Tugas Kecil 3 IF2211 Strategi Algoritma

Penyelesaian Permainan Word Ladder Menggunakan Algoritma UCS, Greedy Best First Search, dan A*

Semester II Tahun 2023/2024

Disusun oleh

Muhammad Dava Fathurrahman 13522114

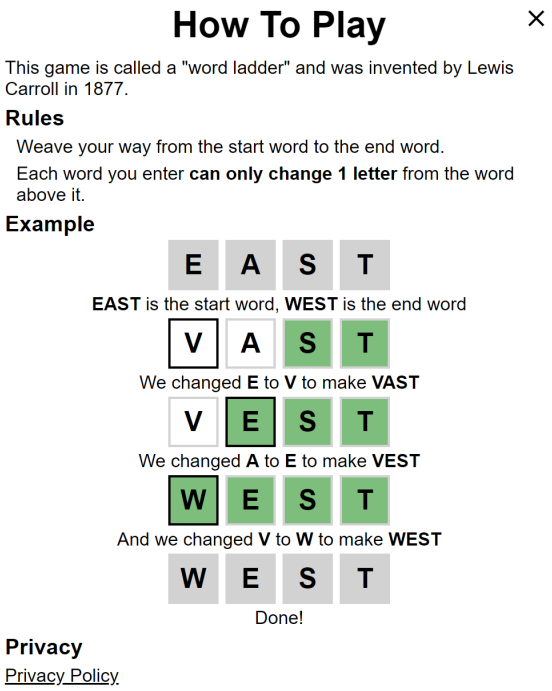
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

I. Deskripsi Tugas

Word ladder (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragrams, laddergrams, atau word golf) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. Word ladder ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara start word dan end word. Banyaknya huruf pada start word dan end word selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.



Gambar 1. Ilustrasi dan Peraturan Permainan Word Ladder
(Sumber: https://wordwormdormdork.com/)

Tentu akan sangat banyak rantai kata dari start word menuju end word sehingga memerlukan waktu yang lebih banyak. Oleh karena itu, Anda diminta untuk membuat sebuah solver permainan tersebut dengan harapan menemukan solusi paling optimal untuk menyelesaikan permainan Word Ladder ini.

II. Implementasi Algoritma

A. Uniform Cost Search

Uniform Cost Search (UCS) adalah metode penelusuran graf yang mempertimbangkan biaya atau cost yang terkait dengan simpul awal. Algoritma ini memulai pencarian dari root node, kemudian dilanjutkan ke node-node

selanjutnya. Simpul yang diekspansi merupakan simpul dengan harga (cost) kumulatif terendah. Ongkos kumulatif simpul n didefinisikan sebagai g(n), yaitu jarak dari akar ke simpul n.

Implementasi algoritma UCS pada penyelesaian permain Word Ladder adalah sebagai berikut,

1. Masukan start word akan dicek apakah tersedia pada kamus lokal. Jika tidak, pesan pemberitahuan akan ditampilkan pada layar.
2. Start word dimasukkan ke dalam priority queue dengan harga nol.
3. Jika priority queue tidak kosong, elemen terdepan priority queue (dijamin terurut mulai dari harga terendah) dikeluarkan
4. Jika simpul yang dikeluarkan adalah simpul tujuan, proses dihentikan. Jika tidak, lanjut ke langkah ke-5 untuk ekspansi simpul.
5. Simpul dimasukkan ke dalam senarai visited dan diekspansi ke simpul tetangganya yang belum pernah dikunjungi, dengan harga kumulatif ditambah satu.
6. Seluruh simpul ekspansi dimasukkan ke priority queue dan kembali ke langkah ke-3.

B. Greedy Best First Search

Greedy Best First Search (GBFS) adalah metode penelusuran graf yang mempertimbangkan biaya atau cost yang terkait dengan simpul awal. Algoritma ini memulai pencarian dari root node, kemudian dilanjutkan ke node-node selanjutnya. Simpul yang diekspansi merupakan simpul dengan harga (cost) terendah. Ongkos simpul n didefinisikan sebagai h(n), yaitu banyaknya huruf yang berbeda dengan huruf pada kata tujuan pada posisi yang sama.

