Journal of Theoretical and Applied Information Technology
15th September 2020. Vol.98. No 17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

COMPARATIVE ANALYSIS OF DEPTH-FIRST SEARCH ALGORITHM AND GREEDY ALGORITHM AT NEAREST ATM SEARCH IN PADANG SIDEMPUAN CITY

DIAN RACHMAWATI[1*], SYAHRIL EFENDI[1], ADI SYAHPUTRA SITUMORANG[1]

[1]Departemen Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi,

Universitas Sumatera Utara

Jl. Universitas No. 9-A, Kampus USU, Medan 20155, Indonesia

[*]E-mail: dian.rachmawati@usu.ac.id

ABSTRACT

The Automated Teller Machine (ATM) is needed when users need immediate banking transactions. But sometimes ATM has some problems such as ATMs are being broken, money deposited in ATM depleted, the number of people who queued at ATM, etc. At that time, people needed an alternative ATM located nearby. To make the process of ATM search easier, than it takes a system that serves to find the nearest ATM from the ATM location that the user is visiting. Padang Sidempuan City has many ATMs in various places. This research will make the nearest ATM search system in Padang Sidempuan city. In this system, there is a menu to search by selecting the starting point and ATM of the bank that you want to go and will produce the nearest ATM and the line from the starting point to the ATM. To support the closest ATM search in this system, the deep first search algorithm and the greedy algorithm applied to this system. Then, the performance of both algorithms will be compared based on process time and distance. After implementation and comparison, it is known that the complexity of the depth-first search algorithm is the same as the complexity of the algorithm greedy, (N2). Attesting with a sample of 10 starting points and 1 ATM destination, the depth-first search algorithm has an average running time of 239.9675 milliseconds, and the average distance is 3033.555 meters, while a greedy algorithm has an average running time of 274.8501 milliseconds and the average distance is 2035.2568 meters. So it concludes that the depth-first search algorithm is more efficient in running time than the greedy algorithm. But in generating shorter distances, the greedy algorithm is better than the depth-first search algorithm.

Keywords: *Depth First Search, Greedy, ATM, Shortest Path, Algorithm*

1. INTRODUCTION

The Automated Teller Machine (ATM) is needed when users need immediate banking transactions. But sometimes ATM has some problems such as ATMs are being broken, money deposited in ATM depleted, the number of people who queued at ATM, etc. At that time, people needed an alternative ATM location nearby [4].

Padang Sidempuan city is one of the biggest cities in North Sumatera. In this city, technological developments have begun to grow. In this city, there are already many ATMs at various locations.

In the process of finding the nearest ATM, to simplify the process, we need the shortest path system in finding the nearest ATM.

The shortest path is a method of solving the problem of finding the shortest path from location A to location B using a directed and weighted graph. The weight is the value of the distance between one point to another point. According to Andrew Goldberg, Microsoft Research Researcher Silicon Valley, said there are many reasons why researchers try to find the problem in the shortest way. "The shortest path is an optimization problem that is relevant for a variety of applications, such as network routing, games, circuit design, and mapping." In applying the shortest path, we need several algorithms to solve it, including the depth-first search algorithm and the greedy algorithm [17].

The depth-first search algorithm is an algorithm that is used to search for a point in a graph. The depth-first search algorithm will expand the root

from the starting point and trace from the first root to the inside until the destination point is found. If the destination point is not found, the search will be returned from the previous point.

The greedy algorithm is a search for a path from the starting point to the destination point by expanding the root of the starting point. The greedy algorithm will select the root that selects the smallest distance to continue the search without considering the consequences of the search for the future.

The advantage of DFS is that DFS consumes very little memory space. DFS will reach on the destination node in fewer time periods than BFS if it crosses on the correct path. DFS can find a solution without checking many searches because we might get the desired solution in the first step.The biggest benefit of having Greedy algorithm over others is that it is easy to implement and very efficient in most cases.

Based on the background that has been explained, then the research was conducted to determine the comparison of the performance of the depth-first search algorithm and the greedy algorithm with a case study of finding the nearest ATM in Padang Sidimpuan city.

2. SHORTEST PATH

The shortest path is the search for optimum routes between nodes in the graph [8]. In search of the shortest path, the problem faced is to find the path which is traversed to obtain the most optimum path from one point to another point [7]. There are several kinds of shortest path problems, that is :

1. The shortest path between two points
2. The shortest path between all points pairs
3. The shortest path from a particular point to all other points
4. The shortest path between two points passing through specific points

The shortest path is one of the problems that can be solved using a graph. If given a weighted graph, the shortest problem is how to find a path in the graph that minimizes the number of side weights forming the path [9].

Shortest path completion can be done using algorithms such as a depth-first algorithm, breadth-first algorithm, hill-climbing algorithm, greedy algorithm, Dijkstra, A* and many more algorithms that can be used [13][18].

