MIAKI Presents

Inter University Programming Contest (IUPC)

Hosted By
Department of Computer Science and Engineering
Khulna University of Engineering and Technology

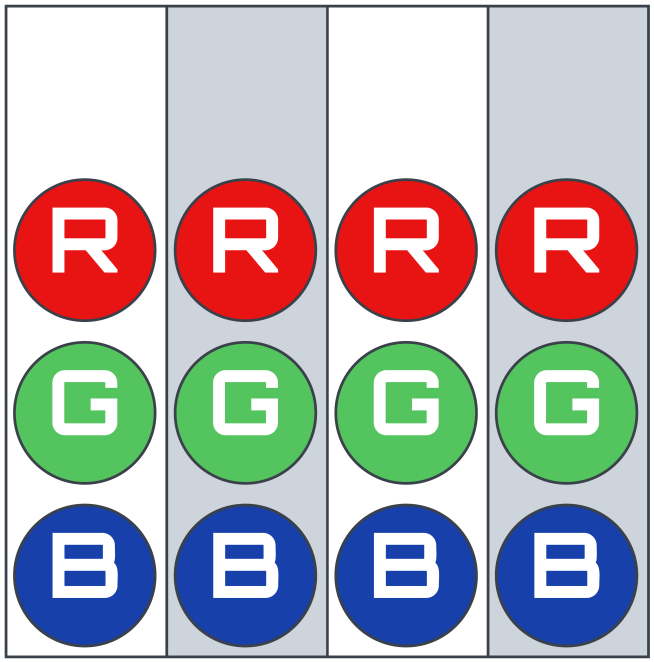
Supported by

Problem A. Perpendicular Parking

Problem Setter: Irfanur Rahman Rafio
Tester: Aminul Haq, Md Sabbir Rahman
Category: Constructive, Math
Total Solved: 10
First to Solve: BUET_hotasha

Since the driving lane is used to only take the frontmost car of one parking lane to another, we can remodel the problem without the driving lane and consider the parking lanes as stacks. We can also represent every university by a separate color for simplicity of explanation.

Initial arrangement for $n = 3$:



Problem Analysis

In a perfect arrangement, all cars of the same color (belonging to the same university) will be in the same lane. This arrangement can also be defined as the arrangement where one lane is empty and in the other lanes, all the cars have the same color as the backmost car of its lane.

To begin, let's define the operation of moving all cars of a specific color c into a designated lane l as 'solving color c in lane-l'. Let's define the 'size' of a lane as the number of cars it contains. Between two lanes, let's define the one with fewer cars as 'smaller.' Finally, let's define two lanes as 'symmetrical' if they have the same size and contain cars of the same colors in the same order.

For solving the first color, we need to first choose a color and a lane. Emptying that lane is necessary because the backmost car of that lane needs to be popped to make way for the color to be solved (unless the colors match). Solving all n colors will solve the problem.

After solving any color, we can forget about that color and the lane chosen for that operation. This is because there is no reason to pop from that lane and since that lane is full, it is not possible to push into that lane either.

