



**um**  
The Learning  
University

# MODUL SISTEM CERDAS

Pegangan Mahasiswa

## SEARCHING

Jurusan Teknik Elektro  
Fakultas Teknik  
Universita Negeri Malang

Disusun oleh :  
Ria Febrianti  
Dr. Hakkun Elmunsyah, S.T., M.T  
Dr. Eng. Anik Nur Handayani, S.T., M.T.

# KATA PENGANTAR

**Bismillahirrahmannirahim,**

Puji syukur kehadiran Tuhan yang Maha Esa atas limpahan rahmat serta hidayahnya, sehingga penulis dapat menyelesaikan modul pembelajaran berbantuan SIPEJAR UM dan *e-Collab Classroom* pada mata kuliah Sistem Cerdas dengan topik materi *Searching* untuk mahasiswa Program Studi S1 Pendidikan Teknik Elektro, Jurusan Teknik Elektro, Fakultas Teknik, Universitas Negeri Malang. Penyajian materi pada modul pembelajaran *Searching* ini merujuk pada Rencana Perkuliahan Semester untuk mata kuliah Sistem Cerdas (PTEL667)

Penulisan modul *Searching* ini diharapkan dapat menumbuhkan motivasi belajar mahasiswa dan meningkatkan efektivitas pelaksanaan perkuliahan Sistem Cerdas di Jurusan Teknik Elektro, Fakultas Teknik, Universitas Negeri Malang. Selain itu, mahasiswa diharapkan mampu menguasai secara tuntas materi *Searching* yang tertulis pada modul pembelajaran terutama pada pelaksanaan pembelajaran secara daring, namun pemanfaatan modul ini juga dapat menunjang kegiatan perkuliahan secara tatap muka. Dengan demikian akan tercapai tujuan perkuliahan Sistem Cerdas yang telah ditetapkan

Penulisan modul *Searching* ini tentu tidak terlepas dari bantuan dan dukungan dari berbagai pihak. Penulis mengucapkan terimakasih kepada pihak yang telah membantu proses penyusunan modul pembelajaran *Searching* ini. Selain itu, penulis menyadari sepenuhnya bahwa isi ataupun penyajian dari modul pembelajaran ini masih jauh dari kata sempurna. Maka dari itu, penulis

mengharapkan masukan berupa kritik dan saran yang dapat membangun untuk perbaikan dan pengembangan modul selanjutnya.

