1^{Η} ΑΣΚΗΣΗ ΣΤΗΝ ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΓΙΩΡΓΟΣ ΧΑΝΤΖΗΑΛΕΞΙΟΥ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 03110749

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΧΡΗΣΤΟΣ ΜΗΤΡΟΠΟΥΛΟΣ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 03110103

Ερώτηση 1

Στο πρόβλημα αυτό ασχοληθήκαμε με εύρεση βέλτιστης λύσης σε χώρο καταστάσεων. Συγκεκριμένα, υλοποιήσαμε αλγόριθμο A* έτσι ώστε τα ρομπότ να βρίσκουν το βέλτιστο μονοπάτι που θα ακολουθήσουν από το αρχικό στον τελικό τους στόχο.

Τα ρομπότ κινούνται υπολογίζοντας το βέλτιστο μονοπάτι που θα ακολουθήσουν χρησιμοποιώντας τον αλγόριθμο A*. Τα ρομπότ μπορούν να κινηθούν μόνο δεξιά – αριστερά και πάνω – κάτω. Για αυτό το λόγο σαν υπο-εκτιμήτρια ευριστική συνάρτηση χρησιμοποιήθηκε η συνάρτηση Manhattan και σαν υπερ-εκτιμήτρια συνάρτηση χρησιμοποιήθηκε η ευκλείδια απόσταση δύο σημείων υψωμένη στο τετράγωνο.

Τα δύο ρομπότ γνωρίζουν τη μορφολογία του χώρου όμως έχουν μία πολύ σημαντική διαφορά, ενώ το ρομπότ 2 θα πρέπει ξεκινώντας από την αρχική του θέση να καταλήξει σε μια προκαθορισμένη θέση, γεγονός το οποίο υλοποιούμε χρησιμοποιώντας τον A* αλγόριθμο, το ρομπότ 1 θα πρέπει κάθε φορά να υπολογίζει την βέλτιστη διαδρομή για να «πιάσει» το ρομπότ 2 με βάση την τρέχουσα θέση του ρομπότ 2. Επομένως στην ουσία επαναυπολογίζω κάθε φορά την βέλτιστη διαδρομή που πρέπει να ακολουθήσει το ρομπότ 1 για να «πιάσει» το ρομπότ 2 , με βάση την τρέχουσα θέση του ρομπότ 2. Η αρχική και η τελική θέση του ρομπότ 2 δίνεται στο αρχείο εισόδου μαζί με το χώρο στο οποίο πρόκειται να κινηθεί. Το ρομπότ 1 έχει μια τυχαία αρχική θέση και αφού κινείται με διπλάσια ταχύτητα από το ρομπότ 2 κάποια στιγμή θα το συναντήσει.

Ερώτηση 2

Η δομή δεδομένων που επιλέξαμε είναι τα Binary Heaps. Τα Binary Heaps είναι μία δεντρική κατασκευή η οποία είναι αποθηκευμένη σε έναν πίνακα και η οποία σε αντίθεση με τις συνηθισμένες δεντρικές δομές δεν χρησιμοποιεί δείκτες για να αναφερθεί στα παιδιά της, αλλά χρησιμοποιεί Indexing. Οι πολυπλοκότητες της συγκεκριμένης δομής είναι: O(n) για δούμε αν ένα σημείο ανήκει στο Binary Heap και O(logn) για εισαγωγή νέου στοιχείου και εξαγωγή.

Η υλοποίηση της δομής δεδομένων γίνεται στο αρχείο graph.js , ενώ η αρχικοποίηση και το πέρασμα παραμέτρων γίνεται στο αρχείο astar.js και ai.html.

Ερώτηση 3

Για την υλοποίηση του A*, οι βασικές συναρτήσεις που σχεδιάστηκαν είναι οι εξής:

- Η συνάρτηση *init* η οποία αρχικοποιεί όλους τους κόμβους στο grid.
- η συνάρτηση *search* η οποία δέχεται σαν ορίσματα το grid το οποίο είναι ο χώρος που κινούνται τα ρομπότ, με βάση την είσοδο που δεχόμαστε στο κύριο πρόγραμμα ανάγνωσης ai.html , μία αρχική και μία τελική θέση, καθώς και μία ευριστική συνάρτηση. Στη συνέχεια υλοποιείται η βασική ιδέα του A* , με το open set που περιέχει τους υπο εξέταση κόμβους και το closed set που περιέχει τους κόμβους που έχουν ήδη εξεταστεί.

