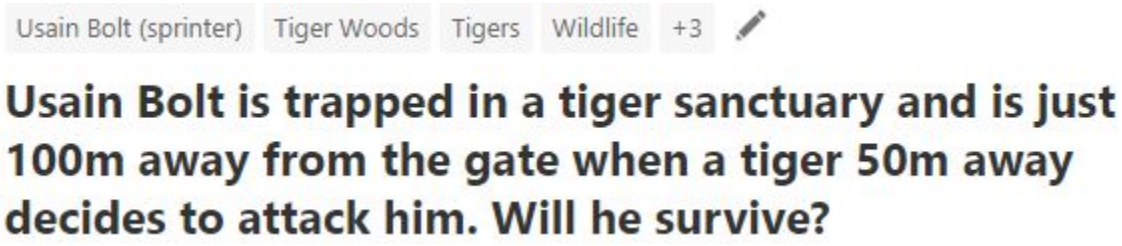
ICPC Amritapuri 2019

Problem discussion

USANBOLT - [BIPIN]

Problem Idea ;)



Total Distance covered by Tiger = ut + (1/2)at^2

Time taken by Bolt to cover the distance = distance_to_gate/(bolt's_speed)

So with the time taken by bolt, calculate the distance travelled by Tiger in the same time frame. If the distance is >= to Bolt's distance to gate, then tiger will the race.

Else Bolt

SDIFFSTR - [Balajiganapathi]

- Observation 1: Since both strings have unique character, the condition F(s, t) <= k means we can use atmost k chars from s in t
- Observation 2: To get lexicographically smallest, we should always try to add a if possible then b if possible and so on
- Solution idea: loop from a to z and add char to output if adding it will keep F(s, t) <= k. Else skip that char and go to next
- Edge case: we may run out of characters to add. If so, length of t will be less than length of s

EXPCAN - [Jatin]

- Let dp[i][j] be the value if the game was played on the subarray i...j
- There are 4 cases, each with probability ¼ : the first two removed candies (Alice, Bob) being one of {(i, i + 1), (j, j - 1), (i, j), (j, i)}
- dp[i][j] = (dp[i + 2][j] + A[i]) / 4 + (dp[i][j - 2] + A[j]) / 4 + (dp[i+1][j-1] + A[i])/ 4 + (dp[i + 1][j - 1] + A[j]) / 4
- Time complexity : O(n^2)

Bonus : Solve for n <= 10^5.

COLINT - [Jatin]

- It turns out that there is a coloring in which every point that is contained in at least two intervals is colored green. Clearly, there can’t exist a better answer.
- Greedy approach : Iterate on the intervals in increasing order of left end. Give every interval a color opposite to that of the interval before it with the maximum right end.
- Proof by induction : Say the above approach gave best possible answer for k - 1 intervals.
- Let the maximum r value of the first k - 1 intervals be R, and the interval being processed be [p, q]
- If p > R, this interval is separate from all previous intervals and doesn’t contribute to answer.

COLINT (contd.)

- If p <= R, then [p, min(q, R)] will be colored green, and the part (min(q, R), q] is contained in only one interval and its color doesn’t matter.
- Time complexity - O(n logn)

TRGRPH (Endagorion)

- If **u is an ancestor of v** in T, then **deg(u) >= deg(v)** in G(T).
  - deg(u) == deg(v) only when path between u and v does not have any other branches.
- Order the vertices in **decreasing** order of degrees in H
- Make vertex with maximum degree the root.
- For all vertices v in order of non-increasing deg(v)
  - Make parent(v) equal to the latest vertex u < v such that u is adjacent to v in H.
- In the end, explicitly check that G(T) = H.

MORTGAGE (GlebsHP)

If we can take a loan of **x** and pay it in **m** days, then we can pay back any loan of less than x in m days . Hence, we can **binary search** for the answer.

It is always optimal to take loan from a lower interest bank. Hence, we take loans from the banks in the order of increasing interests.

