Aplikasi Algoritma UCS dalam Pencarian Susunan Perjalanan Kereta Api dengan Harga Tiket Termurah dari Bandung ke Banyuwangi

Eka Novendra Wahyunadi 13517011
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ekanovendra726@gmail.com

***Abstrak*—Di era informasi ini strategi algoritma berkembang dengan pesat, seiring meningkatnya perkembangan strategi algoritma dalam penyelesaian masalah, implementasi yang dapat dilakukan pun bertambah, karena itulah peluang strategi algoritma dalam membantu menyelesaikan masalah semakin besar, karena selain perkembangan pesat tersebut, implementasi dari strategi algoritma tidak harus berskala besar, bisa dimulai dari hal - hal yang ditemui sehari – hari, seperti masalah pencarian perjalanan dengan harga tiket kereta api termurah untuk mahasiswa Institut Teknologi Bandung yang berasal dari Banyuwangi, masalah tersebut disebabkan oleh belum adanya layanan kereta api yang berangkat dari Bandung dan turun di salah satu stasiun di Banyuwangi, masalah ini dapat diselesaikan menggunakan Algoritma *UCS(Uniform-Cost Search)* yang merupakan variasi dari Algoritma *Dijkstra*.**

Kata kunci—UCS;Algoritma; Dijkstra; Graf;

I. PENDAHULUAN

UCS(Uniform-Cost Search) adalah sebuah algoritma pencarian yang merupakan variasi dari Algoritma pencarian *Dijkstra*. Dalam Algoritma UCS, semua simpul tidak langsung dimasukkan dalam *priority queue*, melainkan jalur yang berada di sumber dimasukkan, dan satu – persatu memasukkan jalur lain jika diperlukan. Dalam setiap langkah di algoritma ini, dilakukan pengecekan apakah sebuah simpul sudah pernah berada di *priority queue* (contohnya menggunakan *array* untuk mencatat simpul yang sudah dikunjungi), jika sebuah simpul sudah berada dalam *queue*, maka dilakukan penghapusan simpul, dan jika simpul tersebut belum pernah berada dalam *priority queue*, maka simpul tersebut dimasukkan dalam *priority queue* tersebut. Variasi dari Algoritma *Dijkstra* ini berguna untuk graf – graf berukuran besar yang terlalu besar untuk direpresentasikan di memori.

Setiap akhir semester banyak mahasiswa Institut Teknologi Bandung yang kembali ke kampung halaman masing – masing dalam rangka liburan. Moda transportasi yang digunakan oleh mahasiswa – mahasiswa tersebut beragam, seperti pesawat terbang, kereta api, bus, dan lain lain. Mahasiswa Institut Teknologi Bandung yang berasal dari Banyuwangi dan ingin pulang ke kampung halamannya tersebut terkadang harus berpikir beberapa kali sebelum membeli tiket pulang jika menggunakan moda transportasi kereta api, dikarenakan belum ada layanan kereta api yang melayani pemberangkatan dari salah satu stasiun di Bandung dan berhenti di salah satu stasiun di Banyuwangi, oleh karena itu mahasiswa Institut Teknologi Bandung yang berasal dari Banyuwangi tersebut harus membeli tiket transit terlebih dahulu, dan sebagian mahasiswa ingin mendapatkan harga termurah yang mungkin dari tiket tersebut. Masalah pemilihan tiket termurah ini dapat diselesaikan menggunakan algoritma UCS.

II. DASAR TEORI

A. Algoritma Dijkstra

Algoritma *Dijkstra* ditemukan oleh seorang ilmuwan komputer bernama Edsger Dijkstra. Algoritma adalah sebuah algoritma rakus (*greedy algorithm*) yang dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah graf berarah (*directed graph*) dengan bobot-bobot sisi (*edge weights*) yang nilainya tidak negatif.

Misalnya, bila terdapat simpul dari sebuah graf melambangkan kota - kota dan bobot sisi (*edge weights*) melambangkan jarak antara kota - kota tersebut, maka algoritma *Dijkstra* dapat digunakan untuk menemukan jarak terpendek antara dua kota.

Algoritma ini menerima input berupa sebuah graf berarah yang berbobot (*weighted directed graph*) G dan sebuah simpul awal s dengan G dan V adalah himpunan semua simpul dalam graf G.

Setiap sisi dari graf ini adalah pasangan simpul (u,v) yang melambangkan hubungan dari simpul u ke simpul v. Himpunan semua sisi disebut E.