Malang, Juni 2021

Penulis

# DAFTAR ISI

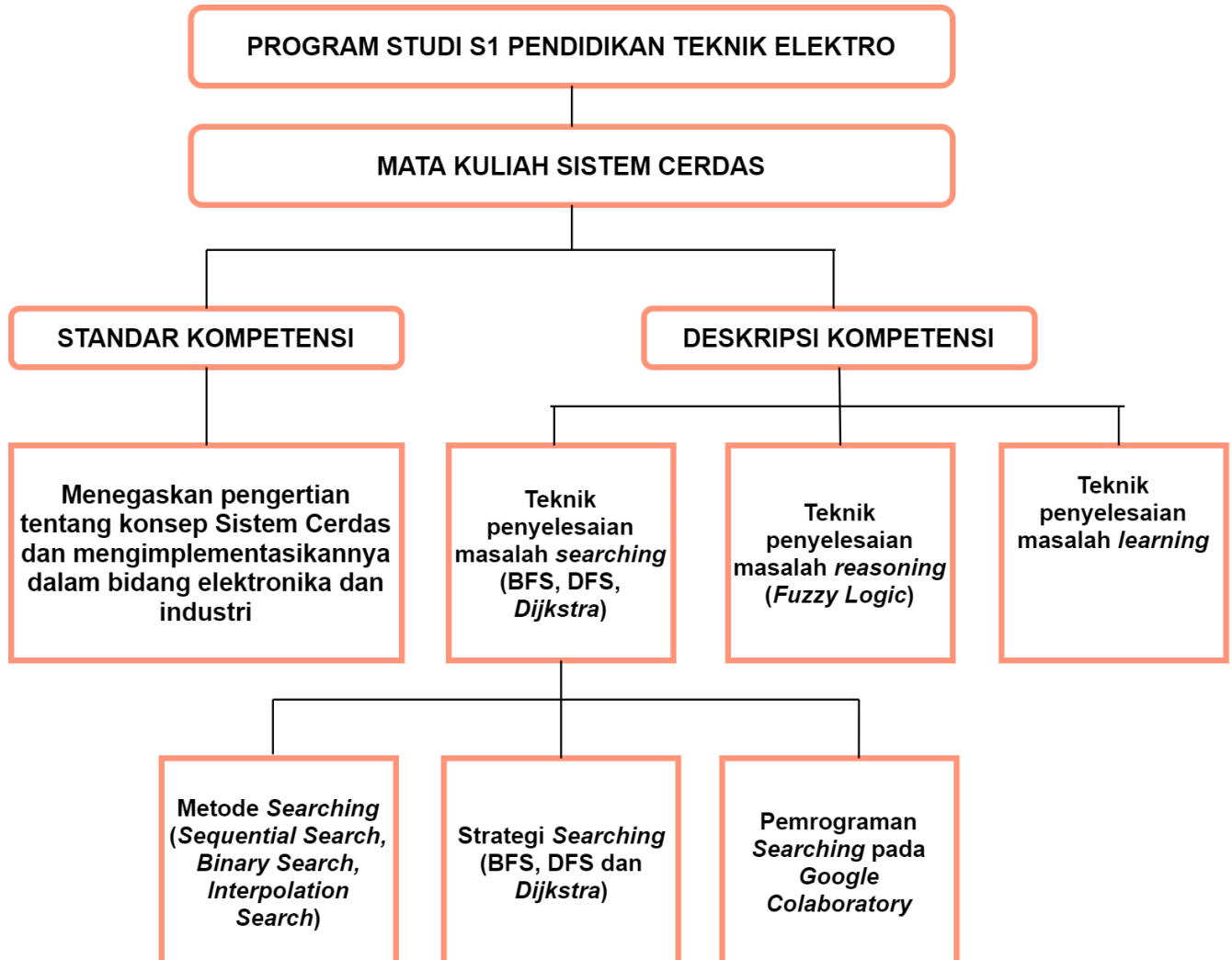
HALAMAN SAMPUL .....	
KATA PENGANTAR.....	ii
DAFTAR ISI .....	iv
DAFTAR GAMBAR.....	vi
PETA KEDUDUKAN MODUL .....	vii
<b>PENDAHULUAN</b> .....	1
Standar Kompetensi.....	2
Deskripsi .....	2
Prasyarat.....	3
Petunjuk Penggunaan Modul .....	3
Tujuan Akhir .....	3
Indikator Penguasaan Kompetensi .....	4
<b>PEMBELAJARAN</b> .....	5
<i>SEARCHING</i> .....	6
Jabaran Materi .....	6
1. Metode <i>Searching</i> .....	7
A. Sequential Search (linear search) .....	7
B. Binary Search .....	7
C. Interpolation Search.....	8
2. Strategi <i>Searching</i> .....	10
A. <i>Breadth-First Search</i> (BFS).....	10
B. <i>Depth-First Search</i> (DFS).....	12
C. <i>Dijkstra</i> .....	14
3. Pemrograman <i>Searching</i> pada <i>Google Colaboratory</i> .....	17
Menjalankan fungsi BFS pada <i>Google Colaboratory</i> .....	18
Menjalankan fungsi DFS pada <i>Google Colaboratory</i> .....	20
Menjalankan fungsi <i>Dijkstra</i> pada <i>Google Colaboratory</i> .....	20

Rangkuman Materi.....	24
Materi Pengayaan .....	25
<b>EVALUASI</b> .....	30
Tes Individu .....	31
Referensi .....	32

