Implementation of Prim's and Kruskal's Algorithms' on Maze Generation

Fauzan Hilmi Ramadhian 13512003
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
fauzan_hilmi@students.itb.ac.id

Maze is an appealing subject to discuss in mathematics and computer science. An interesting discussion about it is how the solution path of the maze can be found by self-running algorithms. The maze generation is interesting too; how can such complex structure created without direct human interventions? In this paper, two spanning-tree-based methods of generating mazes, that is, Prim's and Kruskal's algorithms, will be discussed. There will be discussed as well about how their performances and mazes they produce are compared to each other.

Index Terms— **Kruskal's algorithm, maze generation, minimum spanning tree, Prim's algorithm**

I. Introduction

Maze is a graphical puzzle which consists of complex structure with a series of interconnecting pathways. The goal of the puzzle is to connect the start point with the finish point by a single route.

Maze can be created or implemented on various shapes and forms. There are mazes on 2 dimensions, 3 dimensions, and even on 4 dimensions surface (there are "past" and "future" portals on 4 dimensions maze). There is maze on a paper which can be solved by pencil and rubber; there is also 3D maze that was made by construction of blocks or bushes which the solver must go inside the maze from entrance to the exit. Another thing to note is that maze is a game which can be played without any significant boundaries. There are no any restrictions to play this game. Besides that, maze is a fun game to play with. Every person can play this game alone by her/himself or do a challenge with her/his friends or families. Those things above make maze a popular game to play with anywhere, by anyone.

Besides to play with, maze is also interesting and fun to be analyzed and explored. The main topics about maze study are its generations and solution findings. There are many maze generation and solution finding methods that have been discovered.

The maze generation methods are basically classified into four types. They are graph-based method, recursive division method, simple algorithm, and cellular automaton algorithm. The maze generations that are discussed in this paper are focused on the graph-based method; more precisely, the spanning tree generation algorithms.

On a spanning tree based maze generation, a maze can be generated by starting from a set of cells (or matrix of cells on 2D maze generation) with wall sites between them. This set of cells can be viewed as a connected graph with nodes represented the cells and edges represented the walls. Then, the algorithm proceeds by constantly adding nodes or edges to the maze with certain rules until there are no more nodes or edges in the set of cells that can be added. Two of spanning tree based maze generations; Prim's and Kruskal's algorithms, will be covered further on this paper.

II. Theory

A. Graph

Graph is defined as a pair of two sets as (V,E), which V represents a non-null set of vertices or nodes and E represents a set of edges. Graph is used to represent links or connections between some objects, where the objects are represented by nodes and the links are represented by edges.

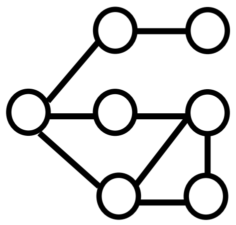

Fig. 2.1. An example of graph. The circles represent nodes and the lines represent edges.

There are some basic terms about graph. Here are the explanations of them.

i. *Undirected Graph*

Undirected graph is a kind of graph which the edges don't have particular direction. So, the ordering of two adjacent nodes which is connected by an edge is unnecessary.

ii. *Directed Graph*

Contradicted to undirected graph, the edges in directed graph have direction to determine the initial vertex and the terminal vertex. Ordering of two adjacent nodes becomes necessary now as (v_a,v_b) and (v_b,v_a) represents two different edges. In this kind of graph, edges sometimes referred as *arcs*. That is, arc is defined as directed edge

iii. *Adjacent*

Two different nodes are said to be adjacent to each other if they are connected with an edge. In other words, in a graph *G*, vertex v_a is said adjacent to vertex v_b if (v_a,v_b) is an edge in G.

iv. *Incident*

An edge is said to be incident to be incident to two different nodes if the nodes are connected by that edge. In other words, in a graph G, if there is exist an edge $e = (v_a,v_b)$, e is adjacent to v_a and v_b.

