WEEK4 Maze

郁思敏 10153901225

codes:

**1.main.cpp**

#include <iostream>

#include "Queue.h"

#include "maze.h"

using namespace std;

int main()

{

int row, col;

Maze mymaze;

Pos in;

Pos out;

cout << "please input the row(<10) of your maze:" << endl;

cin >> row;

cout << "please input the column(<10) of your maze:" << endl;

cin >> col;

mymaze.InitMatrix(row, col);

mymaze.InitMaze();

// mymaze.PrintMatrix();

cout << "please input the row and column of your start point:" << endl

<< "please input the row and column of your end point:" << endl;

cin >> in.cur\_row >> in.cur\_col >> out.cur\_row >> out.cur\_col;

if (mymaze.BFS(in, out))

{

cout << "The nearest way to the end point:" << endl;

mymaze.PrintTheNearstWay(in, out);

// cout << "Detailed:" << endl;

// mymaze.PrintDetailed();

}

else

cout << "There is no way from start to the end" << endl;

return 0;

}

**2.Queue.h**

#ifndef QUEUE\_H\_INCLUDED

#define QUEUE\_H\_INCLUDED

#define MAX\_QUEUE\_SIZE 10

enum Error\_code{success, underflow, overflow};

template <class Queue\_entry>

class CirQueue

{

public:

CirQueue(); //new CirQueue.

bool empty() const; //if queue is empty, return true. Otherwise return false.

bool full() const; //if queue is full, return true. Otherwise return false.

Error\_code append(const Queue\_entry &item); //append new item in the rear of the queue.

Error\_code serve(); //similar to pop in stack. Remove the first item in a queue.

Error\_code retrieve(Queue\_entry &item) const; //similar to top in stack. Get the first item in a queue.

Error\_code retrieve\_and\_serve(Queue\_entry &item); //While get the first item in the queue, remove it.

int size() const; //return count

void clear(); //clear the queue

private:

int count; //the number of the member in a queue.

int front, rear;

Queue\_entry entry[MAX\_QUEUE\_SIZE];

};

#endif // QUEUE\_H\_INCLUDED

**3.Queue.cpp**

#include "Queue.h"

template <class Queue\_entry>

CirQueue<Queue\_entry>::CirQueue() //new CirQueue.

{

count = 0;

front = 0;

rear = MAX\_QUEUE\_SIZE-1;

}

template <class Queue\_entry>

bool CirQueue<Queue\_entry>::empty() const //if queue is empty, return true. Otherwise return false.

{

return count==0;

}

template <class Queue\_entry>

bool CirQueue<Queue\_entry>::full() const //if queue is full, return true. Otherwise return false.

{

return count==MAX\_QUEUE\_SIZE;

}

template <class Queue\_entry>

Error\_code CirQueue<Queue\_entry>::append(const Queue\_entry &item) //append new item in the rear of the queue.

{

if(count >= MAX\_QUEUE\_SIZE)

return overflow;

else

{

count++;

rear = ( (rear+1) == MAX\_QUEUE\_SIZE)?0:(rear+1);

entry[rear] = item;

return success;

}

}

template <class Queue\_entry>

Error\_code CirQueue<Queue\_entry>::retrieve(Queue\_entry &item) const //similar to top in stack. Get the first item in a queue.

{

if (count <= 0)

return underflow;

else

{

item = entry[front];

return success;

}

}

template <class Queue\_entry>

Error\_code CirQueue<Queue\_entry>::serve() //similar to pop in stack. Remove the first item in a queue.

{

if(count <= 0)

return underflow;

else

{

count--;

front = ((front+1)== MAX\_QUEUE\_SIZE)?0:(front+1);

return success;

}

}

template <class Queue\_entry>

Error\_code CirQueue<Queue\_entry>::retrieve\_and\_serve(Queue\_entry &item) //While get the first item in the queue, remove it.

{

if (count <= 0)

return underflow;

else

{

item = entry[front];

count--;

front = ((front+1) == MAX\_QUEUE\_SIZE)?0:(front+1);

return success;

}

}

template <class Queue\_entry>

int CirQueue<Queue\_entry>::size() const //return count

{

return count;

}

template <class Queue\_entry>

void CirQueue<Queue\_entry>::clear() //clear the queue

{

count = 0;

front = 0;

rear = MAX\_QUEUE\_SIZE-1;

}

**4.maze.h**

#ifndef MAZE\_H\_INCLUDED

#define MAZE\_H\_INCLUDED

#include "Queue.h"

#include "Queue.cpp"

#define MAX\_MAZE\_SIZE 10

typedef struct Poistion

{

int cur\_row; //current row

int cur\_col; //current column

int pre\_row; //previous row

int pre\_col; //previous column

} Pos;

class Maze

{

private:

int row, col; //the row and col of your input maze.

Pos position; // a temp position ready to put in queue.

Pos detailed[MAX\_MAZE\_SIZE][MAX\_MAZE\_SIZE]; //the passerby point's detailed info :current(x,y) and previous(x,y)

int matrix[MAX\_MAZE\_SIZE][MAX\_MAZE\_SIZE]; //store the maze you put in.

int noted[MAX\_MAZE\_SIZE][MAX\_MAZE\_SIZE]; //if a position has been added, mark it as 1.

