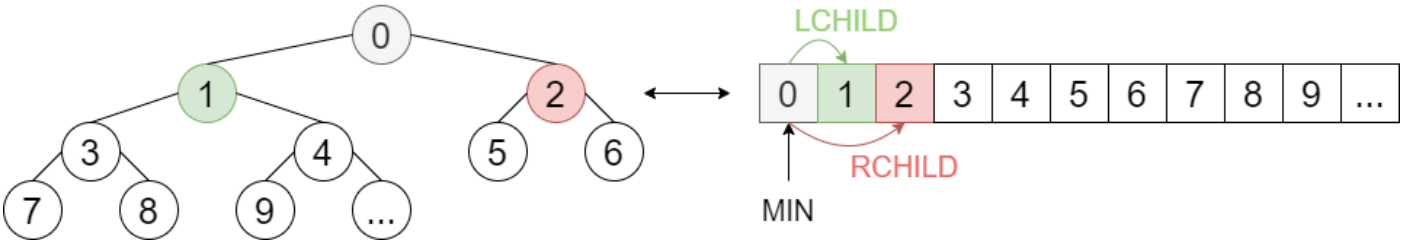
Hrytsyna Anastasiia-Solomiia
ID 105323

At the beginning, out of the input map I create matrix(table) of NODEs. In each NODE I remember its coordinates in map (x;y) and its edge (1 – C, D, P; 2 – H). Also I initialize a structure TOLL, where I count the number of the princesses and remember the pointers to all of my destinations (dragon, princess(es)).

After that I launch main part of the program (dijkstra's algorithm) and find way to the dragon. If this way exists and my Popolvar has enough time to kill the dragon – I look for all princesses (and all possible variants of ways between them. Then compare them (with function compare ☺ and choose the smallest/quickest one). At the end I unite way to dragon with way to all princesses, transform the output to the array of a necessary type and return it to tha mein function (where I'll print this array).

MIN HEAP FUNCTION

At first, when I initialize min_heap, I use structure HEAP, where I put the max number of the possibly inserted nodes (n*m) and create the array of nodes (actually min_heap table, to which I'll perform all future insertions). To the table I always insert the whole NODEs – add it to the end of the array and then hepify_bottom_top (swap if it's needed), so that parent remains smaller than its children. If I want to pop_min – I replace the 'root of the tree' ([0] position) with the last NODE in the table and then perform hepify_top_bottom (swap with the smaller child – if it's necessary). I didn't implement decrease_key function in the min heap, as in our case difference between edges isn't greater than 1, so the smallest possible key will be added at the first try to the heap, so we don't have to rewrite it.



DIJKSTRA'S ALGORITHM

My dijkstra's algorithm is pretty simple. At first I initialize all weights & pointers to the previous NODEs. Then I create min heap table and add starting NODE there. Next step is to get the min out of the heap (its root) and add all of its neighbours to the heap table (also mark its new weight and pointer to the previous NODE – the popped one). If our popped NODE is the finish one or heap table is already empty – we stop the loop, if not – we repeat popping&adding again and again… After all, if we reached the finish NODE we recreate our way back and return array (WAY) of coordinates like this:

0	1	2	3	4	5	…
weight	size	y1	x1	y2	x2	…

Where *weight* – time for reaching from the start NODE to the finish one; *size* – quantity of the elements in the array (all x&y + 2 (weight and size)).