Bobot *(weights)* dari semua sisi dihitung dengan fungsi sebagai berikut:

$$W = E \rightarrow [0, \infty)$$

Sehingga w(u,v) adalah jarak tak-negatif dari simpul u ke simpul v.

Ongkos (*cost*) dari sebuah sisi dapat dianggap sebagai jarak antara dua simpul, yaitu jumlah jarak semua sisi dalam jalur tersebut. Untuk sepasang simpul s dan t dalam V, algoritma ini menghitung jarak terpendek dari s ke t.[3]

Berikut adalah contoh *pseudocode* dari algoritma *Dijkstra* ini:

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:                 // Initialization
        dist[v] := infinity                     // initial distance from source to vertex v is set to infinite
        previous[v] := undefined                // Previous node in optimal path from source
    dist[source] := 0                           // Distance from source to source
    Q := the set of all nodes in Graph          // all nodes in the graph are unoptimized - thus are in Q
    while Q is not empty:                       // main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:               // where v has not yet been removed from Q.
            alt := dist[u] + dist_between(u, v)
            if alt < dist[v]                    // Relax (u,v)
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

Gambar 1. *Pseudocode* Algoritma *Dijkstra*[1]

B. *Algoritma UCS*

Uniform-*Cost Search* mirip dengan algoritma *Dijkstra*. Dalam algoritma ini dari keadaan awal akan dikunjungi simpul yang berdekatan dan akan memilih simpul yang memiliki *cost* paling sedikit, kemudian kita akan memilih simpul yang memiliki *cost* paling sedikit berikutnya dari semua simpul yang belum dikunjungi dan berdekatan dari simpul yang sudah dikunjungi, dengan cara ini kita akan mencoba untuk mencapai simpul tujuan. Algoritma ini akan menyimpan *priority queue* yang akan memberikan simpul dengan *cost* paling sedikit berikutnya dari semua simpul yang berdekatan dari simpul yang sudah dikunjungi.[4]

```
Input: Graph G = (V, E), Start state s, Set of goal states F
Output: Path P ⊆ E
1 begin
2 let Frontier = {s};
3 let Explored = {s};
4 while Frontier != [] do
5 let v ∈ Frontier, such that Path(s,v) has lowest cost;
6 let Frontier = Frontier\{v};
7 if v ∈ F then
8 return Path(s,v)
9 end
10 else
11 let Explored = {v}∪ Explored;
12 let Frontier = Frontier∪{v-subscript(1), . . . , v-
subscript(n)},
where (v, v-subscript(i)) ∈ E and v-subscript(i) !∈
Explored.
13 end
14 end
15 return failure;
16 end
```

Gambar 2. *Pseudocode* Algoritma UCS[2]

III. METODE APLIKASI ALGORITMA UCS

Metode aplikasi dari Algoritma UCS ini dalam menyelesaikan kasus perjalanan dengan tiket termurah ini adalah dengan cara memisalkan kota keberangkatan dan kota tujuan sebagai simpul – simpul, lalu harga tiket sebagai bobot dari sisi antara simpul yang memisalkan kereta api dan simpul yang memisalkan tempat tujuan. Untuk bobot dari simpul yang memisalkan tempat keberangkatan ke simpul yang memisalkan kereta api, diberikan nilainya 1. Alasan mengapa nilai untuk bobot dari simpul yang memisalkan tempat keberangkatan ke simpul yang memisalkan kereta api diberikan nilainya 1, adalah karena nilai 0 dipakai untuk mengecek apakah suatu simpul dengan simpul lain memiliki sisi yang menghubungkan antara keduanya.

Setelah menentukan simpul awal dan simpul akhir beserta bobot sisinya, pasangan simpul awal, simpul akhir, dan bobotnya dimasukkan ke dalam sebuah *adjacentMatrix*, dengan cara untuk setiap pasangan simpul awal i dan simpul akhir j, nilai dari bobot sisinya dimasukkan ke dalam elemen *matrix* yang memiliki nilai baris i dan nilai kolom j.

Setelah dimasukkan ke dalam *adjacentMatrix*, data tersebut harus dimasukkan ke dalam algoritma UCS yang sudah dibuat, dalam metode yang saya gunakan, fungsi serta prosedur yang digunakan adalah sebagai berikut :

- Fungsi uniformCostSearch yang mencari jalur perjalanan dengan *cost* paling sedikit, dan mengembalikan nilai harga perjalanan tersebut.
- Prosedur addFrontiersToQueue yang berfungsi memasukkan simpul – simpul yang sudah dikunjungi ke dalam sebuah himpunan yang berisi simpul yang dikunjungi. Selain itu prosedur ini juga berfungsi untuk memasukkan simpul saat ini ke dalam *priority queue* untuk diproses lebih lanjut.
- Fungsi getMinCost yang berfungsi untuk menghapus simpul yang akan diekspansi dalam *priority queue* dan mengembalikan simpul tersebut untuk diproses lebih lanjut.
- Fungsi printPath yang berfungsi mencetak jalur perjalanan dengan harga tiket termurah hasil dari pengolahan Algoritma UCS.