Implementasi algoritma UCS pada penyelesaian permain Word Ladder adalah sebagai berikut,

1. Masukan start word akan dicek apakah tersedia pada kamus lokal. Jika tidak, pesan pemberitahuan akan ditampilkan pada layar.
2. Start word dimasukkan ke dalam priority queue.
3. Jika priority queue tidak kosong, elemen terdepan priority queue (dijamin terurut mulai dari harga terendah) dikeluarkan
4. Jika simpul yang dikeluarkan adalah simpul tujuan, proses dihentikan. Jika tidak, lanjut ke langkah ke-5 untuk ekspansi simpul.
5. Simpul dimasukkan ke dalam senarai visited dan diekspansi ke simpul tetangganya yang belum pernah dikunjungi.

6. Seluruh simpul ekspansi dimasukkan ke priority queue dan kembali ke langkah ke-3.

C. A* Search (A Star Search)

A* Search adalah metode penelusuran graf yang mempertimbangkan biaya atau cost yang terkait dengan simpul awal. Algoritma ini memulai pencarian dari root node, kemudian dilanjutkan ke node-node selanjutnya. Simpul yang diekspansi merupakan simpul dengan harga (cost) kumulatif terendah ditambah estimasi harga dari simpul n ke simpul tujuan. Ongkos kumulatif simpul n didefinisikan sebagai f(n) = g(n) + h(n), yaitu jarak dari akar ke simpul n.

Implementasi algoritma A* Search pada penyelesaian permain Word Ladder adalah sebagai berikut,

1. Masukan start word akan dicek apakah tersedia pada kamus lokal. Jika tidak, pesan pemberitahuan akan ditampilkan pada layar.
2. Start word dimasukkan ke dalam priority queue dengan harga nol.
3. Jika priority queue tidak kosong, elemen terdepan priority queue (dijamin terurut mulai dari harga terendah) dikeluarkan
4. Jika simpul yang dikeluarkan adalah simpul tujuan, proses dihentikan. Jika tidak, lanjut ke langkah ke-5 untuk ekspansi simpul.
5. Simpul dimasukkan ke dalam senarai visited dan diekspansi ke simpul tetangganya yang belum pernah dikunjungi, dengan harga kumulatif ditambah satu.
6. Seluruh simpul ekspansi dimasukkan ke priority queue dan kembali ke langkah ke-3.

III. Kode Sumber

A. Kelas Node

Kelas Node merepresentasikan simpul dalam struktur pohon dengan atribut kata (word) dan simpul induk (parent), serta memiliki metode untuk mengakses kata, mendapatkan simpul induk, dan mendapatkan jalur dari akar ke simpul saat ini dalam bentuk string. Kelas Node digunakan pada metode pencarian Greedy Best First Search (GBFS) karena simpul tidak perlu menyimpan informasi harga kumulatif.

```
public class Node {
    protected String word;
    protected Node parent;

    public Node(String word, Node parent){
```

```
        this.word = word;
        this.parent = parent;
    }

    public String getWord(){
        return this.word;
    }

    public Node getParent(){
        return this.parent;
    }

    public String get_path_from_root(){
        if (this.parent == null){
            return this.word;
        }

        return this.parent.get_path_from_root() + " " + this.word;
    }
}
```

B. Kelas ExtendedNode

Kelas ExtendedNode merupakan turunan dari kelas Node yang menambahkan atribut jarak (distance) dan memiliki metode untuk mengakses jarak serta mendapatkan jalur dari akar ke simpul saat ini dalam bentuk string. Kelas ini digunakan pada metode pencarian UCS dan A* karena simpul perlu menyimpan informasi harga kumulatif.

```
public class ExtendedNode extends Node{
    private int distance;

    public ExtendedNode(String word, Node parent, int distance){
        super(word, parent);
        this.distance = distance;
    }

    public int get_distance(){
        return this.distance;
    }

    @Override
    public String get_path_from_root(){
        if (this.parent == null){
            return this.word;
        }
```

```
        return this.parent.get_path_from_root() + " " + this.word;
    }
}
```

