GENERATING AND SOLVING OF THE MAZE BY USING KRUSKAL'S AND FLOYD'S ALGORITHM

Suzana Marković
Academy of Business Applied Studies Belgrade, suzana.markovic@bpa.edu.rs

Jasmina Novakovic
Academy of Business Applied Studies Belgrade

Anja Marković
Faculty of Electrical Engineering, Belgrade

Abstract

Graphs are used in computer systems as models for presenting certain objects and mutual relationships among them. The simplest data structure, which is used for presenting graphs in computer programs, is the adjacency matrix. This matrix enables easy manipulation of nodes and edges so it is used in the realization of the maze. A maze implies a set of connected passages in which movement is possible in a space filled with obstacles (walls). Following conditions should be determined when generating the maze: set of maze's nodes, initial state - entrance to maze and end state - exit from maze. Kruskal algorithm was used to generate the maze. In solving the maze, it is necessary to determine a set of possible actions - the path choice (the next node of the maze) at each step and the transition determined by the relations between the maze's nodes. Finding the best path to exit the maze was implemented using the Floyd algorithm.

Keywords: Graph, spanning tree, adjacency matrix, Kruskal's algorithm, Floyd's algorithm, maze.

INTRODUCTION

Algorithm is one of the basic building blocks for the implementation of program systems [1]. It is a precisely defined procedure realized by a final set of commands, which a computer should execute for solving a problem. Algorithm analysis determines the use of computer resources. The tendency is to minimize the most expensive resources – execution time and memory requirements.

Graphs are used in computer systems as models for presenting certain objects and mutual relationships among them. They can be used in numerous ways: maps of cities and roads among them, maps of city transportation with marked stops, the connection of computers in a computer network, pages of a certain website connected by hyperlinks, etc. Graphs are, in mathematical terms, binary relations which are visually presented by points (nodes) and lines (edges), which are used to connect those points. More precisely, a graph is a couple of sets (V, E) –> G = (V, E), where V is a final blank set, the elements of which are nodes (vertices) (also called nodes or points), and set E presents binary relations of elements of set V, the elements of which are called (edges) (also called links or lines).

One of the basic operations in the graph is the movement through a series of nodes connected with edges. The undirected simple graph is connected if there is a path between every two nodes. Otherwise, it consists of more than one connected component. A tree is a special case of a graph, i.e. the simplest type of connected graphs (by removing any line from the tree, it becomes disconnected). A graph in which all components are connected stand for free trees themselves is called a forest.

G is an undirected, connected graph. Spanning tree of the graph G is any tree which contains all the nodes and only a certain number of edges of graph G, is required for connecting all nodes so that there is no loop (a loop is simple path which starts and ends in one node).

In the weighted graph, all edges have their weight which is expressed numerically. If (V,T) is a connected subgraph of the weighted graph G = (V, E), whose total weight of edges is the least, then (V,T) is a free tree.

The simplest data structure which is used for presenting graphs in computer programs, and which enables easy manipulation of nodes and edges is the adjacency matrix – a two-dimensional series, in which the numbers of rows and columns are equal to the number of graph nodes. Graph G has n nodes, which are numerated from 1 to n [2]. This graph can be presented by a square matrix M, of dimensions nxn, the members of which m[i,j] have binary values m[i,j]=1, if edge (i,j) belongs to a set of edges, and m[i,j]=0 if the edge does not belong to set of edges of graph G.

The rest of this paper presents the way of generating and solving the maze by using the graph theory and algorithms, which solve the problems of generating trees, and passing through it by choosing the most favorable paths.

INTRODUCTION TO THE MAZES

A maze implies a set of connected passages in which movement is possible [3] in a space filled with obstacles (walls). Every passage has one or more fields and two ends. One field is the entrances, and another is the exit of the maze. The entrance, exit, ends of the passage, as well as the fields, which are mutual for both passages made up the nodes of the maze. The aim is to find the way from the entrance to the exit through maze nodes. In generating the maze, it is necessary to determine the following states: set of maze nodes, the initial state – entrance in the maze and the final state – exit from the maze. In solving the maze, it is necessary to determine the set of possible actions – the choice of the path (the next maze node) in each step and the transitions, which are determined by links between the maze nodes.