v. *Path*

A path is a sequence of edges that started at a vertex and travels from vertex to vertex along the edges in a graph. Formally, in a graph *G*, a path with n length from vertex v_a to v_b is defined as a sequence of n edges e_1 , ... , e_n of *G* such that e_1 is associated with vertex pair (v_0,v_1), e_2 is associated with (v_1,v_2), and so on, with e_n is associated with (v_{n-1},v_n), where $v_0 = v_a$ and $v_n = v_b$.

vi. *Connected*

Two different nodes are said to be connected if there is a path between them. If every pair of two nodes in an undirected graph *G* is connected, *G* is a *connected graph*. That is, an undirected graph $G = (V,E)$ is said to be a connected graph if every pair of nodes v_a and v_b in set *V* has a path that connects each other. Otherwise, *G* is said to be an *unconnected graph*.

vii. *Subgraph*

A graph *G1* is said to be a subgraph of graph *G* if *G* "contains" *G1*. Formally, a graph $G1 = (V1,E1)$ is a subgraph of graph $G = (V,E)$ if $V1 \subseteq V$ and $E1 \subseteq E$.

viii. *Spanning Subgraph*

A subgraph *G1* of graph *G* is said to be a spanning subgraph if *G1* contains all the nodes in *G*. That is, a subgraph $G1 = (V1,E1)$ of graph $G = (V,E)$ is a spanning subgraph if $V1 = V$.

ix. *Circuit*

A circuit is a path where the starting vertex is the same with the terminal vertex. In other words, a circuit is a path with sequence of n edges e_1 , ... , e_n in graph *G* where e_1 is associated with (v_0,v_1), e_2 is associated with (v_1,v_2), and so on, with e_n is associated with (v_{n-1},v_n) and $v_0 = v_n$.

x. *Weighted Graph*

A graph is said to be a weighted graph if each of its edges has a value or weight.

xi. *Graph Weight*

In a weighted tree *G*, graph weight is defined as the sum of all edge weights in *G*.

B. Tree

Tree is graph with some specializations. A tree is defined as an unconnected graph which doesn't contain any circuits. These are some basic terms about tree.

i. *Forest*

Forest is defined as a set of disjoint trees. Forest can also be defined as unconnected graph which doesn't contain any circuits.

ii. *Spanning Tree*

Spanning tree is defined as spanning subgraph which is also a tree. Every connected graph has at least one spanning tree.

iii. *Minimum Spanning Tree*

In a weighted graph G, minimum spanning tree is a spanning tree with the least weight among the others spanning tree in G. A weighted graph can has more than one minimum spanning tree.

Minimum spanning tree is considered as the most important spanning tree because of its wide applications. So, knowing how to generate a minimum spanning tree from a weighted graph is important. There are some techniques to do that, two of them are discussed here.

1) Prim's Algorithm

Prim's algorithm was discovered by Robert Prim on 1957. It is a technique to generate a minimum spanning tree from a weighted graph by choosing an edge with the least weight, and then put it into the spanning tree. Next, select one of the remaining edges with the least weight so that the edge is incident with a node already in the tree and not forming a circuit in the tree. Then, add the edge to the tree. Repeat the process until n-1 edges have been added (n is the amount of nodes in graph at the beginning). The pseudo-code of the algorithm is as follows.

```
procedure Prim (input G:Graph, output T:Tree)
{I.S     : Graph G = (V,E) is defined
 F.S     : Minimum spanning tree T = (V,E') is
           defined
Process : Generating a minimum spanning tree
from a graph by Prim's algorithm}

Declaration
i, p, q, u, v, n : integer

Algorithm
n ← |V|
Find an edge (p,q) with the least weight from set
E
T ← {(p,q)}
for i=1 to n-2 do
        Select an edge (u,v) wih the least weight
        from E and is incident with a node in T
        if (u,v) isn't forming circuit in T then
                T ← T ∪ {(u,v)}
        endif
endfor
```