C. Kelas App

Kelas App merupakan kelas titik masuk program yang terdiri atas konstruktor pembangun antarmuka pengguna grafis, metode memuat fail kamus, metode mencari simpul tetangga.

```
public class App extends JFrame {
    public static Set<String> dictionary;

    private JTextField startWordField;
    private JTextField endWordField;
    private JComboBox<String> algorithmComboBox;
    private JTextPane resultPane; // Text area to display ladder
information

    public App() {
        setTitle("Word Ladder Solver");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(450, 400);
        setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.insets = new Insets(5, 5, 5, 5);

        searchButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String startWord =
startWordField.getText().toUpperCase();
                String endWord = endWordField.getText().toUpperCase();
                String searchAlgorithm = (String)
algorithmComboBox.getSelectedItem();

                // Get dictionary words with same length from the file
and store them in a set
                loadDictionary("src/dictionary.txt", startWord);

                // Perform the search algorithm and update the result
area
                performSearch(startWord, endWord, searchAlgorithm);
            }
```

```
        });
    }

    private void performSearch(String startWord, String endWord, String
searchAlgorithm) {
        StringBuilder result = new StringBuilder();

        // Check for valid input and dictionary initialization
        if (!isValidInput(startWord, endWord)) {
            resultPane.setText("Invalid input or dictionary not
initialized.");
            return;
        }

        // Check if the start word and the end word are in the dictionary
        if (!dictionary.contains(startWord) ||
!dictionary.contains(endWord)) {
            resultPane.setText("Start word or end word not in the
dictionary.");
            return;
        }

        long startTime = System.currentTimeMillis();
        long endTime;
        long executionTime;

        App.Pair<Integer, List<String>> wordLadder = null;

        switch (searchAlgorithm) {
            case "Uniform Cost Search (UCS)":
                UCS ucs = new UCS(endWord);
                ExtendedNode startUCSNode = new ExtendedNode(startWord,
null, 0);
                wordLadder = ucs.search(startUCSNode);

                break;
            case "Greedy Best First Search (GBFS)":
                GBFS gbfs = new GBFS(endWord);
                Node startGFBSNode = new Node(startWord, null);
                wordLadder = gbfs.search(startGFBSNode);

                break;
            case "A* Search":
                AStar astar = new AStar(endWord);
                ExtendedNode startAStarNode = new ExtendedNode(startWord,
null, 0);
                wordLadder = astar.search(startAStarNode);

                break;
```

```
        }

        endTime = System.currentTimeMillis();

        if (!wordLadder.getSecond().isEmpty()){
            int size = wordLadder.getSecond().size();
            for (int i = 0; i < size; i++) {
                String word = wordLadder.getSecond().get(i);
                if (i == size - 1) {
                    result.append("<font color='green'>[" + i + "] " +
word + "</font>");
                }
                else if (i == 0) {
                    result.append("<font color='blue'>[" + i + "] " +
word + "</font>");
                }
                else {
                    result.append(word);
                }
                result.append("<br>");
            }
        }
        else{
            result.append("No ladder found.");
        }

        executionTime = endTime - startTime;
        result.append("<br>" + wordLadder.getFirst() + " node has been
visited in " + executionTime + " ms.");

        resultPane.setText(result.toString());
    }

    private boolean isValidInput(String startWord, String endWord) {
        if (!startWord.matches("[A-Z]+") || !endWord.matches("[A-Z]+")) {
            return false; // Alphabetic check failed
        }

        if (dictionary == null) {
            return false; // Dictionary not initialized
        }

        if (startWord.length() != endWord.length()) {
            return false; // Word length mismatch
        }

        return true;
    }
```

```
    private static void loadDictionary(String filePath, String startWord)
{
        dictionary = new HashSet<>();
        try {
            File file = new File(filePath);
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String word = scanner.nextLine();
                if (word.length() == startWord.length()) {
                    dictionary.add(word.toUpperCase());
                }
            }
            scanner.close();
        }
        catch (FileNotFoundException e) {}
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            App app = new App(); // Create an instance of the application
            app.setVisible(true);
        });
    }


    public static List<String> getNeighbor(String word) {
        List<String> nextWords = new ArrayList<>();
        for (int i = 0; i < word.length(); i++) {
            char[] chars = word.toCharArray();
            for (char c = 'A'; c <= 'Z'; c++) {
                chars[i] = c;
                String newWord = new String(chars);
                if (!newWord.equals(word) &&
dictionary.contains(newWord)) {
                    nextWords.add(newWord);
                }
            }
        }

        return nextWords;
    }
}
```

