BLG336E

Analyis of Algorithm 2

Project1

Name : Lütfü Sirac Küçükarabacıoğlu

ID: 504211521

1.

a.

class node {

public:

int x;

int y;

bool isFull;

bool isVisited;

node \* right;

node \* left;

node \* up;

node \* down;

node(int X,int Y){

x = X;

y = Y;

isFull = false;

right = NULL;

left = NULL;

up = NULL;

down = NULL;

isVisited = false;

};

};

First class is node class which holds nodes of map’s. These nodes also include itself neigbhours, itself coordinates, node is visited before by searching algorithm or not and is the node inlcude a part of ship.

class map{

public:

node \* head;

int numberofFullNode;

map(){

head = NULL;

numberofFullNode = 0;

}

};

Second class is map which is include a head pointer and number of the nodes which include ships.

class ship{

public:

int xbegin;

int xend;

int ybegin;

int yend;

int size;

char axis;

ship(int Xbegin,int Ybegin,int Xend,int Yend){

xbegin = Xbegin;

xend = Xend;

ybegin = Ybegin;

yend = Yend;

if(Xbegin != Xend){

size = abs(Xbegin - Xend)+1; // Added bcs we need size of the ship also in the xend point

axis = 'x';

}

else if(Ybegin != Yend){

size = abs(Ybegin - Yend)+1;

axis = 'y';

}

else{

size = 1;

axis = 'x';

}

}

};

Third class is chip class. This class includes starting coordinates, ending coordinate, size and axis information which is ship parallel.

class player{

public:

string algo\_s;

int algorithm;

int x;

int y;

int sizeofMap;

int numberofShips;

vector<ship\*> ships;

map Map;

int numberofNodesKeptInMemory;

int numberofVisitedNode;

};

Last class is player class. This class a master class which it holds searching algorithm, Player’s map initial point for the searching algorithm, size of the map, number of ships inside the map, also a dynamic array of ships, player’s visited node number during the search algorithm and number of nodes kept in the memory during the search which algorithm uses.

b.

Pseudo Code:

|  |
| --- |
| main:  GET beginingTime  GET pathofPlayer1, pathofPlayer2  CALL readPlayer with pathofPlayer1, pathofPlayer2  CALL buildGraph with P1sizeofMap RETURNING P1MapHead  CALL buildGraph with P2sizeofMap RETURNING P2MapHead  Player1numberofNodesKeptInMemory = P1sizeofMap\*P1sizeofMap  Player2numberofNodesKeptInMemory = P2sizeofMap\*P2sizeofMap  CALL filltheMap with Player1  CALL filltheMap with Player2  CREATE an int called Winner  CALL starttheGame RETURNING Winner  GET endTime  CALL log with Winner, (endTime-beginingTime)  ENDmain  readPlayer:  CREATE int Counter as 0  WHILE getline gives new lines from file  IF counter == 0 and line == "DFS" THEN  Player1's algorithm = DFS  ELSEIF counter == 0 and line == "BFS" THEN  Player1's algorithm = BFS  ELSEIF counter == 1 THEN  SET Player1's initial points  ELSEIF counter == 2 THEN  SET Player1's map size  SET Player1's number of ships  ELSE  CREATE a ship with koordinates  PUSH the ship into Player1's ship vector  ENDIF  SET counter as counter +1  ENDWHILE  SET counter as 0  WHILE getline gives new lines from file  IF counter == 0 and line == "DFS" THEN  Player2's algorithm = DFS  ELSEIF counter == 0 and line == "BFS" THEN  Player2's algorithm = BFS  ELSEIF counter == 1 THEN  SET Player2's initial points  ELSEIF counter == 2 THEN  SET Player2's map size  SET Player2's number of ships  ELSE  CREATE a ship with koordinates  PUSH the ship into Player2's ship vector  ENDIF  SET counter as counter +1  ENDWHILE  ENDreadPlayer  buildGraph:  CREATE node \* 2D array with sizeofmap which is comes as argument  FOR i in sizeofmap  FOR j in sizeofmap  CREATE a new node with address of node[i][j]  IF j != 0  nodeList[i][j-1]'s right = nodeList[i][j];  nodeList[i][j]'s left = nodeList[i][j-1];  ENDIF  IF i != 0  nodeList[i-1][j]'s down = nodeList[i][j];  nodeList[i][j]'s up = nodeList[i-1][j];  ENDIF  ENDFOR  ENDFOR  RETURN node[0][0]  ENDbuildGraph  filltheMap:  CREATE size as numberofships of Player as argument  CREATE temp vector as Player's ships  FOR i in size  placetheShip with Player, and temp[i]  ENDFOR  ENDfilltheMap  placetheShip:  CREATE node \* temp as ship comes as argument  WHILE temp->x < ship->xbegin  temp = temp->down  ENDWHILE  WHILE temp->y < ship->ybegin  temp = temp->right  ENDWHILE  IF ship->axis equals 'x'  FOR i in ship's size  temp's node is assined as shipnode  temp = temp->down  player->Map's number of ship nodes + 1  ENDFOR  ENDIF  IF ship->axis equals 'y'  FOR i in ship's size  temp's node is assined as shipnode  temp = temp->right  player->Map's number of ship nodes + 1  ENDFOR  ENDIF  ENDplacetheShip  starttheGame:  CREATE int Winner as 0  IF Player1's algorithm == "BFS"  CALL BFSGame RETURNING Winner  ENDIF  IF Player1's algorithm == "DFS"  CALL DFSGame RETURNING Winner  ENDIF  RETURN Winner  ENDstarttheGame  BFSGame:  CREATE node \* tempP2 as Player2.Map.Head  CREATE node \* tempP1 as Player1.Map.Head  FOR i in Player1's initial point x  tempP2 = tempP2->down  ENDFOR  FOR i in Player1's initial point x  tempP2 = tempP2->right  ENDFOR  FOR i in Player2's initial point x  tempP1 = tempP1->down  ENDFOR  FOR i in Player2's initial point x  tempP1 = tempP1->right  ENDFOR  CREATE queue<node\*> P1\_q  CREATE queue<node\*> P2\_q  PUSH tempP2 to P1\_q  PUSH tempP1 to P2\_q  Player1's numberofNodesKeptInMemory++  Player2's numberofNodesKeptInMemory++  WHILE P1\_q not empty and P2\_q not empty  CREATE node \* temp = P1\_q.front()  POP P1\_q  Player1's numberofVisitedNode++  SET temp->isVisited as TRUE  IF temp is ship node  SET temp->isFull = false  Player2's Map.numberofFullNode--  ENDIF  IF temp->up is not NULL and temp->left is not visited  PUSH temp->up to P1\_q  Player1's numberofNodesKeptInMemory++  SET temp->up as Visited  ENDIF  IF temp->left is not NULL and temp->left is not visited  PUSH temp->left to P1\_q  Player1's numberofNodesKeptInMemory++  SET temp->left as Visited  ENDIF  IF temp->down is not NULL and temp->left is not visited  PUSH temp->down to P1\_q  Player1's numberofNodesKeptInMemory++  SET temp->down as Visited  ENDIF  IF temp->right is not NULL and temp->left is not visited  PUSH temp->right to P1\_q  Player1's numberofNodesKeptInMemory++  SET temp->right as Visited  ENDIF  IF Player2' Map.numberofFullNode is equals 0  RETURN 1  SET node \* temp = P2\_q.front()  POP P2\_q  Player2's numberofVisitedNode++  SET temp->isVisited as TRUE  IF temp is ship node  SET temp->isFull = false  Player1's Map.numberofFullNode--  ENDIF  IF temp->up is not NULL and temp->left is not visited  PUSH temp->up to P2\_q  Player2's numberofNodesKeptInMemory++  SET temp->up as Visited  ENDIF  IF temp->left is not NULL and temp->left is not visited  PUSH temp->left to P2\_q  Player2's numberofNodesKeptInMemory++  SET temp->left as Visited  ENDIF  IF temp->down is not NULL and temp->left is not visited  PUSH temp->down to P2\_q  Player2's numberofNodesKeptInMemory++  SET temp->down as Visited  ENDIF  IF temp->right is not NULL and temp->left is not visited  PUSH temp->right to P2\_q  Player2's numberofNodesKeptInMemory++  SET temp->right as Visited  ENDIF  IF Player1' Map.numberofFullNode is equals 0  RETURN 2  ENDWHILE  RETURN -1  ENDBFSGame    DFSGame:  CREATE node \* tempP2 as Player2.Map.Head  CREATE node \* tempP1 as Player1.Map.Head  FOR i in Player1's initial point x  tempP2 = tempP2->down  ENDFOR  FOR i in Player1's initial point x  tempP2 = tempP2->right  ENDFOR  FOR i in Player2's initial point x  tempP1 = tempP1->down  ENDFOR  FOR i in Player2's initial point x  tempP1 = tempP1->right  ENDFOR  CREATE int winner as 0  CALL DFSrec with tempP2, tempP1 , 1 , winner  RETURN winner  ENDDFSGame  DFSrec:  IF winner is not 0  RETURN    IF(turn is equals 1)  Player1's numberofVisitedNode++  CREATE boolean empty\_room as FALSE  IF P1node->up is not NULL and P1node->up is not shipnode  SET empty\_room as TRUE  ENDIF  IF P1node->left is not NULL and P1node->left is not shipnode  SET empty\_room as TRUE  ENDIF  IF P1node->down is not NULL and P1node->down is not shipnode  SET empty\_room as TRUE  ENDIF  IF P1node->right is not NULL and P1node->right is not shipnode  SET empty\_room as TRUE  ENDIF  IF P1node is ship node  SET P1node is not ship node  Player2's numberofFullNode--  ENDIF  IF Player2's numberofFullNode is equals 0  SET winner as 1  RETURN  ENDIF  SET P1node->visited as TRUE  IF P1node->up is not NULL and P1node->up is not visited and winner is not 0  CALL DFSrec with P1node->up,P2node,2,winner  Player1's numberofNodesKeptInMemory++  ENDIF  IF P1node->left is not NULL and P1node->left is not visited and winner is not 0  CALL DFSrec with P1node->left,P2node,2,winner  Player1's numberofNodesKeptInMemory++  ENDIF  IF P1node->down is not NULL and P1node->down is not visited and winner is not 0  CALL DFSrec with P1node->down,P2node,2,winner  Player1's numberofNodesKeptInMemory++  ENDIF  IF P1node->right is not NULL and P1node->right is not visited and winner is not 0  CALL DFSrec with P1node->right,P2node,2,winner  Player1's numberofNodesKeptInMemory++  ENDIF  ENDIF  IF(turn is equals 2)  Player2's numberofVisitedNode++  CREATE boolean empty\_room as FALSE  IF P2node->up is not NULL and P2node->up is not shipnode  SET empty\_room as TRUE  ENDIF  IF P2node->left is not NULL and P2node->left is not shipnode  SET empty\_room as TRUE  ENDIF  IF P2node->down is not NULL and P2node->down is not shipnode  SET empty\_room as TRUE  ENDIF  IF P2node->right is not NULL and P2node->right is not shipnode  SET empty\_room as TRUE  ENDIF  IF P2node is ship node  SET P2node is not ship node  Player1's numberofFullNode--  ENDIF  IF Player1's numberofFullNode is equals 0  SET winner as 1  RETURN  ENDIF  SET P2node->visited as TRUE  IF P2node->up is not NULL and P2node->up is not visited and winner is not 0  CALL DFSrec with P1node,P2node->up,2,winner  Player2's numberofNodesKeptInMemory++  ENDIF  IF P2node->left is not NULL and P2node->left is not visited and winner is not 0  CALL DFSrec with P1node,P2node->left,2,winner  Player2's numberofNodesKeptInMemory++  ENDIF  IF P2node->down is not NULL and P2node->down is not visited and winner is not 0  CALL DFSrec with P1node,P2node->down,2,winner  Player2's numberofNodesKeptInMemory++  ENDIF  IF P2node->right is not NULL and P2node->right is not visited and winner is not 0  CALL DFSrec with P1node,P2node->right,2,winner  Player2's numberofNodesKeptInMemory++  ENDIF  ENDIF  ENDDFSrec |