3. GRAPH

A graph is a set of connected nodes through a path (edge) [14]. A graph consists of two sets, namely set V, which contains a set of vertices (nodes) that cannot be empty, and the set E containing the edge (path) set [15]. The graph is divided into four parts, that is:

1. Directed and Weighted Graph
   Each side has a direction and weight (value) between one vertex and another vertex.
2. Directed and Not Weighted Graph
   Each side only has a direction between one vertex with another vertex, no weight (value)
3. Not Directed and Weighted Graph
   Each side does not have a direction but weights one vertex with another vertex
4. Not Directed and Not Weighted Graph
   Each side has no direction and has no weight between one vertex with another vertex.

4. ALGORITHM

An algorithm is a collection of logical steps in problem-solving that are arranged systematically [10]. Specifically, an algorithm is a unique method that is appropriate and consists of a series of steps that are structured and written consistently, which will be to solve a problem with the help of a computer [2].

According to Donald E. Knuth, the characteristics of the algorithm that is :

1. The algorithm has start and end process. In other words, the algorithm has a restricted step.
2. Each step in the algorithm is defined precisely and unambiguous (double meaning)
3. The algorithm has an input or initial conditions to be processed.
4. The algorithm has an output or terminal condition as a result of the process.
5. The algorithm must be valid so that it can produce output in logical time.

5. DEPTH FIRST SEARCH ALGORITHM

Depth First Search is a search that runs by expanding he first root child of the graph and goes deeper and deeper until the destination node is found, or until find a knot that doesn't have any more children [3][16]. Then, research will return to the node that has not been traced [6]. In a non-recursive

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

implementation, all expanded nodes are added to LIFO (Last In First Out) stack [11].

Following is an example of the DFS algorithm in searching for a point in a graph. For example, given a graph, the starting point is A, and the destination point is G. Determine the path from A to G.

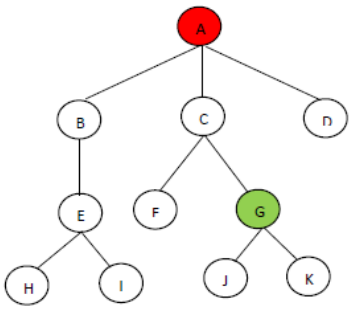


Figure 1. Example of DFS algorithm in searching for a point in a graph

In figure 1, the relationship between trajectories is not bidirectional. All tracks only run from top to bottom. A has a path to B, C, and D, but B, C, and do not have a path to A. Each circular circle in the graph above is the vertex. The Node A is the parent of nodes B, C, and D, while nodes B, C, and D are child nodes A and so on down. The following is the completion of the graph above :

- We will use two info lists to save the steps we take, namely Open and Close. Open is the node to be checked, while Close is the node that has been verified. Because this is the first step, Close is still empty, and Open will contain the initial node, which is A.
  Open : {A}
  Close : { }
- Check whether node A has children. If so, add the child to Open, and add A to Close because it's already checked.
  Open : {B,C,D}
  Close : {A}
- Open now contains three nodes. For depth-first search, always search for the first node of Open. Because B is not a goal, check whether node B has children. If yes, add the child to Open replacing node B which was added to Close
  Open : {E,C,D}
  Close : {A,B}
- Because E is not a goal, check wether node E has children. If yes, add the child to Open replacing node E, which was added to Close.
  Open : {H,I,C,D}
  Close : {A,B,E}
- Because H is not a goal, check whether node H has children. If not, delete H from Open and add H to Close.
  Open : {I,C,D}
  Close : {A,B,E,H}
- Now we check node I. Because I is not a goal and has no children, delete I from Open and add I to Close.
  Open : {C,D}
  Close : {A,B,E,H}
- Now we check node C. Because C is not a goal, check whether node C has children. If so, add the child to Open replacing node C that was added to Close.
  Open : {F,G,D}
  Close : {A,B,E,H,C}
- Because F is not a goal and has no children, delete F from Open and add F to Close.
  Open : {G,D}
  Close : {A,B,E,H,C,F}
- Now we check node G. Because G is the goal, then we stop until this step. Delete G from Open and add G to Close. The final Open and Close are as follows :
  Open : {D}
  Close : {A,B,E,H,C,F,G}

The final path to take the depth-first search algorithm is the last value of Close, which is A, B, E, H, C, F, G.

6. GREEDY ALGORITHM

The greedy algorithm is an algorithm that solves problems step by step. The greedy algorithm is one algorithm that is often used in solving optimization problems. The greedy algorithm has the principle of 'take what you can get now,' the intention is to make the best choice at this time without regard to making these choices in the future [1]. The workings of the greedy algorithm are as follows :

1. Visit a point on the graph.
2. Remove all possible points to be visited from the current position. Calculate the distance between each point to the current point
3. Mark the current point as the point visited.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. Take the point that has the smallest distance, and make that point the current point
5. If the current point is the destination point, then the search is complete. If the current point is not a destination point, repeat the search from step 2

Following is an example of the greedy algorithm in searching for the nearest path. Determine the route from point A to G.