2) Kruskal’s Algorithm

Joseph Kruskal invented this minimum spanning tree creation algorithm on 1956. To carry out this algorithm, first, arrange the edges of the graph G based on their weights. Next, create a forest T that contains all of the edges in G that are unconnected each other. Then, successively as many as n-2 times, add the least edge in G to T so that no circuits are formed in T. In the end, T is a minimum spanning tree from G. The pseudo-code of the algorithm is as follows.

```
procedure Kruskal (input G:Graph, output T:Tree)
{I.S      : Graph G = (V,E) is defined
 F.S      : Minimum spanning tree T = (V,E') is
            defined
 Process  : Generating a minimum spanning tree from
            a graph by Kruskal's algorithm}

Declaration
i, p, q, u, v, n : integer

Algorithm
n ← |V|
T ← {}
while number of T edges < n-1 do
            Select an edge (u,v) with the least weight
            from E
            if (u,v) isn't forming circuit in T then
                    T ← T ∪ {(u,v)}
            endif
endwhile
```

C. Maze

There are various types and kinds of mazes. Here will be explained three important types of maze based on their routings and their graph representations.

1) Sparse Maze

The first type is *sparse maze.* The passages on sparse maze don’t pass on all cells. The unpassed cells become inaccessible cells that the solver cannot go through. Solving this kind of maze is not too difficult because the solution path can easily spotted.

An example of a 3x3 sparse maze and its graph representation is as follows.

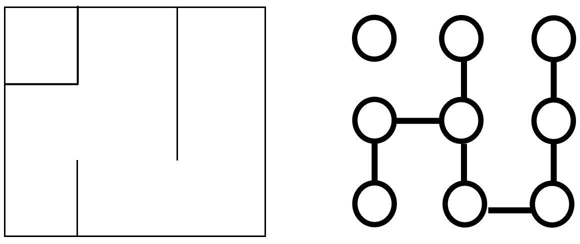

Fig. 2.2. An example of sparse maze and its graph representation.

Since there must be at least one unconnected node, this maze type cannot be created by minimum spanning tree creation algorithms.

2) Braid Maze

The second type is *braid maze*. Although braid maze doesn’t have dead-ends, it haves junctions. The junctions connect multiple paths to form circles or loops. This is what makes braid maze quite difficult to solve as the loops can cause confusion when the solver is finding out the path to the finish. However, the loops can also make the maze easier to solve because it gives more than one possible solution paths.

An example of 4x4 braid maze and its graph representation is as follows.

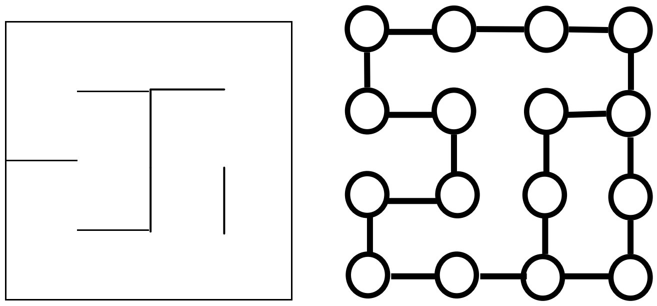

Fig. 2.3. An example of braid maze and its graph representation.

This maze type cannot be created by minimum spanning tree creation algorithm since its graph representation contains circuits, and thus is not a tree.

3) Perfect Maze

The third type is *perfect maze.* As the name suggests, this is the most perfect kind of maze that can give difficult challenge to do. In perfect maze, there are no circles or loops. So, there is only one right path that connects start point to finish point. What makes perfect maze difficult to solve is that the maze has many dead-ends that the solver must try to avoid of. Sometimes perfect maze is referred as *simply-connected maze.*

An example of a 4x4 perfect maze and its graph representation is as follows.

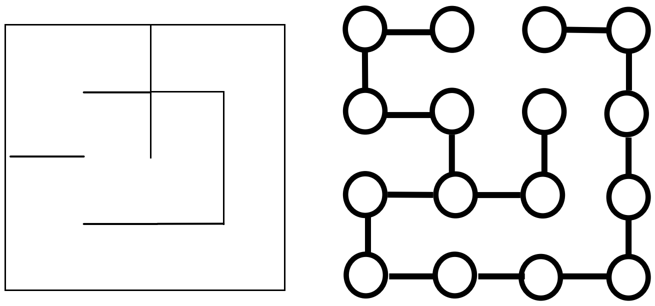

Fig. 2.4. An example of perfect maze and its graph representation.