IV. SIMULASI APLIKASI ALGORITMA UCS

Dalam simulasi ini saya mencoba mengimplementasikan algoritma tersebut dalam bahasa pemrograman Java, program ini menerima input berupa file yang dibaca pada program dan berisi daftar kemungkinan kereta dan harga untuk beberapa tempat transit, dan menjalankan aplikasi berdasarkan file tersebut. Berikut adalah tabel yang berisi daftar harga yang saya gunakan dalam simulasi ini.

TABEL 1. DAFTAR HARGA KERETA

Tempat Berangkat	Tempat Tujuan	Nama Kereta Api	Harga (dalam Rupiah)
Bandung	Surabaya	Argo Wilis	480.000
Bandung	Surabaya	Mutiara Selatan	335.000
Bandung	Surabaya	Turangga	480.000
Bandung	Malang	Malabar	260.000
Bandung	Yogyakarta	Lodaya	200.000
Bandung	Yogyakarta	Pasundan	88.000
Bandung	Malang	Mutiara Selatan	350.000
Surabaya	Banyuwangi	Probowangi	56.000
Surabaya	Banyuwangi	Wijayakusuma	170.000
Malang	Banyuwangi	Tawang Alun	62.000
Yogyakarta	Banyuwangi	Sri Tanjung	94.000

Dari tabel tersebut, dapat dibuat file harga.txt yang isinya sebagai berikut:

```
1 6 1
1 7 1
1 8 1
1 9 1
1 10 1
1 11 1
1 12 1
6 2 480
7 2 335
8 2 480
9 2 260
10 3 350
11 4 88
12 4 200
4 13 1
2 14 1
2 15 1
3 16 1
13 5 94
14 5 170
15 5 56
16 5 62
```

(File harga.txt)

Dari file harga.txt tersebut dapat diperoleh data berupa adjacentMatrix yang isinya sebagai berikut:

0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 480 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 335 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 480 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 260 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 350 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 88 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 200 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 94 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 17 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 56 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 62 0 0 0 0 0 0 0 0 0 0

Matriks inilah yang selanjutnya akan diproses untuk menemukan harga tiket termurah dengan mempergunakan algoritma UCS.