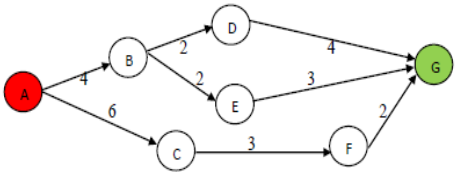


Figure 2. The greedy algorithm in searching for the nearest path

In Figure 2, the relationship between points is only one direction, so from A, it can go to B, but B can't go to A, etc. The following is the completion of the graph above :

- Take point A, take out the points that can be visited from A, that is points B and C. Calculate the distance from A to B and A to C.
  AB = 4
  AC = 6
  Because A to B is the smallest distance, then B becomes the current point. Mark A as visited. Is B the goal? Not. Continue searching.
- Take point B, take out the points that can be visited from point B, that is points D and E. Calculate the distance from B to D and B to E.
  BD = 3
  BE = 2
  Because B to E is the smallest distance, E is the current point. Mark B as visited. Is Ethe goal? Not. Continue searching.
- Take point E, take out the points that can be visited from point E, that is points G. Calculate the distance from E to G.
  EG = 3
  Because there is only one branch from point E, which is G, then G becomes the current point. Is G the goal? Yes. Search complete.

The path obtained by the greedy algorithm is A-B-E-G with a total distance of 4+2+3=9. The search results can be seen in figure 3.

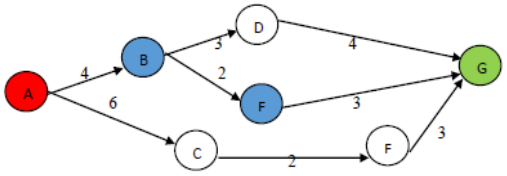


Figure 3. The search results

7. ALGORITHM COMPLEXITY

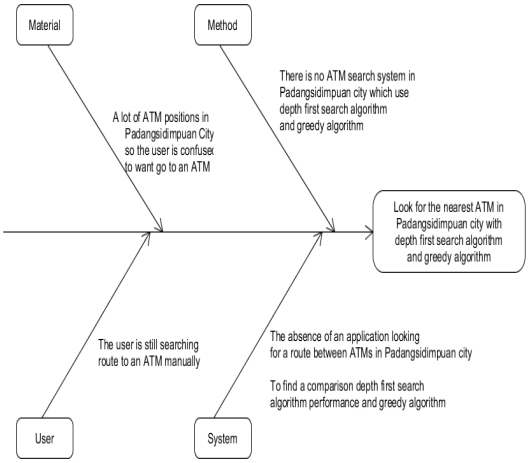
The complexity of the algorithm consists of 2 types, namely time complexity and space complexity [5].

1. Time Complexity (T(n)), measured by the number of computational stages needed to run the algorithm as a function of incoming size n. The complexity of time is divided into three types, that is :
   a. Tmax(n) that is time complexity for the worst case. The need for maximum time.
   b. Tmin(n) that is time complexity for the best case. The need for the minimum time.
   c. Tavg(n) that is time complexity for the average case. Need for time on average.
2. Space Complexity (S(n)), measured from memory used by the data structure contained in the algorithm as a function of the incoming size n.

8. ISHIKAWA DIAGRAM

Ishikawa diagram is a diagram that shows the causes of a problem to be solved. Ishikawa diagrams are also called fish diagrams (fishbone) because they are like fish bones. Ishikawa's diagram was introduced in 1968 by Kaoru Ishikawa. The use of Ishikawa is generally to prevent impacts and expand the quality of an event (the causes of a problem). Ishikawa diagram of the system to be built can be seen in figure 4.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



. Figure 4. Ishikawa Diagram

9. GENERAL ARCHITECTURE

The general architecture is a representation of a structure that describes the process or flow of a system to be built. The general architecture of the system to be made can be seen in figure 5.

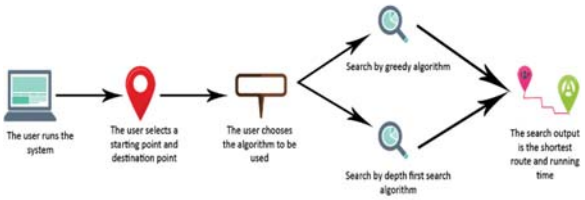


Figure 5. The General Architecture

1. First, the user runs the system.
2. Then the user selects a starting point and destination point
3. After that, the user chooses the algorithm to be used
4. Then the system will search for a route from the starting point to the destination point with depth-first search algorithm or greedy algorithm, depending on user choice
5. After that, the system will output the results of the search in the form of a path from the starting point to the destination point and running time