- Η συνάρτηση *heuristic2* η οποία είναι η υπερεκτιμήτρια (non-admissible) ευριστική που επιλέχθηκε και στην ουσία χρησιμοποιεί ως απόσταση 2 σημείων την ευκλείδια απόσταση υψωμένη στο τετράγωνο.
- Η συνάρτηση *manhattan* η οποία είναι η υποεκτιμήτρια συνάρτηση και βρίσκει τη manhattan distance δύο σημείων.
- Η συνάρτηση *neighbors* η οποία παίρνει σαν το grid που είναι ο χώρος που κινούνται τα ρομπότ και ένα κόμβο και μας επιστρέφει όλους του γείτονές του προς τις κατευθύνσεις που μπορεί να κινηθεί(North,South,East,West).

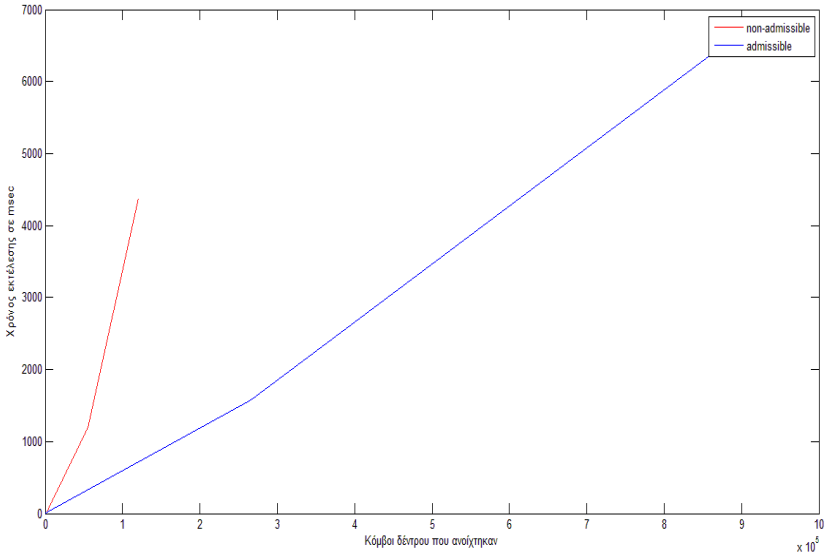
Ερώτηση 4

Ως admissible heuristic χρησιμοποιήθηκε η συνάρτηση απόστασης Manhattan. Ο λόγος που είναι admissible είναι διότι ένα robot χρειάζετα τόσα βήματα όσα δείχνει η απόσταση Manhattan για να φτάσει στο στόχο στην καλύτερη περίπτωση, αν δεν συναντήσει εμπόδια. Διαφορετικά χρειάζεται περισσότερα βήματα.

Ως non-admissible heuristic χρησιμοποιήθηκε η συνάρτηση $x^2 + y^2$. Αυτή η συνάρτηση είναι non-admissible heuristic. Ο λόγος που είναι non-admissible είναι διότι το αποτέλεσμα που επιστρέφει θα είναι πάντα μεγαλύτερο από το βέλτιστο μονοπάτι αν δεν υπήρχαν εμπόδια.

Ερώτηση 5

Τα στατιστικά αποτελέσματα που προκύπτουν μετά από είσοδο με χώρο κίνησης για τα ρομπότ 10x10, 20x20,40x40,50x50,500x500,1000x1000 φαίνονται στο παρακάτω διάγραμμα. Να σημειώσουμε ότι τα αρχεία εισόδου δημιουργήθηκαν με τυχαίο τρόπο και ότι τα διαγράμματα έγιναν με το πρόγραμμα MATLAB και όπως φαίνεται και στο διάγραμμα στον άξονα των x έχουμε του κόμβους του δέντρου που ανοίχτηκαν κατά την εκτέλεση και στον άξονα των y έχουμε το χρόνο εκτέλεσης σε msec. Επιπλέον με μπλε γραμμή απεικονίζεται η admissible ευριστική και με κόκκινη γραμμή η non-admissible ευριστική.