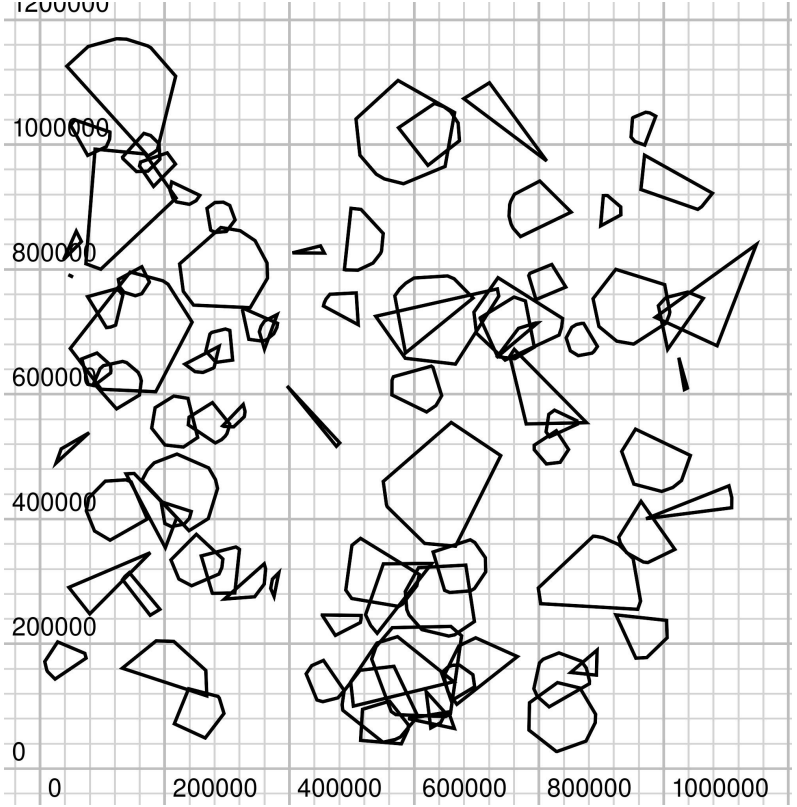
While repaying it is always better to payback first a bank with higher interest since next time we will also have to pay the interest on payback amount which will be higher for higher interest bank.

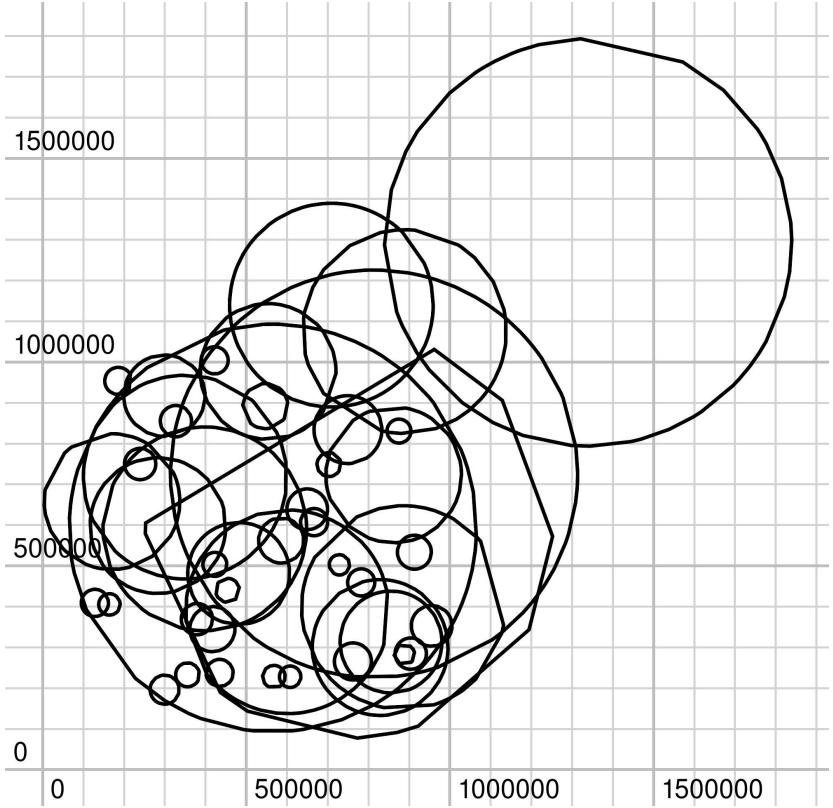
Cautions while implementing: power function and multiplications might lead to large values. We need to remember that if at any point outstanding loan exceeds m*t then we cannot pay it.