10. FLOWCHART

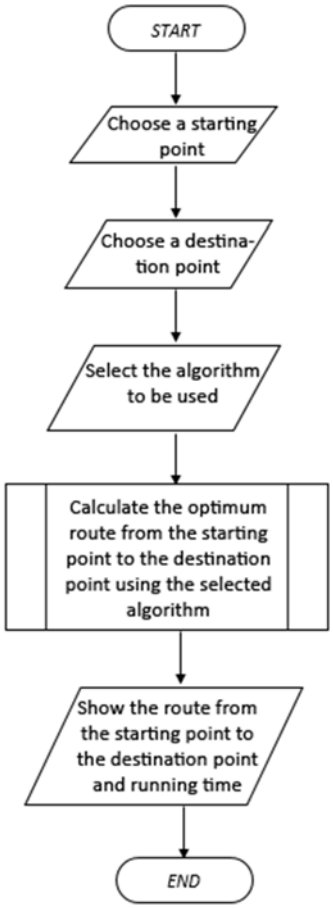


Figure 6. System Flowchart

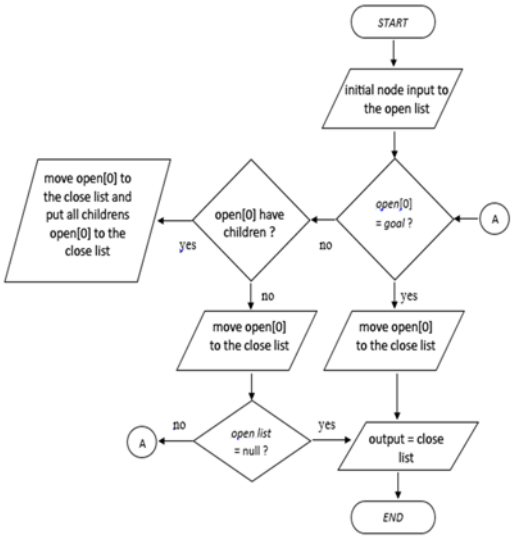


Figure 7. Depth First Search Flowchart

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

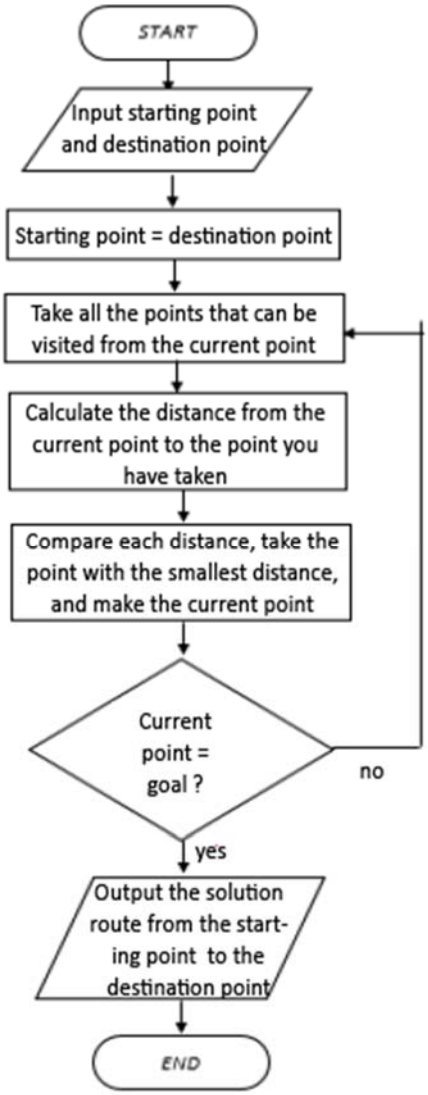


Figure 8. Greedy Algorithm Flowchart

11. IMPLEMENTATION OF DEPTH FIRST SEARCH AND GREEDY ALGORITHM

The start page is the page that the system first displays when the user runs the system. This page will run immediately without any user input. The start page will run a progress bar, and when the progress bar is full, the system will display the main page [12]. The start page display can be seen in figure 9.

Figure 9. Start Page

The main page is the page where users input ATM searches. On this page, the user selects the starting point and destination ATM and the algorithm to be used. Then the user presses the START button. Then the system will search according to the algorithm chosen by the user. On this page, the system will also display search results in the form of a route, distance, and running time. The main page display can be seen in figure 10.

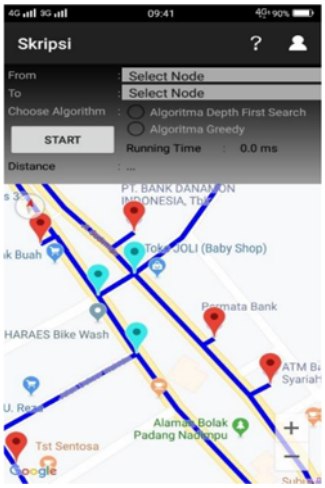


Figure 10. Main Page

Journal of Theoretical and Applied Information Technology
15th September 2020. Vol.98. No 17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

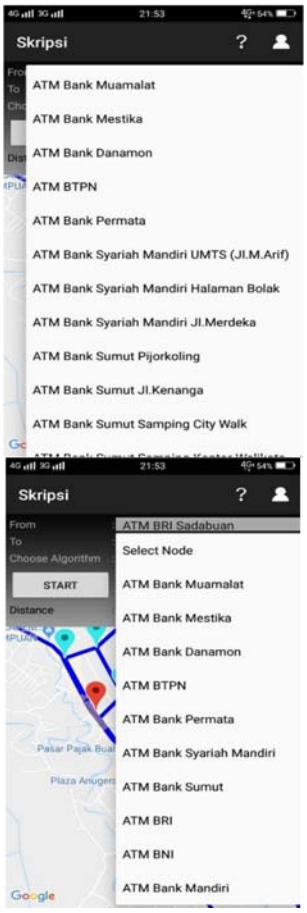


Figure 11. Main Page for Looking location

12. COMPARISON OF DFS AND GREEDY ALGORITHM

In the research, the parameters used for comparing DFS and Greedy algorithms are the distances in meters, running time application in milliseconds, and the complexity of the algorithm (big theta). The results of the two algorithms can be seen in the table 1, 2 and 3.