## DAFTAR GAMBAR

Gambar		Halaman
Gambar 1	Cara Kerja <i>Sequential Search</i> dan <i>Binary Search</i> ....	8
Gambar 2	Interpolasi Search.....	9
Gambar 3	Cara Kerja BFS.....	10
Gambar 4	Cara Kerja DFS.....	13
Gambar 5	<i>Dijkstra</i> .....	15
Gambar 6	Penerapan BFS dan DFS.....	18
Gambar 7	Penerapan <i>Dijkstra</i> .....	20
Gambar 8	Studi Kasus <i>Search</i> .....	25
Gambar 9	<i>Graph</i> Studi Kasus <i>Searching</i> .....	26
Gambar 10	Rute dari Desa A ke Desa E.....	27

# PETA KEDUDUKAN MODUL



# 1

## PENDAHULUAN

**Standar Kompetensi**

**Deskripsi**

**Prasyarat**

**Petunjuk Penggunaan Modul**

**Tujuan Akhir**

**Indikator Penguasaan  
Kompetensi**





## Standar Kompetensi

Menganalisis tahapan teknik penyelesaian masalah *searching* menggunakan BFS, DFS, dan *Dijkstra* dan mengimplentasikan dalam bidang teknik elektronika.



## Deskripsi

Modul ini merupakan bahan ajar virtual yang dapat membantu mahasiswa untuk mempelajari salah satu jenis teknik penyelesaian masalah yaitu *searching*, yang meliputi BFS, DFS, dan *Dijkstra*. Modul pembelajaran *searching* ini terdiri dari tiga bab yaitu pendahuluan, pembelajaran, dan evaluasi. Pada bab pertama (pendahuluan), berisi penjelasan standar kompetensi, deskripsi modul, prasyarat, petunjuk penggunaan modul, tujuan akhir, dan indikator penguasaan kompetensi.

Bab kedua (Pembelajaran) berisi tentang penyajian materi pembelajaran yang dilengkapi dengan materi pengayaan. Materi yang disajikan pada modul meliputi metode *searching*, strategi *searching* (BFS, DFS, dan *Dijkstra*), dan pemrograman *searching* pada *google colaboratory*

Bab ketiga (evaluasi), berisi tes individu sebagai bentuk evaluasi penguasaan mahasiswa terhadap materi yang disajikan. Tes individu yang disajikan berisi soal tentang penerapan *searching* (BFS, DFS, dan *Dijkstra*) dengan langkah penyelesaian secara manual. Setelah menyelesaikan modul pembelajaran, diharapkan mahasiswa mampu menganalisis teknik penyelesaian masalah pada sistem cerdas menggunakan strategi *searching* (BFS, DFS, dan *Dijkstra*).



## Prasyarat

Untuk mendukung proses pembelajaran, mahasiswa diharapkan mampu menguasai beberapa pengetahuan seperti:

1. Mengetahui algoritma *boolean*.
2. Mengetahui jenis-jenis *agent* dan *environment*.
3. Mengetahui *knowledge based system* dalam kecerdasan buatan.



## Petunjuk Penggunaan Modul

1. Pelajari daftar isi dan peta kedudukan modul untuk mengetahui modul *searching* yang akan dipelajari.
2. Pelajari uraian materi yang disajikan pada setiap kegiatan pembelajaran.
3. Pahami kesulitan yang ditemukan dari uraian materi yang disajikan, tanyakan pada instruktur/dosen pada saat kegiatan pembelajaran.
4. Pahami dan selesaikan soal evaluasi yang disajikan pada modul.



## Tujuan Akhir

1. Memahami konsep algoritma *searching*
2. Mengidentifikasi strategi algoritma *searching* (BFS, DFS, dan *Dijkstra*).
3. Menguraikan penyelesaian masalah dengan menggunakan algoritma *searching* menggunakan langkah matematis.
4. Merepresentasikan algoritma *searching* dengan bahasa pemrograman *python* pada *google colaboratory*.



