

# **Tugas Kecil III IF2211 Strategi Algoritma**

## **PENYELESAIAN PERMAINAN WORD LADDER MENGUNAKAN ALGORITMA Uniform Cost Search, Greedy Best First Search, dan A\***

*Diajukan sebagai pemenuhan tugas kecil III.*



Oleh:

13522093 - Matthew Vladimir Hutabarat

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2024**

## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB 1</b>	
<b>ANALISIS DAN IMPLEMENTASI</b>	<b>3</b>
3.1. Definisi dan Implementasi $f(n)$ , $g(n)$ , dan $h(n)$	3
3.2. Implementasi Algoritma UCS, Greedy Best-First Search, dan A*	3
3.3. Analisis Teoritis Algoritma UCS, Greedy Best-First Search, dan A*	4
<b>BAB 2</b>	
<b>SOURCE CODE PROGRAM</b>	<b>6</b>
<b>BAB 3</b>	
<b>PENGUJIAN</b>	<b>19</b>
<b>BAB 4</b>	
<b>ANALISIS HASIL</b>	<b>21</b>
<b>LAMPIRAN</b>	<b>24</b>

## BAB 1

### ANALISIS DAN IMPLEMENTASI

#### 3.1. Definisi dan Implementasi $f(n)$ , $g(n)$ , dan $h(n)$

1. Fungsi  $g(n)$  adalah biaya yang dibutuhkan dari node root hingga ke node  $n$ . Pada program ini, fungsi  $g(n)$  akan dipakai untuk implementasi algoritma UCS,  $f(n) = g(n)$ . Biaya fungsi ini bernilai tinggi kedalaman dari kata awal ke kata  $n$ .
2. Fungsi  $h(n)$  adalah biaya estimasi dari node  $n$  ke node goal. Pada program ini, fungsi  $h(n)$  akan dipakai untuk implementasi Greedy Best-First Search,  $f(n) = h(n)$ . Rumus biaya estimasi pada program ini menghitung jarak antar karakter node  $n$  dengan karakter node goal. Sebagai contoh, Anggap node  $n$  adalah "SING" dan node goal adalah "SANG". Besar biayanya menjadi 1. Maka nilai  $h(n) = 1$ .
3. Fungsi  $f(n)$  adalah biaya estimasi dari node awal ke node goal. Fungsi ini menjumlahkan fungsi  $g(n)$  dan  $h(n)$ . Pada program ini, fungsi  $f(n)$  akan dipakai untuk implementasi algoritma A\*. Fungsi  $h(n)$  pada program ini adalah *admissible*. Jumlah kata berbeda menunjukkan bahwa nilai heuristik bernilai sama dengan nilai sesungguhnya untuk mencapai node goal. Jika jumlah karakter berbeda 5, maka langkah menuju tujuan optimal sebanyak 5 kali. Sehingga  $h(n) = h^*(n)$ . Berdasarkan teori yang diajarkan di kuliah, Algoritma A\* yang  $h(n)$  *admissible* menjamin hasil yang optimal.

#### 3.2. Implementasi Algoritma UCS, Greedy Best-First Search, dan A\*

- Algoritma UCS
  1. Inisiasi program dengan memasukkan node *start* ke antrian *wordQueue* sehingga node *start* menjadi elemen di antrian dengan index ke-0.
  2. Masukkan node start ke map kata-kata yang sudah dikunjungi.
  3. Cari kemungkinan kata-kata di kamus yang bisa dicapai dari antrian node *wordQueue* dengan index ke-0 dengan mengubah satu karakter ke karakter lain. Kemudian, masukkan kata tersebut ke *list of string wordNextMove*.
  4. Jika di *list wordNextMove* terdapat kata tujuan. Maka hubungkan node index ke-0 dengan kata tujuan. Setelah itu ubah serangkai node tersebut ke *list of string solution* dan terminasi program dengan mengembalikan nilai *list of string solution*.
  5. Jika tidak ada kata tujuan di *list wordNextMove*. Maka setiap kata di *list wordNextMove* dihubungkan ke node dengan index ke-0 dan masukkan ke map kata-kata yang sudah dikunjungi. Kemudian hasil node tersebut dimasukkan ke antrian *wordQueue* dengan menambah 1 nilai costnya.
  6. Hapus node index ke-0 pada *wordQueue*.
  7. Jika antrian di *wordQueue* panjangnya 0 maka tidak ada lagi kata di *wordQueue* yang bisa dicari sehingga solusi tidak ditemukan. Maka terminasi program dan kembalikan *list of string* kosong.
  8. Ulang ke langkah 2 hingga solusi ditemukan.