Every classical rectangular maze can be presented by a graph so that the maze nodes represent graph nodes, and edges comprise the paths (passes) through the selected nodes. Mazes do not have unconnected parts because there is a unique path between every two points, so the appropriate graph which fits them is actually a tree which can be considered as a maze outline scheme.

Entertainment and brain training may be considered as primary goals of maze application [4]. Logic mazes are special mazes with specifically defined rules altering the usual way of maze solving. Also, there are life size mazes as tourist attractions and finally application of mazes can be found in the video game and film industry.

According to Walter D. Pullen [4], there are seven categories mazes can be classified by:

- dimension
- hyper-dimension
- topology
- tessellation
- routing
- texture
- focus

According to *dimension*, there are two-dimensional mazes (rectangle) and three-dimensional (cube). This paper focuses on two-dimensional mazes.

Hyper-dimension category refers to the dimension of the object the solver moves through the maze. The *topology* category divides mazes according to the space they exist in (normal and plainair). The *tessellation* categorization divides mazes into several groups according to the shape of their basic units they are composed of – cells (orthogonal, omega, theta, crack, fractal). The *routing* category classifies mazes into several subcategories according to the properties of their passage (path) system (perfect, braid, unicursal, sparseness). Identifying a maze's *texture* can be done by observing the maze or by expressing it using mathematical evaluation (symmetry, river).

Walter D. Pullen uses *focus* category to distinguish various methods of maze creation. There are two basic algorithmic ways: wall adders (focusing on wall positioning) and passage carvers (concentrate on paths and cell positioning).

There are two basic ways of maze generation: algorithmic and non-algorithmic. Algorithmic methods create mazes according to a predefined step order. This paper puts focus on graph based algorithms. Graph based maze generation algorithms create mazes by building a spanning tree.

KRUSKAL'S AND FLOYD'S ALGORITHM

The minimum spanning tree for the given weighted graph G = (V, E) finds a subset of

edges T, so that subgraph (V,T) is connected and the total weight of edges in T is minimum. It is called minimum because it has the lowest price, and spanning because it encompasses all nodes of the given graph G.

A graph can have numerous spanning trees, the structures of which are completely different. However, the aim is to find a spanning tree with the lowest possible price. There are numerous algorithms which solve that successfully. The property of the graph division enables a gradual selection of its edges which belong to the minimum spanning tree. One of the representatives of such an approach is Kruskal's algorithm [5]. This algorithm is applied in real life in numerous situations: in computer networks, it finds the shortest path between two nodes (computers) taking into consideration the type of the connection (telephone line, optic cable, etc.); in transport infrastructure, it finds the shortest path between two nodes (city, street) considering the type of the path, etc.

The application of Kruskal's algorithm for the purposes of creating a made will be explained in more detail.

Aside from that, one of the basic algorithm problems related to graphs is finding the shortest paths between its nodes. The problem consists of the fact that for the given node s in the weighted graph G to determine the shortest paths from that node to all other nodes in the graph, whereby the complexity of finding the shortest path from the given node s do another given node does not become more simple in relation to the previous one. In other words, the problem of the shortest paths between all nodes comes down to finding the shortest paths between every two nodes of the weighted graph. One of the representatives of such an approach which is used in this paper is Floyd's algorithm [2]. This algorithm finds its application in determining the shortest distance between all nodes (cities) in a certain transport network.

The application of Floyd's algorithm for the purposes of solving the maze is presented in more detail as follows.

GENERATING THE MAZE

A graph as a data structure is used for presenting the maze in this paper that is its matrix representation, presented in the form of the adjacency matrix. If between nodes iij there is a branch, then m[i,j]=1; otherwise, m[i,j]=INT_MAX, except on the main diagonal, where m[i,j]=0 because those elements present edges from the nodes to themselves.