Table 1. Results of Depth First Search Algorithm

No	Starting Point	Destination ATM	ATM Obtained	Distance (meter)	Running time (ms)
1	ATM Bank Muamalat	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	2459.783	289.156
2	ATM Bank Mestika	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	2066.181	300.169
3	ATM Bank Danamon	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Halaman Bolak	176.581	211.165
4	ATM BTPN	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	2906.538	303.354
5	ATM Bank Syariah Mandiri UMTS	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	5059.327	188.435
6	ATM Bank Sumut Pijorkoling	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	9329.967	329.213
7	ATM BRI Sadabuan	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	1820.751	259.902
8	ATM BNI STKIP	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri UMTS	384.378	132.156
9	ATM Bank Mandiri Pajak Buah	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	1202.317	252.867
10	ATM BNI dan Bank Mandiri Hotel Sitamiang	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri UMTS	4929.727	133.258

Table 2. Results of the Greedy Algorithm

No	Starting Point	Destination ATM	ATM Obtained	Distance (meter)	Running time (ms)
1	ATM Bank Muamalat	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Halaman Bolak	943.17	255.706
2	ATM Bank Mestika	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Halaman Bolak	2312.574	276.854
3	ATM Bank Danamon	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Halaman Bolak	176.581	257.752
4	ATM BTPN	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Halaman Bolak	1389.925	261.389
5	ATM Bank Syariah Mandiri UMTS	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	1314.445	289.698
6	ATM Bank Sumut Pijorkoling	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Halaman Bolak	8232.968	301.579
7	ATM BRI Sadabuan	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	1820.751	344.626
8	ATM BNI STKIP	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri UMTS	384.378	133.795
9	ATM Bank Mandiri Pajak Buah	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Jl. Merdeka	1202.317	347.352
10	ATM BNI dan Bank Mandiri Hotel Sitamiang	ATM Bank Syariah Mandiri	ATM Bank Syariah Mandiri Halaman Bolak	2575.459	279.75

Table 3. Comparison results

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

No	Distance Comparison (meter)		Running time Comparison (ms)	
	Depth First Search	Greedy	Depth First Search	Greedy
1	2459.783	943.17	289.156	255.706
2	2066.181	2312.574	300.169	276.854
3	176.581	176.581	211.165	257.752
4	2906.538	1389.925	303.354	261.389
5	5059.327	1314.445	188.435	289.698
6	9329.967	8232.968	329.213	301.579
7	1820.751	1820.751	259.902	344.626
8	384.378	384.378	132.156	133.795
9	1202.317	1202.317	252.867	347.352
10	4929.727	2575.459	133.258	279.75
Average	3033.555	2035.2568	239.9675	274.8501