Hrytsyna Anastasiia-Solomiia
ID 105323

LOOKING FOR THE FASTEST WAY

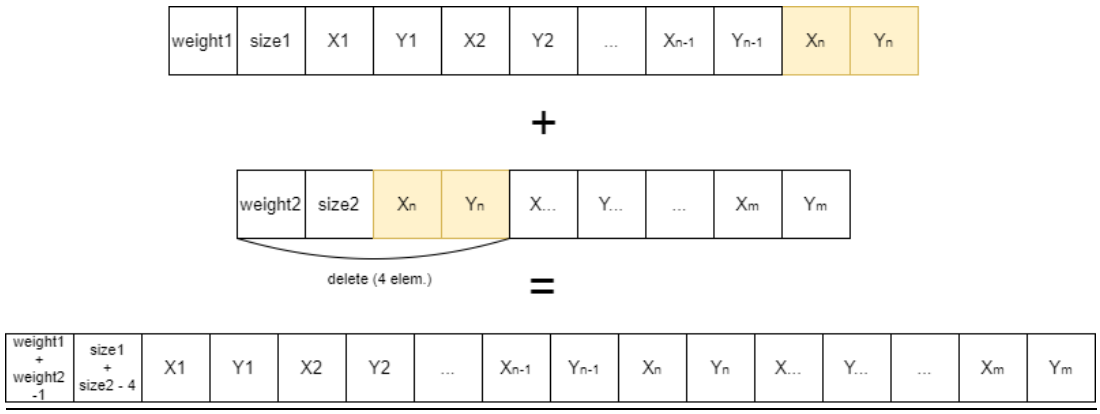
To compare all ways between princesses and dragon I call function “compare” ☺. Then I create one table (size of which depends on the number of princesses – at the example it is table for 4 princesses) with all information about distance between P. On the diagonal I put ways from the dragon to each P, another part of the table I filled with ways between princesses (at the left corner all of the coordinates are twisted, but the weight remains the same).

Drak		(P1)	(P2)	(P3)	(P4)
		0	1	2	3
(P1)	0	way1	way12	way13	way14
(P2)	1	way21	way2	way23	way24
(P3)	2	way31	way32	way3	way34
(P4)	3	way41	way42	way43	way4

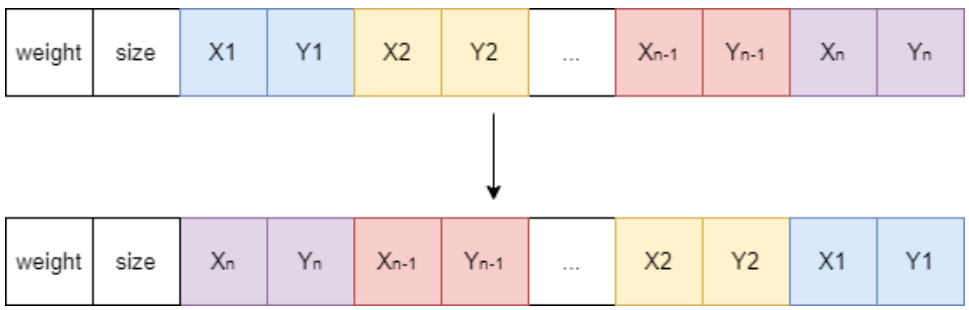
Then I use algorithm “permutation”, that produce all possible variations of the princesses order (for example P1P2P3P4, P2P1P3P4, P1P2P4P3…). In all cases I am finding the weight of the whole way (with this order) and compare it to the previous one (looking for a min). If it is smaller – I have found the new min, so I have to remember it and unite ways.

ADDITIONAL FUNCTIONS

- Unite arrays



- Twist coordinates



Hrytsyna Anastasiia-Solomiia
ID 105323

Time Complexity:

- **Dijkstra's algorithm:**

The main part: O((E+V)*LogV) = O((4+V)*LogV) = O((4+(n*m))*Log(n*m)) = O(n*m*Log(n*m))
Initialization: O(n*m)
Create_way: O(n*m)
Filling_information: O(n*m)

TOGETHER: O(n*m*Log(n*m)) + O(n*m) + O(n*m) + O(n*m) = O(n*m*Log(n*m)) + 3 * O(n*m) = O(n*m*Log(n*m)) = **O(x*Log(x)),** where x=n*m (number of the NODEs)

- **Zachran_princezne:**

Create_map: O(n*m)
Compare: (P*P)/2 * twist_coordinates = ((P^2)/2) * O(n*m) = O((P^2)*n*m)
Permutations * unite_arrays: O(P! * P * (n*m+2))
Dijkstra's algorithm 1+P+(P*P)/2 times: O((P^2) * (n*m)*Log(n*m))
Unite_arrays: O(n*m+2)*P = O(P*n*m)
Destroy_matrix: O(n*m)
Unite_arrays(final): O(n*m+2)*P = O(P*n*m)