CirQueue<Pos> que;

public:

void InitMatrix(int row\_, int col\_); //Initiate position and noted\_matrix.

void InitMaze(); //Input your maze.

int GetNeighbor(Pos cur, Pos neigh[4]); //search whether one point can go to four directions, if can, store it in the array.

bool BFS(Pos in, Pos out); //return whether you can find a way to solve the maze.

void PrintTheNearstWay(Pos in, Pos out); //search from back to forth, then store it in the array and print it.

void PrintMatrix(); //print the matrix you just put in.

void PrintDetailed(); //print the detailed\_matrix.

};

#endif // MAZE\_H\_INCLUDED

**5.maze.cpp**

#include <iostream>

#include <cstring>

#include "maze.h"

#include "Queue.h"

using namespace std;

void Maze::InitMatrix(int row\_, int col\_) //Initiate position and noted\_matrix.

{

row = row\_;

col = col\_;

memset(noted, 0, sizeof(noted));

}

void Maze::InitMaze() //Input your maze.

{

int i, j;

cout << "please input your maze(1:walls, 0:roads)." << endl;

for (i=0; i<row; i++)

{

for (j=0; j<col; j++)

{

cin >> matrix[i][j];

}

}

}

bool Maze::BFS(Pos in, Pos out) //return whether you can find a way to solve the maze.

{

int i;

bool find = false;

int neighbor;

Pos neigh[4];

que.append(in);

while (!que.empty())

{

que.retrieve(position);

detailed[position.cur\_row][position.cur\_col]=position;

noted[position.cur\_row][position.cur\_col]=1; //note that position has been added.

que.serve();

if (position.cur\_row==out.cur\_row && position.cur\_col==out.cur\_col)

{

find = true;

break;

}

neighbor = GetNeighbor(position, neigh);

for (i=0; i<neighbor; i++)

{

que.append(neigh[i]);

detailed[neigh[i].cur\_row][neigh[i].cur\_col]=neigh[i];

}

}

return find;

}

int Maze::GetNeighbor(Pos cur, Pos neigh[4]) //search whether one point can go to four directions, if can, store it in the array.

{

int cnt=0;

int x = cur.cur\_row;

int y = cur.cur\_col;

//down

if ((x+1)<row && matrix[x+1][y]!=1 && noted[x+1][y]!=1)

{

neigh[cnt].cur\_row = x+1;

neigh[cnt].cur\_col = y;

neigh[cnt].pre\_row = x;

neigh[cnt].pre\_col = y;

cnt++;

}

//right

if ((y+1)<col && matrix[x][y+1]!=1 && noted[x][y+1]!=1)

{

neigh[cnt].cur\_row = x;

neigh[cnt].cur\_col = y+1;

neigh[cnt].pre\_row = x;

neigh[cnt].pre\_col = y;

cnt++;

}

//up

if ((x-1)>=0 && matrix[x-1][y]!=1 && noted[x-1][y]!=1)

{

neigh[cnt].cur\_row = x-1;

neigh[cnt].cur\_col = y;

neigh[cnt].pre\_row = x;

neigh[cnt].pre\_col = y;

cnt++;

}

//left

if ((y-1)>=0 && matrix[x][y-1]!=1 && noted[x][y-1]!=1)

{

neigh[cnt].cur\_row = x;

neigh[cnt].cur\_col = y-1;

neigh[cnt].pre\_row = x;

neigh[cnt].pre\_col = y;

cnt++;

}

return cnt;

}

void Maze::PrintTheNearstWay(Pos in, Pos out) //search from back to forth, then store it in the array and print it.

{

int cnt=0, i;

Pos cur\_pos;

Pos result[row\*col];

position = out;

while (position.cur\_col!=in.cur\_col || position.cur\_row!=in.cur\_row) //when not meet in, store in the array.

{

result[cnt++] = position;

cur\_pos = position;

position.cur\_row = detailed[cur\_pos.cur\_row][cur\_pos.cur\_col].pre\_row;

position.cur\_col = detailed[cur\_pos.cur\_row][cur\_pos.cur\_col].pre\_col;

}

result[cnt]=position;

for (i=cnt; i>=0; i--) //print the array.

{

cout << result[i].cur\_row << " " << result[i].cur\_col << endl;

}

}

void Maze::PrintMatrix() //print the matrix you just put in.

{

int i, j;

cout << "Following is the maze you put in." << endl;

for (i=0; i<row; i++)

{

for (j=0; j<col; j++)

{

cout << matrix[i][j] << ' ';

}

cout << endl;

}

}

void Maze::PrintDetailed() //print the detailed\_matrix.

{

int i, j;

for (i=0; i<row; i++)

{

for (j=0; j<col; j++)

{

cout << i << " " << j << ":"

<< detailed[i][j].cur\_row << " "

<< detailed[i][j].cur\_col << " "

<< detailed[i][j].pre\_row << " "

<< detailed[i][j].pre\_col << " "

<< endl;

}

}

}