13. DEPTH FIRST SEARCH ALGORITHM COMPLEXITY

No	Kode Program	C	#	C#
1	Stack<Graph> stack = new Stack<>();	C1	1	C1
2	Stack<Double> s_jarak = new Stack<>();	C2	1	C2
3	Stack<PolylineOptions> s_line = new Stack<>();	C3	1	C3
4	Stack<Graph> visiteds = new Stack<>();	C1	1	C1
5	Stack<Double> jaraks = new Stack<>();	C2	1	C2
6	Stack<PolylineOptions> polylineOptionses = new Stack<>();	C3	1	C3
7	ArrayList<Graph> titik = new ArrayList<>();	C1	1	C1
8	ArrayList<Double> jarak = new ArrayList<>();	C2	1	C2
9	ArrayList<PolylineOptions> line = new ArrayList<>();	C3	1	C3
10	String a = "";	C4	1	C4
11	double total=0.0;	C5	1	C5
12	s_jarak.add(0.0);	C2	1	C2
13	s_line.add(new PolylineOptions().add(new LatLng(0, 0), new LatLng(0, 0)));	C3	1	C3
14	stack.add(start);	C1	1	C1
15	while (!stack.isEmpty()) {	C6	n	nC6
16	Graph element = stack.pop();	C1	n	nC1
17	double abc = s_jarak.pop();	C5	n	Nc5
18	PolylineOptions def = s_line.pop();	C3	n	nC3
19	visiteds.add(element);	C1	n	nC1
20	jaraks.add(abc);	C2	n	nC2
21	polylineOptionses.add(def);	C3	n	nC3
22	element.visited = true;	C1	n	nC1
23	if (element != goal) {	C7	n	nC7
24	ArrayList<Graph> neighbours = element.getNeighbour();	C1	n	nC1
25	ArrayList<Double> distance = element.getDistance();	C2	n	nC2
26	ArrayList<PolylineOptions> polyline = element.getPolylineOptions();	C3	n	nC3
27	for (int i=0; i<neighbours.size(); i++) {	C8	n^2	n^2C8
28	Graph n = neighbours.get(i);	C1	n^2	n^2C1
29	double x = distance.get(i);	C5	n^2	n^2C5
30	PolylineOptions p = polyline.get(i);	C3	n^2	n^2C3
31	if (n != null && !n.visited) {	C7	n^2	n^2C7
32	titik.add(n);	C1	n^2	n^2C1
33	jarak.add(x);	C2	n^2	n^2C2
34	line.add(p);	C3	n^2	n^2C3
35	} }			
36	if (titik.isEmpty()) {	C7	n	nC7
37	visiteds.pop();	C1	n	nC1
38	jaraks.pop();	C2	n	nC2
39	polylineOptionses.pop();	C3	n	nC3
40	Graph grap = visiteds.pop();	C1	n	nC1
41	double dobel = jaraks.pop();	C5	n	nC5
42	PolylineOptions po = polylineOptionses.pop();	C3	n	nC3
43	stack.add(grap);	C1	n	nC1
44	s_jarak.add(dobel);	C2	n	nC2
45	s_line.add(po);	C3	n	nC3
46	} else {	C9	n	nC9
47	for (int j=0; j<titik.size(); j++) {	C8	n^2	n^2C8
48	stack.add(titik.get(j));	C1	n^2	n^2C1
49	s_jarak.add(jarak.get(j));	C2	n^2	n^2C2
50	s_line.add(line.get(j));	C3	n^2	n^2C1
51	} }			
52	} else {	C9	n	nC9
53	break; }	C10	n	nC10
54	titik.clear();	C1	n	nC1
55	jarak.clear();	C2	n	nC2
56	line.clear(); }	C3	n	nC3
57	for (int j=0; j<jaraks.size(); j++) {	C8	n	nC8
58	total += jaraks.get(j); }	C5	n	nC5
59	if (total < total_dfs) {	C7	1	C7
60	total_dfs = total;	C5	1	C5
61	mMap.clear();	C11	1	C11
62	for (int k = 0; k < polylineOptionses.size(); k++) {	C8	n	C8
63	mMap.addPolyline(polylineOptionses.get(k).color(Color.BLUE)); }	C12	n	nC12
64	mMap.addMarker(marker.get(graph.indexOf(goal)). icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory. HUE_MAGENTA)));	C13	1	C13
65	mMap.addPolyline(line_atm.get(graph.indexOf(goal)). color(Color.MAGENTA)); }	C12	1	C12
66	a = a + "(" + String.format("%.3f",total_dfs) + " metres)";	C4	1	C4
67	txt.setText(a);	C14	1	C14
68	for (int i=0; i<graph.size(); i++) {	C8	n	nC8
69	graph.get(i).visited = false; }	C1	n	nC1

Based on the calculation of the complexity in the table, column C is the process of what happened, column # represents the number of operations executed, and column C# is the result of multiplication between column C and column #. From the calculation of complexity in the table, obtained complexity (T(n)) as follows :

$$T(n) = 4C_1 + 4C_2 + 4C_3 + 2C_4 + 2C_5 + C_7 + C_{11} + C_{12} + C_{13} + C_{14} + 8nC_1 + 5nC_2 + 7nC_3 + 3nC_5 + nC_6 + 2nC_7 + 3nC_8 + 2nC_9 + nC_{10} + nC_{12} + 3n^2C_1 + 2n^2C_2 + 2n^2C_3 + n^2C_5 + n^2C_7 + 2n^2C_8$$

$$T(n) = (4C_1 + 4C_2 + 4C_3 + 2C_4 + 2C_5 + C_7 + C_{11} + C_{12} + C_{13} + C_{14})\, n^0 + (8C_1 + 5C_2 + 7C_3 + 3C_5 + C_6 + 2C_7 + 3C_8 + 2C_9 + C_{10} + C_{12})\, n^1 + (3C_1 + 2C_2 + 2C_3 + C_5 + C_7 + 2C_8)\, n^2$$

$$T(n) = \theta(n^2)$$

In the calculation of complexity based on the table, the depth-first search algorithm complexity value is $\theta(n^2)$.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