- Algoritma Greedy Best-First Search
  1. Inisiasi program dengan memasukkan kata *start* ke antrian *list of string wordQueue*.
  2. Masukkan kata *start* ke map kata kata yang sudah dikunjungi.
  3. Ambil kata dengan index terakhir di *list wordQueue* dan cari kata-kata pada kamus yang bisa dicapai dengan mengubah 1 karakter pada kata tersebut kecuali kata kata yang sudah ada di map kata yang sudah dikunjungi.
  4. Jika tidak ada kata yang bisa dicapai maka terminasi program dan kembalikan *list* kosong.
  5. Setiap kata tersebut dihitung biaya estimasinya dengan rumus  $h(n)$  diatas. Lalu biaya estimasi termurah akan dicatat.
  6. Masukkan kata termurah tersebut ke *list wordQueue* dan ke map kata-kata yang sudah dikunjungi.
  7. Jika kata termurah tersebut adalah kata tujuan maka terminasi program dan kembalikan *list wordQueue* sebagai solusi.
  8. Jika kata termurah tersebut bukan kata tujuan maka ulangin langkah 3.
- Algoritma A\*
  1. Inisiasi program dengan memasukkan node *start* dan hitung biaya  $f(n)$  kata tersebut ke antrian *wordQueue*. Kemudian masukkan kata tersebut ke map kata kata yang sudah dikunjungi.
  2. Ambil kata dengan index ke-0 pada antrian *wordQueue*. Lalu cari kata-kata pada kamus yang bisa dijangkau dari kata tersebut dengan mengubah 1 karakter pada kata tersebut kecuali kata-kata yang sudah dikunjungi, maka masukkan kata kata tersebut ke *wordNextMove*.
  3. Jika tidak ditemukan kata yang bisa dijangkau. Maka terminasi program dan kembalikan *list* kosong.
  4. Jika kata tujuan terdapat di *wordNextMove*. Maka hubungkan node index ke-0 dengan kata tujuan. Setelah itu ubah serangkai node tersebut ke *list of string solution* dan terminasi program dengan mengembalikan nilai *list of string solution*
  5. Jika kata tersebut tidak ada di *wordNextMove*. Maka setiap kata di *list wordNextMove* dihubungkan ke node dengan index ke-0 dan masukkan ke map kata-kata yang sudah dikunjungi. Kemudian hasil node tersebut dimasukkan ke antrian *wordQueue* dengan nilai biaya total  $h(n)$  dan  $g(n)$ .
  6. Hapus node index ke-0 pada *wordQueue*.
  7. Jika antrian di *wordQueue* panjangnya 0 maka tidak ada lagi kata di *wordQueue* yang bisa dicari sehingga solusi tidak ditemukan. Maka terminasi program dan kembalikan *list of string* kosong.
  8. Ulang langkah 2 hingga mendapatkan solusi

### 3.3. Analisis Teoritis Algoritma UCS, Greedy Best-First Search, dan A\*

Pada permasalahan *word ladder*, UCS sama dengan BFS karena biaya node yang baru pasti lebih mahal daripada kata kata yang sudah ditemukan sehingga UCS harus menyelesaikan semua node di 1 level dahulu baru pergi ke level berikutnya. Hal

itu sama dengan BFS yang harus menyelesaikan 1 level dahulu baru pergi ke level berikutnya.

Secara teoritis, pada kasus *word ladder* UCS lebih efisien daripada A\* karena pada algoritma UCS simpul dengan nilai  $g(n)$  terkecil diperiksa terlebih dahulu. Sementara pada algoritma A\*, yang diperiksa adalah nilai  $g(n) + h(n)$  yang mana dapat terjadi backtracking ke simpul dengan nilai  $g(n)$  yang lebih kecil. Selain itu, pada prosesnya A\* masih lakukan pencarian meski solusi ditemukan untuk memastikan solusi yang dicapai adalah yang paling optimal.

Permasalahan dari GBFS adalah tidak bisa menjamin memberikan solusi yang optimal karena GBFS hanya mengambil biaya paling murah untuk setiap langkahnya dan tidak bisa backtracking. Akibat GBFS bisa terjebak ke kata kata yang tidak bisa melampaui kata kata lain.