Ερώτηση 6

Παρατηρώντας τα διαγράμματα βγάζουμε τα εξής συμπεράσματα:

- Για μικρά αρχεία εισόδου οι διαφορές είναι ελάχιστες και αμελητέες.
- Όσο μεγαλώνουν όμως οι χώροι καταστάσεων η non-admissible heuristic εκτελείται αρκετά πιο γρήγορα και σχεδόν στο μισό χρόνο για χώρο καταστάσεων 1000x1000 (τελευταίο σημείο των δυο γραφικών παραστάσεων)
- Επιπλέον η non-admissible ανοίγει για εξέταση αρκετά λιγότερους κόμβους από την admissible για το ίδιο μέγεθος χώρου καταστάσεων.
- Και για τις δύο Heuristic συναρτήσεις η σχέση μεταξύ των κόμβων που ανοίχτηκαν και του χρόνου εκτέλεσης είναι τμηματικά γραμμική με την Non-admissible όμως να έχει σαφώς πιο απότομη κλίση.

Θα περίμενε κανείς λοιπόν αφού η non-admissible heuristic είναι πιο γρήγορη και πιο οικονομική σε μνήμη να είναι η επικρατέστερη επιλογή μας κάθε φορά που θέλουμε να υλοποιήσουμε τον A*, ή οποιοδήποτε άλλο αλγόριθμο αναζήτησης. Αυτό δεν συμβαίνει γιατί το trade-off για να πετύχει η non-admissible γρηγορότερο χρόνο και λιγότερους υπο εξέταση κόμβους είναι ότι συμβιβάζεται με υποβέλτιστες λύσεις, σε αντίθεση με την admissible η οποία μπορεί να είναι πιο αργή και να εξετάζει περισσότερους κόμβους για ίδια μεγέιη χώρων καταστάσεων, αλλά βρίσκει πάντα τη βέλτιστη λύση.

Τελικά η επιλογή ανάμεσα σε non-admissible και admissible είναι στην κρίση του προγραμματιστή, ανάλογα κάθε φορά με τις απαιτήσεις της εφαρμογής σε ταχύτητα και μνήμη και την ακρίβεια που θέλουμε να έχουμε στη λύση.

Ερώτηση 7

Η υλοποίηση έγινε χρησιμοποιώντας τη γλώσσα JavaScript σε συνδυασμό με την Html για την απεικόνιση των αποτελεσμάτων σε ιντερνετικό περιβάλλον. Το κυρίως πρόγραμμα είναι το αρχείο ai.html στο οποίο διαβάζουμε το αρχείο εισόδου και τυπώνουμε τα αποτελέσματα και τα βοηθητικά αρχεία που χρησιμοποιούνται είναι το graph.js στο οποίο υλοποιούνται τα Binary Heaps και το astar.js στο οποίο υλοποιείται ο A* αλγόριθμος αναζήτησης. Τα αποτελέσματα που τυπώνουμε είναι το σημείο συνάντησης και οι διαδρομές που ακολούθησαν τα δύο ρομπότ μέχρι να συναντηθούν,καθώς και ο χρόνος εκτέλεσης και οι κόμβοι που εξετάστηκαν. Επιπλέον εάν το ρομπότ 1 προσπαθώντας να πιάσει το ρομπότ 2 εγκλωβιστεί, τυπώνεται η είσοδος του χρήστη στην οθόνη, ενώ εάν ο χρήστης δώσει αρχική ή τελική θέση για το ρομπότ 2 η οποία είναι εμπόδιο εμφανίζεται alert message που τον ενημερώνει.

Η υλοποίηση της άσκησης με την non-admissible heuristic function έχει ανέβει και στον προσωπικό μας web server στον okeano σε αυτή τη διεύθυνση: http://83.212.115.132/ai-robots-A*/ai.html , ενώ Offline μπορεί κανείς να τρέξει το πρόγραμμα μέσω οποιουδήποτε browser έχοντας όλα τα αρχεία στον ίδιο φάκελο.