POLCON - [Vaibhav]

- The given set of polygons can be treated as a Partially Ordered Set (Poset).
  - a <= a for all a in S (the set of polygons) [Reflexivity]
  - if a <= b and b <= a, then a = b [Antisymmetry]
  - if a <= b and b <= c then a <= c [Transitivity]
  - Think of this as a Directed Acyclic Graph (D) with an edge i -> j iff i <= j.
- The problem reduces to: Find the longest **antichain** in the Poset.
  - *"An antichain in a partially ordered set is a set of elements **no two of which are comparable to each other**"*
- Dilworth's Theorem
  - *"In any finite partially ordered set, the largest antichain has the **same size as the smallest chain decomposition**"*
- Can be solved using Konig's Theorem
  - Find Maximum Bipartite matching (M) on the bipartite graph (G) with edges (u, v) such that the directed edge u -> v exists in D.
- Length of the longest antichain = |S| - |M|
- How to check if Polygon A is contained inside Polygon B?
  - Create a convex hull (H) with the points of both A and B.
  - Check that no point of B is part of H.

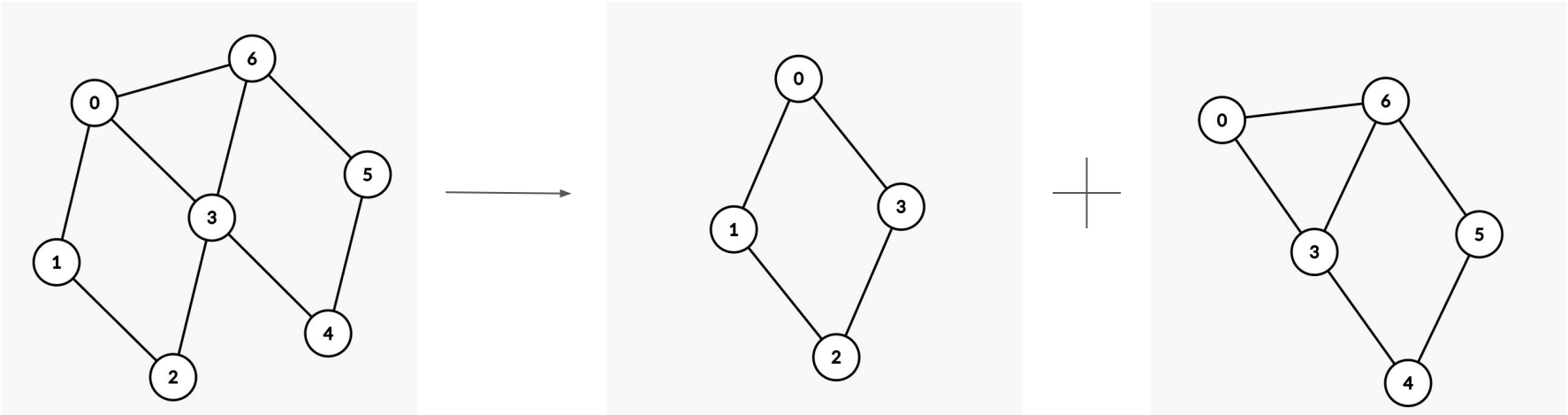
POLCON


1200000
1000000
800000
600000
400000
200000
0
0
200000
400000
600000
800000
1000000


1500000
1000000
500000
0
0
500000
1000000
1500000

COLPOLLY - [Pranjal]

- We can choose any chord in the polygon and divide the polygon into two different polygons, P1 and P2.
- Polygons P1 and P2 will share at most 2 nodes between them. Below, only nodes 0 and 3 are common.