Exploration

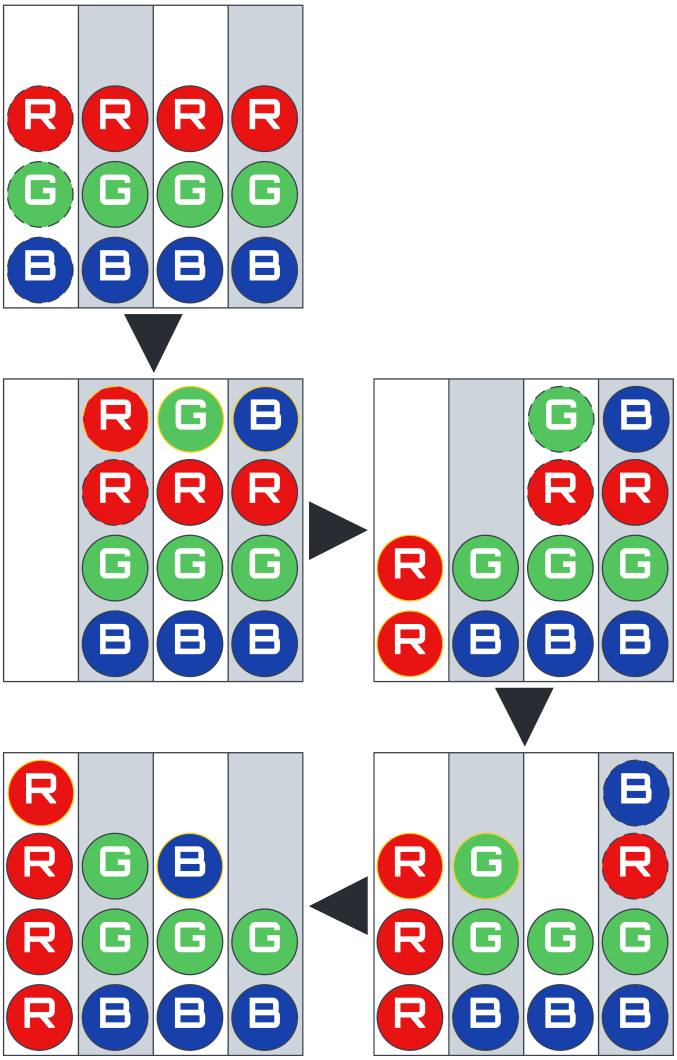
Let's explore the case $n = 3$ to try to generate an idea.

Solving First Color

For the arrangement shown above, it is intuitive that solving R first will be the easiest, as otherwise those cars will get in the way.

For solving R, we can follow these steps:

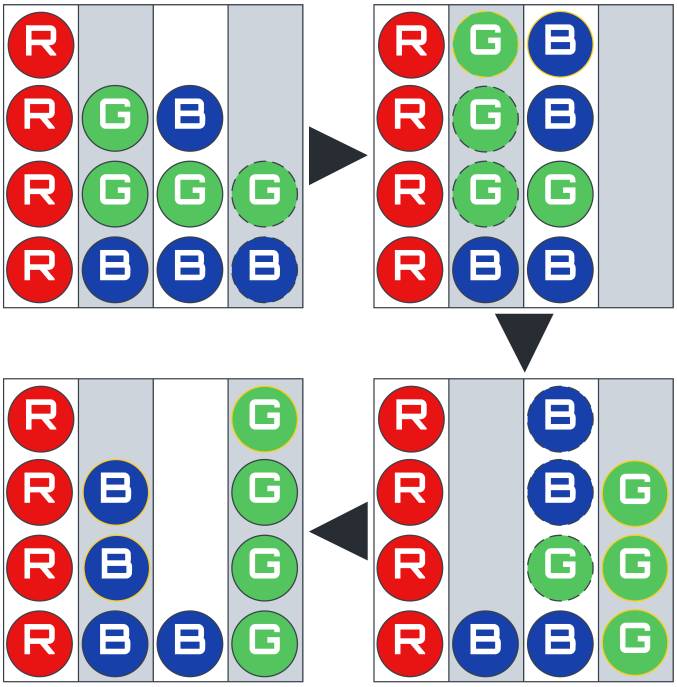
1. Empty lane-1 by driving the cars into lane-2, lane-3, and lane-4 respectively.
2. Drive the 2 Rs from lane-2 to lane-1.
3. Drive the G from lane-3 to lane-2, then drive the R from lane-3 to lane-1.
4. Drive the B from lane-4 to lane-3, then drive the R from lane-4 to lane-1.



Solving Second Color

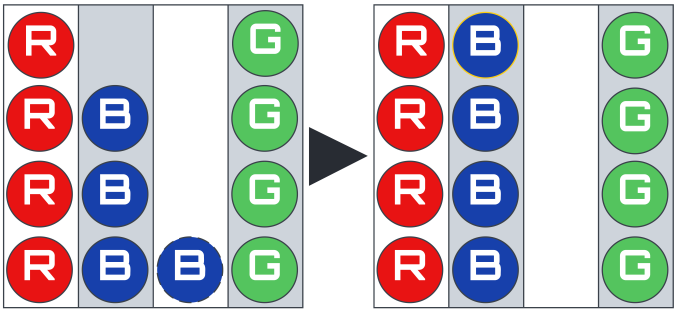
Next, we will be solving G. For that, we can follow these steps:

1. Empty lane-4 by driving the cars into lane-2 and lane-3 respectively.
2. Drive the 3 Gs from lane-2 to lane-4.
3. Drive the 2 Bs from lane-3 to lane-2, then drive the G from lane-3 to lane-4.