D. Kelas UCS

Kelas UCS adalah implementasi algoritma Uniform Cost Search (UCS) yang menggunakan struktur data priority queue untuk mengelola simpul yang akan

dieksplorasi berdasarkan prioritas jarak terpendek. Metode search menerima simpul awal dan mencari jalur terpendek dari simpul awal ke simpul tujuan (endWord). Selama pencarian, simpul yang akan dieksplorasi disimpan dalam frontier, sementara simpul yang telah dieksplorasi disimpan dalam visited. Algoritma berhenti ketika simpul tujuan ditemukan atau semua simpul yang dapat dieksplorasi telah dieksplorasi. Metode search mengembalikan pasangan nilai yang berisi jumlah simpul yang telah dieksplorasi dan jalur terpendek dari simpul awal ke simpul tujuan jika ditemukan, atau jumlah simpul yang telah dieksplorasi dan daftar kosong jika jalur tidak ditemukan.

```
public class UCS {
    private String endWord;
    private PriorityQueue<ExtendedNode> frontier;
    private Set<String> visited;

    public UCS(String endWord) {
        this.endWord = endWord;
        this.frontier = new
PriorityQueue<>(Comparator.comparingInt((currentNode) ->
currentNode.get_distance()));
        this.visited = new HashSet<>();
    }

    public App.Pair<Integer, List<String>> search(ExtendedNode startNode)
{
        if (startNode == null) {
            return new App.Pair<>(visited.size(),
Collections.emptyList());
        }

        frontier.offer(startNode);

        while (!frontier.isEmpty()) {
            ExtendedNode currentNode = frontier.poll();

            if (currentNode.getWord().equals(endWord)) {
                return new App.Pair<>(visited.size()+1,
Arrays.asList(currentNode.get_path_from_root().split(" ")));
            }

            visited.add(currentNode.getWord());

            for (String nextWord :
App.getNeighbor(currentNode.getWord())) {
                if (!visited.contains(nextWord)) {
                    int newCost = currentNode.get_distance() + 1;
                    ExtendedNode nextNode = new ExtendedNode(nextWord,
currentNode, newCost);
                    frontier.offer(nextNode);
```

```
                }
            }
        }

        return new App.Pair<>(visited.size(), Collections.emptyList());
    }
}
```

E. Kelas GDBS

Kelas GBFS (Greedy Best-First Search) merupakan implementasi dari algoritma pencarian heuristik yang menggunakan pendekatan greedy untuk mencari jalur terpendek dari simpul awal ke simpul tujuan (endWord). Dalam kelas ini, sebuah PriorityQueue digunakan untuk mengelola simpul yang akan dieksplorasi berdasarkan heuristik, yaitu jumlah karakter yang tidak sesuai berdasarkan posisinya antara kata saat ini dan kata tujuan. Metode search menerima simpul awal dan mencari jalur terpendek ke simpul tujuan. Algoritma berhenti saat simpul tujuan ditemukan atau tidak ada lagi simpul yang dapat dieksplorasi. Metode search mengembalikan pasangan nilai yang berisi jumlah simpul yang telah dieksplorasi dan jalur terpendek dari simpul awal ke simpul tujuan jika ditemukan, atau jumlah simpul yang telah dieksplorasi dan daftar kosong jika jalur tidak ditemukan.

```
public class GBFS {
    private String endWord;
    private PriorityQueue<Node> prioQueue;
    private Set<String> visited;

    public GBFS(String endWord) {
        this.endWord = endWord;
        this.prioQueue = new
PriorityQueue<>(Comparator.comparingInt((currentNode) -> {
            String a = currentNode.getWord();
            int missChar = 0;
            for (int i = 0; i < a.length(); i++) {
                if (a.charAt(i) != endWord.charAt(i)) {
                    missChar++;
                }
            }

            return missChar;
        }));
        this.visited = new HashSet<>();
    }

    public App.Pair<Integer, List<String>> search(Node startNode) {
        if (startNode == null) {
```

```
            return new App.Pair<>(visited.size(),
Collections.emptyList());
        }

        // Add the start word to the prioQueue
        prioQueue.offer(startNode);
        visited.add(startNode.getWord());

        while (!prioQueue.isEmpty()) {
            Node currentNode = prioQueue.poll();
            String currentWord = currentNode.getWord();

            if (currentWord.equals(endWord)) {
                return new App.Pair<>(visited.size() + 1,
Arrays.asList(currentNode.get_path_from_root().split(" ")));
            }

            for (String nextWord : App.getNeighbor(currentWord)) {
                if (!visited.contains(nextWord)) {
                    Node nextNode = new Node(nextWord, currentNode);
                    prioQueue.offer(nextNode);
                    visited.add(nextWord);
                }
            }
        }

        return new App.Pair<>(visited.size(), Collections.emptyList());
    }
}
```