COLPOLLY - [Contd]

- Recursively color both of the polygons individually, with minimum possible colors.
- Now merge both of the polygons, while merging them there can be conflict of color at the two shared nodes. To resolve this, we permute colors in one of the polygons, let’s say P2, to match the shared nodes’ color.
- In base case, we will have a simple polygon whose minimum coloring is <= 3 (= x), and as we are not creating any additional colors in merging the polygons, the minimum coloring of the original polygon is also <=3 (= x).

COLPOLLY - Greedy?

DSatur: At each step take the vertex which has maximum nodes already colored and give it an unique color

Difficult to beat this for planar graph

For Polynomial Tree graphs, it has been proved this gives optimal answer

REF:

https://en.wikipedia.org/wiki/Greedy_coloring#Adaptive

"The smallest hard-to-color graph for algorithm DSATUR" R. Janczewski ∗, M. Kubale, K. Manuszewski, K. Piwakowski1

MOBCNT

Weight of leaves under root is fixed and is sum of all a_i (let's say s).

Now depth 1 children of root will have s/2, depth 2 children of root will have s/4 and so on. So this will let us know number of leaves we need to have at each depth.

So we can start with root. And choose the children needed for leaves in a level (let's say there are n nodes available and we need r as leaves. Then ${}^{n}C_{r}$ ways to do so). And add 2 children to the nodes which was not chosen as leaves. And move to next level.

PERMDIST - [Mikhail]

- Assume u < v. If p[u] < p[v], the answer is 1.
- Else, let the shortest path be u, u_1, u_2, ..., u_k, v. If two consecutive directions (u_i - u_{i-1}) were the same, we could skip the middle vertex. So, in the optimal path, the index increases and decreases alternatively.
- Let L[u] = max index t < u, such that p[u] > p[t] and p[i] > p[t] for all i < t
- Let R[u] = max index > u with value > p[u] (The maximum index on the right that one can jump from index u.)
- It turns out that it is optimal to alternate between L and R while v can't be directly reached, i.e. there is an optimal path of the form u -> L[u] -> R[L[u]] -> L[R[L[u]]]... OR u -> R[u] -> L[R[u]] -> R[L[R[u]]....

PERMDIST (proof of correctness)

- Consider first u_i > v if it exists. p[ui] must be > p[v] as otherwise p[u_{i-1}] < p[u_i] < p[v], and we could have reached v directly from u_{i - 1}. As u_i > v and p[u_i]> p[v], we can go from u_i -> v.
- Say the optimal path looks like (u, r, r’, ….., v)
- Consider r < u and r != L[u]. If r > L[u], then r' also has an edge from L[u], as p[L[u]] < p[r] < p[r'] and L[u] < r < r'. If r < L[u], let r'' be the first index greater than r in the path (there must be one as we eventually reach v). Then r'' has an edge from L[u].

PERMDIST(proof of correctness contd.)

- Consider r > u and r != R[u]. Clearly, r must be < R[u] by definition. By induction, r' = L[r]. If p[r'] < p[R[u]], then we can replace r by L[u] in the path. Otherwise, p[r'] > p[R[u]] > p[u]. As p[r'] = p[L[r]] is a prefix minimum, r' must be < u. But then, as p[r'] > p[u], we can directly go to r' from u, getting a better path
- The above two points imply that from r is either L[u] or R[u].
- If we keep applying this logic on the further nodes of the path as well, we can convert any optimal path into a path of the alternating L and R type.

PERMDIST (Implementation)

- Its trivial to find all L and R values in O(n logn).
- Given u, consider two paths u -> L[u] -> R[L[u]] -> L[R[L[u]]]... and u -> R[u] -> L[R[u]] -> R[L[R[u]]....
- In these sequences we need to find the first index i, which is > v, or which has p[i] < p[v].
- This can be done using jump pointers and sparse table approach in O(n logn) precomputation time and O(logn) query time.

COMPLEXG - [Kevin]

- There are only 13 possible costs.
- The possible angles are multiples of 45 degrees, and the magnitudes are powers of sqrt(2).
- Expand the graph: For every node x, create 9 new nodes: (x, c) for each c in {0, 1, -1, i, -i, 1+i, 1-i, -1+i, -1-i}.
- Edges are now assumed to have cost 0.
- Whenever you seem forced to visit a node that should look like (x, 2a+2bi), just have it go to (x, a+bi) and replace the cost of the edge by 1.