Τα αρχεία είναι τα εξής:

ai.html

```
<!--//Artificial Inteligence project -NTUA//
//Christos Mitropoulos - 03110103//
//George Chantzialexiou - 03110749//
////////////////////////////////////-->
<!DOCTYPE html>
<html lang="en">


<script type='text/javascript' src='graph.js'></script>
<script type='text/javascript' src='astar.js'></script>



<head>
   <title>Artificial Intelligence </title>
   <h1>Artificial Intelligence Project by Christos Mitropoulos and George Chantzialexiou</h1>
   <p>Robots Movement with A* Algorithm <br>If process is taking too long, try reloading the
page, robot 1 cannot reach robot 2 because it is stuck <br> </p>
</head>
<body>
   <style>
 #byte_content {
   margin: 5px 0;
   max-height: 100px;
   overflow-y: auto;
   overflow-x: hidden;
 }
 #byte_range { margin-top: 5px; }
 #cont {margin-top: 10px}
</style>

<input type="file" id="files" name="file" /> Click to Load input File:
<span class="readBytesButtons">
 <button>Load</button>
</span>
<div id="byte_range"></div>
<div id="byte_content"></div>

<div id="cont"></div>

<script>

   var obj = new Object();
   var Test = "";
   var GraphData;
   var res = new Array;
   obj.message = "";
   var j =0;
   var k=0;
   var graph = new Graph([]);
 function readBlob(opt_startByte, opt_stopByte) {

   var files = document.getElementById('files').files;
   if (!files.length) {
     alert('Please select a file!');
     return;
   }
```

```
  var file = files[0];
  var start = parseInt(opt_startByte) || 0;
  var stop = parseInt(opt_stopByte) || file.size - 1;

  var reader = new FileReader();

  // If we use onloadend, we need to check the readyState.
  reader.onloadend = function(evt,Test) {
    if (evt.target.readyState == FileReader.DONE) { // DONE == 2
 //     document.getElementById('byte_content').textContent = evt.target.result;
      document.getElementById('byte_range').textContent =
          ['Read bytes: ', start + 1, ' - ', stop + 1,
           ' of ', file.size, ' byte file'].join('');
 //     Test=evt.target.result;
 obj.message = evt.target.result;
  //  document.getElementById("cont").innerHTML = Test;
 // console.log(obj.message);


    }
  };

  var blob = file.slice(start, stop + 1);
//  reader.readAsBinaryString(blob);
  reader.readAsText(blob,"UTF-8");
}



document.querySelector('.readBytesButtons').addEventListener('click', function(evt,Test) {

  if (evt.target.tagName.toLowerCase() == 'button') {
    var startByte = evt.target.getAttribute('data-startbyte');
    var endByte = evt.target.getAttribute('data-endbyte');
    readBlob(startByte, endByte);

  }
}, false);

  document.getElementById('byte_content').textContent =obj.message;
//  console.log(obj.message);

function myFunction()
{
  var start = +new Date();
  document.getElementById("mess").innerHTML = obj.message;
  var a=0;

  for ( i=0;i<6;i++){
     a = obj.message.charCodeAt(k);
     while (a!=32 && a!=10 ) {  // to 32 k to 10 einai Space & NewLine ASCII values
        k+=1;
        a = obj.message.charCodeAt(k);
     }
     k=k+1;
     res[i] = parseInt(obj.message);
     obj.message = obj.message.substring(k);
     k=0;
```

```
    }


    var Tixos=new Array ;
    j=0;
   for (i=0;i<obj.message.length;i++){
      if (obj.message[i]=="O"){
         Tixos[j]=1;
         j +=1;
      }
      else if (obj.message[i]=="X"){
         Tixos[j]=0;
         j+=1;
      }
   }

var grid;
   grid=new Array(res[0]);
   for(i=0;i<res[0];i++){
     grid[i]=new Array(res[1]);  //create a two dimensional array after reading the file
