Universidad Católica San Pablo Departamento de Ciencia de la Computación

Practica Backtracking

Integrantes:
Becerra Sipiran, Cledy Elizabeth
Oviedo Sivincha, Massiel
Villanueva Borda, Harold Alejandro

Universidad Católica San Pablo, Arequipa – Perú

CODE HORSE WALK

```
#include <iostream>
#include <chrono>

/*
int move_x[8] = {1, 1, 2, 2, -1, -1, -2, -2};
int move_y[8] = {2, -2, 1, -1, 2, -2, 1, -1}; // */

/*int move_x[8] ={-2, -2, -1, 1, 2, 2, 1, -1};
int move_y[8] = {-1, 1, 2, 2, 1, -1, -2, -2}; // */

/*int move_x[8] ={2, 1 , -1, -2, -2, -1, 1, 2};
int move_y[8] = {1, 2, 2, 1, -1, -2, -2, -1};// */


int move_x[8] = {1,2,2,1,-1,-2,-2,-1};
int move_y[8] = {2,1,-1,-2,-2,-1,1,2};// */

/*int move_x[8] ={2, 2 , -2, -2, 1, 1, -1, -1};
int move_y[8] = {1, -1, 1, -1, 2, -2, 2, -2};// */

void print(int n, int **matrix){
    for(int i = 0; i < n; ++i) {
        for(int j = 0; j < n; ++j) {
            std::cout << *(*(matrix + i) + j ) << "\t";
        }
        std::cout<<std::endl;
    }
}

bool check(int x, int y, int **matrix, int n){
    //Verífica que no se sobrepase los limites de la matriz y que la
posición a verificar no se haya tomado previamente.
```

```
    if(x < 0 or x > n - 1 or y < 0 or y > n - 1 or *(*(matrix + x) + y ) >
0)
        return false;
    return true;
}

int getDegree(int **matrix, int n, int x, int y){
    //Verifica los caminos posibles, en caso de no tener caminos posibles
se retorna -1
    int res = -1;
    for(int i = 0; i < 8; ++i)
        if(check(x + move_x[i], y + move_y[i], matrix, n))
            res += 1;
    return res;
}

bool solve(int n, int **matrix, int curr_x, int curr_y, int pos){
    int id1 = -1;
    int min_degree1 = n + 1;
    //matrix[curr_x][curr_y] = pos;
    *(*(matrix + curr_x) + curr_y ) = pos;
    if(pos == n*n){
        print(n, matrix); //Encuentra la solución, imprime el array
        return true;
    }
    for(int i = 0; i < 8; ++i){
        int new_x = curr_x + move_x[i];
        int new_y = curr_y + move_y[i];
        if(check(new_x, new_y, matrix, n)){
            int degree = getDegree(matrix, n, new_x, new_y);
            if(degree <= min_degree1){
                min_degree1 = degree;
                id1 = i;
            }
        }
    }

    if(id1 == -1){
        std::cout << "no hay solucion " << std::endl;
        return false;
    }
    int n_x = curr_x + move_x[id1];
    int n_y = curr_y + move_y[id1];

    solve(n, matrix, n_x, n_y, pos + 1);
    return false;
}
```

```
void horse_9(int n, int x, int y){
    int pos_x = x;
    int pos_y = y;

    int** matrix = new int*[n];

    for(int i = 0; i < n; ++i) {
        *(matrix + i) = new int[n];
        for(int j = 0; j < n; ++j) {
            *(*(matrix + i) + j ) = -1;
        }
    }
    //print(n, matrix);
    int pos = 1; //Contador de las posiciones tomadas, se va reemplazando
en la matriz.
    auto start = std::chrono::system_clock::now();
    solve(n, matrix, pos_x, pos_y, pos);
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - start;
    //std::time_t end_time = std::chrono::system_clock::to_time_t(end);
    std::cout << "tiempo " << elapsed_seconds.count() << std::endl;
}

int main(){
    horse_9(20, 0, 0);
}
```