### Indikator Penguasaan Kompetensi

Indikator penguasaan kompetensi dari tujuan akhir, yaitu mahasiswa dapat:

1. Mengetahui konsep dasar pengetahuan sistem cerdas dengan teknik penyelesaian masalah *searching*
2. Menganalisis strategi *searching* dalam menyelesaikan masalah
3. Mengetahui langkah merepresentasikan teknik penyelesaian masalah *searching* pada *google colaboratory*.

# 2

## PEMBELAJARAN

***Metode Searching (Sequential Search, Binary Search, Interpolation Search)***

***Strategi Searching (BFS, DFS dan Dijkstra)***

***Pemrograman Searching pada Google Colaboratory***

# SEARCHING



## Jabaran Materi

1. Metode Searching
2. Strategi Searching
3. Pemrograman Searching pada Google Colaboratory



## Uraian Materi

Pencarian data sering disebut dengan *table look* atau *storage* dan *retrieval information* yang merupakan suatu proses untuk mengumpulkan sejumlah informasi dalam komputer. Setelah informasi terkumpul, kemudian dilakukan pencarian kembali informasi yang diperlukan secara cepat (Suryadi, 2014). Algoritma pencarian (*searching algorithm*) adalah alur atau algoritma untuk mendapatkan suatu data dalam sekumpulan data berdasarkan kunci (*key*) atau acuan data. Setelah selesai melakukan proses pencarian, akan diperoleh dua kemungkinan yaitu data yang dicari tidak ditemukan (*unsuccessful*) atau data ditemukan (*successful*).

Pencarian (*searching*) merupakan suatu proses yang mendasar dalam pemrograman. Proses pencarian dapat dilakukan dengan dua metode yaitu pencarian statis (*static searching*) dan pencarian dinamis (*dynamic searching*). Pada pencarian statis rekaman data yang diperoleh dianggap tetap, sedangkan pada pencarian dinamis banyaknya rekaman data yang diperoleh bisa berubah-ubah. Hal tersebut dikarenakan adanya penambahan atau penghapusan pada rekaman data. Terdapat dua macam teknik pencarian yaitu pencarian sekuensial

dan pencarian biner. Perbedaan dari kedua teknik tersebut terletak pada keadaan data. Pencarian sekuensial (*sequential search*) digunakan apabila data dalam keadaan tidak teratur atau acak, sedangkan pencarian biner (*binary search*) digunakan apabila data sudah dalam keadaan urut.

*Searching* pada umumnya memiliki tiga metode yaitu *sequential search*, *binary search*, dan *interpolation search*. Proses pencarian dilakukan dengan menggunakan tiga strategi yaitu *Depth-First Search* (DFS), *Breadth-First Search* (BFS), dan *Dijkstra*.



## 1. Metode Searching

Metode yang digunakan untuk membandingkan data pada *searching* ada dua yaitu *sequential search*, *binary search*, dan *interpolation search* (Sitorus, 2015).

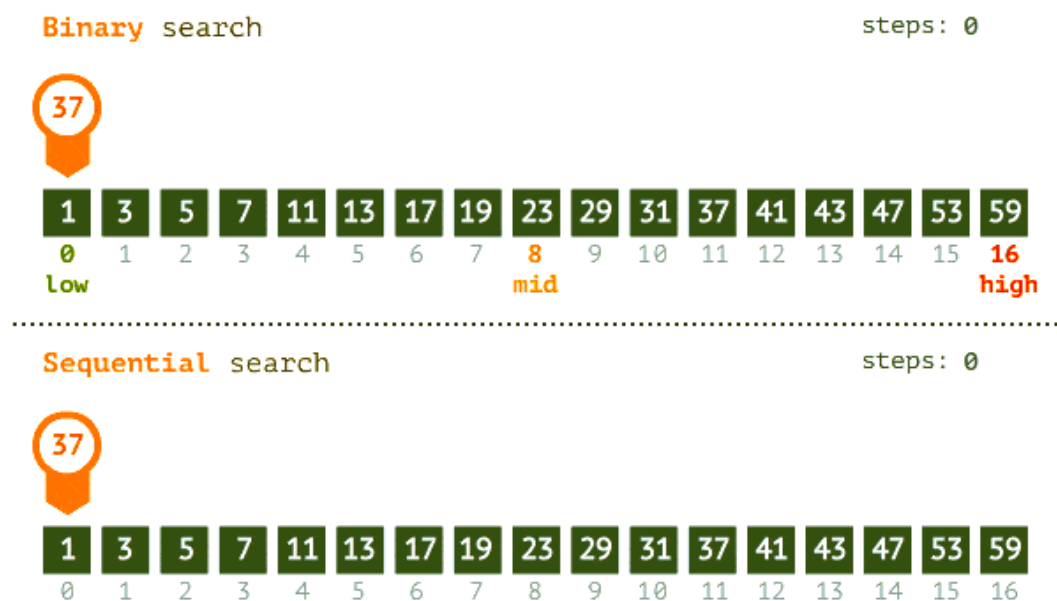
### A. **Sequential Search (linear search)**