F. Kelas AStar

Kelas AStar adalah implementasi dari algoritma A* yang merupakan kombinasi dari algoritma Uniform Cost Search (UCS) dan Greedy Best-First Search. PriorityQueue digunakan untuk mengelola simpul yang akan dieksplorasi berdasarkan nilai heuristik. Metode search menerima simpul awal dan mencari jalur terpendek ke simpul tujuan. Algoritma berhenti saat simpul tujuan ditemukan atau tidak ada lagi simpul yang dapat dieksplorasi. Metode search mengembalikan pasangan nilai yang berisi jumlah simpul yang telah dieksplorasi dan jalur terpendek dari simpul awal ke simpul tujuan jika ditemukan, atau jumlah simpul yang telah dieksplorasi dan daftar kosong jika jalur tidak ditemukan.

```
public class AStar {
    private String endWord;
    private PriorityQueue<ExtendedNode> prioQueue;
    private Set<String> visited;
```

```
    public AStar(String endWord) {
        this.endWord = endWord;
        this.prioQueue = new
PriorityQueue<>(Comparator.comparingInt((currentNode) -> {
            int missChar = 0;
            String word = currentNode.getWord();
            for (int i = 0; i < word.length(); i++) {
                if (word.charAt(i) != endWord.charAt(i)) {
                    missChar++;
                }
            }

            return missChar + currentNode.get_distance();
        }));
        this.visited = new HashSet<>();
    }

    public App.Pair<Integer, List<String>> search(ExtendedNode startNode)
{
        if (startNode == null) {
            return new App.Pair<>(visited.size(),
Collections.emptyList());
        }

        prioQueue.offer(startNode);

        while (!prioQueue.isEmpty()) {
            ExtendedNode currentNode = prioQueue.poll();

            if (currentNode.getWord().equals(endWord)) {
                return new App.Pair<>(visited.size()+1,
Arrays.asList(currentNode.get_path_from_root().split(" ")));
            }

            visited.add(currentNode.getWord());

            for (String nextWord :
App.getNeighbor(currentNode.getWord())) {
                if (!visited.contains(nextWord)) {
                    int newCost = currentNode.get_distance() + 1;
                    ExtendedNode nextNode = new ExtendedNode(nextWord,
currentNode, newCost);
                    prioQueue.offer(nextNode);
                }
            }
        }

        return new App.Pair<>(visited.size(), Collections.emptyList());
    }
```

```
}
```