Since its graph is connected, contains all of the nodes (cells) and doesn't contain any circuits, a perfect maze can be described as a spanning tree over the set of the cells. Thus, this type of maze can be created with spanning tree creation algorithm such as Prim's and Kruskal's algorithm that will be discussed further on this paper.

III. MAZE GENERATION

Here are the explanations of how the maze generation works by each algorithm. Note that in the tree representations of the mazes, the nodes are representing the cells and the edges are representing the walls between the cells. Since there are no "weights" in the tree, some adaptations are conducted on the original algorithms to solve the faced problems.

A. Prim's Algorithm Method

The basic principles of Prim's algorithm are choosing a node from the graph, and then choose the edge with the smallest weight that connects the first edge with another edge in the graph, and then choose the edge with the smallest weight that connects the second edge with another edge in the graph, and so on. In maze generation, there are some adaptations that must be applied on the algorithm. First, because the cells are the nodes, we begin with adding a random cell from the cells matrix to the maze. Then, add a cell that is adjacent with the previous added cell to the matrix. Repeat the second process until there are no cells can be added.

Since it is an adapted version of Prim's algorithm that handles non-weighted edges on maze generation, this algorithm is sometimes called *Randomized Prim's Algorithm* (The "randomized" term comes from the fact that the algorithm choose the edge at random instead of the least weighted). The pseudo-code of the algorithm is as follows.

```
procedure PrimMaze (input S:Set of cells,
output M:Maze)
{I.S      : Set of cells S is defined
 F.S      : Maze M is generated
Process  : Generating a perfect maze from a set
of cells by Prim's algorithm}

Declaration
c : cell

Algorithm
Select a random cell c from S
M ← c
while M is not full do
    Select an unmarked random cell c from S
    if c is adjacent to one of the cells in M then
        M ← M ∪ c
    endif
    Mark c so it will not be selected again in S
endwhile
```

For a clearer understanding, let's take a look at an example of 3x3 maze generation by this algorithm. First, let's start with a matrix of 3x3 cells.

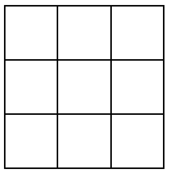

Fig. 2.5. The steps of maze generation by Prim's algorithm (1).

Then, select a cell at random and add it to the maze. Mark the added cell in the matrix.

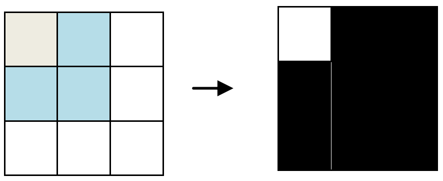

Fig. 2.6. The steps of maze generation by Prim's algorithm (2).

Now, select one of the three cells (the blue-shaded cells) that are adjacent with the first cell that has been added. Add the cell to the maze.

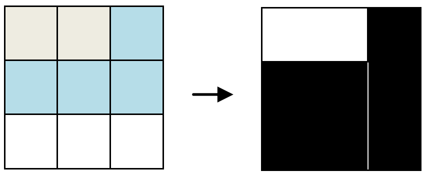

Fig. 2.7. The steps of maze generation by Prim's algorithm (3).

Again, select one of the cells (the blue-shaded cells) that are adjacent with the cells that have been added previously. Add the selected cell to the maze.

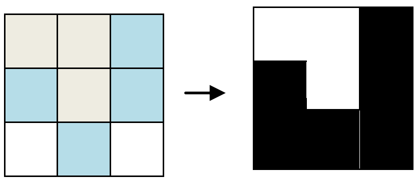

Fig. 2.8. The steps of maze generation by Prim's algorithm (4).

Now, if we select a cell that is adjacent to more than one previously added cells, choose only one cell among them as the "neighbor" of the newly selected cell. Then, add the selected cell to the maze.

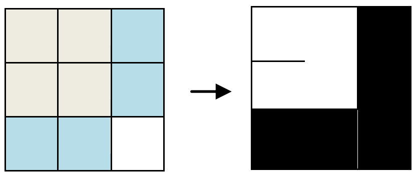

Fig. 2.9. The steps of maze generation by Prim's algorithm (5).