CODE N QUEEN

```
#include <iostream>
#include <chrono>

void print(int **matrix, int n){
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            if(*(*(matrix + j) + i ) == 0)
                std::cout << "." <<  "\t";
            else
                std::cout << "x" <<  "\t";
        }
```

```
        std::cout << std::endl;
    }
}

bool check(int **matrix, int r, int i, int N){
    for(int j = 0; j < N; ++j)
        if(*(*(matrix + j) + i ) == 1)//Barre verticales desde la posición
[j][i] -> i = estático.
            return false;

    for(int j = 0; j < N; ++j)
        if(*(*(matrix + r) + j ) == 1)//Barre horizontales desde la
posición [r][j] -> r = estático.
            return false;

    for(int j = 0; j < N; ++j)//Barre diagonales
        for(int k = 0; k < N; ++k)
            if (((*(*(matrix + j) + k )) == 1) and ((j + k == r + i) or (j
- k == r - i)))
                return false;

    return true;
}

bool solve(int **matrix, int r, int x, int y, int N ){
    if(r == N and *(*(matrix + x) + y ) == 1){
        print(matrix, N); // llega a su fin, se imprime la solución
        return false;
    }
    for(int i = 0; i < N; ++i){
        if(check(matrix, r, i, N) or (x == r and y == i)){
            //La segunda condición solo se cumple cuando se verifica la
posición inicial dentro del bucle.
            //r avanza en cuanto a filas disponibles, i avanza en
columnas.
            matrix[r][i] = 1;
            bool l = solve(matrix, r + 1, x, y, N);
            if (l == 0)
                return false;
            if (x != r and y != i)
                *(*(matrix + r) + i ) = 0;
        }
    }
    return true;
}

void nqueens_9(int n, int x, int y ) {
```

```
    int** matrix = new int*[n];

    for(int i = 0; i < n; ++i) {
        *(matrix + i) = new int[n];

        for(int j = 0; j < n; ++j) {
            *(*(matrix + i) + j ) = 0;
        }
    }

    *(*(matrix + x) + y ) = 1;

    auto start = std::chrono::system_clock::now();
    solve(matrix, 0, x, y, n);
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - start;
    //std::time_t end_time = std::chrono::system_clock::to_time_t(end);
    std::cout << "tiempo " << elapsed_seconds.count() << std::endl;
}

int main() {
    nqueens_9(4, 2, 0);
}
```

CODE MAZE

```
#include <iostream>
#include <vector>
#include <fstream>
#include <utility>
#include <string>
#include <iomanip>
#include <chrono>

//Función para leer un laberinto de un archivo, si el siguiente
std::string es "x" da su posición
void readMaze(std::vector<std::vector<std::string>>& maze, int& y, int& x,
const std::string& fileName){
    std::ifstream file;
    file.open(fileName);
    if(!file.is_open()){
        std::cout << "No se pudo abrir el archivo" << std::endl;
```

```
        return;
    }
    char c;
    int y_ = 0, x_ = 0;
    maze.resize(1);
    while(file.get(c)){
        std::cout << c;
        if(c == 'x'){
            y = y_;
            x = x_;
        }
        if(c == '\n'){
            std::vector<std::string> row;
            maze.push_back(row);
            ++y_;
            x_ = 0;
        }
        else{
            maze[maze.size()-1].push_back(std::string(1, c));
            ++x_;
        }
    }
    std::cout << std::endl;
    file.close();
}

void print_maze(std::vector<std::vector<std::string>>& maze){
    for(auto & y : maze){
        for(auto & x : y){
            std::cout << std::setw(6) << x;
        }
        std::cout << std::endl;
    }
}

std::vector<std::pair<int, int>>
get_neighbours(std::vector<std::vector<std::string>>& maze, int y, int x){
    std::vector<std::pair<int, int>> neighbours;
    if(x < maze[y].size()-1 && (maze[y][x+1] == "." || maze[y][x+1] ==
"y")){
        neighbours.emplace_back(y, x+1);
    }
    if(x > 0 && (maze[y][x-1] == "." || maze[y][x-1] == "y")){
        neighbours.emplace_back(y, x-1);
    }
    if(y < maze.size()-1 && (maze[y+1][x] == "." || maze[y+1][x] == "y")){
        neighbours.emplace_back(y+1, x);
```

```
    }
    if(y > 0 && (maze[y-1][x] == "." || maze[y-1][x] == "y")){
        neighbours.emplace_back(y-1, x);
    }
    return neighbours;
}

bool solve(std::vector<std::vector<std::string>>& maze, int y, int x, int&
cont){
    if(maze[y][x] == "y")
        return true;
    if(maze[y][x] != "x" && maze[y][x] != "y"){
        maze[y][x] = std::to_string(cont);
        ++cont;
    }
    std::vector<std::pair<int, int>> neighbours = get_neighbours(maze, y,
x);
    for(auto & neighbour : neighbours){
        if(solve(maze, neighbour.first, neighbour.second, cont)){
            return true;
        }
        if(maze[y][x] != "x"){
            maze[y][x] = std::to_string(cont);
            ++cont;
        }
    }
    return false;
}

void maze_9(const std::string& fileName){
    std::vector<std::vector<std::string>> maze;
    int y, x, cont = 1;
    readMaze(maze, y, x, fileName);
    std::cout << "y: " << y << " x: " << x << std::endl;
    //print_maze(maze);
    auto start = std::chrono::system_clock::now();
    std::cout << "Solucion: " << ((solve(maze, y, x, cont)) ? "Si" : "No")
<< std::endl;
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - start;
    //std::time_t end_time = std::chrono::system_clock::to_time_t(end);
    std::cout << "tiempo " << elapsed_seconds.count() << std::endl;
    print_maze(maze);
}

int main(){
    maze_9("C:/Users/win 10/Documents/CLION/pruebas/maze1.txt");
```

```
}
```