Solving Final Color

Finally, we can simply drive the B from lane-3 to lane-2 to solve the problem.



Idea Generalization

It is easily understandable that the 'empty and fill' algorithm described above can be generalized for any value of n.

- Start with the frontmost color as the first color to solve.
- Choose a lane, empty it, and fill it entirely with cars of the chosen color. To achieve this, make minimal extra moves to temporarily relocate other cars that get in the way. Also ensure that all 'moved' cars of the same color are kept together in a single lane for later operations.

- In this way, all lanes will be symmetrical except one which will be smaller.
- Solve the frontmost color from the remaining unsolved colors in the smallest lane in the same manner.
- Again, after solving a color, all lanes will be symmetrical except one.
- Repeat this process until only one color remains unsolved.
- Solve the last remaining color using a single move.

One clean way to implement this is solving the colors from front to back in lanes from right to left. In this way, after solving the i-th color in the lane-$(n+1-i)$, we only need to consider the colors from $i+1$ to n and the lanes from 1 to $n-i$. The sequence of moves can be generated deterministically without keeping track of the actual lanes.

Verification

For solving the 1st color, emptying requires n moves and filling requires $2n$ moves (because the backmost Rs are in level-2 of the arrangement). In total, this requires $3n$ moves.

For solving the 2nd color, emptying requires $n-1$ moves and filling requires $3(n-1)$ moves. In total, this requires $4(n-1)$ moves.

Generalizing this pattern, we can see that solving the i-th color will require $(i+2)(n+1-i)$ moves where $i < n$.

Finally, 1 move is required to solve the last color.

So, the total number of moves required is $\sum_{i=1}^{n-1}((i+2)(n+1-i))+1$.

Expanding this series, we get $\dfrac{(n+1)(n^2+8n-6)}{6}$ moves.

Solving the inequality $\dfrac{(n+1)(n^2+8n-6)}{6} < 50n^2$ for positive integer values of n, we get $n < 291$. Since the constraint of the problem limits the value of n to 200, this solution will always work and the impossible case will never occur.

(Proved)

Alternate Solution

Assume the last color is solved using the same method as the other colors instead of a single move. This increases the total number of moves by $n+1$, which is acceptable within the constraints. The rest of the algorithm remains unchanged.

With this adjustment, every car requires exactly one move to be placed in its correct lane, resulting in $n(n+1)$ moves.

Additionally, temporary moves are needed. For the 1st color, one car requires 1 extra move. For the 2nd color, one car requires 1 extra move, and another requires 2 extra moves (once while solving the 1st color and again while solving the 2nd). For the i-th color, one car requires 1 extra move, another requires 2 and so on up to i.

The total extra moves are the sum of the first n triangular numbers $= 1+(1+2)+(1+2+3)+\cdots+(1+2+\cdots+n)$.

$\therefore$ The total number of moves $= n(n+1) + \sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{n(n+1)(n+8)}{6}$.

For $n \leq 200$, this will not exceed $50n^2$ and so the solution is always possible.

(Proved)

Problem B. Baskets and Stones

Problem Setter: Md Sabbir Rahman
Tester: Nafis Sadique
Category: Combinatorics, Expected Value
Total Solved: 0
First to Solve: N/A

Number the baskets as $1, 2, 3, \cdots$ according to when they came from the stock (1 was originally in front of you). Also number the stones as $1, 2, 3, \cdots$ according to the order when they are picked from pile. Now the expression $X_1^2 + X_2^2 + \cdots$ can be expressed Also

$$\sum_{i=1}^{n}\sum_{j=1}^{n} I(i,j)$$

Where $I(i,j)$ is 1 iff stone i and stone j finally went to same basket. Let us compute this. If $i = j$ then $I(i,j) = 1$ obviously. Otherwise assume $i < j$ then the probabilty of $I(i,j) = 1$ is the probabilty of stone j going to same basket as the one where i is. Assume there are k baskets present when it's time for j to be put in a basket. Then the probabilty is $\frac{1}{k}$. k basket being present during stone j's turn means $k-1+j$ coin flips were done, the last one was tails and $k-1$ of the flips were heads (one basket is originally in front of you). This probability can be expressed as $\binom{k-1+j-1}{k-1}p^{k-1}(1-p)^j$. Multiplying $\frac{1}{k}$ and summing this for $k = 1$ to infinity gives us

