# Homework: Heaps and Priority Queues

This document defines the **homework assignments** for the ["Data Structures" course @ Software University](https://softuni.bg/trainings/1147/Data-Structures-June-2015).

## Implement Decrease Key

Extend your Binary Heap to support the DecreaseKey(T element) operation, that changes the priority of a given key. In a Min Binary Heap this should increase the priority of a given key, moving it higher in the tree structure, e.g. decreasing the price of a given product, increases its priority for the customers.

## A\* Algorithm

You are given a skeleton. Your task is to implement the A\* algorithm in order to find the shortest path from a starting point "P" (Start) to a goal point "\*" (Goal) on a given grid of squares. Player is only allowed to walk up, right, down or left. The AStar class should return the path as IEnumerable<Node>, each entry corresponding to the next cell in the shortest path.

You can read more about the A\* here: <http://web.mit.edu/eranki/www/tutorials/search/>

Or here: <http://www.redblobgames.com/pathfinding/a-star/introduction.html>

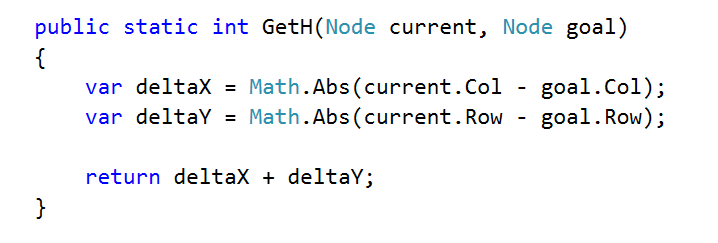
|  |  |  |
| --- | --- | --- |
| **Maze** | **Output Nodes** | **Path** |
| -----  -\*---  WWWW-  ---P- | { "3 3", "3 4", "2 4", "1 4", "1 3", "1 2", "1 1" } | -----  -@@@@  WWWW@  ---@@ |

If there is no path to the goal, return IEnumerable<Node> containing only the start node.

|  |  |  |
| --- | --- | --- |
| **Maze** | **Output Nodes** | **Path** |
| -----  -\*---  WWWWW  ---P- | { "3 3" } | -----  -----  WWWWW  ---@- |

### Hints: H Cost

First of all, implement the method GetH(). **H** is the approximation of the distance from the current node to the goal. Use **Manhattan distance** (total number of squares moved horizontally and vertically to reach the target, ignoring diagonal movement, and ignoring any obstacles that may be in the way)



### Hints: A\* Pseudocode

We need some way to store to cost to a given node and the node that we are coming from.

* OPEN = priority queue containing START
* PARENT = dictionary storing the node from which we have reached a node (following a path)
* COST = dictionary storing cost from the start to a node (following a path)
* PARENT[START] = null
* COST[START] = 0
* while OPEN is not empty:
  + current = remove highest priority item from OPEN
  + if current is the goal 🡪 break
  + for each neighbor of current (up, right, down, left):
    - new cost = COST[current] + 1
    - if neighbor is not in COST or new cost < COST[neighbor]
      * COST[neighbor] = new cost
      * neighbor.F = new cost + HCost(neighbor, goal)
      * OPEN 🡨 neighbor
      * PARENT[neighbor] = current

You can reconstruct the path following PARENT[goal] to the starting node. If there is no path to the goal PARENT[goal] won't be in the dictionary.