14. GREEDY ALGORITHM COMPLEXITY

No	Kode Program	C	#	C#
1	Stack<Graph> visited = new Stack<>();	C1	1	C1
2	Stack<Double> jarak14 = new Stack<>();	C2	1	C2
3	Stack<PolylineOptions> line14 = new Stack<>();	C3	1	C3
4	ArrayList<Graph> titik = new ArrayList<>();	C1	1	C1
5	ArrayList<Double> jarak = new ArrayList<>();	C2	1	C2
6	ArrayList<PolylineOptions> line = new ArrayList<>();	C3	1	C3
7	ArrayList<Graph> titik2 = new ArrayList<>();	C1	1	C1
8	titik2.add(goal);	C1	1	C1
9	String a;	C4	1	C4
10	visited.add(start);	C1	1	C1
11	while (!visited.isEmpty()) {	C5	n	nC5
12	Graph el = visited.pop();	C1	n	nC1
13	visited.add(el);	C1	n	nC1
14	el.visited = true;	C1	n	nC1
15	if (el != goal) {	C6	n	nC6
16	ArrayList<Graph> open_node = el.getNeighbour();	C1	n	nC1
17	ArrayList<Double> open_jarak = el.getDistance();	C2	n	nC2
18	ArrayList<PolylineOptions> open_line = el.getPolylineOptions();	C3	n	nC3
19	for (int i = 0; i < open_node.size(); i++) {	C7	n^2	n^2C7
20	Graph n = open_node.get(i);	C1	n^2	n^2C1
21	Double x = open_jarak.get(i);	C2	n^2	n^2C2
22	PolylineOptions p = open_line.get(i);	C3	n^2	n^2C3
23	if (n != null && !n.visited) {	C6	n^2	n^2C6
24	titik.add(n);	C1	n^2	n^2C1
25	jarak.add(x);	C2	n^2	n^2C2
26	line.add(p);	C3	n^2	n^2C3
27	} }			
28	if (titik.isEmpty()) {	C6	n	nC6
29	visited.pop();	C1	n	nC1
30	jarak14.pop();	C2	n	nC2
31	line14.pop();	C3	n	nC3
32	} else {	C8	n	nC8
33	double min = 4000.0;	C9	n	nC9
34	PolylineOptions pp = null;	C3	n	nC3
35	Graph m = titik.get(0);	C1	n	nC1
36	for (int k = 0; k < titik.size(); k++) {	C7	n^2	n^2C7
37	if (jarak.get(k) < min && !titik.get(k).visited) {	C6	n^2	n^2C6
38	min = jarak.get(k);	C9	n^2	n^2C9
39	pp = line.get(k);	C3	n^2	n^2C3
40	m = titik.get(k);	C1	n^2	n^2C1
41	} }			
42	visited.add(m);	C1	n	C1
43	jarak14.add(min);	C2	n	nC2
44	line14.add(pp); }	C3	n	nC3
45	} else {	C8	n	nC8
46	break; }	C10	n	nC10
47	titik.clear();	C1	n	nC1
48	jarak.clear();	C2	n	nC2
49	line.clear(); }	C3	n	nC3
50	double total = 0.0;	C9	1	C9
51	for (int k=0; k<jarak14.size(); k++) {	C7	n	nC7
52	total += jarak14.get(k); }	C9	n	nC9
53	if (total < total_greedy) {	C6	1	C6
54	total_greedy = total;	C9	1	C9
55	mMap.clear();	C11	1	C11
56	for (int k = 0; k < line14.size(); k++) {	C7	n	nC7
57	mMap.addPolyline(line14.get(k).color(Color.BLUE)); }	C12	n	nC12
58	mMap.addMarker(marker.get(graph.indexOf(goal)). icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory. HUE_MAGENTA)));	C13	1	nC13
59	mMap.addPolyline(line_atm.get(graph.indexOf(goal)). color(Color.MAGENTA)); }	C14	1	nC14
60	a = String.format("%.3f",total_greedy) + " metres";	C4	1	C4
61	txt.setText(a);	C15	1	nC15
62	for (int i=0; i<graph.size(); i++) {	C7	n	nC7
63	graph.get(i).visited = false; }	C1	n	nC1

Based on the calculation of complexity in the table, column C represents what process occurred, column # represents the number of operations executed, and column C# is the result of multiplication between column C and column #. From the calculation of complexity in the table, obtained complexity (T(n)) as follows :

$$T(n) = 4C_1 + 2C_2 + 2C_3 + 2C_4 + C_6 + 2C_9 + C_{11} + C_{13} + C_{14} + C_{15} + 9nC_1 + 4nC_2 + 3nC_3 + nC_5 + 2nC_6 + 3nC_7 + 2nC_8 + 2nC_9 + nC_{10} + nC_{12} + 3n^2C_1 + 2n^2C_2 + 2n^2C_3 + 2n^2C_6 + 2n^2C_7 + n^2C_9$$

$$T(n) = (4C_1 + 2C_2 + 2C_3 + 2C_4 + C_6 + 2C_9 + C_{11} + C_{13} + C_{14} + C_{15})\, n^0 + (9C_1 + 4C_2 + 5C_3 + C_5 + 2C_6 + 3C_7 + 2C_8 + 2C_9 + C_{10} + C_{12})\, n^1 + (3C_1 + 2C_2 + 3C_3 + 2C_6 + 2C_7 + C_9)\, n^2$$

$$T(n) = \theta(n^2)$$

In the calculation of complexity based on the table, the complexity of the greedy algorithm is $\theta(n^2)$.

15. CONCLUSION

The path obtained with the depth-first search algorithm is not necessarily the most optimum path because the depth-first search algorithm searches for points through the first root child and so on until the destination is found. If the destination point is in the first branch, then the path obtained can be the optimum path. If the destination point is not in the first child branch, the path obtained can be the optimum path, and may not be the optimum path.

The path obtained by the greedy algorithm is not necessarily the most optimum path because the greedy algorithm looks for points through children with the lowest distance value without considering the total distance going forward. But the path found is near the optimum path. If the destination point is at the root of the branch with the smallest distance, then the path obtained can be an optimum path. If the destination point is not in the branch of the root of the shortest distance, the path obtained can be the optimum path, and may not the optimum path.