***FINAL*: 2* (O(n*m)+ O(P*n*m)) + O((P^2) * (n*m)*Log(n*m)) + O(P! * P * (n*m+2))+ O((P^2)*n*m)**
where n*m - number of the NODEs, P – number of the princesses

Space Complexity:

- **Dijkstra's algorithm:**

Min_heap: O(n*m)
Path: O(n*m)

TOGETHER: O(n*m) + O(n*m) = **O(n*m)**

- **Zachran_princezne:**

Create_matrix: O(n*m)
Destination: O(P+1)=O(P)
Permutation: O(P!)
Unite_arrays: O(n*m+2)*P = O(P*n*m)
Unite_arrays(final): O(n*m+2)*P = O(P*n*m)
Ways: O((P^2)*n*m)

FINAL: O(P) + O(P!) + O(n*m) + 2* O(P*n*m) + O((P^2)*n*m)
where n*m - number of the NODEs, P – number of the princesses

Hrytsyna Anastasiia-Solomiia
ID 105323

TESTS

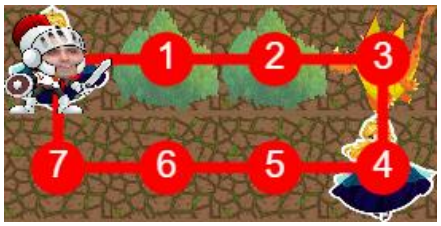
To check if my program runs correctly I have created 13 different maps (with diverse size and different number of princesses). At first I verified my code at small maps, than I enlarged their size. Also, I tried to generate maps that require to use all possible cases/functions of my algorithm (for example using permutations, twisting coordinates, uniting arrays…), so let's take a look on them:

- Test 1

  I check if my Popolvar will kill the dragon first and then come back for a princess.

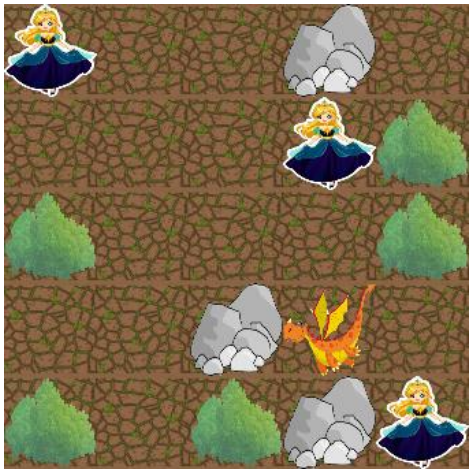
- Test 2

  I check if Popolvar can kill the dragon first and then protect more than 1 princess (2 for example).



- Test 3

  At this test I check the output in case my Popolvar hasn't enough time to kill the dragon (the dragon will wake up earlier and fly away).



Hrytsyna Anastasiia-Solomiia
ID 105323

- Test 4
  Here I check a bigger map and bigger number of the princesses. As there is more than two P – we need to use permutations, to find the shortest way. P1 is in the top right corner, but, as we can see, at first Popolvar will grab the P4 (6;4). That means that we have also used twist_coordinates function (to get to the P3, P2, P1 and P5 at last).

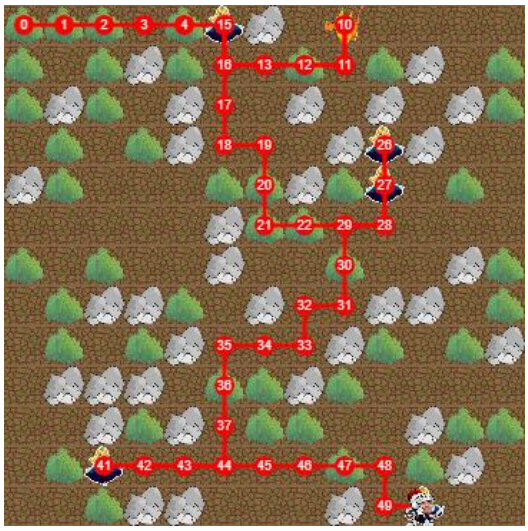
- Test 5
  This test runs for a map, where dragon is surrounded with rocks.