*Sequential search* atau *linear search* merupakan teknik pencarian dengan membandingkan setiap elemen *array* satu per satu secara berurutan dimulai dari elemen pertama hingga elemen terakhir, sampai data yang dicari ditemukan. Metode *sequential search* dapat dikatakan sebagai metode yang paling mudah. Metode ini juga dapat dilakukan terhadap elemen *array* yang sudah teratur atau belum teratur. Proses pada metode *sequential search* bisa dikatakan singkat apabila data yang diolah sedikit, dan akan lama apabila data yang diolah banyak.

### B. **Binary Search**

*Binary search* adalah salah satu metode pencarian pada *array* yang sudah urut. Hal yang harus diperhatikan dalam penggunaan *binary search* yaitu data sudah dalam keadaan diurutkan. *Binary search* dilakukan dengan cara menebak apakah data yang dicari

berada di tengah, setelah itu dilakukan perbandingan antara data yang dicari dengan data yang berada di tengah. Apabila data yang berada di tengah sama dengan data yang dicari maka data ditemukan. Apabila data yang ditengah lebih besar dari yang dicari, maka kemungkinan data yang dicari berada di sebelah kiri dari data tengah, dan data yang berada di sebelah kanan data tengah dapat diabaikan. Data dari bagian kiri yang baru adalah indeks dari data tengah itu sendiri (*upper bound*). Sebaliknya, jika data yang berada di tengah lebih kecil dari data yang dicari, maka kemungkinan data yang dicari berada di sebelah kanan dari data tengah. Data sebelah kanan dari data tengah adalah indeks dari data tengah itu sendiri ditambah 1. Pada prinsipnya cara kerja dari metode *sequential search* dan *binary search* dapat dilihat pada ilustrasi Gambar 1.



**Gambar 1. Cara Kerja Sequential Search dan Binary Search**

### C. *Interpolation Search*

Metode *interpolation search* merupakan pengembangan dari *binary search*. Pada metode *binary search* akan selalu memeriksa

nilai tengah dari setiap array, sedangkan pada metode *interpolation search* dapat menuju ke lokasi yang berbeda berdasarkan key yang diperoleh. Apabila nilai key lebih dekat dengan array yang terakhir, maka metode *interpolation search* akan memulai pencarian dari array yang terakhir. Berikut merupakan contoh penerapan *interpolation search* (Gambar 2), elemen array dimulai dari 0.

1	3	5	7	9	11	13	15
0	1	2	3	4	5	6	7

**Gambar 2. Interpolation Search**

Untuk mendapatkan nilai yang dicari dengan menggunakan metode *interpolation search*, dapat dihitung dengan menggunakan rumus:

$$\text{Poss} = a(L) + \frac{x - L}{h - L} \times a(h) - a(L)$$

Keterangan:

- Poss = posisi
- L = urutan data terendah
- h = urutan data tertinggi
- a(L) = array terendah
- a(h) = array tertinggi
- x = nilai yang dicari

Misalnya nilai yang dicari adalah 9, maka dapat diperoleh perhitungan sebagai berikut:

$$\begin{aligned}
 \text{Poss} &= 0 + \frac{9 - 1}{15 - 1} \times 7 - 0 \\
 &= 0 + \frac{8}{14} \times 7 \\
 &= 0 + \frac{8}{2} \\
 &= 4
 \end{aligned}$$

Maka data yang dicari (9) berada pada elemen array yang ke-4.

