**Artificial Intelligence – Spring 2017**

Project 1

Issued: Mar 14th, 2017 Due: Apr 4th, 2017

## 1. Route finding problem

1.1 BFS and DFS

Instructions:

1. Read DFS\_BFS.h and DFS\_BFS.cpp carefully. Make sure that you understand the definition of structure MGraph, and the function createMGraph(), so that you can give the proper input for the state graph to be searched;
2. Read the comments in dfs(), dfs1() and bfs(), which may give you some basic ideas on organizing your code. Of course some other details are for you to add, so that your code can work properly. Also, remember that the non-recursive implementation of DFS should be based on Stack, which has the property of LIFO(last in, first out), while the implementation of BFS should be based on Queue, which has the property of FIFO(first in, first out);
3. Review BFS,FDS algorithm in Lec2 or any data structure lessons you had before. The main strategy of BFS is to explore the shallowest node first, while the main strategy of DFS is to explore the deepest node first.
4. Remember to print the search result after a successful search. For BFS and DFS, the easy way to print your result is to print a node whenever it is visited during the search;

1.2 A\*

Instructions:

1) Read AStar.h and AStar.cpp carefully. Make sure that you understand the definition of structure Graph and function createGraph(), which are very similar to the definition of MGraph and createMGraph() for DFS/BFS. In AStar.h, there is also a definition of GraphNode for A\* search problem, and you should take a look at it.

2) Review A\* algorithm in Lec2. The core idea of A\* algorithm is to explore (expand) node with smallest g+h value. We need a prior queue, named “openlist” to store nodes that will be explored. A nice property of prior queue is that, elements inside of it are ordered. For example, if openlist is C++ priority\_queue object, the function openlist.top() will return the the element with smallest g+h value in A\* algorithm if openlist is defined properly.

3) Here is a pseudo code for reference:

while openlist is not empty

current=openlist.top() //openlist is priority queue,current is a pointer to GraphNode

openlist.pop() //remove the explored node from openlist

if current==goal

return reconstruct\_path(current, initNode, resultPath) //reverse to get path

closelist.push\_back(current) //push the explored node to closelist

successors=getSuccessors(current) //get neighbor nodes of current node

for (i=0;i<successors,size();++i)

if successors[i] in closelist //do nothing if it is already explored

continue

if successors[i] not in openlist //if it is neither explored nor in openlist

successors[i]->g=current->g+graph.edge(current,successors[i])

successors[i]->h=computeHeuristic(successors[i])

successors[i]->preNode=current

openlist.push(successors[i])

else if current->g+ graph.edge(current,successors[i])>=successors[i]->g

continue; //if the cost from current to successor is bigger than previous cost of successor, do not update

successors[i]->g=current->g + graph.edge(current,successors[i]) //else update

successors[i]->h=computeHeuristic(successors[i])

successors[i]->preNode=current

In the above pseudo code, openlist and closelist are defined as follows:

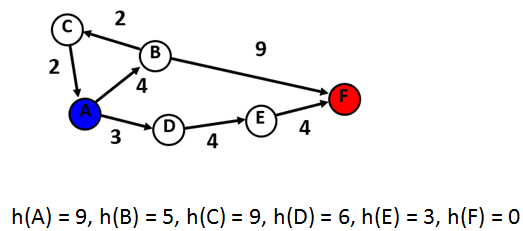
priority\_queue<GraphNode\*, vector<GraphNode\*>, cmpLarge> openlist;

vector<GraphNode\*> closelist;

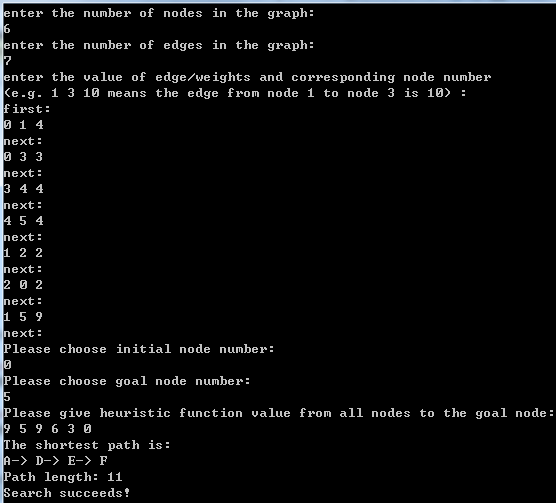
note that they store pointers of GraphNode objects.

4) You need to fill in function AStarShortestPathSearch () and write some other functions. For example, check whether an GraphNode is in openlist or closelist, get the successors of current node, and reconstruct path when goal node has been reached.