   }




            k=0;
            for(i=0;i<res[0];i++){

              for(j=0;j<res[1];j++){


                  if(Tixos[k]==0){ //the symbol we are reading is X which is wall

                           grid[i][j]=0;//document.write("the symbol we are reading is XI  <br>");
            }
                        else if(Tixos[k]==1){ //the symbol is O which is open path

                           grid[i][j]=1; //document.write("the symbol is Omikron  <br>");
            }

                        //document.write(grid[i][j]+"<br>");

                    k++;

                  }

       }
 var graph = new Graph(grid);

 console.log(graph);
 console.log(grid);
//document.write("graph created");

var start_robot2 = graph.nodes[res[2]][res[3]];
var end_robot2 = graph.nodes[res[4]][res[5]];

var random1=Math.floor(Math.random() * (graph.input.length-1));// console.log(random1);
```

```
var random2=Math.floor(Math.random() * (graph.input[0].length-1));// console.log(random2);
while (graph.nodes[random1][random2]==0) {
   random1=Math.floor(Math.random() * (graph.input.length-1));// console.log(random1);
   random2=Math.floor(Math.random() * (graph.input[0].length-1));// console.log(random2);
}
var start_robot1=graph.nodes[random1][random2]; // random choice of robot's 1 start



var robot1_path= new Array;   //i create an array where i will store all the squares robot1 has
been
var robot2 = astar.search(graph.nodes, start_robot2, end_robot2);  // robot2 array with A* path
for robot2
var i=0;
robot1_path[0]=start_robot1;
if (start_robot1.type==0 || start_robot2.type==0 || end_robot2.type==0)
  {alert("Error, cannot go there");}
else{
while((i <robot2.length) && ((astar.manhattan(robot2[i],start_robot1))>1)  ){
 var robot1=astar.search(graph.nodes, start_robot1, robot2[i]); //robot1 array with A* path to
robot2's current place
    start_robot1=robot1[1];//move robot1 two squares
            robot1_path[i+1]=start_robot1; //store all the robot1 paths
i++;

}

}
j=i-1; //last square of robot2
if(i==robot2.length && start_robot1!=robot2[j]){ // if robot2 reached his final destination and
robot1 hasn't find yet we need to see if robot1 can still go there

            robot1=astar.search(graph.nodes, start_robot1, robot2[j]); //robot1 array with A* path
to robot2's current place
    document.write("meeting point is "+robot1[robot1.length-1]+" <br> ");
    robot1_path=robot1_path.concat(robot1);
            document.writeln(" and robot's 1 path is "+robot1_path);
            document.writeln("<br> and robot's 2 path from it's starting point is "+robot2);
}
 else if (start_robot1==robot2[j]) {
 document.writeln("meeting point is "+start_robot1+"<br>  ");
 document.writeln("robot's 1 path is "+robot1_path+"<br> and robot's 2 path from it's starting
point is ");
  for(k=0;k<=j;k++) document.write(robot2[k]);
 }

else if ((astar.manhattan(robot2[i],start_robot1))==1){
 document.writeln("meeting point is "+robot2[i+1]+"<br> ");
document.writeln("robot's 1 path is "+robot1_path.concat(robot2[i+1])+"<br> and robot's 2 path
from it's starting point is");
for(k=0;k<=i+1;k++)
    document.writeln(robot2[k]);



}
  var end =  +new Date();  // log end timestamp
var diff = end - start;

document.write("<br>The time for finding robot 2 is "+ diff +" ms")
```

```
document.write("<br> And the graphs that were opened during search are
"+open_graph_counter);//console.log(diff);

 }



</script>

<button type="button" onclick = "myFunction()"> Execute</button>
<p id = "mess" > </p>
</body>
</html>
```