Repeat these processes until the maze is full.

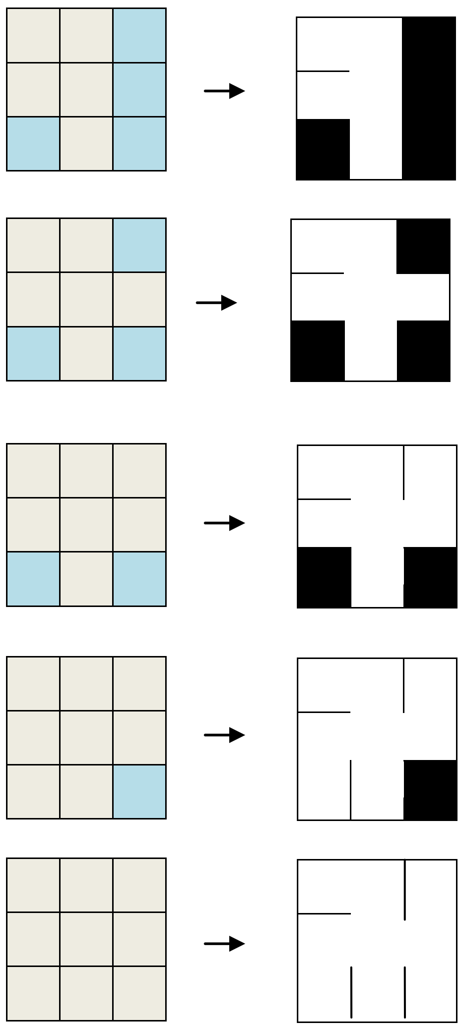

Fig. 2.10. The steps of maze generation by Prim's algorithm (6).

At this point, the algorithm terminates because there are no more cells that can be selected from the matrix. Now, add "a little touch" to complete the maze, that is, add two "holes" on the outer wall to mark the start and finish point of the maze.

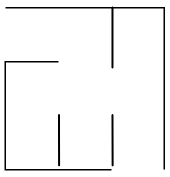

Fig. 2.11. Final result of the maze.

Here it is, a perfect maze generated by Prim's algorithm is finished.

B. Kruskal's Algorithm Method

Just like Prim's algorithm, this adapted version of Kruskal's algorithm is sometimes referred as *Randomized Kruskal's Algorithm*. The algorithm started by creating a forest of all nodes in cells matrix. Then, select a random edge (wall between two cells) and add it with two cells that are connected with it to the maze, creating a tree or a set of cells. After that, select another random edge that doesn't connect two cells in a same "tree" and add it to the maze. The processes finished when there are no edges left to be selected, or there is only one tree remaining in the cells matrix.

The pseudo-code of the algorithm is as follows.

```
procedure KruskalMaze (input S:Set of cells, output M:Maze)
{I.S      : Set of cells S is defined
 F.S      : Maze M is generated
Process  : Generating a perfect maze from a set of cells by Kruskal's algorithm}

Declaration
e : edge
c_1, c_2 : cell

Algorithm
Select a random edge e = (c_1, c_2) in S
M ← {( c_1, c_2)}
while number of tree in S >1 do
    Select a random edge e = (c_1, c_2) in S with c_1 and c_2 is in different tree
    M ← M ∪ {( c_1, c_2)}
    Unify c_1 and c_2 in S into a single tree
endwhile
```

Here is an example of a 3x3 maze generation by Kruskal's algorithm. The first step is creating a 3x3 cells matrix with a forest of 9 disjoint trees (in this case, 9 nodes). Note that different colors represent different trees or sets of cells.

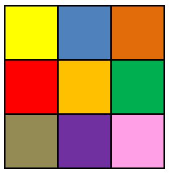

Fig. 2.12. The steps of maze generation by Kruskal's algorithm (1).

Next, add a random edge from matrix to the maze. In this case, the added edge is the edge between cell (2,2) and (2,1).