Untuk kode sumber dari program yang saya gunakan dalam simulasi ini, terdiri dari kelas yang memiliki fungsi dan prosedur sebagai berikut:

```
/*
Nama : Eka Novendra Wahyunadi
NIM : 13517011
 */

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.*;

public class UCSBandungBanyuwangi
{
    private PriorityQueue<Node> priorityQueue;
    private Set<Integer> dikunjungi;
    private int[] harga;
    private int numberOfNodes;
    private int[][] adjacencyMatrix;
    private LinkedList<Integer> path;
    private int[] parent;
    private int source, destination;
    private static final int MAX_VALUE =
1000000;
```

```
    private UCSBandungBanyuwangi(int
numberOfNodes)
    {
        this.numberOfNodes = numberOfNodes;
        this.dikunjungi = new HashSet<>();



 this.priorityQueue =
        new PriorityQueue<>(numberOfNodes,
        new Node());
        this.harga =
        new int[numberOfNodes + 1];
        this.path = new LinkedList<>();
        this.adjacencyMatrix =
new int[numberOfNodes + 1][numberOfNodes + 1];
        this.parent =
        new int[numberOfNodes + 1];
    }

    private int uniformCostSearch(int[][]
adjacencyMatrix, int source, int destination)
    {
        int evaluationNode;
        this.source = source;
        this.destination = destination;

        for (int i = 1; i <=
        numberOfNodes; i++)
        {
            harga[i] = MAX_VALUE;
        }

        for (int sourcevertex = 1;
        sourcevertex <= numberOfNodes;
        sourcevertex++)
        {
            if (numberOfNodes >= 0)

System.arraycopy(adjacencyMatrix[sourcevertex]
, 1, this.adjacencyMatrix[sourcevertex], 1,
numberOfNodes);
        }

        priorityQueue.add(
        new Node(source, 0));
        harga[source] = 0;

        while (!priorityQueue.isEmpty())
        {
            evaluationNode = getMinCost();
            if (evaluationNode == destination)
            {
                return harga[destination];
            }
            dikunjungi.add(evaluationNode);

addFrontiersToQueue(evaluationNode);
        }
        return harga[destination];
    }

    private void addFrontiersToQueue(int
evaluationNode)
    {
        for (int destination = 1; destination
<= numberOfNodes; destination++)
        {
            if
(!dikunjungi.contains(destination))
            {
                if
(adjacencyMatrix[evaluationNode][destination]
!= MAX_VALUE)
                {
                    if (harga[destination] >
adjacencyMatrix[evaluationNode][destination]
                                +
harga[evaluationNode])
                    {
                        harga[destination] =
adjacencyMatrix[evaluationNode][destination]
                                    +
harga[evaluationNode];
                        parent[destination] =
evaluationNode;
                    }

                    Node node = new
Node(destination, harga[destination]);

priorityQueue.remove(node);
                    priorityQueue.add(node);
                }
            }
        }
    }

    private int getMinCost()
    {
        Node node = priorityQueue.remove();
        return node.node;
    }

    private void printPath()
    {
        path.add(destination);
        boolean found = false;
        int vertex = destination;
        while (!found)
        {
            if (vertex == source)
            {
                found = true;
                continue;
            }
            path.add(parent[vertex]);
            vertex = parent[vertex];
        }
```

```
    System.out.println("Jalur dari tiket
     kereta termurah adalah");
    Iterator<Integer> iterator =
     path.descendingIterator();
    String x = "";
    while (iterator.hasNext())
    {

        switch (iterator.next()){
            case 1:x = "Bandung";
                     break;
            case 2:x = "Surabaya";
                     break;
            case 3:x = "Malang";
                     break;
            case 4:x = "Yogyakarta";
                     break;
            case 5:x = "Banyuwangi";
                     break;
            case 6:x = "Argo Wilis";
                     break;
            case 7:x = "Mutiara Selatan";
                     break;
            case 8:x = "Turangga";
                     break;
            case 9:x = "Malabar";
                     break;
            case 10:x = "Mutiara Selatan";
                     break;
            case 11:x = "Pasundan";
                     break;
            case 12:x = "Lodaya";
                     break;
            case 13:x = "Sri Tanjung";
                     break;
            case 14:x = "Wijayakusuma";
                     break;
            case 15:x = "Probowangi";
                     break;
            case 16:x = "Tawang Alun";
                     break;
        }
        if (x!="Banyuwangi") {
            System.out.print(x + " - ");
        }else{
            System.out.println(x);
        }
    }
}

public static void main(String[] args)
{
    int[][] adjacency_matrix;
    int number_of_vertices;
    int source;
    int destination;
    int hargatotal;
    adjacency_matrix = new int[17][17];
        for (int i = 1; i <= 16; i++)
        {
            for (int j = 1; j <= 16; j++)
            {
            adjacency_matrix[i][j]=0;
            }
        }
        try{
            BufferedReader in =
         new BufferedReader(new
         FileReader("harga.txt"));
            String s;
            while((s = in.readLine())
                 != null){
                String[] var = s.split(" ");

adjacency_matrix[Integer.parseInt(var[0])]
[Integer.parseInt(var[1])] =
Integer.parseInt(var[2]);
            }

        }catch(Exception e){
            e.printStackTrace();
        }

        try
        {
            number_of_vertices = 16;

            for (int i = 1; i <= 16; i++)
            {
                for (int j = 1; j <= 16; j++)
                {
                    if (i == j)
                    {
                        adjacency_matrix[i][j]
                          = 0;
                        continue;
                    }
                    if (adjacency_matrix[i][j]
                          == 0)
                    {
                        adjacency_matrix[i][j]
                          = MAX_VALUE;
                    }
                }
            }
            source = 1;
            destination = 5;

            UCSBandungBanyuwangi
         uniformCostSearch = new
         UCSBandungBanyuwangi
         (number_of_vertices);
            hargatotal =
uniformCostSearch.uniformCostSearch
(adjacency_matrix,source, destination)- 2;
            uniformCostSearch.printPath();
```

```
            System.out.println("\nHarga tiket
            termurah adalah  Rp" + hargatotal +
            ".000");

            } catch (InputMismatchException
            inputMismatch)
            {
                System.out.println("Error");
            }
        }
}

class Node implements Comparator<Node>
{
    int node;
    private int cost;

    Node()
    {
    }

    Node(int node, int cost)
    {
        this.node = node;
        this.cost = cost;
    }

    @Override
    public int compare(Node node1, Node node2)
    {
        if (node1.cost < node2.cost)
            return -1;
        if (node1.cost > node2.cost)
            return 1;
        if (node1.node < node2.node)
            return -1;
        return 0;
    }

    @Override
    public boolean equals(Object obj)
    {
        if (obj instanceof Node)
        {
            Node node = (Node) obj;
            return this.node == node.node;
        }
        return false;
    }
}
```

