Path Finding Documentation

*Iain Dowling Advanced Diploma of Professional Games Development*

*Assessment 3 ( Game Artificial Intelligence )*

Overview

For this assessment we are required to demonstrate knowledge of various artificial intelligence pathfinding techniques and behaviours, as well as implementing said techniques into our Tiny Tanks game.

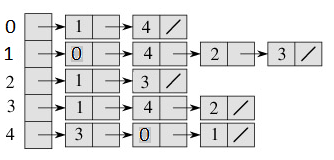
Structure of Graph Data

* **Node object edge objects**

The nodes and edges are both objects. The nodes contain an array of edges, and each edge contains a pointer to another node.

* **Adjacency List.**

An array of Linked lists. Each list in the array shows the connections for one node to other nodes.

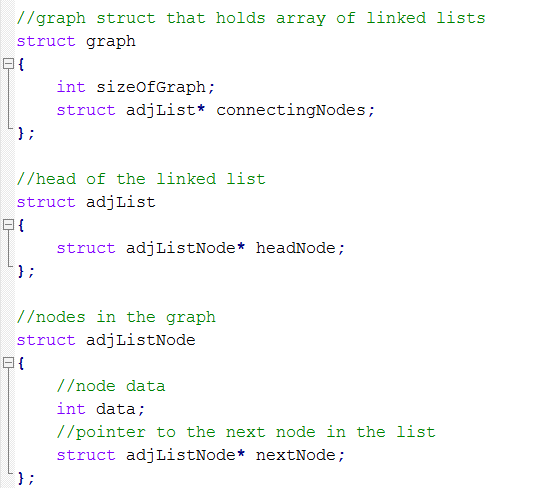


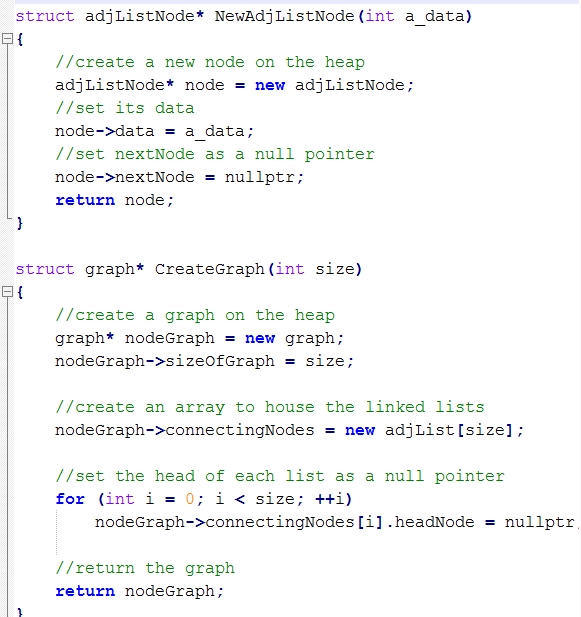
The example on the right shows the connections for a graph of 5 nodes.  
  
The first node in the Graph, represented by a 0, is connected to the 2nd and 5th nodes, represented by 1 and 4 respectively.

*Container type:*

Depending on whether the Graph is dynamic or not, an array of linked lists or a vector of linked lists could be used to represent the adjacency list:

* + An array of Linked lists would be the most memory efficient option for a graph that is not dynamic. As such, the array would not be expected to resize and so the data contained will stay uniform in memory. The array would allow fast retrieval of data via the array indexing operator.
  + A Vector of Linked lists would be the better option for a graph that is expected to be dynamic as it would allow for the addition and deletion of nodes via its resizing capabilities. The drawback is that each time the vector resizes, the data inside will be shuffled around in memory, making accessing it more memory inefficient. Retrieving data from a vector is as easy as using an array as the vector can also make use of the array indexing operators.
  + Linked lists would be the best option to hold the connection data for both arrays and vectors as they are dynamic and would allow new connections to be made and old connection to be deleted easily.

*Code snippet:*

**

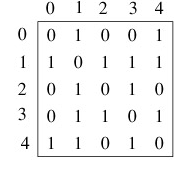
*Pros and cons:*

Adding and removing nodes is easier with this method. This method is also more memory efficient for larger graphs with lots of connections. However it is harder to implement and accessing connections requires looking through an entire linked list that could potentially be hundreds of connections large.

* **Adjacency matrix.**

A 2D array of size [ j ^2 ] where [ j ] equals the number of nodes in the graph.

An intersection between a column and row shows the relationship between the nodes with the respective column/row index.



The example on the right shows the connections for a graph of 5 nodes.