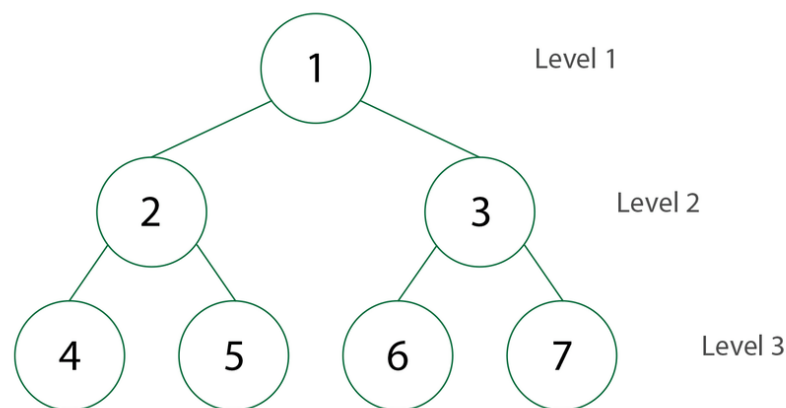




## 2. Strategi Searching

### A. *Breadth-First Search (BFS)*

Strategi *searching* BFS merupakan suatu pencarian yang dilakukan pada semua *node* di setiap level secara berurutan dari kiri ke kanan (Suyanto, 2014). Apabila pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya, demikian seterusnya sampai ditemukan solusi. Penggunaan strategi BFS dalam *searching* dapat diperoleh solusi yang ditemukan merupakan yang paling baik (optimal), akan tetapi strategi pencarian BFS harus menyimpan semua *node* dalam suatu antrian (*queue*). *Queue* tersebut digunakan untuk mengacu *node* bertetangga yang akan dikunjungi sesuai urutan pengantrian. Ilustrasi cara kerja BFS dapat dilihat pada Gambar 3, dimana algoritma BFS merupakan algoritma pencarian yang metodenya mencari pada setiap level *tree*.