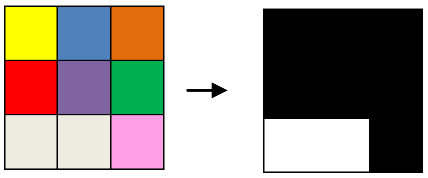

Fig. 2.13. The steps of maze generation by Kruskal's algorithm (2).

Note that when an edge is added, two cells that are connected to it are unified into a single tree. Now, let's add another edge to the maze.

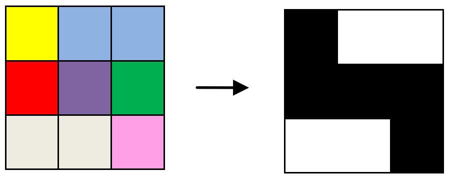

Fig. 2.14. The steps of maze generation by Kruskal's algorithm (3).

Keep adding more edges until there is only one tree remaining in the matrix.

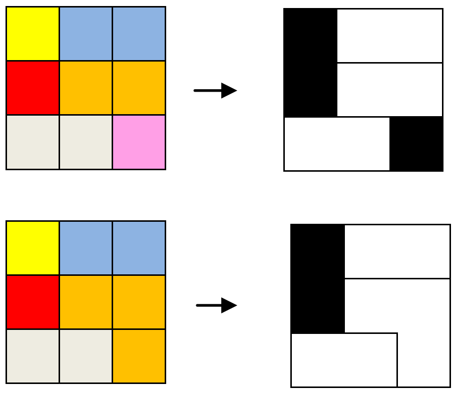

Fig. 2.15. The steps of maze generation by Kruskal's algorithm (4).

Now add the edge between (2,2) and (3,2) so that the orange and grey tree unified into one tree.

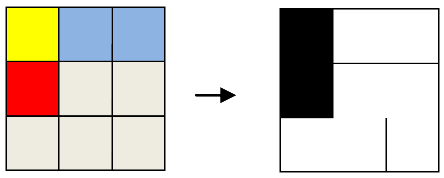

Fig. 2.16. The steps of maze generation by Kruskal's algorithm (5).

Again, add more edges.

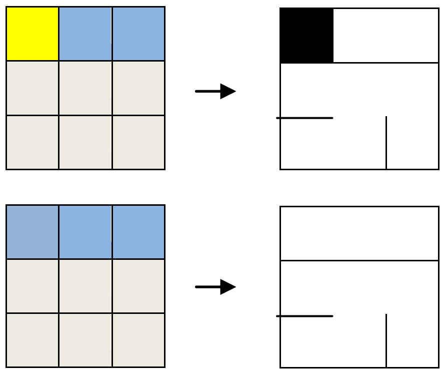

Fig. 2.17. The steps of maze generation by Kruskal's algorithm (6).

At this point, the process is not yet stopped because there are still two trees left behind in the matrix. The adding of the edge between (1,2) and (2,2) unify the trees and hence, marks the final step of the algorithm.

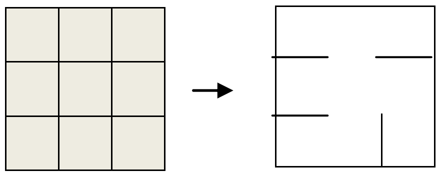

Fig. 2.18. The steps of maze generation by Kruskal's algorithm (7).

Finally, add random start and finish point to complete the maze.

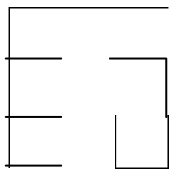

Fig. 2.19. Final result of the maze.

IV. ANALYSIS

The basic principle of randomized Prim's and Kruskal's algorithm is same, that is, generating a perfect maze based on minimum spanning tree algorithm. The basic difference between them is that Prim's is focused on adding nodes or cells to the maze while Kruskal's is focused on adding the edges. In theory, they can generate maze that is similar to each other. However, there must be some differences on their performances and the mazes they are generating as the implication of the algorithm difference between them.

Here is a table that lists head-to-head performance statistics between the two.

Table 4.1 Maze generation performance statistics by Prim's and Kruskal's Algorithm.

