- Each zone has c_i cost
- Job depends on subset of zones
- Job is done if it's subset is bought
- $n \leq 100$, $m \leq 100$

B. Canton Courier

Problem solution

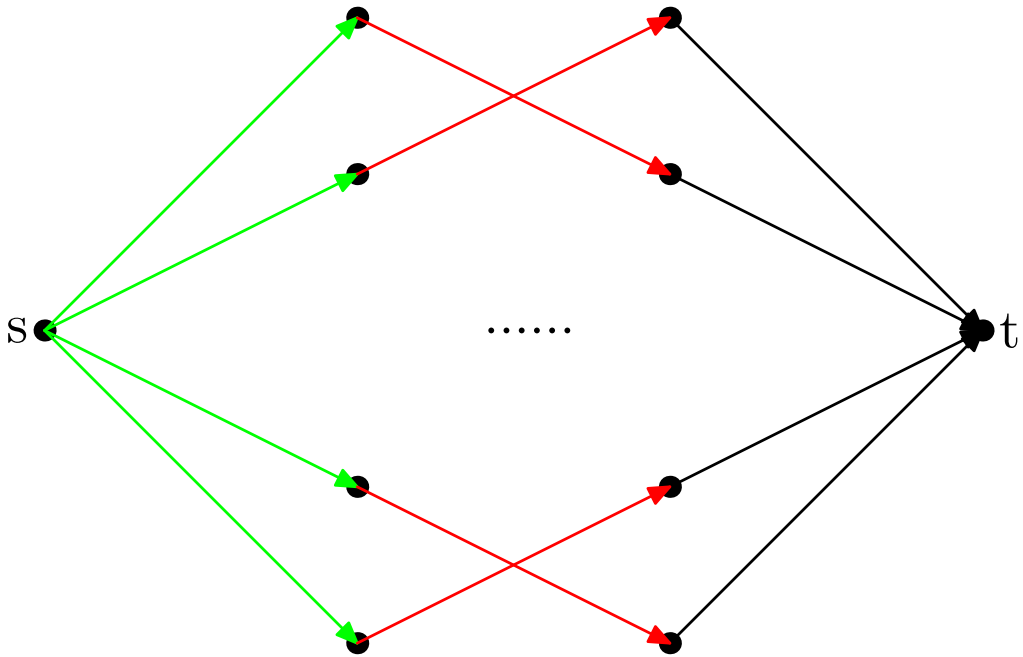
- Let's build a network with n vertices in first level and m vertices in second level
- Green edges: capacity p_i
- Red edges: capacity $+\infty$, connect jobs with corresponding set of zones
- Black edges: capacity c_i
- The answer is $\sum p_i - C$, where C is size of minimum cut



B. Canton Courier

Explanation

- Any cut in this network represents a solution
- Green edges in cut represent jobs we haven't done
- Red edges in cut represent zones we bought
- We need to minimize size of cut



C. Merlin's Orb

Problem statement

- N circumferences of different radii randomly pop op on the surface of a sphere. Count the expected number of intersection points.

C. Merlin's Orb

Solution

- Answer is the sum of intersection points for all pairs of circles.
- Each pairs of circles has 2 intersection points with probability p and 0 with probability $1 - p$.
- Let α_i be the angle between the sphere axis and any point on i circle if it's center is on the north pole: $\sin(\alpha_i) = \frac{r_i}{R}$
- Consider 2 circles with radii r_1 and r_2. Without loss of generality, assume $r_1 \geq r_2$ and the center of the first center is the north pole. Probability that these two don't intersect is the ratio of areas of 2 sphere caps to the sphere area and is equal to $\cos(\alpha_1 - \alpha_2) - \cos(\alpha_1 - \alpha_2) = 2\sin(\alpha_1)\sin(\alpha_2)$.
- So the answer is $\sum_{i<j} 2\sin(\alpha_i)\sin(\alpha_j)$, which can be computed in linear time.

D. Multidrink

Problem statement

- Given a tree with N ($N \leq 500000$) vertices.
- Find a permutation of vertices such that any 2 adjacent vertices are no farther than 2.
- Start in vertex 1 and end in vertex N.