Maze realization is performed in C programming language. The following options are available from the defined menu:

1. Creation of the maze
1.1. Creation of the maze by entering coordinates of the passage
1.2. Generating of the maze
2. Creation of the passage
3. Creation of walls
4. Adding nodes
5. Deleting nodes
6. Destroying mazes
7. Solving mazes
7.1. Path from entrance to exit
7.2. Path between certain coordinates
8. Printing of maze
8.1. Maze printing on the standard output
8.2. Printing maze to a file
9. Program abortion

For *generating the maze*, two options are used. The first option enables the creation of the maze by entering coordinates of the passage through a standard entrance. The second option is more significant for the purposes of this paper and it enables the generating of the maze by using Kruskal's algorithm.

In Kruskal's algorithm, it is started from the forest of unconnected nodes of a graph, of which every node presents a specially connected component, without any edge, and then an edge is added one by one until the forest becomes a tree [6]. For the given tree, it is guaranteed that it is a minimum spanning tree. Then, an edge is chosen by applying the function rand(), by choosing four pseudo-coincidental numbers which signify coordinates of the two nodes in the adjacency matrix. If both nodes are in the same connected component, an edge is rejected so that the cycle disappears from the graph, and in contrast, an edge is joined to the set of graph edges. An algorithm ends when n-1 edge is joined because then all nodes are in one

connected component. At the end of this option, through a standard entrance to the maze, coordinates of entrance are entered, as well as all the coordinates of all exits from the maze (if there are several).

For the realization of this algorithm, sets must be used. An auxiliary node structure is used, realized by the following code (1):

```
typedef struct node {
int id;
struct node *parent;
struct node *left;
struct node *right;
} Node;                                    (1)
```

Through the given structure, every graph node in the beginning is presented in the form of the structure node. When an edge between two graph nodes is found, those two nodes are entered into the same connected component, that is in the same disjunctive set, so that certain nodes (with the chosen Ids) are connected by an auxiliary node which presents a new root of the tree in which the given two nodes can be found. The algorithm ends when all initiated nodes are entered into one tree. The realization of an algorithm is presented by the following code (2).

```
pair1 = parent(mat[i][j]); // the first selected
node
pair2 = parent(mat[x][y]); // the second
selected node
if (pair1->id == pair2->id) continue; // in the
same connected component
else {
pair = malloc(sizeof(Node));
pair->parent = NULL;
pair->id = n++; cnt++; //cnt is a node
counter; n is id of a node
pair->left = pair1;
pair->right = pair2;
pair1->parent = pair;
pair2->parent = pair;
int no1 = (x*lav->column) + y;
lav->mat[number][no1] = lav-
>mat[no1][number] = 1; // branch becomes
valid
}                                          (2)
```

According to the menu defined on the previous page, what follows are options for additional adjustment of the maze, the addition and deleting elements, as well as for destroying the maze. After creating the maze, the menu options (2. Creation of the passage and 3. Creation of the walls) serve for adding the passage and for adding the walls in the maze. That is achieved through simple positioning of elements of the adjacency matrix, determined by the entered coordinates, on the value of 1 for adding a passage, that is the value of 0 when creating the wall.

The next two options of the defined menu provide the opportunity of adding and deleting nodes. When adding nodes (option 4) in the graph, it is necessary to allocate the new types and columns in the adjacency matrix, considering that it must be squared, whereas when deleting nodes (option 5) the column in which the node is located is being deleted. The created maze can be destroyed by choosing option 6 from the defined menu. Destroying the maze (option 6) is performed by deallocating the adjacency matrix and other required structures created when generating the maze, like the series of exits from the maze is dynamically allocated.

SOLVING THE MAZE

In order to find the exit from the maze, two options are used. One option is to find a path to one of the existing exits from the maze. Another option is to find the path between two random nodes in the graph. In both cases, Floyd's algorithm is used [2]. Floyd's algorithm is used for finding the shortest distance between all sets of nodes in the graph and reconstruction of the discovered shortest paths. For the realization of this algorithm, it is necessary to allocate a new matrix of the predecessor T, which has the same dimension as the adjacency matrix. Element t[i,j] identifies the node which is a direct predecessor of the node j on the shortest path from node i. If there is no path between nodes i and j, then t[i,j]=0. A copy is made from the adjacency matrix in the form of the matrix D which contains currently the shortest distance between nodes. The algorithm is presented as follows (3).

