Priority queue

use library heaps

- add_to_heap
  - key is child you want to add to heap
- min_of_heap
  - gives back next key and its cost
- empty_heap
  - will use
- when you find the cost add it to the heap

will use findall

```
node(state, cost):-
findall(A, cost(State, A), L),
sum_list(L,Cost).
```

*This only gets manhattan distance, will need to get G cost as well

Can add the parent of the node, so you know what parent you used to get to this state

Node:

- parent or list of parents
- cost (g and h)

use library lists

- as you build a path, add it to a list

heap: list of things I know about but haven't done anything with yet

Also need to know the list of things you've seen, so you don't go there again

- Can use seen list (set of nodes)

when you make a list of children, add it to the heap. after you take it off the heap,

for all ages and names, there exists a name such that for this name there is an age, place the result in a list. Will end up with list of lists of the pairs ordered by age. Instead of age, we use cost. The problem with this is you have to resort each time, that's why heaps are better. Heaps just insert in the correct place as you go. But we can use either.

```
setof([A,N], N^ age(N,A), [H|T).
setof([C,S], C^ cost(S,C), [H|T).
```

L = [[4, sharon],[8, cathy],...]

to get the minimum element: use H

- H will be the top element: [4, sharon]
- T will be the rest of the list

pick the state with the lowest overall cost to get to the next state

A* Algorithm Steps:

Goal: determine the shortest path, where the path is a sequence of states

1. **Start at the starting state, add it to the open list**
2. **Look at all states adjacent to the starting state, add them to the open list and save the starting square as each of their parent states.** How do figure out which states are adjacent to the starting state?
3. Remove the starting state from the open list, add it to the closed list
4. Choose one of the states from the open list, the state with the lowest F cost.
    a. Add the previous state as either the parent or one of the parents in the list of parents, depending how it is implemented.
    b. Calculate F = G + H
        i. G: the cost to move from starting state to the current state, add together the costs of each of the different squares when moving from the starting position to the current position. <- Is this addition the correct way to do it? Add the G cost of the parent to the cost to move from the previous state to this state.
        ii. H: the cost to move from the current state to the goal state, again add together the costs of each of the different squares? Drop the chosen state from the open list, add it to the closed list.
5. Check all adjacent states, and add them to the open list, only if they are not on the open list already. Make the previous state the parent of all the adjacent states.
6. If an adjacent state is already on the open list, check to see if the G score for that state is lower if we use the current state to get there. If the G cost is lower, change the state's parent to the previous square and recalculate its F and G costs, if not, don't change its parent.
7. Again, look at all the states in the open list and choose the one with the lowest F cost
8. Repeat the process until the target state is added to the closed list.

How do figure out which states are adjacent to the starting state?

The adjacent states are the child states produced by examining all the valid moves from the current position. If you look at the sample code I put in testAStarPieces.pl line 181 actually performs this.

Is this addition the correct way to do it?

No this description is incorrect for g. g is simply the number of steps taken since the initial state. for example, at the initial state, g is zero. for any of the direct children of the initial state, g = 1. for any children of these states, g=2. etc. Make sense? if you keep a list of parents with the node, then g is simply the length of the list of parent states.

1) Add the starting square (or node) to the open list.
2) Repeat the following:

a) Look for the lowest F cost square on the open list. We refer to this as the current square.
b) Switch it to the closed list.
c) For each of the 8 squares adjacent to this current square …

- If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
- If it isn't on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
- If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. If you are keeping your open list sorted by F score, you may need to resort the list to account for the change.

d) Stop when you:

- Add the target square to the closed list, in which case the path has been found (see note below), or
- Fail to find the target square, and the open list is empty. In this case, there is no path.

3) Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is your path.