$$\sum_{k=1}^{\infty}\binom{k-1+j-1}{k-1}p^{k-1}(1-p)^j\frac{1}{k}$$

Also note that this expression is independent of i. So in fact, we can say that for all expression $I(a,b)$ with $a \neq b$ and $\max(a,b) = j$ this expression will contribute to final result. So the contribution of j is

$$\begin{aligned}
&2(j-1)\sum_{k=1}^{\infty}\binom{k-1+j-1}{k-1}p^{k-1}(1-p)^j\frac{1}{k}\\
=&2(j-1)(1-p)^j\sum_{k=1}^{\infty}\binom{k-1+j-1}{k-1}p^{k-1}\frac{1}{k}\\
=&2(j-1)(1-p)^j\sum_{x=0}^{\infty}\binom{x+j-1}{x}p^x\frac{1}{x+1}
\end{aligned}$$

We will use a well known fact from generating functions: $\sum_{x=0}^{\infty}\binom{x+y}{x}p^x = (1-p)^{-y+1}$. Integrating both sides of this from 0 to p gives us $\sum_{x=0}^{\infty}\binom{x+y}{x}\frac{p^{x+1}}{x+1} = \frac{(1-p)^{-y}-1}{y}$. which implies

$$\sum_{x=0}^{\infty} \binom{x+y}{x} \frac{p^x}{x+1} = \frac{(1-p)^{-y}-1}{py}$$

Pluggin this in and summing for all j gives us (excluding $j = 1$)

$$\begin{aligned}
&\sum_{j=2}^{n} 2(j-1)(1-p)^j \frac{(1-p)^{-j+1}-1}{p(j-1)} \\
=&\sum_{j=2}^{n} 2\frac{(1-p)-(1-p)^j}{p}
\end{aligned}$$

Adding the contribution of all $I(i, i)$ and using geometric sum, the final reduction makes this

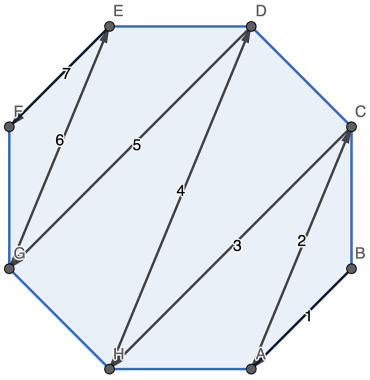
$$\begin{aligned}
&n + 2\left((n-1)\frac{1-p}{p} - \frac{\frac{(1-p)^{n+1}-(1-p)^2}{(1-p)-1}}{p}\right) \\
=&n + 2\left((n-1)\frac{1-p}{p} + \frac{(1-p)^{n+1}-(1-p)^2}{p^2}\right) \\
=&n + 2\left(\frac{(np-1)(1-p)+(1-p)^{n+1}}{p^2}\right)
\end{aligned}$$

This is the answer. You just need to compute modular exponentiation.

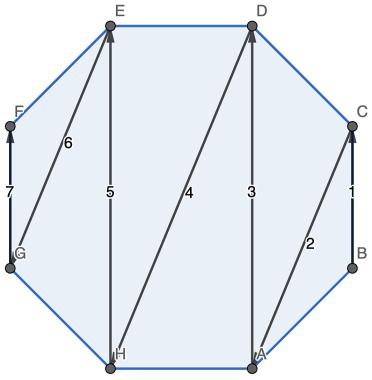
Problem C. Effective Rendering

Problem Setter: Md. Muhiminul Islam Osim
Tester: Hasinur Rahman, Arnob Sarker, Md Sabbir Rahman
Category: Geometry, Observation
Total Solved: 0
First to Solve: N/A

The first and foremost observation of finding a valid $GL\_TRIANGLE\_STRIP$ sequence is to get the point and then alternately take the points from the two sides of the points already taken. There is no valid $GL\_TRIANGLE\_STRIP$ sequence possible without taking this zigzag pattern. Now, we consider each of the boundary points of the polygon as the start of the $GL\_TRIANGLE\_STRIP$ sequence. We take the second point from the left or right of the first point. This choice creates two possible zigzag sequences (clockwise and anticlockwise) for each starting point (see the figure below). Then we try to minimize the maximum area of a triangle from each candidate sequence.



