A Mini Project Report

on

KNIGHT'S TRAVAIL

In Subject: Data Structure & Discrete Mathematics

by

Student Name	Roll No	Prn No.
Aditya Gaikwad	271002	22010088
Ganesh	271018	22010331
Harshit Kumar Singh	271021	22011072
Ishwar Zatke	271022	22010600

Department of Artificial Intelligence and Data Science

VIIT

2021-2022

Contents

Sr. No.	Topic		Page No.
Chapter-1	**Introduction**		
	1.1	Introduction	03
	1.2	Requirements	03
	1.3	Design & Problem Statement	04
	1.4	Proposed work	04
Chapter-2	**Methodology**		
	2.1	Breadth First Search	05
	2.2	Methodology	06
	2.3	Program Code	07
	2.3	Outcomes	16
	2.4	Challenges	17
Chapter-3	**Conclusion**		17
	References		17

Introduction:

Knight's travail is a path finding algorithm for knight in a chess game where a knight should reach the target position from a given position in a minimum number of steps.

As we know the path of knight is two steps ahead and a step in its left or right direction otherwise a step ahead and two steps in its left or right direction.

Thus, there a possibility of 8 different position to be the position where knight can jump on.

These creates a graph where the position can be nodes and the target position is thus spotted. GUI adds a flavor to the program which actually shows the path required. Thus the minimum number of steps are obtained.

The algorithm works on Breadth-first search algorithm. The algorithm is for traversing the graph, where you visit from a node to its next node,

But in these approach you doesn't traverse through the visited node again, thus finding the path.

These can be used here where you can't backtrack to the cell which is visited obtained by one step of knight, so you can back track the path and doesn't make it an infinite loop.

Requirements:

C++ 32 bit compiler

Compiler supporting graphics.h header file to implement GUI

Design and problem statement:

- Creating a script for a board game and a night.
- Treating all possible moves of the knight as children in the tree structure.
- Ensuring that any move does not go off the board.
- Choosing a search algorithm for finding the shortest path in this case.
- Applying the appropriate search algorithm to find the best possible move

  from the starting square to the ending square.

Proposed Work:

We would use breadth first search algorithm where we will find the minimum steps taken by knight to reach the target position. We would make a chess board and assigning the cells of the board an ordered pair, making it a coordinate system.
Thus when I want to insert the source position of knight or target position, I will just pass the coordinations.
A visited array would be created to mark the coordinates of the cell where we would store the coordinates of the visited cells on the board by knight.
Then we will have a distance variable where we will store the number of steps taken by knight, counted from source cell of the knight.
We will use a queue to store all next cells on which knight can step on and check the above conditions one by one.
We implemented graphics to display the path for minimum steps and the buttons to choose the position and given the details about the number of moves taken by knight.

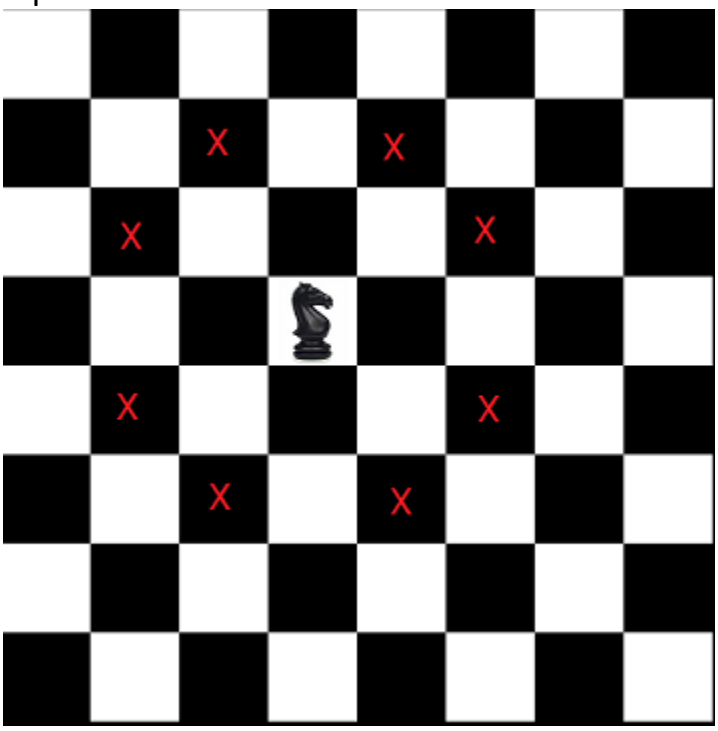
Breadth First Search-

Breadth-first search (**BFS**) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