astar.js

```
//Artificial Inteligence project -NTUA//
//Christos Mitropoulos - 03110103//
//George Chantzialexiou - 03110749//
////////////////////////////////////////
// Implements the astar search algorithm in javascript using a binary heap.
var  open_graph_counter=0;
var astar = {
   init: function(grid) {
      for(var x = 0, xl = grid.length; x < xl; x++) {
         for(var y = 0, yl = grid[x].length; y < yl; y++) {
            var node = grid[x][y];
            node.f = 0;
            node.g = 0;
            node.h = 0;
            node.cost = node.type;
            node.visited = false;
            node.closed = false;
            node.parent = null;
         }
      }
   },
   heap: function() {
      return new BinaryHeap(function(node) {
         return node.f;
      });
   },
   search: function(grid, start, end,  heuristic) {
      astar.init(grid);
            //this is where we choose the heuristic function
     // heuristic = astar.manhattan;    //admissible heuristic
       heuristic =astar.heuristic2 ;   //non-admissible heuristic

      var openHeap = astar.heap();

      openHeap.push(start);

      while(openHeap.size() > 0) {

         // Grab the lowest f(x) to process next.  Heap keeps this sorted for us.
         var currentNode = openHeap.pop();

         // End case -- result has been found, return the traced path.
         if(currentNode === end) {
```

```
            var curr = currentNode;
            var ret = [];
            while(curr.parent) {
              ret.push(curr);
              curr = curr.parent;
            }
            return ret.reverse();
          }

          // Normal case -- move currentNode from open to closed, process each of its neighbors.
          currentNode.closed = true;

          // Find all neighbors for the current node. Optionally find diagonal neighbors as well (false
by default).
          var neighbors = astar.neighbors(grid, currentNode);

          for(var i=0, il = neighbors.length; i < il; i++) {
            var neighbor = neighbors[i];

            if(neighbor.closed || neighbor.isWall()) {
              // Not a valid node to process, skip to next neighbor.
              continue;
            }

            // The g score is the shortest distance from start to current node.
            // We need to check if the path we have arrived at this neighbor is the shortest one we
have seen yet.
            var gScore = currentNode.g + neighbor.cost;
            var beenVisited = neighbor.visited;

            if(!beenVisited || gScore < neighbor.g) {

              // Found an optimal (so far) path to this node.  Take score for node to see how good it
is.
              neighbor.visited = true;
              neighbor.parent = currentNode;
              neighbor.h = neighbor.h || heuristic(neighbor.pos, end.pos);
              neighbor.g = gScore;
              neighbor.f = neighbor.g + neighbor.h;

              if (!beenVisited) {
                // Pushing to heap will put it in proper place based on the 'f' value.
                openHeap.push(neighbor);
                                open_graph_counter++;

              }
              else {
                // Already seen the node, but since it has been rescored we need to reorder it in the
heap
                openHeap.rescoreElement(neighbor);
              }
            }
          }
        }

        // No result was found - empty array signifies failure to find path.
        return 1;
      },
            heuristic2: function(pos0, pos1) {
```

```
    var dx = Math.abs(pos1.x - pos0.x)
    var dy = Math.abs(pos1.y - pos0.y)
    return (dx * dx + dy * dy);
  },
  manhattan: function(pos0, pos1) {


    var d1 = Math.abs (pos1.x - pos0.x);
    var d2 = Math.abs (pos1.y - pos0.y);
    return d1 + d2;
  },
  neighbors: function(grid, node) {
    var ret = [];
    var x = node.x;
    var y = node.y;

    // West
    if(grid[x-1] && grid[x-1][y]) {
      ret.push(grid[x-1][y]);
    }

    // East
    if(grid[x+1] && grid[x+1][y]) {
      ret.push(grid[x+1][y]);
    }

    // South
    if(grid[x] && grid[x][y-1]) {
      ret.push(grid[x][y-1]);
    }

    // North
    if(grid[x] && grid[x][y+1]) {
      ret.push(grid[x][y+1]);
    }

    return ret;
  }
};
```

