CERC 2018

Presentation of solutions

December 2, 2018

Silence of the Lamps

- Note that there are only 16850052 ($\approx$ 1.7e7) such triples for 10^6
- You can safely iterate over all solutions, so just use three cycles to find out number of occurences of every possible volume.
- Now do a prefix sum and you're good to go.
- This way you can precompute all results at once and then simply retrieve a result for every query in $O(1)$.

Clockwork ||ange

- Observe that the maximal answer could be $\log_2 b$ (even with only one bit on)
- You can do brute force, with break after $\log_2 b$ steps (best without trivial duplicites).

Clockwork ||ange

- Observe that the maximal answer could be $\log_2 b$ (even with only one bit on)
- You can do brute force, with break after $\log_2 b$ steps (best without trivial duplicites).

Complexity is "no worse" than $O(b^{\log_2 b})$.

Matrice

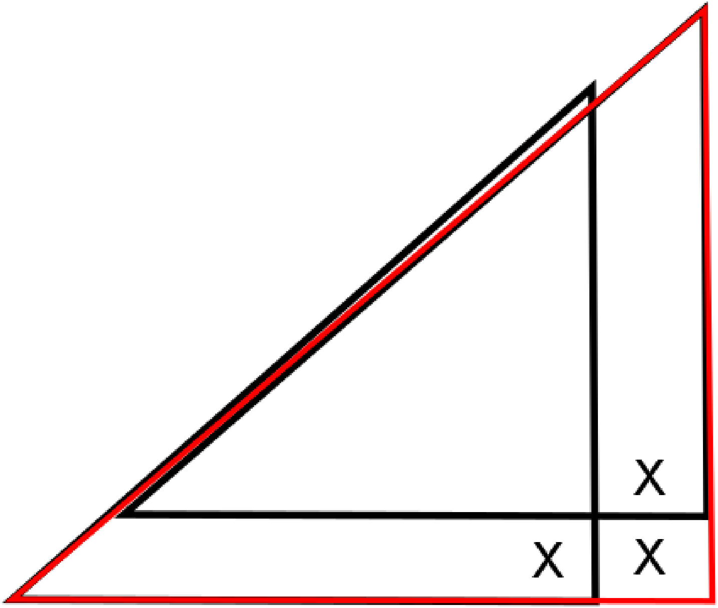
- The biggest size of one kind of triangle on coordinates **x**, **y** could be found as 1 more than biggest triangle on coordinates [x-1][y] and [x][y-1].
- Obviously only if the characters are equal.
- Make similar approach for the other 3 kinds or rotate the matrix.

Matrice

- The biggest size of one kind of triangle on coordinates **x**, **y** could be found as 1 more than biggest triangle on coordinates [x-1][y] and [x][y-1].
- Obviously only if the characters are equal.
- Make similar approach for the other 3 kinds or rotate the matrix.

Complexity $O(N^2)$

Matrice



Tree Gump

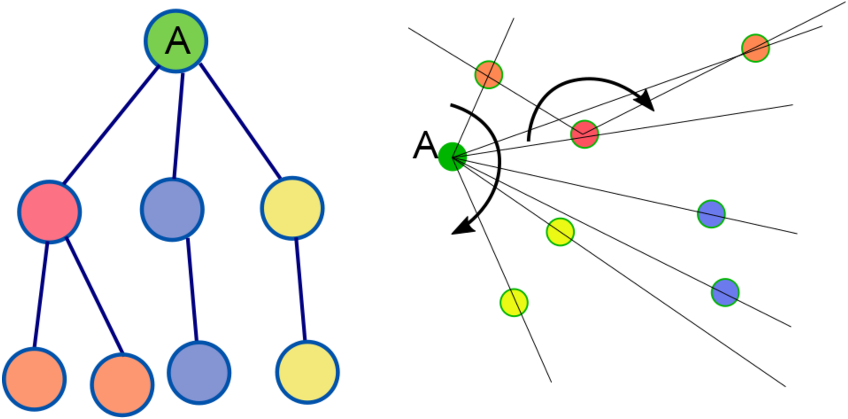
- Pick the lower leftmost point (or some other extreme).
- Recursively, sort the points by angle and divide them to children according to their sizes.
- Note that if this is done properly, the "root" of tree is always some extreme and the rest is in some kind of "fan" (the points are on one side of it).

Tree Gump

- Pick the lower leftmost point (or some other extreme).
- Recursively, sort the points by angle and divide them to children according to their sizes.
- Note that if this is done properly, the "root" of tree is always some extreme and the rest is in some kind of "fan" (the points are on one side of it).