## BAB 2

### SOURCE CODE PROGRAM

#### 1. Node.java

```
public class Node {  
    private Node previousNode;  
    private String current;  
    private Node nextNode;  
  
    public Node(Node currentNode) {  
        this.current = currentNode.current;  
        this.previousNode = currentNode.previousNode;  
        this.nextNode = currentNode.nextNode;  
    }  
  
    public Node(String current) {  
        this.current = current;  
        this.nextNode = null;  
        this.previousNode = null;  
    }  
  
    public String getValue() {  
        return this.current;  
    }  
  
    public Node getNextNode() {  
        return this.nextNode;  
    }  
  
    public Node getPreviousNode() {  
        return this.previousNode;  
    }  
  
    public int getNodeLength() {  
        Node newNode = new Node(this);  
        int count = 0;  
        while (newNode.previousNode != null) {  
            count++;  
            newNode = newNode.previousNode;  
        }  
    }  
}
```

```
        return count;
    }

    public boolean isEqual(Node currentNode) {
        if (this.current.equals(currentNode.current)) {
            return false;
        }

        return true;
    }

    public void concatNode(Node nextNode) {
        this.nextNode = nextNode;
        nextNode.previousNode = this;
    }

    public List<String> convertNodeToArrayFromBackward() {
        List<String> solution = new ArrayList<>(100);
        solution.add(this.current);

        while(previousNode != null) {
            solution.add(previousNode.current);
            previousNode = previousNode.previousNode;
        }

        return solution;
    }

    public void displayNodeBackward() {

        Node newNode = new Node(this);
        System.out.print("(");
        while(newNode.previousNode != null) {
            System.out.print(newNode.current + ", ");
            newNode = newNode.previousNode;
        }
        System.out.print(newNode.current);

        System.out.print(")");
    }
}
```

```
}  
}
```

## 2. Pair.java

```
public class Pair {  
    private Node currentNode;  
    private int value;  
  
    public Pair(){  
        this.value = 100;  
        this.currentNode = new Node("");  
    }  
  
    public Pair(Node currentNode, int value){  
        this.currentNode = currentNode;  
        this.value = value;  
    }  
  
    public Node getNode() {  
        return this.currentNode;  
    }  
  
    public int getValue() {  
        return this.value;  
    }  
  
    public void displayPair(){  
        System.out.print("(");  
        System.out.print(currentNode.getValue());  
        System.out.print(",");  
        System.out.print(this.getValue());  
        System.out.print(")");  
    }  
}
```

## 3. PriorityQueue.java

```
public class PriorityQueue {  
    private List<Pair> wordQueue;  
  
    public PriorityQueue() {  
        this.wordQueue = new ArrayList<>();  
    }  
}
```



```
public PriorityQueue(int n) {
    this.wordQueue = new ArrayList<>(n);
}

public Pair getPair(int index) {
    return this.wordQueue.get(index);
}

public int getLength(){
    return this.wordQueue.size();
}

public Pair getLastPair(){
    return this.wordQueue.get(wordQueue.size() - 1);
}

public void insertPair(Pair newPair){
    int newValue = newPair.getValue();

    if (this.wordQueue.size() == 0) {
        this.wordQueue.add(newPair);
    } else {
        for(int i = 0; i < this.wordQueue.size(); i++){
            if (this.wordQueue.get(i).getValue() > newValue){
                this.wordQueue.add(i, newPair);
                break;
            }

            if (i == (this.wordQueue.size() - 1)){
                this.wordQueue.add(i + 1, newPair);
                break;
            }
        }
    }
}

public void deletePair(Pair pairToDelete){
    for(int i = 0; i < this.wordQueue.size(); i++){
        if
(this.wordQueue.get(i).getNode().getValue().equals(pairToDelete.getNode().
getValue())){
```

```
        this.wordQueue.remove(i);  
        break;  
    }  
}  
  
public boolean isEmpty(){  
    return this.wordQueue.size() == 0;  
}  
  
public void displayWordQueue(){  
    Node nodeToDisplay;  
    System.out.print("[");  
    for(int i = 0; i < this.wordQueue.size(); i++){  
        nodeToDisplay = new Node(this.wordQueue.get(i).getNode());  
        // nodeToDisplay.displayNodeBackward();  
        System.out.print(nodeToDisplay.getValue());  
        if (i != (this.wordQueue.size() - 1)){  
            System.out.print(",");  
        }  
    }  
    System.out.println("]");  
}  
}
```