- Test 6
  At this case one of the princess is surrounded with rocks.

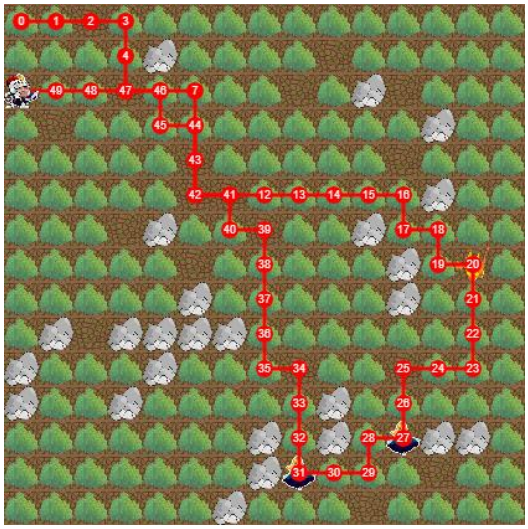
Hrytsyna Anastasiia-Solomiia
ID 105323

In all of the next cases I generate bigger and bigger maps, with different number of the princesses.

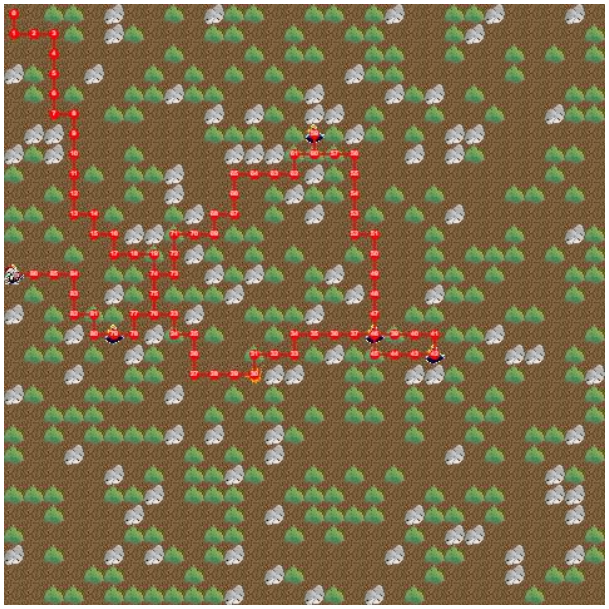
- Test 7



- Test 8

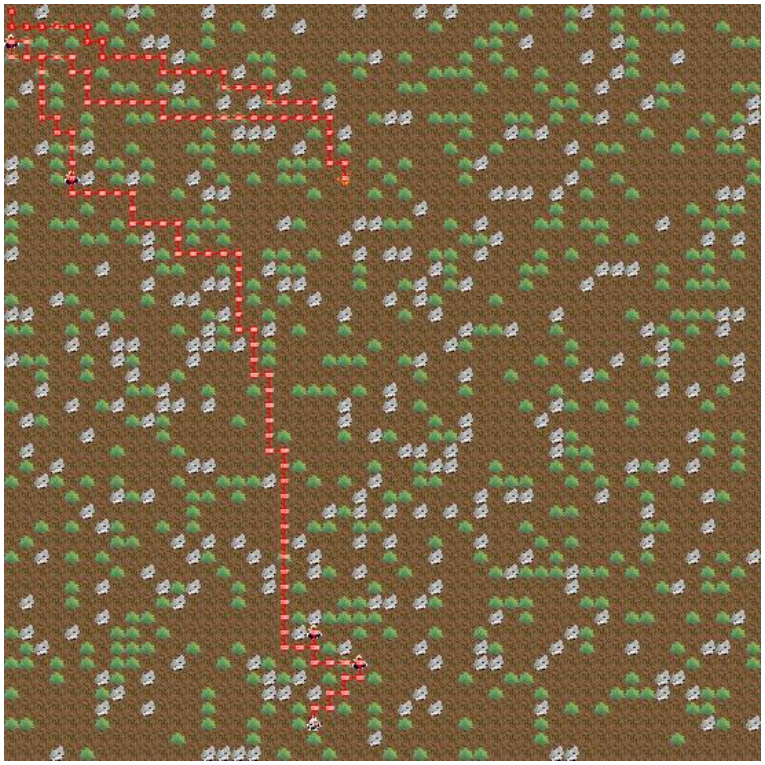


- Test 9