Complexity $O(N^2 \log N)$

Tree Gump



Stones

Stones

- Firstly let us take look on how to solve this problem if $A == B$
- That would be an impartial game solvable with nimbers.
- Actually it is not hard to observe that the behaviour of such nimber sequence – it repeats in modulo A (and B obviously).
- But what if $A \neq B$? The game is impartial!
- Note that if the size of the biggest pile is less or equal than minimum of A and B, it is still an impartial game.

Stones

- Let us WLOG fix $A > B$ for this case ($A < B$ would work vice versa).
- An obvious observation is that player with A has an advantage.
- In fact he could use principle of tied hand, thinking $A == B$ (in this case we could use nimbers again)

Stones

- Let us WLOG fix $A > B$ for this case ($A < B$ would work vice versa).
- An obvious observation is that player with A has an advantage.
- In fact he could use principle of tied hand, thinking $A == B$ (in this case we could use nimbers again)
- Lets say player with A is playing:
- Imagine the sum of games is not zero. In this case he could simply follow the rules of nim with $A == B$ and win.
- If the sum of games is zero, he can play a 0-move (not changing the xor) and continue playing as if $A == B$ after that.

Stones

- But what if B is plaing first?
- Note that if there are at least two piles bigger than B, then fate of the second player is sealed since he can't escape player A playing the 0-move.
- But as soon there is just one such pile, he does have a chance – but he certainly has to make a move to that particular pile.
- Check whether B can play to that pile to get 0 xor and such that the pile has at most B elements.

Stones

- But what if B is plaing first?
- Note that if there are at least two piles bigger than B, then fate of the second player is sealed since he can't escape player A playing the 0-move.
- But as soon there is just one such pile, he does have a chance – but he certainly has to make a move to that particular pile.
- Check whether B can play to that pile to get 0 xor and such that the pile has at most B elements.

Complexity $O(N)$

The ABCD Murderer

- There are multiple solutions for this problem.
- Each of them finds the longest string from dictionary which starts on given index and the rest is common.

The ABCD Murderer

- There are multiple solutions for this problem.
- Each of them finds the longest string from dictionary which starts on given index and the rest is common.
- Once you figure out how to find the longest string, what remains is to greedily iterate and find the length of the longest string which can be added to already matched part.

The ABCD Murderer - Jury solution

- Create Aho–Corasick automaton using all strings from dictionary.
- We don't need the exact indices of matches, we care only about the length of longest word which can be matched and ends at given index of the ransom text. That can be precomputed in linear time.
- Overall complexity is linear to sum of lengths of the input strings.

The ABCD Murderer - Example

- Ransom note: abecedadabra
- Dictionary: abec, ab, ceda, dad, ra

- |abecedadabra
- abec|edadabra
- abeceda|dabra
- abecedad|abra
- abecedadab|ra
- abecedadabra|

The ABCD Murderer - Alternative approach

- Create suffix array from all concatenated strings from dictionary.
- Make LCP array.
- Iterate over it, keeping minimum from last dictionary string to find the number.

Complexity depends on SA – best $O(N)$, but $O(N \log^2 N)$ is also sufficient

The ABCD Murderer - Alternative approach 2

- Divide strings on input by those which are longer then $\sqrt{N}$ and the rest.
- For those which are longer use KMP.
- The shorter shall be put into a trie.
- Afterward test each index of text in trie.

Complexity of this approach is $O(N\sqrt{N})$, also ok

The ABCD Murderer - Alternative approach 3

- For each distinct string length, put all hashes of all patterns of the length into some fast search data structure (hash map / map for example) and use rolling hash over the text.

The ABCD Murderer - Alternative approach 3

- For each distinct string length, put all hashes of all patterns of the length into some fast search data structure (hash map / map for example) and use rolling hash over the text.
- Observe that there is no more than $\sqrt{N}$ distinct pattern lengths.

The ABCD Murderer - Alternative approach 3

- For each distinct string length, put all hashes of all patterns of the length into some fast search data structure (hash map / map for example) and use rolling hash over the text.
- Observe that there is no more than $\sqrt{N}$ distinct pattern lengths.
- We also recommend double hashing!

Complexity of this approach is $O(N\sqrt{N})$

Shooter Island

- Let us have $N \cdot M$ grid and maintain DSU on it.
- This could us answer the queries pretty fast!
- But how to maintain this?

Shooter Island

- Let us have $N \cdot M$ grid and maintain DSU on it.
- This could us answer the queries pretty fast!
- But how to maintain this?
- If we would process the whole range we shooted it would time out.
- But obviously we don't want to process again nodes which were already processed.

