2023 SBU ICPC Selection Contest Editorial

October 12, 2023

A - The Fancy New Hard Disk

Setter: *Tanzir Pial,* ***Validator:*** *Yimin Zhu*
Editorialist: *Nicholas Tarsis* ***Tag:*** *Implementation*

To see if the disk can fit through the door, it suffices to compare the diameter of the disk, or twice the radius with the width and length. If the minimum of the length and width is greater than or equal to the diameter, then the disk will fit through. This solution runs in $O(1)$ time for each testcase.

B - Weird Patterns by Roth

Validator: *Yimin Zhu*
Editorialist: *Jacky Xie* ***Tag:*** *Implementation*

$F(n)$ generates the Fibonacci sequence, $F(0) = 0, F(1) = 1, F(2) = 1, F(3) = 2, F(4) = 3, F(5) = 5, \ldots$.

Notice that the drawings are (vertically) symmetric, and consist of groups of two lines with the same # pattern. For $n = 5$, from top to bottom, we have two rows of 1 #, then two rows of 1, then 2, 3, 5, which is the Fibonacci sequence, matching $F(1)$ through $F(5)$.

The other part of the pattern is how far the #'s are from the sides. Counting the dots to the side, we have 0, then 1, then 2, 3, 4, so they increment one by one. The last two rows have no gap between the #'s on the left on right sides, so the width of the drawing will depend on that, $w = 2((n-1) + F(n))$.

$$\overbrace{\circ\circ\circ\circ}^{(5-1)}\overbrace{\#\#\#\#\#}^{F(5)}\overbrace{\#\#\#\#\#}^{F(5)}\overbrace{\circ\circ\circ\circ}^{(5-1)}$$

We get $n \leq 20$, $F(20) = 6765$, so the size of the drawing is $2 \cdot 20 \cdot 2(19 + 6765) = 271360$ characters. We can simply generate the drawing line by line using the process described above.

C - Walking Around Roth Just Creates Problems!

Problem Setter: *Tasnim Imran Sunny*
Validator: *Kenny Zhang*
Editorialist: *George Ivanchyk* ***Tag:*** *Digit DP, Combinatorics*

This problem can be solved with a common technique called "Digit DP".

Define $S(x) = F(1) + ... + F(x)$ and $dp_{i,j}$ = S(j00....00), where the last number has length i, the first digit is j and all the next $(i-1)$ digits are 0. Also, we can assume for convenience that $dp_{i,10} = dp_{i+1,1}$.

$S(j00....00)$ can be split into two parts: from 1 to $(j-1)0...0$ and from $(j-1)0...0$ to $j0....0$. First part is equal to $dp_{i,j-1}$, and the second part is $(j-1)$ times the sum of $F(x)$ for x between 0....0 and 10....0, which is equal to $dp_{i,1}$ for $i = 2$.

However, for $i > 2$ we should also consider that this sum is counted with leading zeros, so we have to subtract all numbers that start with 0, that is $dp_{i-1,1}$.

Considering all this, DP transition function has the following form:

$$dp_{i,j} = dp_{i,j-1} + dp_{i,1} * (j-1) - dp_{i-1,1} * (i > 2) \tag{1}$$

After that, we want to calculate $F(a) + ... + F(b) = S(b) - S(a-1)$, which can be solved by calculating $S(b)$ and $S(a-1)$ separately.
To calculate $S(x)$ for a given $x = d_1d_2...d_k$, we split this sum into 2 parts: $S(d_100...0)$ and $(S(x) - S(d_100...0))$. First sum is simply $dp_{d_1,k}$, and the second sum is $d_1 * S(d_2...d_k)$, which can be called recursively.

As before, we have to account for the numbers that start with leading zeros after the first recursive call, and for each of them subtract $dp_{i-1,1}$

Complexity: $O(log(b))$ per test case.

D - Upstate-NYC-Stony Brook

Problem Setter: *Tanzir Pial,* ***Validator:*** *Kenny Zhang*
Editorialist: *Tahsin Ahmed* ***Tag:*** *Segment/Fenwick Tree/ Policy Based Data Structures*

Any intersection between segments must start and end at one of the given endpoints. Therefore, we first compress event points by sorting the event points and assigning to each event point its rank.

We now consider how to find the number of intersecting LIRR train times for some (x_i, y_i) Upstate train time. Let $start[i]$ and $end[j]$ be the # of LIRR train times starting at the i^{th} ranked and ending at the j^{th} ranked times respectively. Then the # of LIRR train times that intersect Upstate train (x_i, y_i) is given by

$$\sum_{k=0}^{rank(y_i)} start[k] - \sum_{k=0}^{rank(x_i)-1} end[k]$$

The problem thus reduces to efficiently updating and querying a prefix sum data structure. This can be done in logarithmic time using either a fenwick tree or a segment tree. In total we use four fenwick/segment trees, two for start points and two for end points.