#### 4. MyDictionary.java

```
public class MyDictionary {  
    private HashSet<String> dictionary;  
  
    public MyDictionary() {  
        dictionary = new HashSet<>();  
        loadDictionary("src/backend/util/Dictionary.txt");  
    }  
  
    private void loadDictionary(String filename) {  
        try (BufferedReader reader = new BufferedReader(new  
FileReader(filename))) {
```

```
        String line;
        while ((line = reader.readLine()) != null) {
            dictionary.add(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public boolean isValidWord(String word) {
    return dictionary.contains(word);
}

public List<String> findAllPossibleWord(String word, String
end, Map<String, Boolean> visitedWord) {
    // word = word.toLowerCase();
    List<String> tempPossibleWord = new ArrayList<>();
    // Dictionary dictionary = new Dictionary();

    String resetWord = new String(word);
    for (int i = 0; i < word.length(); i++) {
        word = new String(resetWord);

        for (int j = 0; j < 26; j++) {
            String modifiedString;
            if (i != 0) {
                modifiedString = word.substring(0, i) + ((char) (97 +
j)) + word.substring(i + 1);
            } else {
                modifiedString = ((char) (97 + j)) +
word.substring(1);
            }

            if (!modifiedString.equals(resetWord) &&
visitedWord.get(modifiedString) == null) {
                if (this.isValidWord(modifiedString)) {
                    tempPossibleWord.add(modifiedString);
                }
            }
        }
    }
}
```

```
    }  
  
    return tempPossibleWord;  
}  
  
}
```

### 5. Greedy.java

```
public class Greedy {  
    public Greedy(){};  
    public void displayListString(List<String> wordNextMove) {  
        System.out.print("[");  
        for(int i = 0; i < wordNextMove.size(); i++){  
            System.out.print(wordNextMove.get(i));  
            if (i != wordNextMove.size() - 1){  
                System.out.print(", ");  
            }  
        }  
  
        System.out.println("]");  
    }  
  
    public String findCheapestCost(String currentWord, String end,  
Map<String, Boolean> visitedWord, MyDictionary dictionary) {  
        List<String> word = dictionary.findAllPossibleWord(currentWord,  
end, visitedWord);  
        System.out.println("Reference : " + currentWord);  
        displayListString(word);  
        if (word.size() == 0){  
            return "Not Found";  
        }  
  
        char[] charTarget = end.toCharArray();  
        int cheapestCost = 100;  
        String cheapestWord = word.get(0);  
        for(int i = 0; i < word.size(); i++){  
            int cost = 0;  
            char[] charWord = word.get(i).toCharArray();  
  
            for(int j = 0; j < charWord.length; j++){  
                if(charWord[j] != charTarget[j]){
```

```
        cost ++;
    }
}

if (cost < cheapestCost){
    cheapestCost = (int) cost;
    cheapestWord = new String(word.get(i));
}

}

return cheapestWord;
}

// PriorityQueue -> Pair -> Node -> String
public List<String> algorithmGreedy(String start, String end,
List<String> wordQueue, Map<String, Boolean> visitedWord, MyDictionary
dictionary){

    if(wordQueue.isEmpty()){
        wordQueue.add(start);
        visitedWord.put(start, true);
        return algorithmGreedy(start, end, wordQueue, visitedWord,
dictionary);
    } else {
        String nextWord = new String(wordQueue.get(wordQueue.size() -
1));

        String cheapestWord = findCheapestCost(nextWord, end,
visitedWord, dictionary);
        if (cheapestWord.equals("Not Found")){
            return new ArrayList<>();
        }

        wordQueue.add(cheapestWord);
        visitedWord.put(cheapestWord, true);

        if (cheapestWord.equals(end)){
            visitedWord.put(cheapestWord, true);
            return wordQueue;
        }
    }
}
```

```
    }

    return algorithmGreedy(start, end, wordQueue, visitedWord,
dictionary);
}
}
}
```