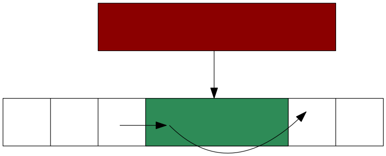
Shooter Island

- Let us have $N \cdot M$ grid and maintain DSU on it.
- This could us answer the queries pretty fast!
- But how to maintain this?
- If we would process the whole range we shooted it would time out.
- But obviously we don't want to process again nodes which were already processed.
- To do this we can maintain another DSU for each line which maintains the last element in component.

Shooter Island

- Let us have $N \cdot M$ grid and maintain DSU on it.
- This could us answer the queries pretty fast!
- But how to maintain this?
- If we would process the whole range we shooted it would time out.
- But obviously we don't want to process again nodes which were already processed.
- To do this we can maintain another DSU for each line which maintains the last element in component.

Complexity $O(N \cdot M \cdot \alpha(N) + Q \cdot N \cdot \alpha(N))$



Dog

$$s(t) = s + vt + at^2/2$$

Compute the time it takes the dog to reach the top of his jump (set $s = 0, v = 0$).

$$s(t) = at^2/2$$

$$t_{topdog} = \frac{2 \cdot s(t)}{a}$$

Dog - by binary search

- Observe that the dog will always jump with maximal velocity.
- Binary search for the earliest time when the dog can get the frisbee – we have exact position of the frisbee:

$$X = V_f \cdot \min(T - T_f, F(H_f))$$

$$Y = H_f - (T - T_f)^2/2$$

- X coordinate ok iff $X/V_d \leq T - T_d$.
- Y coordinate ok iff $Y/V_d \leq T - T_d$.
- Check if dog can get to the frisbee in time – check X and Y coordinates separately.

Complexity $O(1)$

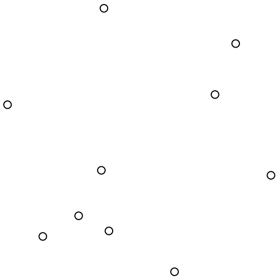
Dog - by exact formula

- There are 3 main cases:
  - frisbee will fall to the ground and dog will fetch it,
  - dog will have to wait until he can jump for frisbee,
  - dog can jump for frisbee immediately.
- Compute minimal time to cover X and Y distance separately and take minimum.

Complexity $O(\log U)$ where $U = 10^{15}$.

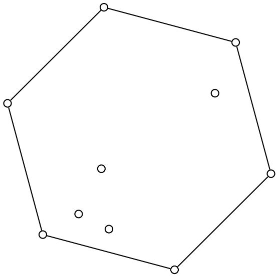
The Incredible Hull

- Observe the behavior of the process:



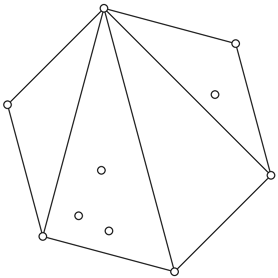
The Incredible Hull

- Observe the behavior of the process:
- Firstly, we consider points (i.e machines) forming convex hull



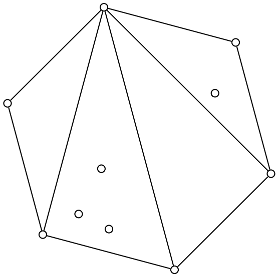
The Incredible Hull

- Observe the behavior of the process:
- Firstly, we consider points (i.e machines) forming convex hull
- Then, the hull will be divided into triangles, where all edges go from the most profitable machine to all others.



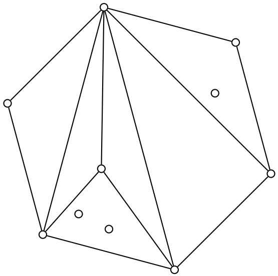
The Incredible Hull

- Observe the behavior of the process:
- Firstly, we consider points (i.e machines) forming convex hull
- Then, the hull will be divided into triangles, where all edges go from the most profitable machine to all others.
- All the nonempty triangles are then recursively split into 3 triangles.



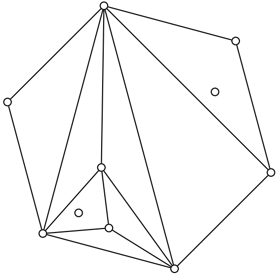
The Incredible Hull

- Observe the behavior of the process:
- Firstly, we consider points (i.e machines) forming convex hull
- Then, the hull will be divided into triangles, where all edges go from the most profitable machine to all others.
- All the nonempty triangles are then recursively split into 3 triangles.