Time Complexity: $\mathcal{O}((n+m+q)\log(n+m+q))$.

E - Can Disco Lights Distract Contestants Enough

Validator: *Kenny Zhang*
Editorialist: *Greg Zborovsky* ***Tag:*** *Matrix Exponentiation, Graph Theory*

For each test case, we are given an undirected graph with N ($0 \le N \le 100$) nodes and M ($0 \le M \le \frac{N(N-1)}{2}$) edges, a reach R ($0 \le R \le 10$), and a time P ($0 \le P \le 10^9$) in minutes. We are also given an initial coloring for the nodes.

Our goal is to compute the colors of the of each of the nodes after P minutes.

The update rule for colors is

$$\text{color}(u,t) = \text{color}(u,t-1) + \sum_{v \in \text{adjacent}(u,(t-1 \bmod R)+1)} \text{color}(v,t)$$

where $\text{color}(u,t)$ is the color node u at time t and $\text{adjacent}(u,k)$ is the set of nodes whose shortest path to u is of length k.

The key observation is that the update rule for colors can be re-written as matrix multiplication.

Let us first solve the following simpler problem:

$$\text{color}(u,t) = \text{color}(u,t-1) + \sum_{v \in \text{adjacent}(u,1)} \text{color}(v,t-1)$$

Note that $\text{color}(u,t)$ is just the sum of its own color from the previous timestep and all of its neighbors' color values from the previous timestep. Rewritten with matrices it looks like:

$$\text{colors}(t) = \boldsymbol{A}\,\text{colors}(t-1)$$

where $\boldsymbol{A}$ is the adjacency matrix of the given graph, or in other words, the matrix of shortest paths of length 1. $\text{colors}(t)$ is a vector of length N where $\text{colors}(t)_i = \text{color}(i,t)$. Applying repeated matrix multiplications, we find that:

$$\text{colors}(t) = \boldsymbol{A}^t\,\text{colors}(0)$$

Naively, computing $\text{colors}(P)$ will take $\Theta(N^3 P)$ time, but using fast matrix exponentiation, we can compute $\text{colors}(P)$ in $\Theta(N^3 \log P)$.

Returning to the original problem, we need to be able to compute the adjacent function for all lengths and shortest path lengths $\in [1,R]$. We can easily do this by running the Floyd-Warshall algorithm to find all-pairs shortest paths and then constructing R adjacency matrices: $\boldsymbol{A}_{1\leq i\leq R}$ where $\boldsymbol{A}_i =$ the adjacency matrix where there is an edge between nodes u and v if and only if the shortest path between u and v has length i. Note that we can now re-write the original update rule as:

$$\text{colors}(t) = \boldsymbol{A}_{(t \bmod R)+1} \dots \boldsymbol{A}_2 \boldsymbol{A}_1 \left(\boldsymbol{A}_R \dots \boldsymbol{A}_2 \boldsymbol{A}_1\right)^{\lfloor \frac{t}{R} \rfloor} \text{colors}(0)$$

.

We can compute $(\boldsymbol{A}_R \dots \boldsymbol{A}_2 \boldsymbol{A}_1)^{\lfloor \frac{t}{R} \rfloor}$ quickly by first computing $\boldsymbol{A}_R \dots \boldsymbol{A}_2 \boldsymbol{A}_1$ and then performing fast matrix exponentiation on the resulting matrix.

The total time complexity is $\Theta(N^3)$ to run Floyd-Warshall, $\Theta(N^2 R)$ to compute the adjacency matrices, $\Theta(N^3 R)$ to compute $\boldsymbol{A}_R \dots \boldsymbol{A}_2 \boldsymbol{A}_1$, and $\Theta(N^3 \log P)$ for fast matrix exponentiation. The total time complexity is $\Theta(N^3(R + \log P))$.

F - Mandatory Problem about Cats

Validator: *Tanzir Pial, Jiarui Zhang*
Editorialist: *Kevin Cai, Tanzir Pial* ***Tag:*** *Number Theory, Probability*

The probability of choosing a brown kitten for box i would be $(X - A_i)/X$. Because all the boxes are independent of each other, the probability of picking only brown kittens would be

$$[(X - A_1) * (X - A_2) * \dots * (X - A_N)]/X^N$$

If any $A_i = X$, then output 0. If all $A_i = 0$, then output 1.

Notice that keeping track of count of the prime factors of X suffices since the denominator only has those factors. Therefore we do not need to prime factorize all $(X - A_i)$. The number of prime factors of X is $< O(log(X))$ and the prime factorization can be done in $\mathcal{O}(\sqrt{X})$. Then for every $(X - A_i)$, we can iterate through the prime factors of X and try to divide $(X - A_i)$ as many times as we can and update the count of that prime factor for the denominator.

Time Complexity: $\mathcal{O}((N \log X) + \sqrt{X})$.