The depth-first search algorithm and the greedy algorithm have the same complexity value, which is $\theta(n^2)$. On testing with ten starting points towards the ATM of Bank Syariah Mandiri, obtained an average distance with a depth-first search algorithm of 3033,555 meters. In contrast, with the greedy algorithm, the average is 2035.2568 meters. Then the greedy algorithm is more effective than the depth-first search algorithm, although the average running time with the depth-first search algorithm is faster than the greedy algorithm, which is 239.9675 ms versus 274.8501 ms. ATM Bank points destination obtained varies depending on the starting point and the algorithm used. Fewer ATM points are selected, the smaller the value of running time algorithm used.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] Ali, Nur Aima. 2017. *Penerapan Algoritma Genetika dan Perbandingannya dengan Algoritma Greedy dalam Penyelesaian Knapsack Problem*. Skripsi. Fakultas Sains dan Teknologi. Universitas Islam Negeri Alauddin: Makassar.

[2] Anasta, Bayu Angga. 2018. *Analisis Perbandingan Algoritma Run-Length Encoding dan Algoritma Fibonacci Code dalam Kompresi Citra*. Skripsi. Fakultas Ilmu Komputer dan Teknologi Informasi. Universitas Sumatera Utara: Medan.

[3] Bismantoko, Demas Haryo, Sriyanto dan Wiwik Budiman. (2015). Sistem Informasi Transportasi Umum Terintegrasi di Kota Semarang Menggunakan Algoritma Depth first search (DFS). *Jurnal Teknik Industri, x*(x), 1-9.Boneh, Dan, and Ramarathnam Venkatesan. "Breaking RSA may not be equivalent to factoring." *Advances in Cryptology—EUROCRYPT'98* (1998): 59-71.

[4] Erika, Orien Rindy, Didik Kurniawan, dan Febi Eka Febriansyah. (2016). Aplikasi Pencarian Letak ATM Berbasis Android dengan GIS di Kota Bandar Lampung. *Jurnal Komputasi*, *4*(1), 27-35.

[5] Gautama, Elliana. (2017, 30 Juni). *Kompleksitas Algoritma*. Diakses 2 November 2018, dari https://dosen.perbanas.id/kompleksitas-algoritma/

[6] Prasetiyo, Budi, dan Maulidia Rahmah Hidayah. (2014). Penngunaan Metode Depth First Search (DFS) dan Breadth First Search pada Strategi game Kamen Rider Decade Versi 0.3. *Scientific Journal of Informatics, 1(2), 161-167*.

[7] Jiang, Tuping, Gang Ren, and Xing Zhao. (2013). Evacuation Route Optimization Based on Tabu Search Algorithm and Hill-Climbing Algorithm. *Procedia Social and Behavioral Sciences, 96, 865-872*.

[8] Lestari, Sasti. 2018. *Perbandingan Algoritma Floyd-Warshall dan Bellman-Ford dalam Pencarian Jarak Terpendek Antar ATM di Kota Tebing Tinggi*. Skripsi. Fakultas Ilmu Komputer dan Teknologi Informasi. Universitas Sumatera Utara: Medan.

[9] Lubis, Henny Syahriza. 2009. *Perbandingan Algoritma Greedy dan Djikstra untuk Menentukan Lintasan Terpendek*. Skripsi. Fakultas Matematika dan Ilmu Pengetahuan Alam. Universitas Sumatera Utara: Medan.

[10] Munir, Rinaldi. 2014. *Matematika Diskrit*. Bandung: Informatika Bandung.

[11] Pribadi, Feddy Setio dan Anggraini Mulwinda. (2011). Pencarian Rute Terpendek dengan Menggunakan Algoritma Depth First, Breadth First, dan Hill Climbing (Studi Comparative). *Jurnal Sains dan Teknologi, 9(1), 1-10*.

[12] Priyantoro, Aris, dan Khabib Mustofa. (2014). Pengembangan Aplikasi Pencarian Rute Terbaik Pada Sistem Operasi Android (Studi Kasus Rute Trans Jogja). *Berkala MIPA, 24(1), 72-88*.

[13] Verma, Madhusi, and K.K. Shukla. (2013). A Greedy Algorithm for Fuzzy Shortest Path Problem using Quasi-Gaussian Fuzzy Weights. *International Journal of Fuzzy System Applications, 3(2), 55-70*.

[14] Saputra, Ilham. 2018. *Implementasi Algoritma Simple Hill Climbing dan Algoritma Djikstra untuk Mencari Jarak Terpendek*. Skripsi. Fakultas Ilmu Komputer dan Teknologi Informasi. Universitas Sumatera Utara: Medan.

[15] Belalawe, Benyamin Jago, M. Suyanto, dan Amir Fatah Sofyan. (2012). Penentuan Jalur Wisata Terpendek Menggunakan Metode Forward Chaining (Studi Kasus Dinas Pariwisata Kota Kupang). *Jurnal Informatika, C, 9-16.*

[16] Deng, Xinguo, et al. (2010). Combining Breadth-First with Depth-First Search Algorithms for VLSI Wire Routing. *International Cpnference on Advanced Computer Theory and Engineering, 6, 482-486*.

[17] B. V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: theory and experimental evaluation. Mathematical Programming, 73:129-174, 1996.

[18] Dian Rachmawati and Lysander Gustin 2020 J. Phys.: Conf. Ser. 1566 012061