For example, in a chess endgame a chess engine may build the game tree from the current position by applying all possible moves, and use breadth-first search to find a win position for white. Implicit trees (such as game trees or other problem-solving trees) may be of infinite size; breadth-first search is guaranteed to find a solution node if one exists.

In contrast, (plain) depth-first search, which explores the node branch as far as possible before backtracking and expanding other nodes, may get lost in an infinite branch and never make it to the solution node. Iterative deepening depth-first search avoids the latter drawback at the price of exploring the tree's top parts over and over again. On the other hand, both depth-first algorithms get along without extra memory.

Input as:



The above image shows the possible cells where knight can step on. Thus these cells can be treated as next node of the source node on which night is now on. Thus be traversing like this, we can found the target node.

Metodology-

Algorithm for the following-

1. Create an empty queue and enqueue the source cell having a distance of 0 from the source (itself).
2. Loop till queue is empty:
    1. Dequeue next unvisited node.
    2. If the popped node is the destination node, return its distance.
    3. Otherwise, we mark the current node as visited. For each of eight possible movements for a knight, enqueue each valid movement with +1 distance
3. We can find all the possible locations the knight can move to from the given location by using the queue that stores the relative position of knight movement from any location.

Programme Code:-

```
#include <iostream>
#include <queue>
#include <graphics.h>

using namespace std;

int width, height; //Screen Width, Height
int sx = 5, sy=30; // Screen Position x, y

int state; // Current Page
const int state_main = 0;
const int state_details = 1;
int bsx=-1,bsy=-1; //Choosing x, y

int start_x = -1; //Start
int start_y = -1;
int goal_x = -1; //Goal
int goal_y = -1;

class Cell //Nodes of search tree
{
public:
   Cell *prev;
   int x,y;
   Cell(int x,int y)
   {
      this->x = x;
      this->y = y;
      prev = nullptr;
   }
};

Cell *goal; //Goal cell after searching
```

```
void search() //BFS search
{
   if (goal!=nullptr)
   {
      delete goal;
   }
   queue<Cell*> q;
   int movement[8][2] = {{-1,-2}, {-1,2}, {1,-2}, {1,2}, {-2,-1}, {-2,1}, {2,-1}, {2,1} };
   int visited[8][8] = {0};
   Cell *current = new Cell(start_x,start_y);
   q.push(current);
   visited[start_x][start_y] = 1;
   while (true)
   {
      int cx = current->x;
      int cy = current->y;
      if (cx==goal_x && cy==goal_y)
      {
         goal = current;
         break;
      }

      for (int i=0;i<8;i++)
      {
         int x = cx+movement[i][0];
         int y = cy+movement[i][1];
         if (x>-1 && x<8 && y>-1 && y<8 && !visited[x][y])
         {
            visited[x][y] = 1;
            Cell *c = new Cell(x,y);
            c->prev = current;
            q.push(c);
         }
      }
      q.pop();
      current = q.front();
   }
```

```
}

void calibrate() //Calibrates the screen for mouse input
{
   int cx=0,cy=0;
   POINT m;
   for(int i=0;i<3;i++) //Calibration row
   {
      for (int j=0;j<3;j++) //Calibration col
      {
         setfillstyle(SOLID_FILL,CYAN);
         bar(0,0,width,height);
         setfillstyle(SOLID_FILL,WHITE);
         outtextxy(10,10,(char*)"Calibration");
         bar(100+100*j,100+100*i,105+100*j,105+100*i);
         while (!GetAsyncKeyState(VK_LBUTTON));
         GetCursorPos(&m);
         cx += m.x - (100+100*j);
         cy += m.y - (100+100*i);
         while (GetAsyncKeyState(VK_LBUTTON));
      }
   }
   sx = cx/9;
   sy = cy/9;
   setfillstyle(SOLID_FILL,CYAN);
}

string getChessLocation(int x,int y) // Convert x,y location to chess location name
{
   string s = "";
   s+=(char)('a'+x);
   s+=(8-y)+'0';
   return s;
}

void drawDetails() //Draw the details screen
{
```

```
    setbkcolor(CYAN);