D. Multidrink

Solution

- Construct the solution vertex by vertex.
- Write an auxiliary function: take a subtree rooted in v, visit all its descendants and return to v's parent or report there's no such path.
- Consider all children of v: if they are all leaves, just visit them in any order.
- Assume there's a child that's not a leaf, call it u. Then after v, we must go to a child of u.

D. Multidrink

More cases

- If all children of u are leaves, then everything's good.
- If u has a children that's not a leaf, then it should be visited *last*.
- Finally, if u has two or more children that isn't a leaf, there's no answer. Same holds for start vertex v.

D. Multidrink

Even more cases

- So far we've been solving the problem without considering the end vertex.
- Exactly one subtree of the start vertex contains n. Call the function so that subtree containing n is the 'parent'.
- After visiting all other subtrees, go to subtree containg n.
- Similar case handing as before.

E. New Year

Problem statement

- Given a bipartite graph and a number k.
- Color the maximum amount of edges using k colors, so that any two edges sharing the vertex have different colors.

E. New Year

Solution

- Theorem: any bipartite graph with maximum degree k can be edge-colored with k colors.
- Proof: there's a constructive algorithm for coloring edges similar to Kuhn's maximum matching algorithm.
- So we need to find a maximum subgraph with all vertices having degree $\leq k$.
- Use any maximum flow algorithm.

E. New Year

How to color

- Use an algorithm from theorem above.
- Alternatively, augment graph to have the same number of vertices in both parts.
- Then, add edges until each vertex has degree equal to k.
- Theorem: every k-regular bipartite graph contains a perfect matching. Corollary: it also contains k edge-disjoint perfect matchings.
- Just find a perfect matching k times, color each matching in it's own color.

F. Packing T-Shirts

Problem statement

- Solve knapsack problem for a specific weight sequence.
- $\sum_{i \in A} w_i = c$
- It's guaranteed that solution exists.

F. Packing T-Shirts

Solution

- Weights $w_i = (ra_i) \mod q$ are created from a *superincreasing* sequence a_i: $\sum_{n=1}^{i-1} a_n < a_i$
- Every weight can be represented by at most one subset of superincreasing sequence, and this subset can be obtained using greedy algorithm.
- $\sum_{i \in A} w_i = c$
- $\sum_{i \in A} (w_i r^{-1}) \mod q = (cr^{-1}) \mod q$
- $\sum_{i \in A} a_i = (cr^{-1}) \mod q$
- Since $\sum_{i=1}^{n} a_i < q$ and solution always exists, the only possible solution is the one retrieved from superincreasing greedy algorithm.

G. Fancy Runway

Problem statement

- Bit vector $a = (a_0, a_1, ..., a_n)$
- Every iteration — new vector
  $a' = (a_1, a_2, ..., a_n, a_0 \oplus a_{j_1} \oplus a_{j_2} ... \oplus a_{j_{k-1}})$
- Find when we will reach some particular state b (or determine that it will never happen)
- $n \leq 32$

G. Fancy Runway

Useful observations

- Can find such matrix B that $a' = B \times a$
- Can find state after T iterations quick enough ($B^T \times a$)
- This sequence will have cycle after at most 2^n iterations
- Can find previous state of a (guaranteed by the constraint about a_0)

G. Fancy Runway

Solution

- b_i — state b reverted on i steps
- Find b, b_{-1}, b_{-2}, ..., $b_{-2^{\frac{n}{2}}}$
- Find $a_{2^{\frac{n}{2}}}$, $a_{2\times 2^{\frac{n}{2}}}$, $a_{3\times 2^{\frac{n}{2}}}$, ..., a_{2^n}
- If these sets have common element — we found an answer
- If they don't — answer doesn't exist
- Might use some bit operations for quick matrix multiplication

H. Scouting Camp

Problem statement

- Set of points on plane
- Complete graph
- Distance between points — $|x_1 - x_2| + |y_1 - y_2|$
- Find MST
- $N \leq 20000$

H. Scouting Camp

Good enough solution

- Prim's algorithm
- $O(N^2)$
- Will probably work
- Optimization: store all vertices in list, remove the vertex after we added it to our MST