## 6. UCS.java

```
public class UCS {
    public UCS() {}

    public boolean foundEnd(List<String> wordNextMove, String target){
        for(int i = 0; i < wordNextMove.size(); i++){
            if (wordNextMove.get(i).equals(target)){
                return true;
            }
        }

        return false;
    }

    public void displayListString(List<String> wordNextMove) {
        System.out.print("[");
        for(int i = 0; i < wordNextMove.size(); i++){
            System.out.print(wordNextMove.get(i));
            if (i != wordNextMove.size() - 1){
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }

    public List<String> invertListString(List<String> solution){
        List<String> newSolution = new ArrayList<>(100);
        for(int i = 0; i < solution.size(); i++){
            newSolution.add(0,solution.get(i));
        }

        return newSolution;
    }
}
```

```
}

    public List<String> algorithmUCS(String start, String end,
PriorityQueue wordQueue, Map<String, Boolean> visitedWord, MyDictionary
dictionary, int nodeVisited){
        List<String> wordNextMove = new ArrayList<>();

        if (wordQueue.isEmpty()){
            Node startNode = new Node(start);
            wordQueue.insertPair(new Pair(startNode,
startNode.getNodeLength()));
            visitedWord.put(start, true);

        }

        wordNextMove =
dictionary.findAllPossibleWord(wordQueue.getPair(0).getNode().getValue(),
end, visitedWord);
        nodeVisited++;
        if (foundEnd(wordNextMove, end)){
            Node nodeSolution = wordQueue.getPair(0).getNode();
            nodeSolution.concatNode(new Node(end));

            List<String> solution =
nodeSolution.getNextNode().convertNodeToArrayFromBackward();
            solution = invertListString(solution);
            solution.add(String.valueOf(nodeVisited));
            return solution;
        }

        Pair pairTemplate = wordQueue.getPair(0);
        Node newNode;
        Node nodeToConnect;

        for(int i = 0; i < wordNextMove.size(); i++){

            if (visitedWord.get(wordNextMove.get(i)) == null){
                nodeToConnect = pairTemplate.getNode();
                newNode = new Node(wordNextMove.get(i));
                nodeToConnect.concatNode(newNode);
            }
        }
    }
}
```

```
        wordQueue.insertPair(new Pair(newNode,
newNode.getNodeLength()) );
        visitedWord.put(wordNextMove.get(i), true);
    }
}
wordQueue.deletePair(pairTemplate);
if(wordQueue.getLength() == 0){
    return new ArrayList<>();
}
return algorithmUCS(start, end, wordQueue,visitedWord, dictionary,
nodeVisited);
}
}
```

#### 7. Astar.java

```
public class Astar {
    public Astar(){};

    public int calculateCost(Node word, String end){
        int cost = 0;
        char[] charWord = word.getValue().toCharArray();
        char[] charTarget = end.toCharArray();

        for(int j = 0; j < charWord.length; j++){
            if(charWord[j] != charTarget[j]){
                cost++;
            }
        }

        cost += word.getNodeLength();
        return cost;
    }

    public void displayListString(List<String> wordNextMove){
        System.out.print("[");
        for(int i = 0; i < wordNextMove.size(); i++){
            System.out.print(wordNextMove.get(i));
            if (i != wordNextMove.size() - 1){
                System.out.print(", ");
            }
        }
        System.out.print("]");
    }
}
```



```
    }

    }

    System.out.println("]");
}

public List<String> invertListString(List<String> solution){
    List<String> newSolution = new ArrayList<>(100);
    for(int i = 0; i < solution.size(); i++){
        newSolution.add(0,solution.get(i));
    }

    return newSolution;
}

public boolean foundEnd(List<String> wordNextMove, String target){
    for(int i = 0; i < wordNextMove.size(); i++){
        if (wordNextMove.get(i).equals(target)){
            return true;
        }
    }

    return false;
}

public List<String> algorithmAstar(String start, String end,
PriorityQueue wordQueue, Map<String,Boolean> visitedWord, MyDictionary
dictionary,int nodeVisited){
    List<String> wordNextMove = new ArrayList<>();

    if (wordQueue.isEmpty()){
        Node startNode = new Node(start);
        wordQueue.insertPair(new Pair(startNode,
calculateCost(startNode, end)));
        visitedWord.put(start, true);
    }

    wordNextMove =
dictionary.findAllPossibleWord(wordQueue.getPair(0).getNode().getValue(),
end, visitedWord);
```

```
nodeVisited++;
if (foundEnd(wordNextMove, end)){
    Node nodeSolution = wordQueue.getPair(0).getNode();
    nodeSolution.concatNode(new Node(end));

    List<String> solution =
nodeSolution.getNextNode().convertNodeToArrayFromBackward();
    solution = invertListString(solution);
    solution.add(String.valueOf(nodeVisited));
    return solution;
}

Pair pairTemplate = wordQueue.getPair(0);
Node newNode;
Node nodeToConnect;

for(int i = 0; i < wordNextMove.size();i++){

    if (visitedWord.get(wordNextMove.get(i)) == null){
        nodeToConnect = pairTemplate.getNode();
        newNode = new Node(wordNextMove.get(i));
        nodeToConnect.concatNode(newNode);
        wordQueue.insertPair(new Pair(newNode,
calculateCost(newNode, end)) );
        visitedWord.put(wordNextMove.get(i), true);
    }
}

wordQueue.deletePair(pairTemplate);
if(wordQueue.getLength() == 0){
    return new ArrayList<>();
}

return algorithmAstar(start, end, wordQueue,visitedWord,
dictionary, nodeVisited);
}
}
```