hasil eksekusi dari program tersebut adalah sebagai berikut:

```
Jalur dari tiket kereta termurah adalah
Bandung - Pasundan - Yogyakarta - Sri Tanjung - Banyuwangi

Harga tiket termurah adalah  Rp182.000
```

V. KESIMPULAN

Aplikasi algoritma UCS terbukti dapat menyelesaikan kasus pencarian harga tiket termurah dari Bandung ke Banyuwangi. Aplikasi dari algoritma UCS ini tentu saja akan sangat berguna, khususnya mahasiswa Institut Teknologi Bandung yang berasal dari Banyuwangi. Selain untuk mahasiswa Institut Teknologi Bandung yang berasal dari Banyuwangi, tentu saja aplikasi ini juga berguna untuk mahasiswa – mahasiswa di seluruh dunia untuk mencari harga tiket termurah jika mereka harus melakukan transit terlebih dahulu sebelum sampai ke kota tujuan . Dari hasil simulasi didapat bahwa harga tiket paling murah saat ini untuk perjalanan dari Bandung ke Banyuwangi adalah Rp182.000 , dengan rincian perjalanan menaiki kereta api pasundan dari Bandung menuju Yogyakarta, lalu dilanjutkan dengan menaiki kereta api Sri Tanjung dari Yogyakarta menuju Banyuwangi. Jika dilihat dari Tabel 1 pada halaman 3, rincian harga untuk perjalanan tersebut adalah Rp88.000 untuk kereta Pasundan dan Rp94.000 untuk kereta Sri Tanjung. Aplikasi dari algoritma ini masih dapat dikembangkan lebih lanjut, seperti menggabungkannya dengan algoritma lain untuk menyelesaikan masalah waktu menunggu kereta api transit, atau mungkin pencarian jadwal kereta api yang paling cepat, dan mungkin saja menyelesaikan masalah pencarian perjalanan paling nyaman untuk penumpang yang ingin beristirahat. Perkembangan dari strategi algoritma masih belum berhenti, dan masih banyak aplikasi yang dapat menyelesaikan masalah ke depannya.

UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa, karena atas limpahan rahmat dan karunia-Nya penulis bisa menulis dan menyelesaikan makalah ini dalam kondisi yang sehat. Penulis juga ingin berterima kasih kepada kedua orang tua penulis, karena berkat bantuan dan dukungan mereka penulis bisa mendapatkan ilmu dan pengetahuan hingga saat ini. Selain itu penulis juga ingin berterima kasih kepada penemu algoritma – algoritma yang saya gunakan, karena melalui penemuannya penulis dapat memahami konsep – konsep penyelesaian masalah yang menjadi dasar dari makalah ini. Tak lupa penulis juga mengucapkan terima kasih kepada para penulis sumber referensi yang telah memberikan saya ilmu yang dibutuhkan untuk menyelesaikan makalah ini.

REFERENSI

[1] H. Thomas Grossmann (Overall), "Dijkstra Algorithm: Short terms and Pseudocode", Gitta.info, 2019. [Online]. Available: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html. [Accessed: 26- Apr- 2019]. [3]H. Thomas Grossmann (Overall), "Dijkstra Algorithm: Short terms and Pseudocode", Gitta.info, 2019. [Online]. Available: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html. [Accessed: 26- Apr- 2019].

[2] U. notation and J. Scofield, "Uniform Cost Search algorithm pseudocode/ocaml notation", Stack Overflow, 2019. [Online]. Available: https://stackoverflow.com/questions/33199587/uniform-cost-search-algorithm-pseudocode-ocaml-notation. [Accessed: 26- Apr- 2019].

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.3: Dijkstra's algorithm, pp.595–601.

[4] "Uniform-Cost Search (Dijkstra for large Graphs) - GeeksforGeeks", GeeksforGeeks, 2019. [Online]. Available: https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/. [Accessed: 26- Apr- 2019].

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, 26 April 2019

Eka Novendra Wahyunadi
13517011