INF 4130 Exercise set 4, 26th Sept 2013 w/ solutions

Exercise 1

Solve exercise 23.6 from the text book (B&P).

Note that it is important that we do not count the empty square when we sum up what is in the wrong spot. If we do, we won't be able to show what we are supposed to. See, for instance, this 2x2-board:

3	1
	2

Here we can get everything in place with three moves (U,R,D), but if we count the empty square we get $h(v) = 4$. The essential point is that when we make a move, only moving a real tile influences the heuristic. This way $h(v)$ is no larger than the number of necessary moves from v, and there is hope of monotonicity.

Monotonicity:

- $h(\text{goal}) = 0$
- If there is an edge from v to w with cost $c(v,w)$, we must have: $h(v) \leq c(v,w) + h(w)$.

1. is obvious, 2. follows from the fact that only one tile is moved when we go from state v to state w, and it is moved only one cell, so the heuristic can at most decrease by 1 when going from v to w, the same as the cost.

Exercise 2

Solve exercise 23.7 from the text book.

If we understood the point of the previous exercise, this one should also be easy.

Monotonicity:

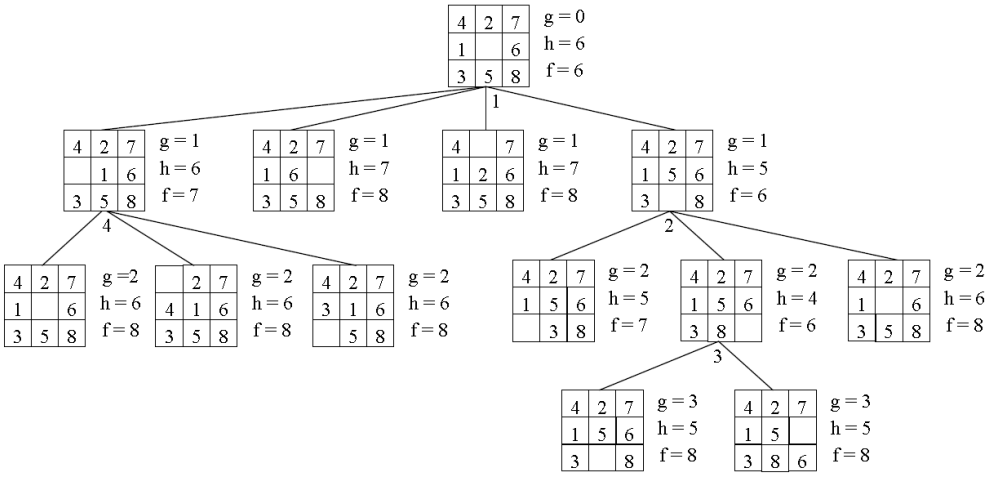
- $h(\text{goal}) = 0$
- If there is an edge from v to w with cost $c(v,w)$, we must have: $h(v) \leq c(v,w) + h(w)$.

1. is obvious, 2. follows from the fact that only one tile is moved when we go from state v to state w, and it is moved only one cell, horizontally or vertically, therefore the Manhattan distance can at most decrease by 1 when going from v to w, the same as the cost.

Exercise 3

Solve exercise 23.8 from the text book.

In the fourth move we have two nodes with $f = 7$, but we assume FIFO for nodes with equal priority. In the nest step the second with $f = 7$ gets expanded further.



Exercise 4

Is your answer regarding monotonicity in 23.7 also valid if we allow moving the hole diagonally?

Monotonicity:

- $h(\text{goal}) = 0$
- If there is an edge from v to w with cost $c(v,w)$, we must have: $h(v) \leq c(v,w) + h(w)$.

The sum of the Manhattan-distances will not be a monotone heuristic if we allow diagonal moves, because it can get larger than the number of moves necessary. Look at:

1	2	3
4		6
7	8	5

Here we get a solution in one move (DR – moving the hole into the square with 5), but the Manhattan-distance gives us $h = 2$.
It is, however, easy to see that if a square is in (x_1, y_1) and its correct position is (x_2, y_2), the fewest number of moves to (x_2,y_2) (if it is alone on the board) is: $\max(|x_2-x_1|, |y_2-y_1|)$. We therefore try with the sum (over all tiles) of this value as our heuristic. For this heuristic we have h(final state) = 0.
To show monotonicity we again have to show that the h-value does not change more than 1 when we make a move. Since only one tile is moved, the question is really whether $\max(a,b)$ can change more than 1 when a and b can increase or decrease by 1, and the same for b (at the same time). It should be clear that this is not the case. (A max-function is a kind of OR.) So the modified heuristic is monotone.