Algorithm	Dead End	Memory	Time	Solution
Kruskal	30%	N^2	21	4.1%
Prim	36%	N^2	43	2.3%

Data was taken from astrolog.org/labyrnth/algorithm, accessed on 8 December 2013, 10.46 pm.

These statistics was extracted from testing of several NxN cells mazes. The dead end statistic measures the approximate percentage of dead ends cells upon of the total cells. The memory statistic represents how much extra memory is required to implement the maze with NxN cells. The time measures the relative time needed to create one maze with the lower number being faster (the fastest is speed 10). The last statistic, solution, represents the length or the “windiness” of the solution path (the path that connects start to finish point). In other words, the solution statistic represents the percentage of cells in the maze that the solution path passes through. A maze with long solution path is less complex in term of the maze complexity.

From the statistics, we can see that Prim’s algorithm generates more dead ends than Kruskal’s algorithm. This happens because Prim’s maze tends to distribute nodes in a short area of cells and thus, tends to have many short dead ends. In terms of memory needed, both algorithms equally needs as many as N^2 memory since both needs a matrix of NxN cells as a “temporary storage” before adding the cells to the maze.

In comparison of the Prim’s and Kruskal’s algorithm time process, Kruskal’s is quite faster. This happens because Kruskal’s maze generate less dead-ends than Prim’s. Lastly, Kruskal’s maze tends to have longer and more “windy” solution path than the Prim’s. So, it can be said that Prim’s maze is more complex than Kruskal’s maze. This stat is related to the fact that Prim’s generated maze tends to have more dead-ends and thus, have less cells in the solution path than the Kruskal’s maze.

In conclusion, Prim’s maze tends to have higher complexity than Kruskal’s because it has more dead-ends and less solution cells percentage. However, Kruskal’s algorithm is better than the Prim’s algorithm in term of process time.

V. CONCLUSIONS

Maze is a kind of graphical puzzle that was and is quite popular around the world. There are several methods on how to draw mazes; one of them is the spanning tree based algorithms. Two commonly used spanning tree algorithm to generate mazes are Prim’s algorithm and Kruskal’s algorithm. In principle, Prim’s algorithm generates the maze by placing cells one by one from the cells matrix to the maze, whereas Kruskal’s algorithm proceeds by placing the edges one by one.

Determining the best algorithm to use between Prim’s and Kruskal’s algorithm is quite difficult. It is because their methods are principally same. However, there are some preferences that should be noted. If a complex and hard-to-solve maze is wanted, Prim’s algorithm is the choice. But, if the processing time is more important, Kruskal’s algorithm is better.

VI. ACKNOWLEDGEMENT

First of all, Author would say thank you to Almighty God because of His mercy and grace Author can finish this paper. Then, Author also wants to express his thanks to Dr. Ir. Rinaldi Munir, M.T. and Mrs. Harlili, whose give helpful advices and assistances. Finally, Author wants to say thank you to his parents and beloved friends who are always give Author strengths and spirits to pass the struggles during the writing of this paper.

REFERENCES

[1] Buck, Jamis. ”Maze Generation: Kruskal's Algorithm,” weblog.jamisbuck.org/2011/1/3/maze-generation-kruskal-s-algorithm, January 3 2011, accessed on December 8 2013, 08.05 pm.
[2] Buck, Jamis. ”Maze Generation: Prim's Algorithm,” weblog.jamisbuck.org/2011/1/10/maze-generation-prim-s-algorithm, January 10 2011, accessed on December 8 2013, 08.07 pm.
[3] Munir, Rinaldi, *Diktat Kuliah IF 2120, Matematika Diskrit, Edisi Keempat, Program Studi Teknik Informatika, STEI ITB,* 2006, pp. VIII-2 – VIII-18, IX-1 – IX-9.
[4] Pullen, Walter D., ”Maze Algorithms,” astrolog.org/labyrnth/algorithm, accessed on December 8 2013, 10.46 pm.
[5] Rosen, Kenneth H., *Discrete Mathematics and Its Applications, Sixth Edition,* Singapore:McGraw-Hill, 2007, pp. 622, 738-740.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 16 Desember 2013

[illegible]

Fauzan Hilmi Ramadhian - 13512003