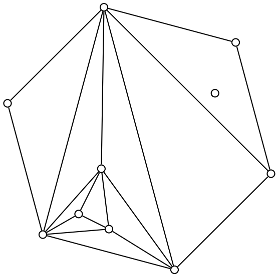
The Incredible Hull

- Observe the behavior of the process:
- Firstly, we consider points (i.e machines) forming convex hull
- Then, the hull will be divided into triangles, where all edges go from the most profitable machine to all others.
- All the nonempty triangles are then recursively split into 3 triangles.



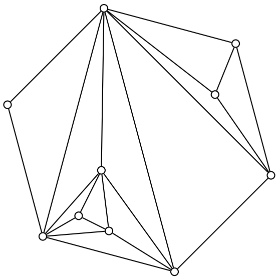
The Incredible Hull

- Observe the behavior of the process:
- Firstly, we consider points (i.e machines) forming convex hull
- Then, the hull will be divided into triangles, where all edges go from the most profitable machine to all others.
- All the nonempty triangles are then recursively split into 3 triangles.

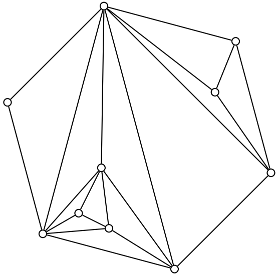


The Incredible Hull

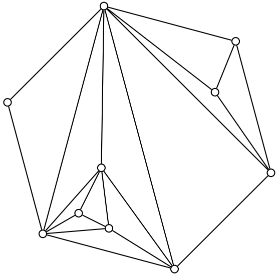
- Observe the behavior of the process:
- Firstly, we consider points (i.e machines) forming convex hull
- Then, the hull will be divided into triangles, where all edges go from the most profitable machine to all others.
- All the nonempty triangles are then recursively split into 3 triangles.



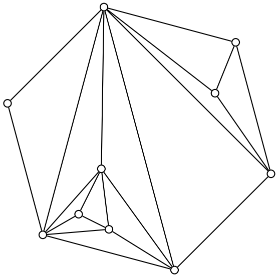
- As we can see, the size of largest clique will be 4 unless all points are on the hull (then the answer is 3).



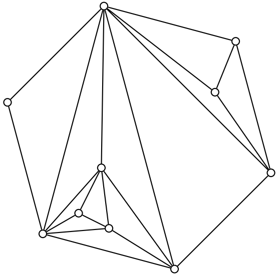
- As we can see, the size of largest clique will be 4 unless all points are on the hull (then the answer is 3).
- Also note that number of such cliques is equal to number of points inside the hull - again unless the answer was 3.



- As we can see, the size of largest clique will be 4 unless all points are on the hull (then the answer is 3).
- Also note that number of such cliques is equal to number of points inside the hull - again unless the answer was 3.
- The only "problem" might be finding the number of involved points.
- This can be done by sorting points by angle and searching for empty triangles.

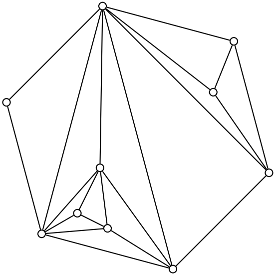


- As we can see, the size of largest clique will be 4 unless all points are on the hull (then the answer is 3).
- Also note that number of such cliques is equal to number of points inside the hull - again unless the answer was 3.
- The only "problem" might be finding the number of involved points.
- This can be done by sorting points by angle and searching for empty triangles.
- It turns out the only thing we need to simulate is the creation of the the hull.



- As we can see, the size of largest clique will be 4 unless all points are on the hull (then the answer is 3).
- Also note that number of such cliques is equal to number of points inside the hull - again unless the answer was 3.
- The only "problem" might be finding the number of involved points.
- This can be done by sorting points by angle and searching for empty triangles.
- It turns out the only thing we need to simulate is the creation of the the hull.

Complexity $O(N \log N)$



The Bridge on River Kawaii

- As first step, let us input all edges and proceed offline.
- As there are just 10 danger levels, we can for each danger level try only those edges whose danger level is lesser/equal.

The Bridge on River Kawaii

- As first step, let us input all edges and proceed offline.
- As there are just 10 danger levels, we can for each danger level try only those edges whose danger level is lesser/equal.
- Lets divide the queries into $\sqrt{N}$ blocks.
- Also for each edge, find the time it started to exist and ceased to exist (or possibly infinite).