IV. Hasil Pengujian

Ksksks
Sss

No.	Payload	Hasil
1		

V. Analisis Hasil Uji

Lorem

VI. Implementasi Bonus

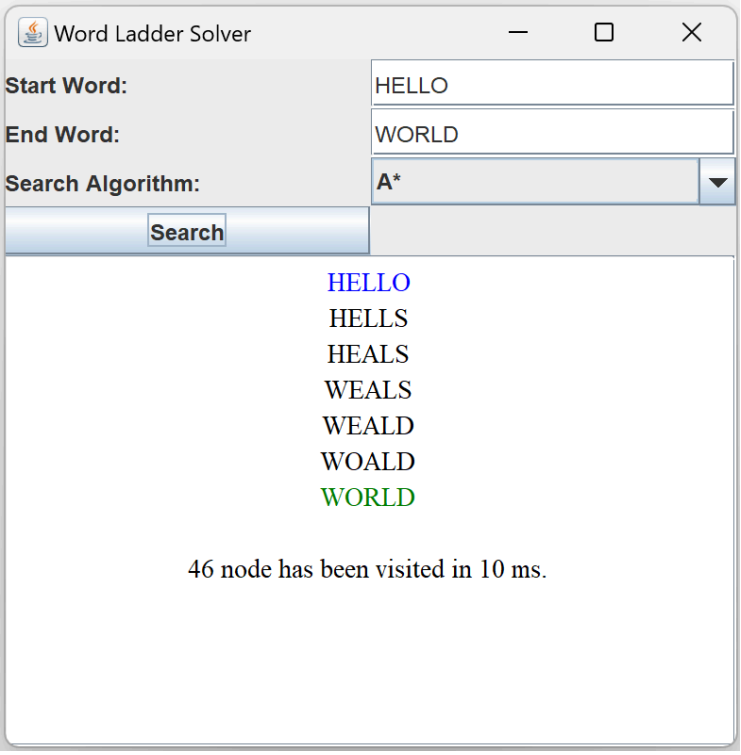
Implementasi GUI dalam kode tersebut menggunakan Java Swing, sebuah toolkit untuk pembuatan aplikasi desktop Java.

Kelas App: Kelas ini mewarisi dari JFrame, yang merupakan jendela utama aplikasi.
Variabel Kelas:
dictionary: Sebuah Set<String> untuk menyimpan kamus kata-kata.
startWordField: JTextField untuk memasukkan kata awal.
endWordField: JTextField untuk memasukkan kata akhir.
algorithmComboBox: JComboBox yang memungkinkan pengguna memilih algoritma pencarian.
resultPane: JTextPane yang menampilkan hasil pencarian.

Konstruktor App:
Mengatur judul jendela, menentukan operasi penutupan ketika jendela ditutup, dan menetapkan ukuran dan tata letak frame.
Membuat panel input menggunakan JPanel dengan tata letak GridLayout.
Menambahkan label dan field input untuk kata awal, kata akhir, dan pilihan algoritma ke panel input.
Membuat tombol "Search" dan menambahkannya ke panel input.
Menambahkan panel input ke bagian atas (BorderLayout.NORTH) frame.

Menginisialisasi resultPane sebagai JTextPane, mengatur tipe kontennya sebagai HTML, dan menambahkannya ke frame di tengah (BorderLayout.CENTER) menggunakan JScrollPane.

ActionListener untuk Tombol "Search":
Menambahkan ActionListener untuk tombol "Search".
Ketika tombol diklik, actionPerformed dipanggil.
Mendapatkan kata awal, kata akhir, dan algoritma pencarian yang dipilih dari input pengguna.
Memuat kamus kata-kata dari file dan menyimpannya dalam dictionary.
Melakukan pencarian dengan algoritma yang dipilih dan memperbarui resultPane dengan hasilnya.



VII. Lampiran

Repositori GitHub : https://github.com/mdavaf17/Tucil3_13522114

VIII. Daftar Pustaka

Munir, Rinaldi. 2021. “Penentuan Rute (Route/Path Planning)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm (Diakses pada 4 Mei 2024)

Kumar, K. Ganesh. 2012. "Midpoint Algorithm: Divide and Conquer Method for Drawing Ellipse." CodeProject.

https://www.codeproject.com/Articles/223159/Midpoint-Algorithm-Divide-and-Conquer-Method-for-D
(Diakses pada 15 Maret 2024)

No	Poin	Ya	Tidak
1	Program berhasil dijalankan.	☑	☐
2	Program dapat melakukan visualisasi kurva Bézier	☑	☐
3	Solusi yang diberikan program optimal	☑	☐
4	Bonus] Program dapat membuat kurva untuk n titik kontrol.	☑	☐
5	[Bonus] Program dapat melakukan visualisasi proses pembuatan kurva.	☑	☐