COMPLEXG

- Thus, we have reduced it to a shortest path on a graph such that:
- (1) You add costs instead of multiplying them.
- (2) Each “weight” is either 0 or 1 (so it’s an almost unweighted graph).
- The weight of an edge represents the exponent we will raise 2 by.

COMPLEXG

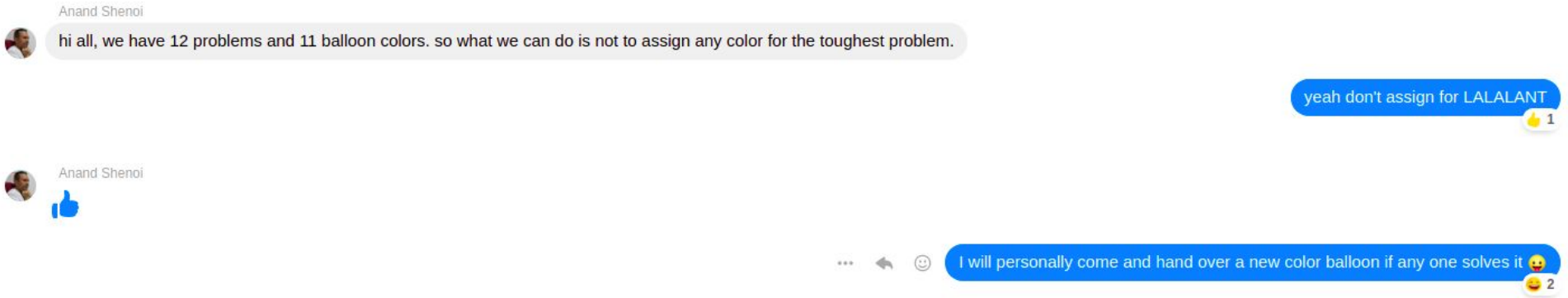
- Now, from (s, +1), we now look at whether we can reach (t, -1), (t, 0), and (t, 1).
- If none is reachable, then there is no real walk from s to t.
- If (t, -1) is reachable, then the answer must be negative.
- The answer is -infinity if there is a node that:
  - is reachable from (s, 1);
  - can reach (t, -1); and
  - belongs to a cycle with at least one edge with positive weight.
- For this, reduce strongly connected components, and detect which SCCs have at least one positive edge.
- Otherwise, the answer is negative and finite. It is actually just the longest path from (s, +1) to (t, -1). This can be computed with DP since we just assumed that there’s no point in going through cycles (since all of them have zero weight).

COMPLEXG

- If (t, -1) is not reachable, then the answer is nonnegative.
- If (t, 0) is reachable, then the answer is 0.
  - No need to do anything else!
- Otherwise, the answer must be positive (and finite). It is actually just the shortest path from (s, 1) to (t, 1), but since weights are 0 or 1, we can just use BFS (using a deque to handle 0-weight edges).