graph.js

```
//Artificial Inteligence project -NTUA//
//Christos Mitropoulos - 03110103//
//George Chantzialexiou - 03110749//
////////////////////////////////////
var GraphNodeType = {
  OPEN: 1,
  WALL: 0
};

// Creates a Graph class used in the astar search algorithm.
function Graph(grid) {
  var nodes = [];

  for (var x = 0; x < grid.length; x++) {
    nodes[x] = [];
```

```
      for (var y = 0, row = grid[x]; y < row.length; y++) {
        nodes[x][y] = new GraphNode(x, y, row[y]);
      }
    }

    this.input = grid;
    this.nodes = nodes;
}

Graph.prototype.toString = function() {
    var graphString = "\n";
    var nodes = this.nodes;
    var rowDebug, row, y, l;
    for (var x = 0, len = nodes.length; x < len; x++) {
      rowDebug = "";
      row = nodes[x];
      for (y = 0, l = row.length; y < l; y++) {
        rowDebug += row[y].type + " ";
      }
      graphString = graphString + rowDebug + "\n";
    }
    return graphString;
};

function GraphNode(x,y,type) {
    this.data = { };
    this.x = x;
    this.y = y;
    this.pos = {
      x: x,
      y: y
    };
    this.type = type;
}

GraphNode.prototype.toString = function() {
    return "[" + this.x + " " + this.y + "]";
};

GraphNode.prototype.isWall = function() {
    return this.type == GraphNodeType.WALL;
};


function BinaryHeap(scoreFunction){
    this.content = [];
    this.scoreFunction = scoreFunction;
}

BinaryHeap.prototype = {
    push: function(element) {
      // Add the new element to the end of the array.
      this.content.push(element);

      // Allow it to sink down.
      this.sinkDown(this.content.length - 1);
    },
    pop: function() {
```

```
    // Store the first element so we can return it later.
    var result = this.content[0];
    // Get the element at the end of the array.
    var end = this.content.pop();
    // If there are any elements left, put the end element at the
    // start, and let it bubble up.
    if (this.content.length > 0) {
        this.content[0] = end;
        this.bubbleUp(0);
    }
    return result;
},
remove: function(node) {
    var i = this.content.indexOf(node);

    // When it is found, the process seen in 'pop' is repeated
    // to fill up the hole.
    var end = this.content.pop();

    if (i !== this.content.length - 1) {
        this.content[i] = end;

        if (this.scoreFunction(end) < this.scoreFunction(node)) {
            this.sinkDown(i);
        }
        else {
            this.bubbleUp(i);
        }
    }
},
size: function() {
    return this.content.length;
},
rescoreElement: function(node) {
    this.sinkDown(this.content.indexOf(node));
},
sinkDown: function(n) {
    // Fetch the element that has to be sunk.
    var element = this.content[n];

    // When at 0, an element can not sink any further.
    while (n > 0) {

        // Compute the parent element's index, and fetch it.
        var parentN = ((n + 1) >> 1) - 1,
            parent = this.content[parentN];
        // Swap the elements if the parent is greater.
        if (this.scoreFunction(element) < this.scoreFunction(parent)) {
            this.content[parentN] = element;
            this.content[n] = parent;
            // Update 'n' to continue at the new position.
            n = parentN;
        }

        // Found a parent that is less, no need to sink any further.
        else {
            break;
        }
    }
```

```
  },
  bubbleUp: function(n) {
    // Look up the target element and its score.
    var length = this.content.length,
      element = this.content[n],
      elemScore = this.scoreFunction(element);

    while(true) {
      // Compute the indices of the child elements.
      var child2N = (n + 1) << 1, child1N = child2N - 1;
      // This is used to store the new position of the element,
      // if any.
      var swap = null;
      // If the first child exists (is inside the array)...
      if (child1N < length) {
      // Look it up and compute its score.
      var child1 = this.content[child1N],
        child1Score = this.scoreFunction(child1);

      // If the score is less than our element's, we need to swap.
      if (child1Score < elemScore)
        swap = child1N;
      }

      // Do the same checks for the other child.
      if (child2N < length) {
        var child2 = this.content[child2N],
          child2Score = this.scoreFunction(child2);
        if (child2Score < (swap === null ? elemScore : child1Score)) {
          swap = child2N;
        }
      }

      // If the element needs to be moved, swap it, and continue.
      if (swap !== null) {
        this.content[n] = this.content[swap];
        this.content[swap] = element;
        n = swap;
      }

      // Otherwise, we are done.
      else {
        break;
      }
    }
  }
};
```