CSCE 313-200: Computer Systems

Homework #1 Part 2 (25 pts)

Due date: 2/6/25

1. Problem Description

In this part, you will perform a single-threaded search using four different techniques.

1.1. Code (25 pts)

Your program must now accept three command-line arguments: planet, cave, and the search method:

```
caveSearch.exe 6 44 BFS
```

The other search options are DFS, bFS, and Astar, where all strings are case-sensitive. The print-outs should contain all of those in Part 1 and additionally the following:

```
Initial position: room 8DCD0AE66BDA38D9, light intensity 0.00
Starting search using BFS...
1) visiting 8DCD0AE66BDA38D9, degree 1, discovered 2
2) Visiting 75EB693C57F16B20, degree 28, discovered 29
3) Visiting C14E6529DAD59EC5, degree 40, discovered 68
4) Visiting 27230CD7014A1D42, degree 7, discovered 74
5) Visiting 64ED8B5283AFFD6B, degree 9, discovered 82
...
8959) visiting F9C145893D12537A, degree 3, discovered 9959
Found exit A9E402E1282F881, 8960 steps, distance 6
```

For every visited room, print the current search step (starting with the initial room where the rover is positioned), room ID, its degree (i.e., number of neighbors), and the total number of unique rooms discovered thus far after processing the current room.

During the search, you cannot allow multiple visits to the same room, which means that you must run duplicate elimination on the discovered rooms. Furthermore, you are not allowed to deploy any *linear-scan* algorithms to maintain this data structure. In fact, you are strictly limited to $O(\log n)$ complexity per found room, where n is the cave size.

1.2. Reading Messages of Unknown Size

Note that robot responses contain an unknown amount of data in each message. Furthermore, this data may not fit into your a-priori allocated buffer. The recommended course of action is to start with a fixed buffer size $B = 128$. After the first call to `ReadFile`, if you observe that the buffer is filled to the limit, you can issue `PeekNamedPipe` to retrieve the amount of additional data still stuck in the pipe. Then, you can resize the initial buffer using `realloc` (or `HeapReAlloc`) and read the second part of the response into an appropriate offset within that buffer. If you prefer to use `new/delete`, you can allocate a new buffer, copy the first received portion into it (i.e., B bytes), delete the old buffer, and issue the second `ReadFile` to consume the remaining data.

Note that when the message is the exact size as the buffer, you will see 0 remaining bytes when peeking at the pipe. In such cases*, it is crucial that you avoid trying to read these 0 bytes* from

the pipe as the kernel will wait for something more to be sent from the CC, which will deadlock your code.

1.3. Exit

When your search issues a MOVE to the exit room, the robot returns a SUCCESS response with no neighbors (i.e., just the header).

1.4. Light Dynamics

The exit and certain other rooms (e.g., with holes in the ceiling) are the only sources of light in the otherwise dark cave. Each planet obeys simple rules of physics: the light gradually fades the further you move from its source. Figure 1 shows one example of light propagation with an exponential decay. In this example, the light is *additive*, meaning that incoming brightness from multiple sources is accumulated; however, the exact dynamics of light propagation are not essential to finding the exit.

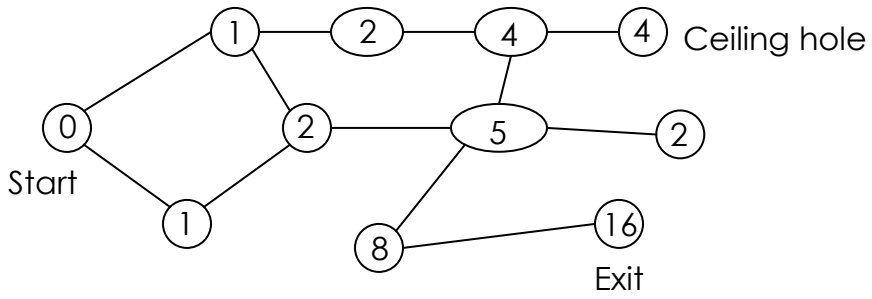

Figure 1. Sample cave (numbers indicate brightness of the room).

1.5. Search Techniques

The studied algorithms – BFS, DFS, bFS, and A* – are explained in more detail below. In all four cases, your program should maintain 1) a set U of *unexplored* rooms, which is built dynamically from MOVE responses; and 2) a set D of *discovered* rooms, which you will use to prevent infinite looping.

In BFS, recall that U is simply a FIFO queue. In DFS, it is a LIFO stack. For bFS, we leverage the observation that brighter rooms are more likely to lead toward the exit and should be preferred compared to darker rooms. Thus, at each step, your program can extract from U the *next brightest* room and attempt to navigate it. While this does not guarantee that you will find the globally optimal (i.e., shortest) path, it ensures that you are likely to get to it quickly. In the worst case, bFS covers the entire graph and thus has the same upper bound on complexity as the two traditional methods.

A* search is similar to bFS, but it additionally takes into account the *distance* from the starting position. Assume each unexplored node i has quality $Q_i = L_i + w / (d_i + 1)$, where L_i is the light intensity of the room, d_i is its distance from the rover, and w is some weight that assigns preference between bFS (i.e., small w) or BFS (i.e., large w). In fact, $w = 0$ produces bFS and $w = \infty$

makes BFS. Unless you find a better constant, use $w = 20$ in this homework. Nodes are then sorted according to their quality and the best one is explored first.