    setfillstyle(SOLID_FILL,CYAN);
    bar(0,0,width,height);
    setcolor(BLUE);
    if (goal!=nullptr)
    {
        int depth = 0;
        Cell *c = goal;
        string cs = "";
        while (c->prev!=nullptr)
        {
            cs = getChessLocation(c->x,c->y) + cs;
            depth++;
            c = c->prev;
        }
        stringstream ss; //To concatenate to string
        ss<<"Moves: "<<depth;
        outtextxy(100,100,(char*)ss.str().c_str());
        outtextxy(100,150,(char*)cs.c_str());
    }
    setfillstyle(SOLID_FILL,WHITE);
    setbkcolor(WHITE);
    setcolor(BLACK);
    bar(10,10,260,60);
    outtextxy(20,20,(char*)"Board");
}

void drawBoard() //Draws the board
{
    for (int i=0;i<8;i++) //Board row
    {
        for (int j=0;j<8;j++) //Board col
        {
            if (j==bsx && i==bsy) //If selection mouse hower
            {
                setfillstyle(SOLID_FILL,YELLOW);
                bar(100+j*50,100+i*50,100+j*50+50,100+i*50+50);
```

```
        }
        else if (j==start_x && i==start_y) //If start
        {
           if ((start_x+start_y)%2) //Black square
           {
              readimagefile("knight
black.bmp",100+50*start_x,100+50*start_y,100+50*start_x+50,100+50*start_y+
50); //Draw the black image
           }
           else //White square
           {
              readimagefile("knight
white.bmp",100+50*start_x,100+50*start_y,100+50*start_x+50,100+50*start_y+
50); //Draw the white image
           }
        }
        else if ((i+j)%2) //Odd is for black square
        {
           setfillstyle(SOLID_FILL,BLACK);
           bar(100+j*50,100+i*50,100+j*50+50,100+i*50+50);
           if (j==goal_x && i==goal_y) //Goal
           {
              setcolor(RED);
              line(100+j*50,100+i*50,100+j*50+50,100+i*50+50);
              line(100+j*50,100+i*50+50,100+j*50+50,100+i*50);
           }
        }
        else //Even is for white square
        {
           setfillstyle(SOLID_FILL,WHITE);
           bar(100+j*50,100+i*50,100+j*50+50,100+i*50+50);
           if (j==goal_x && i==goal_y) //Goal
           {
              setcolor(RED);
              line(100+j*50,100+i*50,100+j*50+50,100+i*50+50);
              line(100+j*50,100+i*50+50,100+j*50+50,100+i*50);
           }
```

```
        }
      }
    }

    setcolor(YELLOW);
    setbkcolor(CYAN);
    for (int i=0;i<8;i++) //X position name (abcdefgh)
    {
      char c[] = {getChessLocation(i,0)[0],0};
      outtextxy(100+i*50+10,60,c);
      outtextxy(100+i*50+10,510,c);
    }
    for (int i=0;i<8;i++) //Y position name (12345678)
    {
      char c[] = {getChessLocation(0,i)[1],0};
      outtextxy(60,100+i*50+10,c);
      outtextxy(510,100+i*50+10,c);
    }


    if (goal!=nullptr) //Draw the path
    {
      setcolor(YELLOW);
      setfillstyle(SOLID_FILL,RED);
      Cell *cur = goal;
      Cell *prev = goal->prev;
      while (prev!=nullptr)
      {
        int x1 = cur->x;
        int y1 = cur->y;
        int x2 = prev->x;
        int y2 = prev->y;
        line(100+50*x1+25,100+50*y1+25,100+50*x2+25,100+50*y2+25); //Line
for path
        //circle(100+50*x1+25,100+50*y1+25,10);
        fillellipse(100+50*x1+25,100+50*y1+25,10,10); //Circle at the cell
        cur = prev;
```

```
            prev = cur->prev;
        }
    }
}

void drawChoose() //Choosing position on board
{
    while (!GetAsyncKeyState(VK_LBUTTON)) //While not clicked
    {
        POINT m;
        GetCursorPos(&m);
        int mx = m.x-sx;
        int my = m.y-sy;
        if (mx>100 && mx<500 && my>100 && my<500) //Clicked on board
        {
            bsx = (mx-100)/50;
            bsy = (my-100)/50;
        }
        else //Clicked outside
        {
            bsx = -1;
            bsy = -1;
        }
        drawBoard();
        delay(10);
    }
}


void drawMain() //Draw main screen
{
    setbkcolor(CYAN);
    setfillstyle(SOLID_FILL,CYAN);
    bar(0,0,width,height);
    drawBoard();
    setfillstyle(SOLID_FILL,WHITE);
    setbkcolor(WHITE);
```