Exercise 5

Is it possible to use the actual cost as our heuristic (it is after all 100% exact and should be good)? Will the actual cost always be monotone? Will we expand a smaller tree? What would the problem be, if any?
To clarify a bit, $h(s)$ is now the cost of the remaining moves along an optimal path from s to a final state. It is of course a bit strange to imagine that we have this as our heuristic, it would constitute a solution to the problem, but hey, stranger things have happened. This exercise can of course tell us something about the behaviour of A* with very good heuristics. If h as described above, we see that the value $f(s) = g(s) + h(s)$ when state s is taken out of the priority queue and placed in the tree, is equal to the shortest path from the start state to a final state, through s. It should therefore be clear that the A*-algorithm will go straight for the best solution and not waste time on sub optimal solutions. Many states may, of course, have the the same f-value, and in these cases the algorithm will follow them "in parallel", the order will depend on how nodes with equal priority are removed from the priority queue. Another point to observe is of course that calculating an exact heuristic is time consuming…

Exercise 6

Show that the straight line (actually the circumference of a great circle, but let's not get into details) between a point and the goal point is a monotone heuristic for finding the shortest path the way it is done in chapter 23.3.3 (page 728).
Monotonicity:

- $h(\text{goal}) = 0$
- If there is an edge from v to w with cost $c(v,w)$, we must have: $h(v) \le c(v,w) + h(w)$.

Let g be our goal. We first observe that we have $h(g) = 0$.
Now let v and w be two places that there is a road with distance $c(v,w)$ between, and let the straight line between them be $a(v,w)$. We have to show that $h(v) \le c(v,w) + h(w)$. We have $h(v) \le a(v,w) + h(w)$ by the triangle inequality, and since $c(v,w) \ge a(v,w)$, we get $h(v) \le a(v,w) + h(w) \le c(v,w) + h(w)$. It is therefore clear that the straight line is a monotone heuristic.

Exercise 7

Assign *g*-, *h*- and *f*-values to the states in figure 23.7 (page 727) and check that we actually avoid expanding the full breadth-first-tree in figure 23.3 (page 719).

Left to the individual student.

Exercise 8

Adjust the DFS procedure below to instead do iterative deepening with one extra level at a time. You should only check once for each node whether it is a goal node, and you need an extra parameter to the procedure DFS. Show how the procedure should be called from a "main" program/procedure for the whole thing to work properly

```
proc DFS(v) {
   if <v is a goal node> then return "…"
   v.visited = TRUE
      for <each neighbor w of v> do
         if not w.visited then DFS(w)
      od
}
```

Things get a little complicated when we search in a graph (as indicated by the variable "visited" in the assignment text). We therefore assume that we search in a tree, so that we never come to an earlier visited node, and therefore do not need the boolean "visited" in the Node class. (The problem arises from the fact that it is tricky to reset the boolen "visited" without doing a separate pass through the graph. This is of course possible, the solution will then be very similar to our tree-approach.)

```
IDDFS(root)                          // Iterative depening DFS
{
  depth = 0
  repeat                             // while(;;), until a solution is found
  {
    result = DLDFS(root, depth)
    if <result is a solution> then
      return result
    depth = depth + 1
  }
}
```

Note: Her the depth is increased exactly i in each search, but we could indeed use larger steps.

```
DLDFS(v, depth)                    // Depth limited DFS
{
  if (depth >= 0 and <v is a goal node>) then
    return v
  else if (depth > 0)
    for <each child w of v> do
      DLDFS(w, depth-1)
  else
    return no-solution
}
```

Exercise 9

Study the example on slide 20 from 12/9 (page 723 on the textbook) to confirm that when $h(v)$ is not monotone, then nodes sometimes will have to be taken back from tree to the priority queue, thus increasing execution time.

The evolution of the tree and the priority queue looks like this:

Dark red indicates a value had to be updated.

Exercise 9

Study the program for the A*-algorithm given on slide 28 from September 17.

Is left to the participants in the group, or to self study

[end]