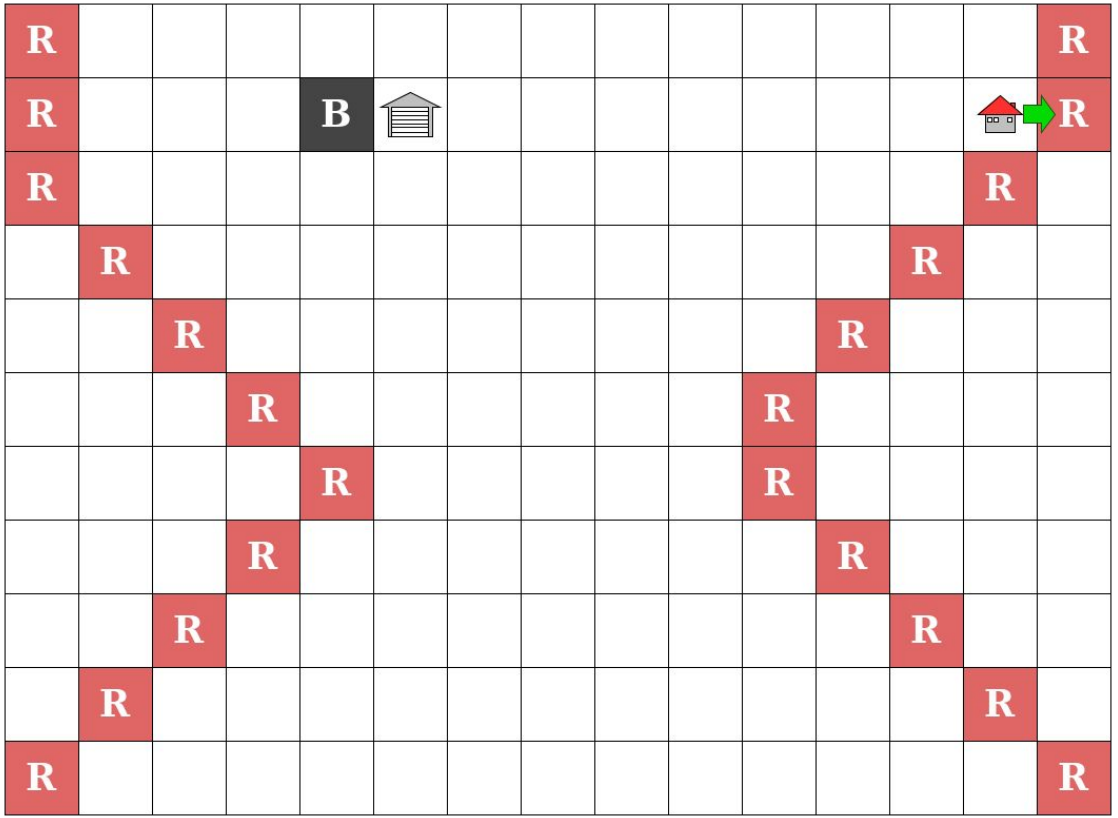
LALALANT - [Kevin]

- For “large enough” dimensions, ignore the special subsets and just attempt to visit all cells.
- It is always doable if min(r,c) >= 3 and max(r,c) >= 4.
- See the following slides for the general constructions.
- Handle the remaining cases separately. (There are only a few of them.)