1.6. Implementing Search

Note that your main loop of the search algorithm must be written once (i.e., without knowing the underlying implementation for U or D). A good starting approach is to define a base class for U with all functions (e.g., `push, pop, size`) declared as pure virtual:

```
class UnexploredRoom {                              // class returned from pop()
    uint64                  ID;                     // room ID
    int                     distance;               // distance from source
};

class Ubase {
        ...
    virtual void            push (uint64 roomID,    // pushes the next room into U
                                  float intensity, int distance) = 0;
    virtual UnexploredRoom  pop (void) = 0;         // pops the next room
    virtual int             size (void) = 0;        // checks the size of U
        ...
};
```

The next step is to inherit from `Ubase` four separate classes `Ubreadth, Udepth, Ubest, Uastar`, each using a different algorithm for managing U and overwriting the virtual functions. The first two use respectively an STL queue and stack that contain pairs (room ID, distance). The other two classes use priority queues that store triples (quality, room ID, distance). These should be sorted in descending order of quality. For bFS, the quality is just the light intensity of the room. For A*, it is Q_i given in Section 1.5.

A similar encapsulating technique applies to the discovered set D, except you won't have to inherit anything from it since there is only one type of D in this homework (i.e., an STL set):

```
class Discovered {
       ...
       bool CheckAdd (uint64 roomID);       // add to set if not there already
};
```

Then, instantiating a BFS search would look something like this:

```
Discovered d;
NodeTuple64 *nt = ...               // get initial room pointer from robot

Ubreadth ub;
Search (&ub, &d, nt->room);         // provide U, D, and initial room
```

and that for DFS:

```
Udepth ud;
Search (&ud, &d, nt->room);
```

Note that the same functionality can be accomplished in C using *function pointers* (i.e., without inheritance), but the C++ version above is probably easier to manage and debug.

1.7. Robot Responses

For Parts 2-3, a maximum of *two* read calls are allowed. After up-sizing the buffer, you may want to retain it for the next read operation. After reaching some critical size, the buffer will sta-

bilize and allow receipt of all future messages in one call to ReadFile. The only exception to this algorithm is the extra-credit portion in Part 3 where the monster will aim to make your code exhaust system RAM and crash. The monster does this by dumping huge messages into the pipe. In such cases, the buffer should be kept only if it is less than 5 KB.

1.8. Traces

This run is from planet 2, cave 1 (initial and final exchanges omitted):

```
Starting search using BFS...
1) Visiting 60C2F3EFD35E9B6B, degree 1, discovered 2
2) Visiting 443A9EAC822367E1, degree 8, discovered 8
3) Visiting 9823C27639428949, degree 37, discovered 38
4) Visiting F98B230317993E18, degree 14, discovered 45
5) Visiting 180ECE05813FBC81, degree 8, discovered 47
...
76) Visiting 756653282582D497, degree 2, discovered 99
77) Visiting 9072D980F494CBEC, degree 4, discovered 99
78) Visiting B20157647C43F450, degree 3, discovered 99
79) Visiting A6127CA5C3DB82C3, degree 1, discovered 99
Found exit DE86B6F20F218E9F, 80 steps, distance 4

Starting search using DFS...
1) Visiting 60C2F3EFD35E9B6B, degree 1, discovered 2
2) Visiting 443A9EAC822367E1, degree 8, discovered 8
3) Visiting B8290D5B1751A30D, degree 3, discovered 10
4) Visiting 828340DCF134BEFE, degree 6, discovered 14
5) Visiting A856FA6A866C30E0, degree 1, discovered 14
...
17) Visiting 78E549C629E1641A, degree 4, discovered 38
18) Visiting 8C6AB86D5C741C94, degree 2, discovered 38
19) Visiting 3B44B813AEFFE593, degree 3, discovered 38
Found exit DE86B6F20F218E9F, 20 steps, distance 13

Starting search using bFS...
1) Visiting 60C2F3EFD35E9B6B, degree 1, discovered 2
2) Visiting 443A9EAC822367E1, degree 8, discovered 8
3) Visiting F98B230317993E18, degree 14, discovered 19
...
7) Visiting FC395132A65451BE, degree 8, discovered 39
8) Visiting E8825E744AF37DEB, degree 6, discovered 40
Found exit DE86B6F20F218E9F, 9 steps, distance 6

Starting search using A*...
1) Visiting 60C2F3EFD35E9B6B, degree 1, discovered 2
2) Visiting 443A9EAC822367E1, degree 8, discovered 8
3) Visiting F98B230317993E18, degree 14, discovered 19
...
9) Visiting FC395132A65451BE, degree 8, discovered 63
10) Visiting E8825E744AF37DEB, degree 6, discovered 64
Found exit DE86B6F20F218E9F, 11 steps, distance 5
```

313 Homework 1 Grade Sheet (Part 2)

Name: _____

Function	Points	Break down	Item	Points
Basic code structure	9	2	Correct U for BFS, DFS, bFS, A*	
		1	Correct D (logN overhead)	
		1	Search loop (approach #2 from slides)	
		1	Initial room in both D and U	
		3	PeekNamedPipe, dynamic buffer size	
		1	Check remainder == 0	
Functionality	14	3	BFS correct on P2	
		3	DFS correct on P2	
		3	bFS correct on P2	
		3	A* correct on P2	
		1	Handles caves without exit (900-999)	
		1	Clean termination	
Printouts	2	2	Prints exit room in hex, # of steps, distance from rover	
Misc*				

Total points: _____