Use the example in Lec2:



After you finish these, run main\_route\_finding.cpp, you will get something like the following:



5) Here is a reference for priority\_queue in C++

<http://www.cnblogs.com/chinazhangjie/archive/2010/10/31/1865653.html>

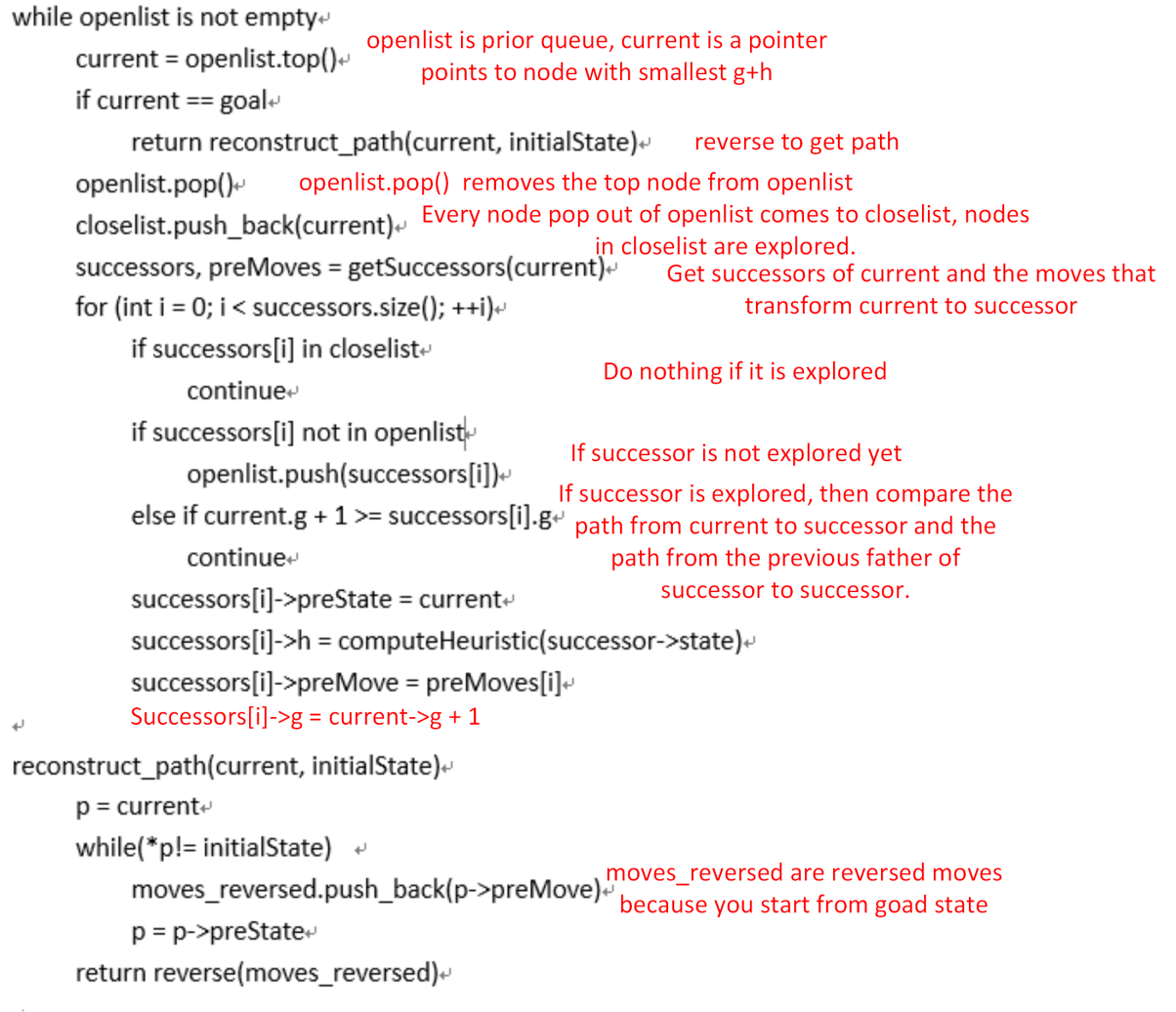
Here is a reference for vector

<http://www.cnblogs.com/wang7/archive/2012/04/27/2474138.html>

## 2.Eight puzzle problem

Instructions:

1. Read eightpuzzle.hpp carefully. Especially, you should read the definition of structure EightPuzzleState, AStarSearchFor8Puzzle() and runMoves(); Notice the definition of moving direction.
2. The same as instruction 2) in 1.2 above for A\* finding shortest path problem;
3. Here is a pseudocode:



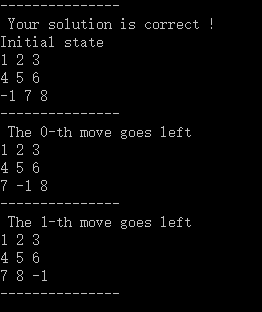
In above pseudocode, openlist and closelist are defined as what I show you in function AStarSearchFor8Puzzle(), i.e.

priority\_queue<EightPuzzleState\*, vector<EightPuzzleState\*>, cmpLarge> openlist;

vector<EightPuzzleState\*> closelist;

Note that they store the pointers of EightPuzzleState objects.

1. You need to fill in function AStarSearchFor8Puzzle() and write some other function like checking whether an EightPuzzleState object is in openlist. After you finish these, run main\_eightpuzzle.cpp, you will get something like the following:



Start from simple examples to debug, you can read line 8 of main\_eightpuzzle.cpp to see how to make it.

1. Zip your BFS\_DFS.hpp, BFS\_DFS.cpp, AStar.hpp, AStar.cpp, eightpuzzle.hpp and eightpuzzle.cpp to XXXXXXXX\_project\_1.zip, XXXXXXX stands for your student ID. Please follow this name rule so I can test your code.
2. The framework is done in limited time, if your find any bugs, please inform me.