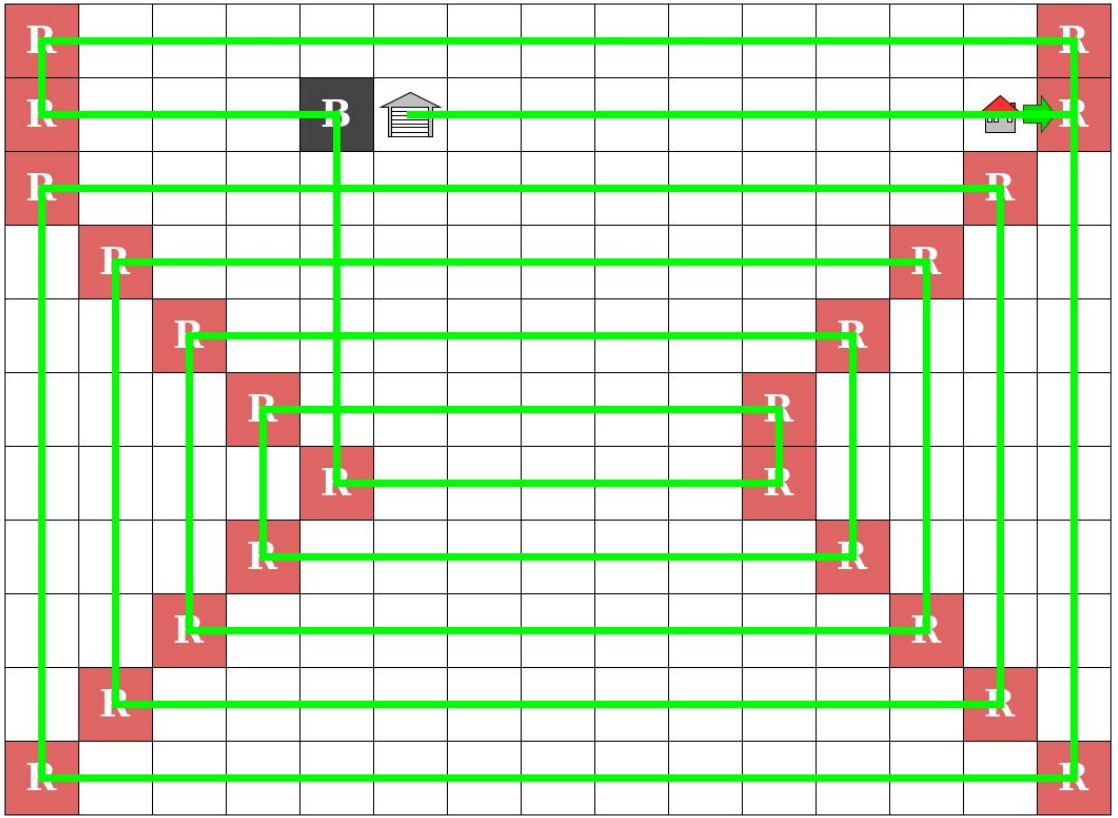
LALALANT

- Follow the path!



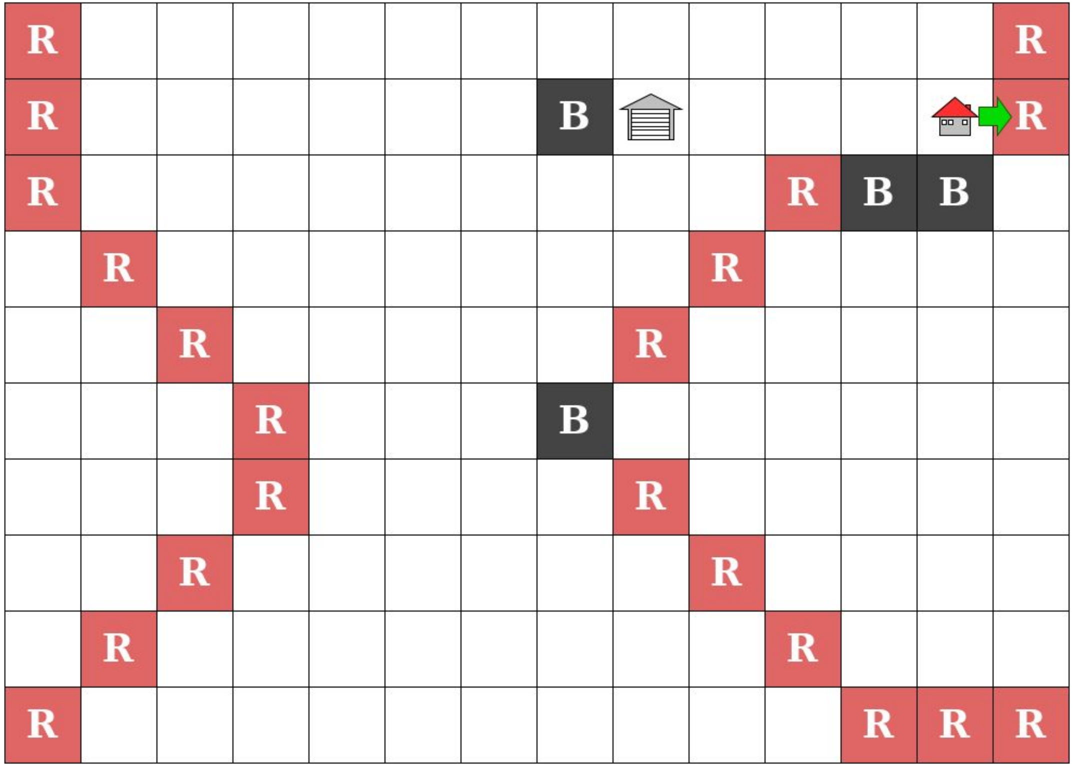
LALALANT

- Follow the path!



LALALANT

- Follow the path!



LALALANT

- Follow the path!