## BAB 3

### PENGUJIAN

#### 1. SING -> TALL

GBFS	UCS	A*
<pre> Enter start word : sing  Enter end word : tall Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 38 milliseconds Node Visited: 6 Solution length : 5 [sing,ting,tang,tank,talk,tall] </pre>	<pre> Enter start word : sing  Enter end word : tall Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 89 milliseconds Node Visited: 947 Solution length : 5 [sing,ting,tang,tank,talk,tall] </pre>	<pre> Enter start word : sing  Enter end word : tall Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 60 milliseconds Node Visited: 46 Solution length : 5 [sing,ting,tang,tank,talk,tall] </pre>

#### 2. BASE -> ROOT

GBFS	UCS	A*
<pre> Enter start word : base  Enter end word : root Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 41 milliseconds Node Visited: 11 Solution length : 10 [base,rase,rose,robe,rode,role,rope,rote,roue,root,root] </pre>	<pre> Enter start word : base  Enter end word : root Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 105 milliseconds Node Visited: 1057 Solution length : 5 [base,case,cast,coat,coot,root] </pre>	<pre> Enter start word : base  Enter end word : root Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 50 milliseconds Node Visited: 75 Solution length : 5 [base,rase,rose,roue,root,root] </pre>

#### 3. PAINT -> BUILD

GBFS	UCS	A*
<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 41 milliseconds Node Visited: 8 Solution length : 7 [paint,faint,saint,suint,quint,quilt,built,build] </pre>	<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 74 milliseconds Node Visited: 515 Solution length : 6 [paint,saint,suint,quint,quilt,built,build] </pre>	<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 49 milliseconds Node Visited: 48 Solution length : 6 [paint,saint,suint,quint,quilt,built,build] </pre>

#### 4. EAT -> MAN

GBFS	UCS	A*
------	-----	----

Tugas Kecil III IF2211 Strategi Algoritma  
Implementasi Algoritma UCS dan A\* dalam Pencarian Rute Terdekat

<pre> Enter start word : eat Enter end word : man Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 45 milliseconds Node Visited: 3 Solution length : 2 [eat,mat,man] </pre>	<pre> Enter start word : eat Enter end word : man Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 47 milliseconds Node Visited: 9 Solution length : 2 [eat,mat,man] </pre>	<pre> Enter start word : eat Enter end word : man Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 45 milliseconds Node Visited: 2 Solution length : 2 [eat,mat,man] </pre>
---	---	---

## 5. EARN -> MAKE

GBFS	UCS	A*
<pre> Enter start word : earn Enter end word : make Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 39 milliseconds Node Visited: 5 Solution length : 4 [earn,barn,bare,mare,make] </pre>	<pre> Enter start word : earn Enter end word : make Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 48 milliseconds Node Visited: 111 Solution length : 4 [earn,barn,bare,mare,make] </pre>	<pre> Enter start word : earn Enter end word : make Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 44 milliseconds Node Visited: 19 Solution length : 4 [earn,barn,bare,mare,make] </pre>

## 6. OOP -> KINDERJOY

GBFS	UCS	A*
<pre> Enter start word : OOP Enter end word : KINDERJOY Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1 Masukan tidak Valid </pre>	<pre> Enter start word : OOP Enter end word : KINDERJOY Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2 Masukan tidak Valid </pre>	<pre> Enter start word : OOP Enter end word : KINDERJOY Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3 Masukan tidak Valid </pre>