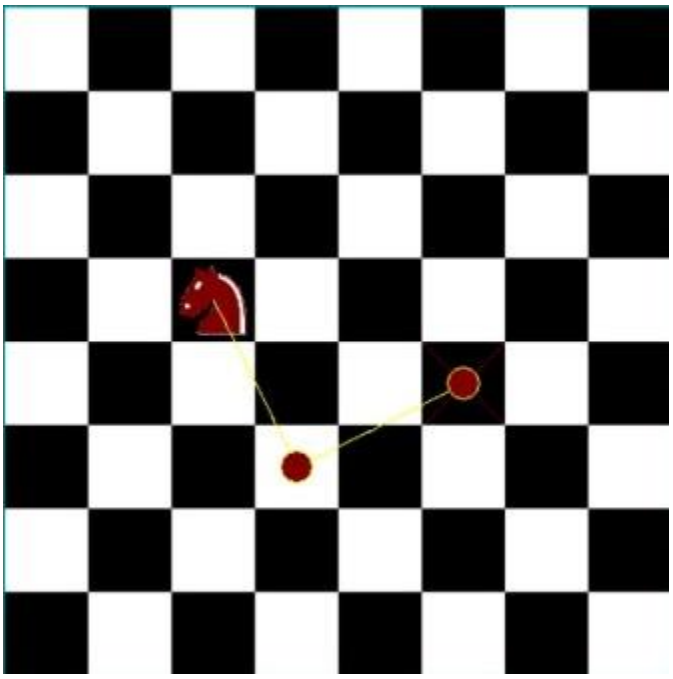
```
   setcolor(BLACK);
   bar(550,100,950,150); //Choose Start button
   outtextxy(560,110,(char*)"Choose Start");
   bar(550,200,950,250); //Choose Goal button
   outtextxy(560,210,(char*)"Choose Goal");
   bar(10,10,260,60); //Details button
   outtextxy(20,20,(char*)"Details");
}

void handleEvents() //Handles mouse and keyboard events
{
   POINT m; //For mouse
   GetCursorPos(&m); //Gets mouse position
   int mx = m.x-sx;
   int my = m.y-sy;
   if (state==state_main) //Main screen
   {
      if (GetAsyncKeyState(VK_LBUTTON)) //Left click
      {
         if (mx>550 && mx<950 && my>100 && my<150) //Choose Start button
         {
            //cout<<"Choose Start"<<endl;
            setfillstyle(SOLID_FILL,WHITE);
            setbkcolor(WHITE);
            setcolor(RED);
            bar(550,100,950,150);
            outtextxy(560,110,(char*)"Choose Start"); //Redraw with red text
            while (GetAsyncKeyState(VK_LBUTTON));
            drawChoose();
            if (bsx!=-1 && bsy!=-1) //Start selected
            {
               start_x = bsx;
               start_y = bsy;
               bsx = -1;
               bsy = -1;
               if (start_x>=0 && start_y>=0 && goal_x>=0 && goal_y>=0) //Both start
and goal available
```