The Bridge on River Kawaii

- As first step, let us input all edges and proceed offline.
- As there are just 10 danger levels, we can for each danger level try only those edges whose danger level is lesser/equal.
- Lets divide the queries into $\sqrt{N}$ blocks.
- Also for each edge, find the time it started to exist and ceased to exist (or possibly infinite).
- For each block, do DSU, while connecting edges which started before the time of block and ending afer it.
- Now we make a graph in each block which consists only of edges which begin or end in the block. As you can see there are at most $\sqrt{N}$ such edges.

The Bridge on River Kawaii

- As first step, let us input all edges and proceed offline.
- As there are just 10 danger levels, we can for each danger level try only those edges whose danger level is lesser/equal.
- Lets divide the queries into $\sqrt{N}$ blocks.
- Also for each edge, find the time it started to exist and ceased to exist (or possibly infinite).
- For each block, do DSU, while connecting edges which started before the time of block and ending afer it.
- Now we make a graph in each block which consists only of edges which begin or end in the block. As you can see there are at most $\sqrt{N}$ such edges.
- Now for each query, check, whether you can reach from start to a node, whose component is equal to component of destination

The Bridge on River Kawaii

- As first step, let us input all edges and proceed offline.
- As there are just 10 danger levels, we can for each danger level try only those edges whose danger level is lesser/equal.
- Lets divide the queries into $\sqrt{N}$ blocks.
- Also for each edge, find the time it started to exist and ceased to exist (or possibly infinite).
- For each block, do DSU, while connecting edges which started before the time of block and ending afer it.
- Now we make a graph in each block which consists only of edges which begin or end in the block. As you can see there are at most $\sqrt{N}$ such edges.
- Now for each query, check, whether you can reach from start to a node, whose component is equal to component of destination

Total coplexity is $O(V \cdot N\sqrt{N})$.

The Bridge on River Kawaii - Alternative solution

- The start is same as of the previous solutions.
- As first step, let us input all edges and proceed offline.
- As there are just 10 danger levels, we can for each danger level try only those edges whose danger level is lesser/equal.
- For each edge, find the time it started to exist and ceased to exist (or possibly infinite).

The Bridge on River Kawaii - Alternative solution

- The start is same as of the previous solutions.
- As first step, let us input all edges and proceed offline.
- As there are just 10 danger levels, we can for each danger level try only those edges whose danger level is lesser/equal.
- For each edge, find the time it started to exist and ceased to exist (or possibly infinite).
- Now - let us proceed Divide et Impera over queries, pointing into the middle and dividing queries into those with lesser time and those with bigger time.
- First step is to recursively solve the left half.
- In second step, tak all edges which are in the left half, and either use DSU or ignore them (depending on time of end).
- As you can observe, the queries in half from time t_b to t_e operate with at most $(t_e - t_b + 1) \cdot 2$ nodes. This means that if we would do the DSU "sparse" we would be able to copy it in each node with overhead linear with size of timelapse.
- If you have only one query and it is question, then simply check whether they are both in same component.

The Bridge on River Kawaii - Alternative solution

- Total coplexity is $O(V \cdot N \cdot log(N) \cdot \alpha^{-1})$.

The Bridge on River Kawaii - Alternative solution

- Total coplexity is $O(V \cdot N \cdot log(N) \cdot \alpha^{-1})$.
- Homework: Can we solve it online? ;-)

Lord of the Kings

- Use BFS to assemble a graph.

Lord of the Kings

- Use BFS to assemble a graph.
- Finding minimal number of tiles to host a helipad is equal to finding a Steiner tree on such graph with cities and palace as terminals.
- This can be solved by parametrized DP.

Lord of the Kings

- We compute Steiner trees for all possible roots and all possible subsets of terminals.

Lord of the Kings

- We compute Steiner trees for all possible roots and all possible subsets of terminals.
- Any such tree can be constructed either by joining two subtrees with the same root but different subsets of the subset forming a partition of the subset.
- OR by extending the rooted tree by another edge, creating a new root.

Lord of the Kings

- We compute Steiner trees for all possible roots and all possible subsets of terminals.
- Any such tree can be constructed either by joining two subtrees with the same root but different subsets of the subset forming a partition of the subset.
- OR by extending the rooted tree by another edge, creating a new root.
  - An extension costs the length of the shortest path between the old and the new root in the graph.
  - We thus also need to precompute APSP – Floyd-Warshall algorithm is sufficient.