## BAB 4

### ANALISIS HASIL

Berdasarkan hasil percobaan, UCS dan A\* selalu memberikan hasil yang optimal sedangkan Greedy Best-First Search tidak selalu memberikan solusi optimal. Hal ini dapat terjadi karena Greedy Best-First Search tidak bisa *backtracking* sehingga jika Greedy Best-First Search terjebak ke kata yang mempunyai cost lebih rendah tapi secara keseluruhan kata tersebut malah membuat solusi menjadi lebih panjang, Algoritma ini tidak bisa menghapus kata tersebut dan harus melanjutkan kata yang ingin dicari dengan kata tersebut. Sebagai contoh pada testcase BASE -> ROOT, GBFS memberikan solusi dengan panjang 10 sedangkan UCS dan A\* memberikan solusi dengan panjang 5.

GBFS	UCS	A*
<pre>Enter start word : base  Enter end word : root Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 41 milliseconds Node Visited: 11 Solution length : 10 [base,rase,rose,robe,rode,role,rope,rote,roue,root,root]</pre>	<pre>Enter start word : base  Enter end word : root Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 105 milliseconds Node Visited: 1057 Solution length : 5 [base,case,cast,cost,coot,root]</pre>	<pre>Enter start word : base  Enter end word : root Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 50 milliseconds Node Visited: 75 Solution length : 5 [base,rase,rose,roue,route,root]</pre>

Berdasarkan hasil percobaan, GBFS menjadi algoritma dengan waktu eksekusi lebih cepat dibandingkan UCS dan A\*. Hal ini terjadi karena GBFS pada setiap langkahnya langsung mengambil sebuah simpul dengan cost terendah saja dan simpul lain diabaikan sedangkan pada UCS dan A\* simpul lain tetap diperhatikan karna ada kemungkinan langkah yang diambil tidak optimal sehingga algoritma UCS dan A\* ingin backtracking. Sementara itu antara UCS dan A\*, A\* relatif lebih cepat dibandingkan dengan UCS karena A\* memilih biaya  $h(n)$  yang menyebabkan terdapat prioritas node dengan biaya yang murah akan diproses duluan sehingga solusi yang didapatkan lebih cepat juga ketemu dibandingkan UCS yang biaya  $g(n)$  tidak terlalu memperhatikan ke-prioritasan dan hanya menganut FIFO.

GBFS	UCS	A*
<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 41 milliseconds Node Visited: 8 Solution length : 7 [paint,faint,saint,suint,quint,quilt,built,build] </pre>	<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 74 milliseconds Node Visited: 515 Solution length : 6 [paint,saint,suint,quint,quilt,built,build] </pre>	<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 49 milliseconds Node Visited: 48 Solution length : 6 [paint,saint,suint,quint,quilt,built,build] </pre>

Berdasarkan hasil percobaan, GBFS cenderung tidak mengambil memori yang besar ketimbang UCS dan A\* karena GBFS pada setiap langkahnya tidak memperdulikan simpul lain maka penggunaan memorinya sedikit sehingga kompleksitas memori  $O(n)$  dengan  $n$  sebagai

kedalaman . UCS dan A\* memiliki kompleksitas memori  $O(b^n)$  dengan b banyak kata kata yang mungkin dari kata referensi dan n sebagai kedalaman.

GBFS	UCS	A*
<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 1  Program execution duration: 46 milliseconds Node Visited: 8 Solution length : 7 [paint,faint,saint,suint,quint,quilt,built,built] Heap Memory Usage: 10485760 </pre>	<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 2  Program execution duration: 71 milliseconds Node Visited: 515 Solution length : 6 [paint,saint,suint,quint,quilt,built,built] Heap Memory Usage: 18874368 </pre>	<pre> Enter start word : paint  Enter end word : build Select Method : 1. GBFS : 2. UCS : 3. A* : Choose 1,2,3 : 3  Program execution duration: 49 milliseconds Node Visited: 48 Solution length : 6 [paint,saint,suint,quint,quilt,built,built] Heap Memory Usage: 10485760 </pre>

### Program Checkpoint

Poin	Ya	Tidak
Program berhasil dijalankan.	V	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma UCS	V	
Solusi yang diberikan pada algoritma UCS optimal	V	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma <i>Greedy Best First Search</i>	V	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma A*	V	
Solusi yang diberikan pada algoritma A* optimal	V	
<b>[Bonus]:</b> Program memiliki tampilan GUI		V

## LAMPIRAN

### LINK REPOSITORY

Link repository GitHub : [Repository](#)