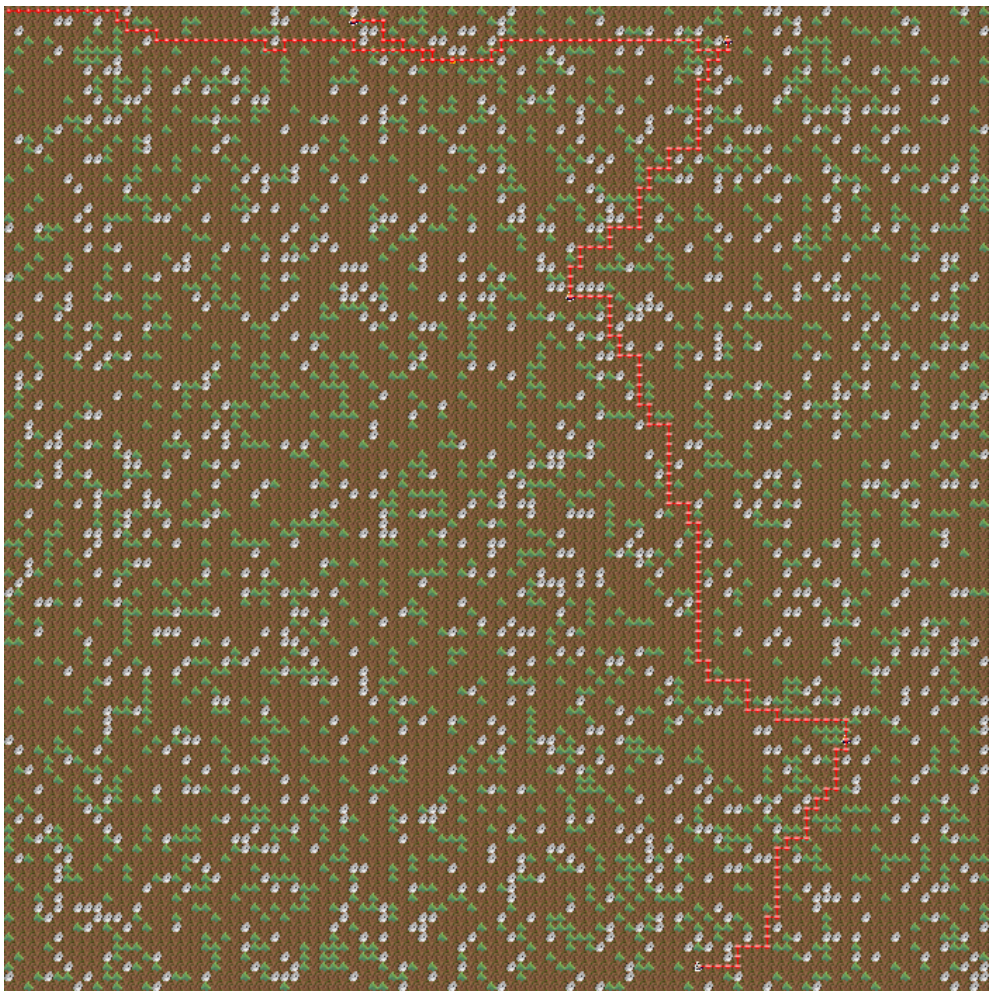


Hrytsyna Anastasiia-Solomiia
ID 105323

- Test 10

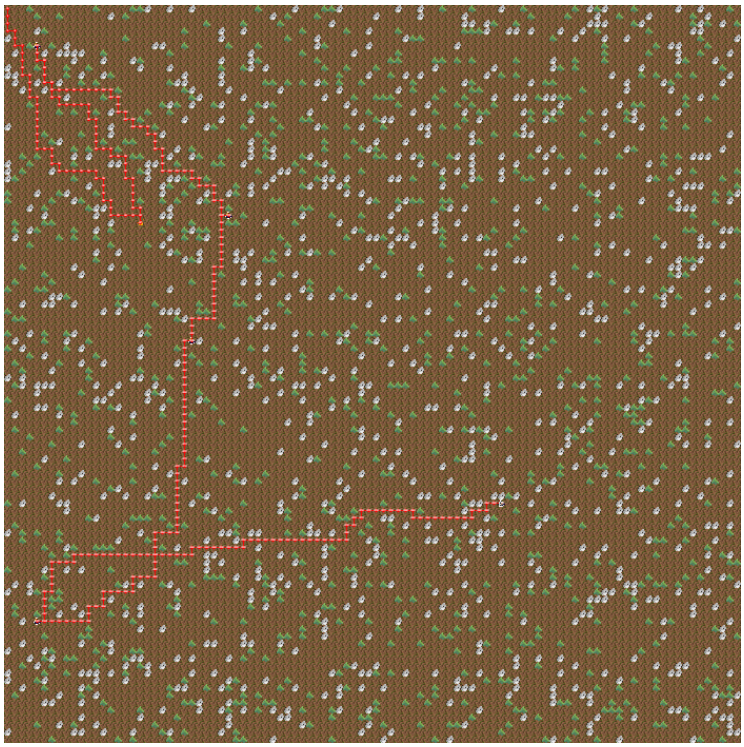


- Test 11

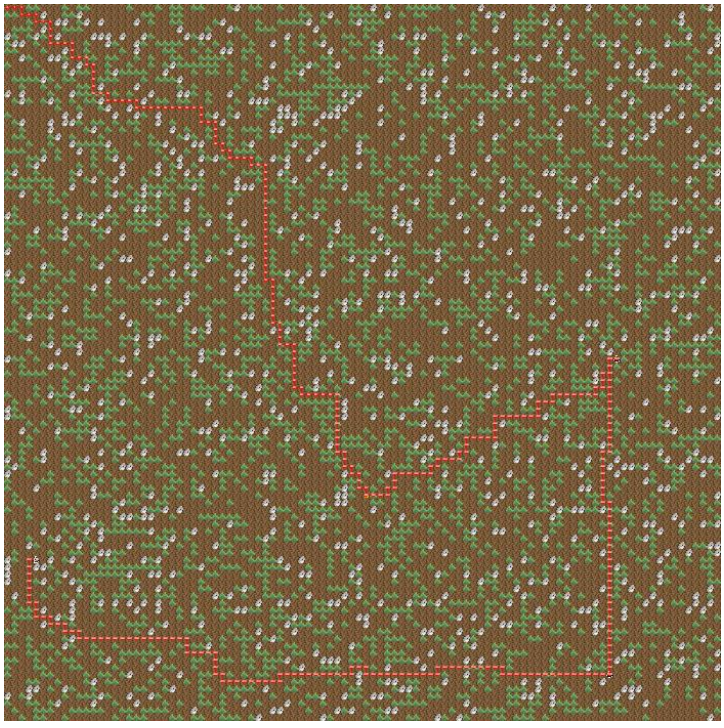


Hrytsyna Anastasiia-Solomiia
ID 105323

- Test 12



- Test 13



By performing these tests I can make a conclusion, that my program works OK. The properness of my results I verified with my friends and nobody of them didn't get the shorter/quicker way at the same maps ☺. In addition I represented my maps and solutions at the pictures (generated by https://popolvar.surge.sh/) and I didn't find any issue with passing thought the rocks or forgetting to pick up some of the princesses for example…(in addition my time and time generated by a site were the same).