Lord of the Kings

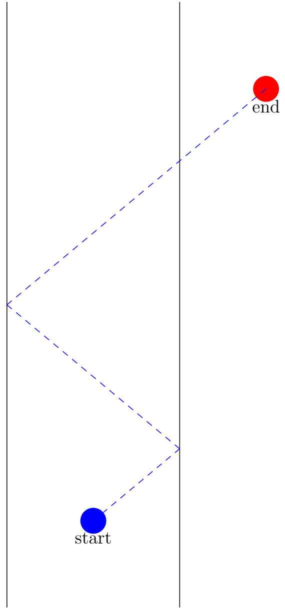
- We compute Steiner trees for all possible roots and all possible subsets of terminals.
- Any such tree can be constructed either by joining two subtrees with the same root but different subsets of the subset forming a partition of the subset.
- OR by extending the rooted tree by another edge, creating a new root.
  - An extension costs the length of the shortest path between the old and the new root in the graph.
  - We thus also need to precompute APSP – Floyd-Warshall algorithm is sufficient.
- For each subset, each root and each subset of subset, we try the join (and similarly with extension).

Lord of the Kings

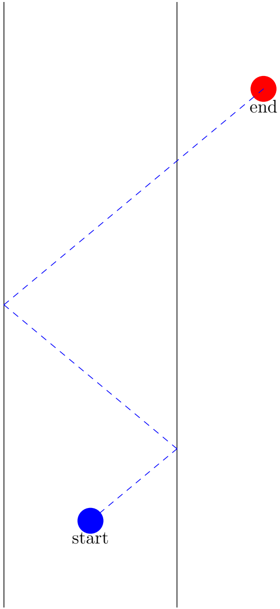
- We compute Steiner trees for all possible roots and all possible subsets of terminals.
- Any such tree can be constructed either by joining two subtrees with the same root but different subsets of the subset forming a partition of the subset.
- OR by extending the rooted tree by another edge, creating a new root.
  - An extension costs the length of the shortest path between the old and the new root in the graph.
  - We thus also need to precompute APSP – Floyd-Warshall algorithm is sufficient.
- For each subset, each root and each subset of subset, we try the join (and similarly with extension).

This leads to complexity of $O(N \cdot 3^T + N^2 \cdot 2^T + N^3)$

Mirrority report

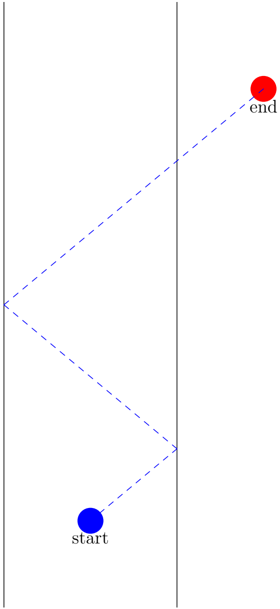


Mirrority report



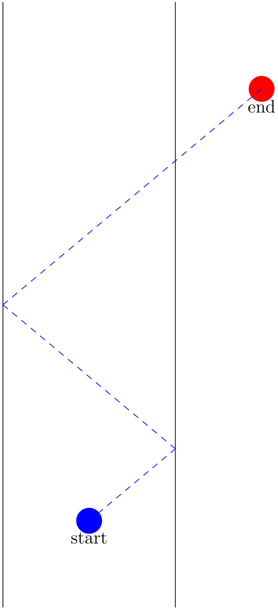
- Since with each reflection mirror dissapear there can be only one solution for each subset permutation.

Mirrority report



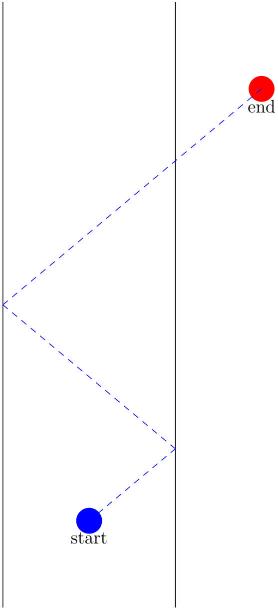
- Since with each reflection mirror dissapear there can be only one solution for each subset permutation.
- $N \leq 8$, we can try every subset and permutation.

Mirrority report



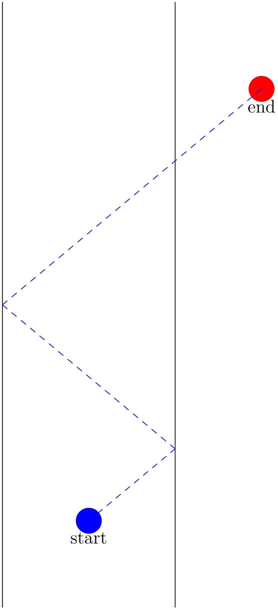
- Since with each reflection mirror dissapear there can be only one solution for each subset permutation.
- $N \leq 8$, we can try every subset and permutation.
- Mirror the plane and its elements for each mirror reflection to model the *reflected* world.