Clockwise zigzag starting from point B



Anti-clockwise zigzag starting from point B

To check if a candidate sequence is a valid sequence covering the whole polygon area without any overlaps among triangles, it is enough to check if the summation of the area of all triangles is equal to the area of the polygon itself. However, there are some other ways to validate a sequence (e.g. checking triangle sides crossing each other and examining if a side is outside the polygon).

Time complexity: from $O(T \cdot n^2)$ to $O(T \cdot n^3)$ depending on the implementation for the validation of a candidate sequence.

Problem D. Dysbinaria

Problem Setter: Rumman Mahmud
Tester: Raihat Zaman Neloy, Md Sabbir Rahman
Category: Data Structure, BBST, Bit-Manipulations
Total Solved: 0
First to Solve: N/A

The immediate solution that comes to mind is to maintain a Treap data structure for each bit, which allows handling all queries in $\mathcal{O}(q \times k \times \log n)$ time. For the rotate operation, we need to split the subarray and swap the Treap nodes. For flip, set, and clear operations, lazy propagation can be used. The memory complexity is $\mathcal{O}(n \times k)$. However, this approach is extremely slow—it is even slower than the brute-force solution, which has a complexity of $\mathcal{O}(n \times q)$. The slowness arises because querying across multiple Treaps is computationally expensive, involving multiple recursive calls along with frequent push-up and push-down checks.

Therefore, we need to consider a slightly different approach. Instead of maintaining k Treaps, we can use a single Treap where each node stores information for all k bits. This modification reduces some overhead.

To optimize further, we can use a bitset to track information such as lazy values for set, clear, flip operations, and counting number of ones. Additionally, push-up and push-down tasks can be performed using bitwise operations. This optimization reduces the complexity of each query from $\mathcal{O}(k \times \log n)$ to $\mathcal{O}(\log n)$, as operations on bitsets of size 31 are considered to have constant time complexity. Note that you can use unsigned intergers instead of bitset as well.

Problem E. Escape Plan

Problem Setter: Arnob Sarker
Tester: Md. Nafis Sadique, Rumman Mahmud, Kazi Md Irshad
Category: Bitmask DP
Total Solved: 8
First to Solve: DU_Primordius

There is no polynomial solution, as the order of visiting stop nodes matters. The non-optimal solution is O(n!) to check each permutation. How to optimize it? Bitmask DP!
There should be two states, A bitmask to keep track of stop nodes and the last node where you stopped. $(2^n \times n)$. DP stores the maximum amount of fuel that you can save by visiting the stop nodes in correct order.
Time Complexity: $O(2^n n^2)$, it can be optimized to $O(2^n n)$.
Memory Complexity: $O(2^n n)$

Problem F. Funny Fairy Tale

Problem Setter: Sharif Minhazul Islam
Tester: Tanzir Pial, Nafis Sadique
Category: FFT
Total Solved: 6
First to Solve: BRACU_Crows

Let's calculate the contribution of each integer in the array individually. Let $A_1, A_2, \ldots, A_{m-1}, A_m$ represent the unique integers in the array, and let C_i be the frequency of the integer A_i.

Each of these integers can be part of a subsequence which has the integer at least once in $2^{C_i} - 1$ ways. In all such cases, the integer contributes to the subsequence's distinct count exactly 1.

However, since we can flip the sign of any integer, all previously considered subsequences can also appear with negative values. For example, the sequence $[1, 1]$ can be transformed into $[-1, 1]$, $[1, -1]$ or $[-1, -1]$, which has 2, 2, and 1 distinct integers, respectively. As a result, the total number of subsequences with 1 distinct integers becomes $2 \times (2^{C_i} - 1)$.