This algorithm is based on the relaxation principle, so that in each moment the highest limit is maintained for evaluating the length of

the shortest path, which is initially equal to the weight of the direct edge or its value is INT_MAX if there is no edge. In the following iterations, it is checked whether the current evaluation can be shortened through another node. The reconstruction of the shortest path between two random nodes i and j can be performed by using the matrix of the predecessor T.

```
for (k = 0; k < n; k++)
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
if (d[i][j] > d[i][k] + d[k][j] && d[i][k] >= 0
&& d[k][j] >= 0 && d[i][k] != INT_MAX
&& d[k][j] != INT_MAX) {
t[i][j] = t[k][j];
d[i][j] = 1;
}                                          (3)
```

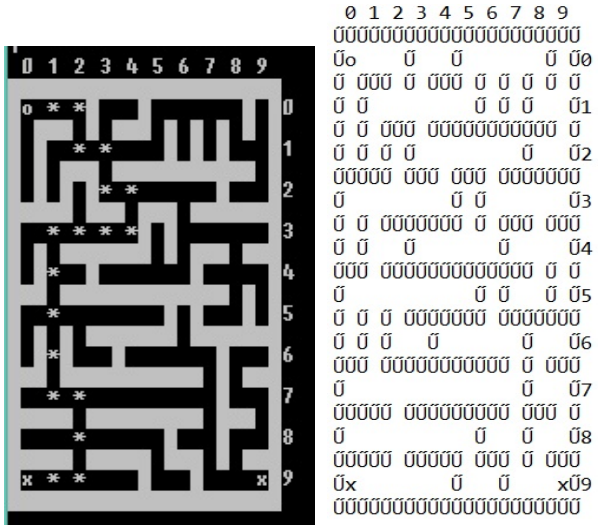

Fig. 1. *Printing the maze through the standard exit (left)/in the file (right)*

Solving the maze (option 7 from the defined menu) solves the whole maze (it finds the path from the entrance to the exit) or only its part (it finds the path between certain coordinates). Printing of maze (option 8) is realized in two ways: printing (drawing) the maze on the standard exit (Fig 1. Left) or printing the maze (Fig 1. Right) on the file. By choosing the second option, a text file is created Lavirint.txt with the stated content.

CONCLUSION

The maze presents a set of connected nodes and edges in which movement is enabled. They are mostly used in games and when created in the physical environment, they enable the practice of the spatial orientation. There are several methods for generating the maze. In this paper, the spanning tree algorithm is used. For generating the maze, Kruskal's algorithm was used, which passes from the forest of unconnected graph nodes, until it obtains a tree which is guaranteed to be minimally spanning. For the purposes of solving the maze, Floyd's algorithm was used. This algorithm was used for finding the shortest distance between all sets of nodes in the graph and the reconstruction of the discovered shortest paths. Drawing the generated maze with the choice of the appropriate entrances and exits and paths between them is performed through the standard exit or an appropriate text file. Further development of this work should be directed towards the use of other algorithms for generating a maze, where their comparative analysis will be performed through an adequate metric.

REFERENCE

[1] Levitin A., The Design and Analysis of Algorithms, Pearson Education, 3rd ed, 2012.

[2] Tomašević M., Algoritmi i strukture podataka, Akademska misao, 2008.

[3] Janičič P., Nikolić M., Veštačka inteligencija, Elektronsko izdanje, Beograd, 2019.

[4] Foltin M., Automated Maze Generation and Human Interaction, Masaryk University Faculty of Informatics, Brno 2011.

[5] Haiming L., Qiyang X., Yong W., Research and Improvement of Kruskal Algorithm, Journal of Computer and Communications, 2017, 5, 63-69.

[6] Ramadhian F. H., Implementation of Prim's and Kruskal's Algorithms' on Maze Generation, Makalah IF2120 Matematika Diskrit – Sem. I Tahun 2012/2013.