G - SBU Campus Network

Validator: *Jiarui Zhang, Kenny Zhang*
Editorialist: *Xiao Sun* ***Tag:*** *Biconnected Components, Articulation Points, Block Cut Tree*

This problem is about biconnectivity. There are two related concepts:

- **Articulation points** (or more commonly, **cut vertices** or **vertices cut**) refer to the vertices whose removal increase the number of connected components in the graph.
- **Biconnected component**: a maximal biconnected induced subgraph. **Biconnected** means the (sub)graph has no cut vertices.

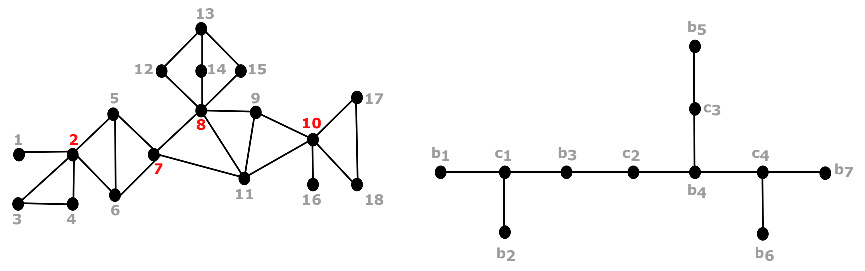

Figure 1: [Source: wikipedia] In the left graph, cut vertices are $[2, 7, 8, 10]$. The right graph is the block-cut tree of the left graph. $b_1 = [1, 2]$, $b_2 = [2, 3, 4], b_3 = [2, 5, 6, 7], b_4 = [7, 8, 9, 10, 11], b_5 = [8, 12, 13, 14, 15], b_6 = [10, 16], b_7 = [10, 17, 18]$.

For example, in Figure 1 there are 4 cut vertices in the left graph. We can also find the biconnected components in the right figure, known as the block-cut tree. In the block-cut tree, each vertex is either a cut vertex (start with label c) or a biconnected component (start with label b) in the original graph. Notice that a cut vertex can be in multiple blocks.

Now back to our original question. We can observe that a fail on a cut vertex is lethal, as it disconnects the graph. So our main concern is the set of cut vertices. There are two cases to consider:

- There are no cut vertices in the graph. In this case, the graph itself is biconnected. So two servers are necessary and sufficient. Clearly one server is not enough as that particular server can fail. Two servers are enough from the definition of biconnectivity (any removal of vertex would not increase the number of connected components). So if any single server fail, other vertices still have a path to another server. Any pair can be chosen as the servers, so there are $C(n, 2)$ choices.

- There is at least one cut vertex in the graph. If we count the number of leaves in the block-cut tree, let the number be K. K servers are necessary and sufficient. If we have less than K servers, then at least one leaf block is not deployed with a server. If the cut vertex in that leaf block fail, all other nodes in the block are dead. K is also sufficient, we can deploy one at any non vertex cut node for each leaf. Then all leave nodes are safe. For other nodes, there are at least two paths to leaves, which also secures their safety. The choice for each leaf is independent. So the number of choices is just the multiplication of all of them.

The overall complexity is $O(m + n)$. Wherer m is the number of edges and n is the number of vertices. We use $O(m + n)$ to find all cut vertices and biconnected components using Tarjan's algorithm (it is similar to the one that finds the strongly connected components (SCC)). To enumerate all leaves we can simply check if the number of cut vertices in the biconnected component is exact 1. Enumeration also takes $O(m + n)$.

H - Peace Maker

Problem Setter: *Yimin Zhu,* ***Validator:*** *Jiarui Zhang*
Editorialist: *Yimin Zhu* ***Tag:*** *Simple Math*

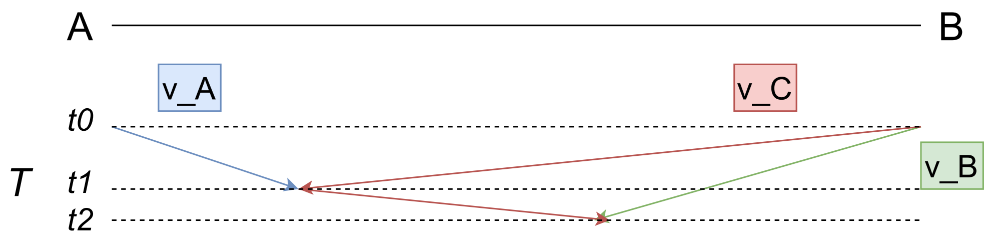

Figure 2: Peace Maker

This problem is simple math. As shown in Fig. 2, A represents Yimin, B represents Tanzir, C represents Jiarui, t represents time, and v represents velocity. They all begin at t_0. A and B meets at t_1 and then B and C meets at t_2. The time period $t_1 - t_0$ is $|AB|/(v_A + v_C)$ and time period $t_2 - t_1$ is $(|AB| - (t_1 - t_0) * (v_A + v_B))/(v_B + v_C)$. Summation of the two time periods will be t_2 and the result is $V_B \times t_2$.