Next, if we analyze the count of subsequences where the integer contributes exactly 2 as distinct integers, it is $3^{C_i} - 2^{C_i+1} + 1$. To generalize this, let's define a polynomial P_i for i distinct integers:

$$P_i = 1 + (2^{C_i+1} - 2)x + (3^{C_i} - 2^{C_i+1} + 1)x^2$$

Here, P_i captures the count of subsequences where the integer A_i contributes with 0, 1, and 2 distinct integers. The final polynomial P is the product of all these individual polynomials:

$$P = P_1 \cdot P_2 \cdot \ldots \cdot P_{m-1} \cdot P_m$$

The coefficient of x^K in P represents the number of subsequences that contain exactly K distinct integers. By pre-computing these polynomials and their products, we can efficiently answer queries about subsequences with a specific number of distinct integers. P has a degree at most n. This can be calculated by divide and conquer and FFT. Overall complexity $O(n \log n)$

Problem G. The Scientist

Problem Setter: Seemanta Bhattacharjee
Tester: Rumman Mahmud
Category: Graph, Shortest Path
Total Solved: 20
First to Solve: SUST Fanatics

For the i^{th} query, we want to check if it is possible to reach state D_i from at least one of the states X_j of the snapshots while satisfying the time constraints.
We calculate all pair shortest path for the graph. For the j^{th} snapshot, if $[L_i, R_i]$ intersects with $[j * T + d[D_i][X_j], (j+1) * T - d[D_i][X_{(j+1)}]]$ (we need to check if this range is valid), for some j between 0 and $S-1$, then it's possible to be at state D_i between time $[L_i, R_i]$, where $d[D_i][X_j]$ is the shortest path distance from node D_i to node X_j.

Problem H. Ticket Dilemma

Problem Setter: Arnob Sarker
Tester: Raihat Zaman Neloy, Muhiminul Islam Osim
Category: Ad hoc
Total Solved: 148
First to Solve: BRACU_Crows

In odd rows, the adjacent seat pairs are 1st-2nd, 3rd-4th, and 4th-5th, while in even rows, the adjacent seat pairs are 1st-2nd, 2nd-3rd, and 4th-5th. Rest is implementation.

Problem I. Travel on the grid II

Problem Setter: Raihat Zaman Neloy
Tester: Nafis Sadique, Kazi Md Irshad, Hasinur Rahman
Category: Dynamic Programming
Total Solved: 29
First to Solve: BRACU_Crows

Since our movement is restricted to $(X+1, Y)$ or $(X, Y+1)$ from the current position (X, Y), we only need to account for the mines and their statuses in the 8 adjacent cells. Furthermore, due to the specific pattern of our movement, we can disregard cells that will no longer be visited. This reduces the adjacent cells we need to consider to just 5. Upon closer examination, we find that it suffices to focus only on the mine statuses at $(X+1, Y+1)$ and $(X+1, Y-1)$. These two values can then be incorporated into the state to compute the minimum number of mines required to reach (N, M). Subsequently, a second dynamic programming (DP) process can be executed to determine the number of ways to achieve this.

Problem J. Earthquake

Problem Setter: Aminul Haq
Tester: Tanzir Islam, Sharif Minhazul Islam
Category: Greedy, DSU, MST
Total Solved: 85
First to Solve: SUST Fanatics

This problem can be solved using a greedy approach based on Kruskal's Minimum Spanning Tree algorithm. First, we need to sort the bridges in descending order of strength, and a Disjoint Set Union (DSU) is used to manage the connected components. Then, iterate over the sorted bridges, and for each bridge, if the villages it connects belong to different components, they are merged using a union operation, reducing the total number of connected components. The strength of the bridge at each union determines the minimum earthquake power required to maintain the current number of components. This ensures that the K-th critical value corresponds to the weakest bridge that keeps at least K groups.

The time complexity is $O(N + M \log M)$ per test case, where N and M are the number of villages and bridges, respectively.

Problem K. The Beast

Problem Setter: Ashraful Islam
Tester: Raihat Zaman Neloy
Category: Binary Search
Total Solved: 92

First to Solve: BRACU_Crows

This problem turnes into solving the following equation: $n \times (1 + \lfloor \frac{1}{2} \rfloor + \lfloor \frac{1}{3} \rfloor + ... + \lfloor \frac{1}{m} \rfloor) = k$, find the minimum value of n where m and k is given. As, we have the floor operation here, we can't use harmonic series to solve the equation, and here comes binary search to help us. We can run binary search on the value of n and find lower bound as the answer.