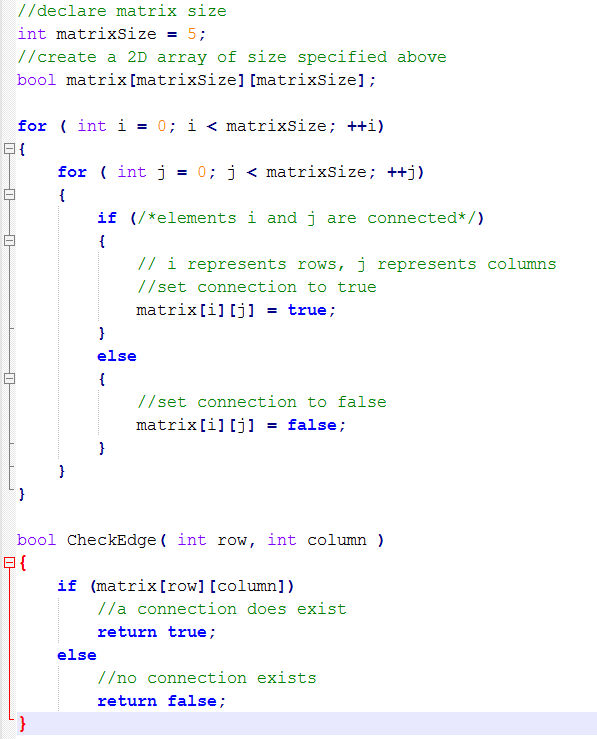
The 1 in the ‘row 2’, ‘column 3’ intersection shows that there is an edge between the 3rd and 4th nodes in the Graph.

*Container type:*

Depending on whether the Graph is dynamic or not, an array or vector could be used to represent the adjacency matrix:

* + A 2D array of nodes would be most memory efficient for a graph that is not dynamic, and as such cannot be expected to resize. It would also allow for fast retrieval of connections data with the use of the array indexing operator.
  + A 2D vector would be more effective for a dynamic array, as it can resize to account for the creation or deletion of new nodes. A vector also allows the use of the array indexing operator so that retrieving connection data would be fast. However, every time the vector resizes its memory will be shifted around, making it less efficient the more it resizes

*Code snippet:*



*Pros and cons:*  
This method is easy to implement and to understand and querying for connections is fast. Adding and removing connections is also fast and easy. However adding and removing nodes can be time consuming and larger graphs with many nodes can take up large amounts of memory.

● Provide 3 methods of storing the data for nodes and edges.

● Show code snippets for each structure used

● for each method, specify what type of data containers you will be using for nodes and

edges and why. (List, Vector, HashMap, Deck, Queue, 2D array etc)

Graph Data Algorithms

From the above methods listed, choose one, and define how the following functions can be

implemented.

● **Finding a node in the graph based on a condition.** *(provide code and written explanation)*

This should be a generic condition that can be changed at runtime, therefore, you could

use lambda functions or function pointers for customising the condition. How will you be

returning the results.

● **Connecting 2 nodes within the graph** *(provide code and written explanation)*

What are some potential errors that could occur when connecting nodes? Will the

pathfinding algorithms you implement work as expected if the same 2 nodes have

duplicate edges? or, could duplicate edges work to your advantage? Why?

● **Adding Nodes to the graph**.*(provide code and written explanation)*

How is the Nodes updated, what are some potential errors? eg 2 nodes placed on top of

each other? is this a problem, or could this be used to your advantage? How/Why?

Will edges be automatically generated with other nodes? What is the criteria for

connecting nodes automatically?

● **Finding Neighbors of a given node** *(provide code and written explanation)*

How are neighbouring nodes stored within your graph, and how will you return a

connected nodes. What return method will you use? custom iterator, return via reference

or return val. Maybe you provide an alternative method.

● **How can this data be saved and loaded from file.** *(provide code and written explanation)*

what file types could you save as? XML, JSON, Custom Text document, Custom Binary

data. Saving and Loading methods.

PathFinding

● How have you implemented the Dijkstra's Path finding algorithm, how is the the path

calculated? (*Provide pseudocode and supporting C++ code Snippets)*

● How is the path stored, what data structures are you using, and what functions are

available for manipulating / getting information from your path?

(*Provide pseudocode and supporting C++ code Snippets)*

○ Path Smoothing, what are some methods you could use?

● How have you modified the Dijkstra's pathfinding Algorithm to perform the AStar

pathfinding algorithm? (*show C++ code Snippets)*

● What are some scenarios you would use the AStar Pathfinding algorithm?

● What are some scenarios you would use the Dijkstra's Pathfinding Algorithm.

● What are the Pro’s and Cons for both Dijkstra's and AStar pathfinding?

Implementation

Show a small code snippet for how you will find a path using your Pathfinding Algorithm. Here is

an example implementation: This implementation will allow for multiple paths to be calculated

simultaneously on the same graph.

// Member variables

Path \*m\_path = nullptr;

PathFinder \*m\_pathUpdater = nullptr;

// called when you want to start finding a path

//m\_

pathUpdater = m\_graph>

FindPath( StartNode, EndNode );

// for Dijkstra's the end node could be a condition

// if a node meets the condition, than a path

// should be returned to that node.

////

during your update loop

**if**( m\_pathUpdater != nullptr )

{

pathUpdater>

Update();

**if**( pathUpdater>

IsPathFound() )

path = pathUpdater>

CalculatePath();

}

**if**( m\_path != null )

{

// TODO: make an entity object follow the path

}

References

List all webpages, textbooks, videos you have used as part of your reasearch for these

pathfinding algorithms.

<http://www.seas.gwu.edu/~simhaweb/alg/lectures/module7/module7.html>

<http://sun.iwu.edu/~sander/CS255/Notes/AdjLists.html>

<http://www.geeksforgeeks.org/graph-and-its-representations/>