H. Scouting Camp

Correct solution

- Binary search for answer (the longest edge in MST)
- Answer check:
  - For every vertex get a "nearby area" (a square around it)
  - Check if all squares are connected
  - Can do it with sweeping line

I. Sequence

Problem statement

- Sequence of N integers
- Split it into maximum possible amount of parts
- Sums of parts — increasing sequence
- $N \leq 4000$

I. Sequence

Solution idea

- Dynamic programming
- $f_{i,j}$ — maximum possible amount of parts for the first i numbers if the last part consists of j elements
- Calculating:

```
for k = i + 1 ..  n do
  if (sum(i + 1, k) > sum(i - j + 1, i))
    f[k, k - i] = max(f[k, k - i], f[i, j] + 1)
```

- $O(N^3)$

I. Sequence

Optimization

- Consider we already calculated f values for first i elements
- Let's do all updates for all j values at once for $O(N \times log_2 N)$
- For every j value calculate pair $(sum(i - j + 1, i), f_{i,j})$
- For every $k > i$ we need to find pair with maximum second value among those with first value less then $sum(i + 1, k)$
- Sort this pairs by first values
- Precalculate maximum second values on prefixes
- Do binary search
- $O(N^2 \times log_2 N)$

J. Sliding Puzzle

Problem solution

- Simulate M steps of sliding puzzle game on the $N \times N$ field
- $N \leq 100$, $M \leq 10^5$
- The easiest problem
- Just do it!

K. Transport

Problem statement

- Consider graph
- All edges have weight a
- We add some new edges with weight b
- Edge (u, v) is added if and only if the shortest distance between u and v in the old graph is $2 \times a$
- Find all shortest distances from s

K. Transport

First idea

- Let's find shortest paths without new edges (BFS)
- Distance between u and v is $t \times a$
- If t is even, distance between u and v in the whole graph is $min(t \times a, \frac{t}{2} \times b)$
- If t is odd, distance between u and v in the whole graph is $min(t \times a, \lfloor \frac{t}{2} \rfloor \times b + a, ?)$

K. Transport

Second idea

- What is ? on the previous slide?
- Consider $b << a$
- In that case we may find some longer path of even length
- And it will be shorter than $\lfloor \frac{t}{2} \rfloor \times b + a$
- So let's do BFS on graph where all vertices have two states — odd and even

K. Transport

New trouble

- Solution still isn't entirely correct
- Consider we are at the vertex $(u, 1)$ and we came from the vertex $(v, 0)$
- We can't go from $(u, 1)$ to $(w, 0)$ if there is an edge (v, w)

K. Transport

Third idea

- Still BFS
- State of the BFS — current last edge on our way (and flag for odd/even)
- This way we can check the situations from the previous slide
- Now complexity of algorithm is $O(E^2)$ (easy to find a test)

K. Transport

Fourth idea

- Let's store all unchecked outcoming edges for every vertex
- Let's remove the edge once we checked it
- This way the complexity is $O(E + T)$
- We check $O(E)$ edges and we skip $O(T)$ edges
- T is the amount of triangles in our graph
- Estimate on T is $O(V\sqrt{V})$

L. Walk Through Byteland

Problem statement

- Consider graph
- Vertices — all binary vectors of length n except k "bad" ones
- Pair of vertices have an edge if and only if Hamming distance $= 1$
- $n \leq 60$, $k \leq 5 \times 10^6$
- Check if vertices s and t are connected

L. Walk Through Byteland

Solution idea

- BFS from s until:
  - We found t (1)
  - We can't go to any new vertex (2)
  - We have passed 10^6 vertices (3)
- Note, that in cases (1) or (2) the problem is solved
- In case (3) — same BFS from t
- If (3) in second BFS – answer is `YES`

L. Walk Through Byteland

Solution explanation

- Our graph is very "branching" (n edges from every vertex)
- Size of BFS levels grows exponentially
- If "bad" vertices block the way from s to t — they do it near s (or t)
- There are too many vertices at distant levels to block the way there
- No formal proof