Mirrority report



- Since with each reflection mirror dissapear there can be only one solution for each subset permutation.
- $N \leq 8$, we can try every subset and permutation.
- Mirror the plane and its elements for each mirror reflection to model the *reflected* world.
- If the particle would pass too close to a corner ignore it.

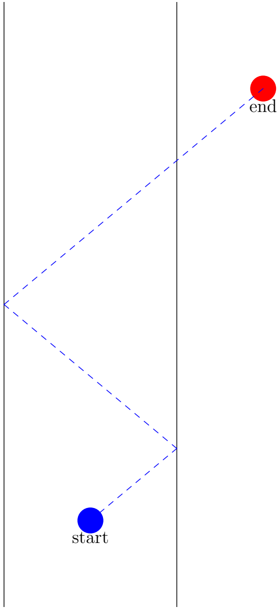
Mirrority report



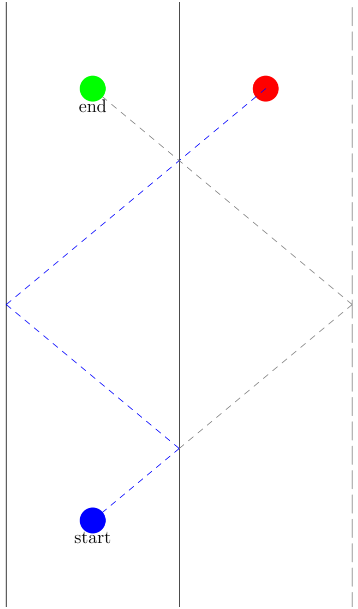
- Since with each reflection mirror dissapear there can be only one solution for each subset permutation.
- $N \leq 8$, we can try every subset and permutation.
- Mirror the plane and its elements for each mirror reflection to model the *reflected* world.
- If the particle would pass too close to a corner ignore it.