c.

readPlayers functions works with O(n) as file lines.

BuildGraph functions works with O(n\*n) when n is size of one side of the map.

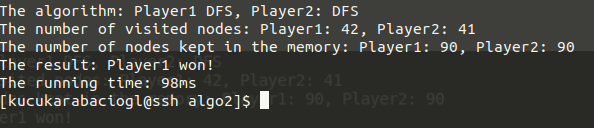
filltheMap and PlacetheShip functions work with O(m\*n) when n is ship count on the map and m is the size of the map.

BFSGame function works with O(n\*n) when n is size of one side of the map.

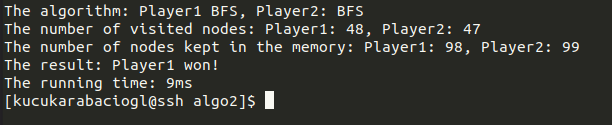
DFSGame function works with O(n\*n) when n is size of one side of the map.

2.

Game2 result:



Game3 result:



As we can see here BFS Algorithm in these circumstances works with less using memory. As can seen on the visited nodes number and nodes kept in the memory number.

But in time BFS algorithm far more faster than DFS algorithm.

3.

If visited node information would not hold algorithm will still works but it will goes again again same nodes which is already visited and processed.

It will increase memory usage and time complexity.

4.

It will increase cost in terms of time and memory usage. Because node number which is has to checked will be increase.