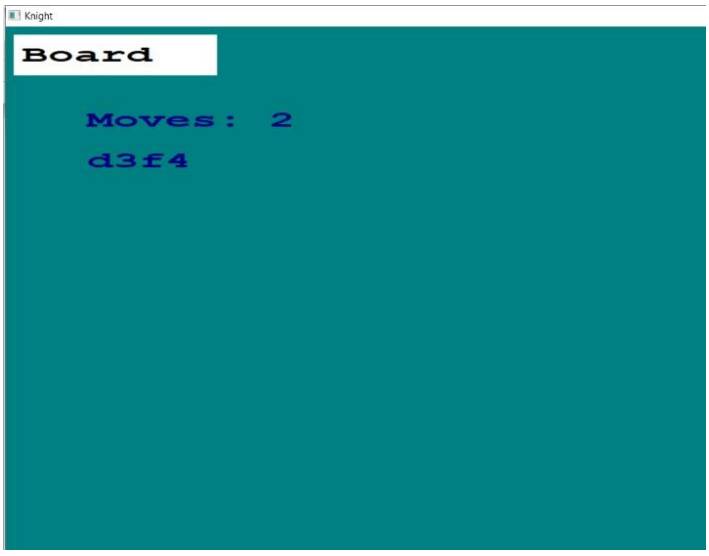
```
            {
               search();
            }
         }
         drawMain();
      }
      else if (mx>550 && mx<950 && my>200 && my<250) //Choose Goal
button
      {
         setfillstyle(SOLID_FILL,WHITE);
         setbkcolor(WHITE);
         setcolor(RED);
         bar(550,200,950,250);
         outtextxy(560,210,(char*)"Choose Goal"); //Redraw with red text
         while (GetAsyncKeyState(VK_LBUTTON));
         drawChoose();
         if (bsx!=-1 && bsy!=-1) //Goal selected
         {
            goal_x = bsx;
            goal_y = bsy;
            bsx = -1;
            bsy = -1;
            if (start_x>=0 && start_y>=0 && goal_x>=0 && goal_y>=0) //Both start
and goal available
            {
               search();
            }
         }
         drawMain();
      }
      else if (mx>10 && mx<260 && my>10 && my<60) //Details button
      {
         while (GetAsyncKeyState(VK_LBUTTON));
         state = state_details;
         drawDetails();
      }
   }
```

```
    }
    else //Details screen
    {
        if (GetAsyncKeyState(VK_LBUTTON)) //Board button
        {
            if (mx>10 && mx<260 && my>10 && my<60)
            {
                while (GetAsyncKeyState(VK_LBUTTON));
                state = state_main;
                drawMain();
            }
        }
    }

    if (GetAsyncKeyState('C')) //Calibration
    {
        while(GetAsyncKeyState('C'));
        calibrate();
        state = state_main;
        drawMain();
    }
}

int main()
{
    width = GetSystemMetrics(SM_CXSCREEN); //Screen width
    height = GetSystemMetrics(SM_CYSCREEN); //Screen height
    initwindow(width,height,"Knight"); //Window created
    setfillstyle(SOLID_FILL,CYAN);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,4); //Text font
    setbkcolor(CYAN);
    state = state_main;
    drawMain();
    while (true)
    {
        handleEvents(); //Handle input
        delay(10);
```

```
    }
    closegraph(); //Close window (never called)
    return 0;
}
```

Outcome:





Conclusion:

- Thus we learned about graphs, trees and its application in data structure
- We learned about breadth first search technique and its implementation in real life problems.
- We learned about 2D arrays and queue data structure.
- We learned about implementation of graphics for these problem statement.

Challenges Faced:

- ❑ Understanding the implementation of the problem statement using BFS algorithm was a bit tricky at first.
- ❑ Making GUI of chess board using graphics.h file was a challenge.

References:

- https://en.wikipedia.org/wiki/Breadth-first_search
- https://www.programmingsimplified.com/c/graphics.h