Complexity $O(N^2 \cdot N!)$

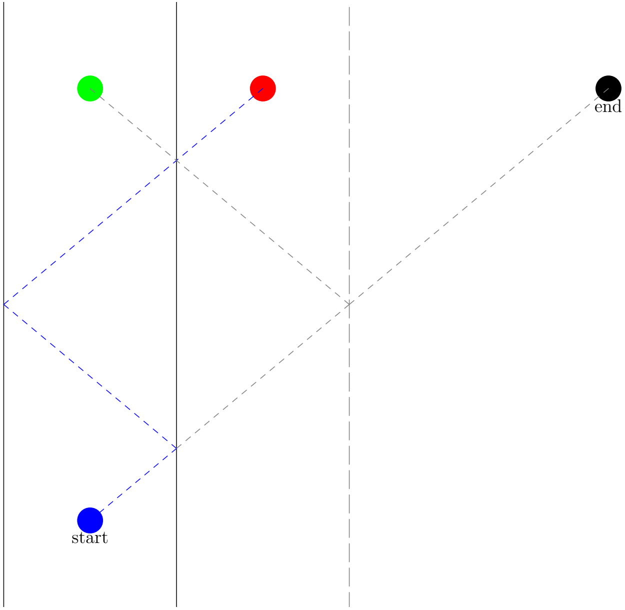
Mirrority report



Mirrority report

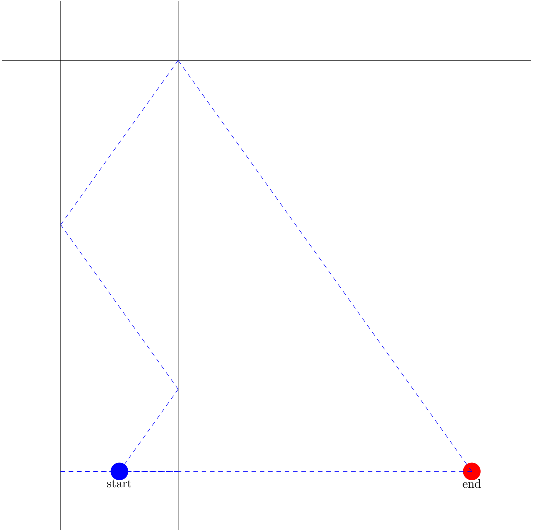


Mirrority report



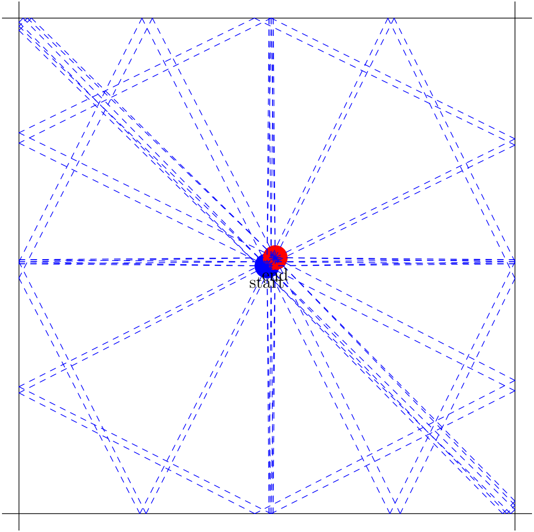
Mirrority report

Special cases – corner which is already gone

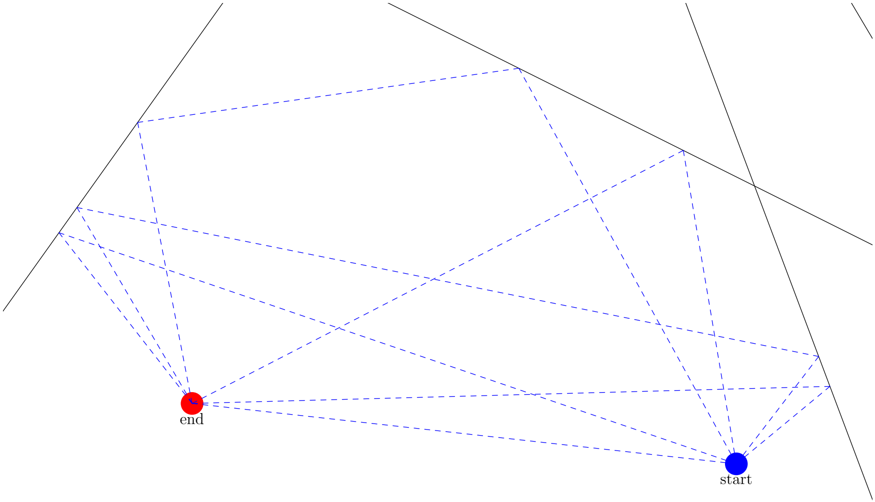


Mirrority report

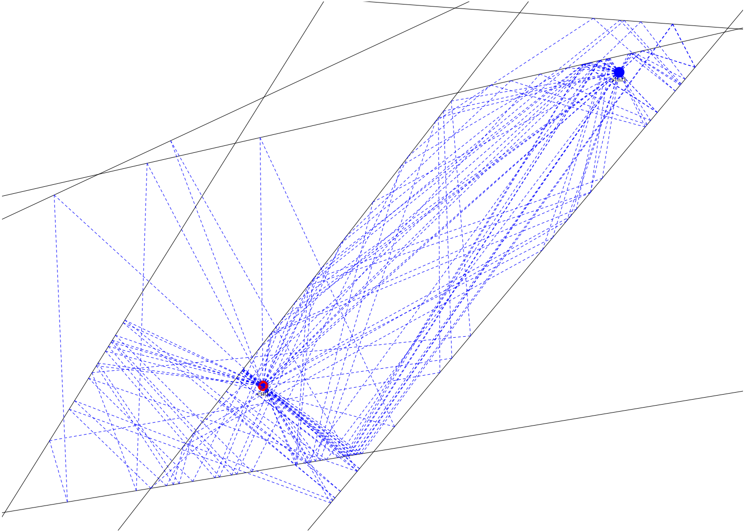
Special cases – ignore corners correctly



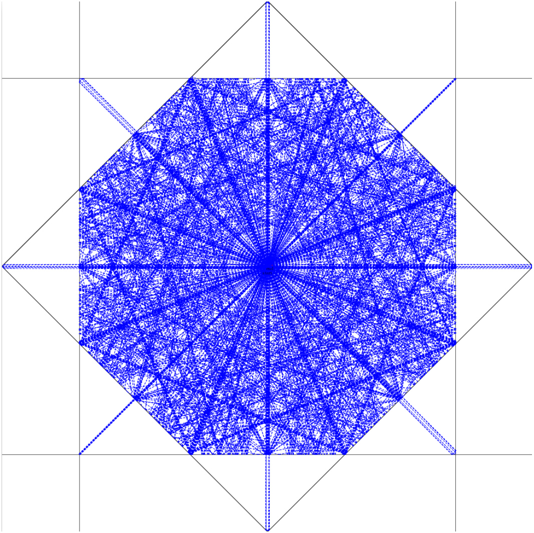
Mirrority report – some cases for amusement



Mirrority report – some cases for amusement



Mirrority report – some cases for amusement



Thank you for your attention!