**Gambar 3. Cara Kerja BFS**

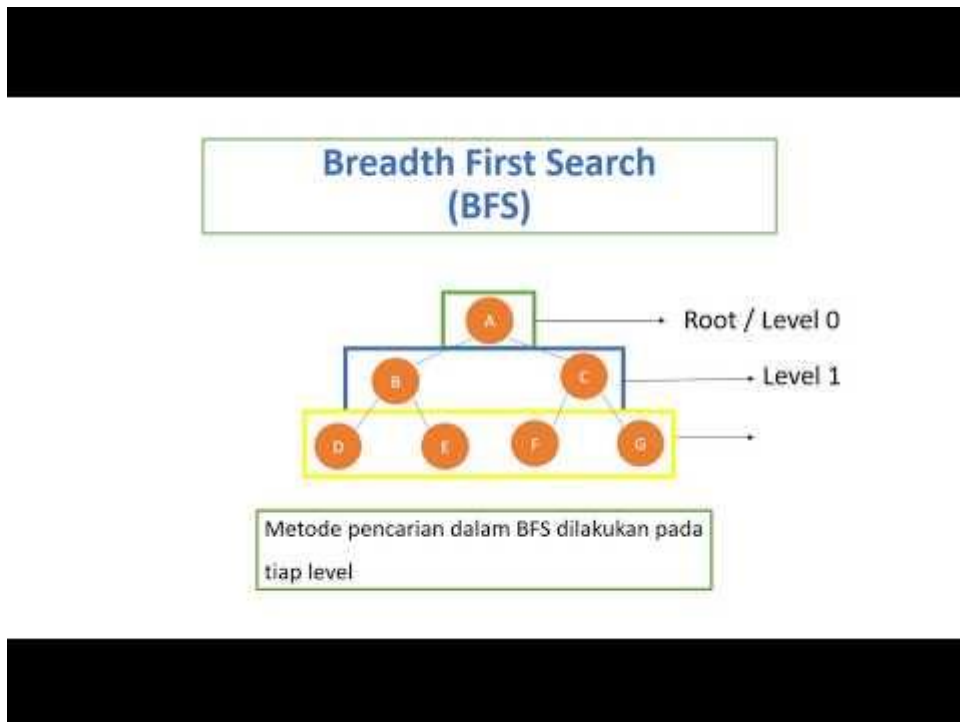
Sumber: <https://dev.to/snird/>

Berikut Langkah-langkah cara kerja BFS berdasarkan ilustrasi Gambar 3, dengan *node* awal 1 dan *node* tujuan 7:

1. Langkah pertama yaitu membuat antrian (*queue*) dan *output*

2. Memasukkan *node* 1 (level 1) dalam *queue* kemudian dimasukkan pada sebuah *output*.
3. Setelah itu, semua *node* pada level berikutnya (Level 2: *node* 2 dan 3) dimasukkan pada *queue* terlebih dahulu kemudian dituliskan pada *output*. Selanjutnya dicek apakah *node* tersebut merupakan solusi. Apabila *node* bukan solusi, maka dilanjutkan pada level berikutnya.
4. Semua *node* pada level berikutnya (Level 3: *node* 4, 5, 6, 7) dimasukkan dalam *queue* terlebih dahulu kemudian dituliskan pada *output*.
5. *Node* 7 merupakan tujuan akhir dari pencarian, maka hasil telah ditemukan dengan urutan *searching* BFS **1 – 2 – 3 – 4 – 5 – 6 – 7**.

Untuk lebih jelasnya terkait strategi *searching* BFS dapat dilihat pada Video 1 berikut:



**Video 1. Strategi Searching BFS**

Kelebihan BFS adalah :

- Tidak menemui jalan buntu
- Algoritma BFS mengunjungi semua *node* dan memastikan bahwa setiap *node* dikunjungi tepat satu kali dan tidak ada *node* yang dikunjungi dua kali (*back tracking*).
- Menjamin ditemukannya solusi (jika solusinya memang ada) dan solusi yang ditemukan pasti yang paling baik

Kelemahan BFS adalah :

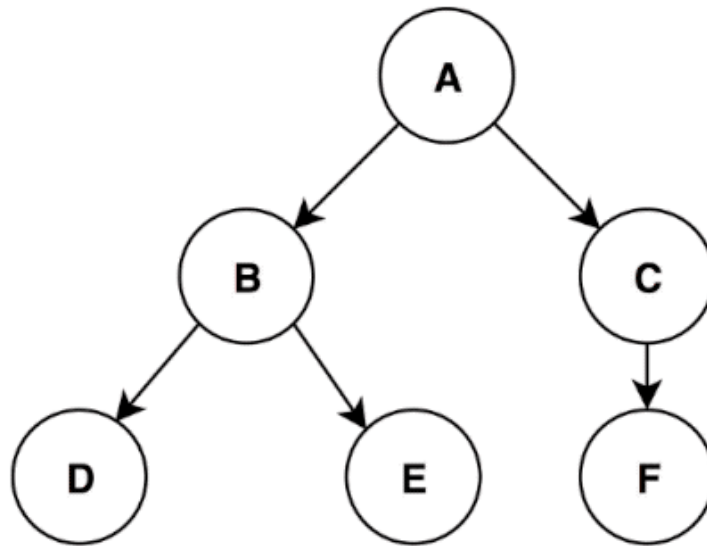
- Penggunaan memori yang cukup besar, karena menyimpan semua *node*.
- Membutuhkan waktu yang lama untuk menemukan solusi, karena menguji tiap  $n$  level untuk mendapatkan solusi pada level ke  $n-1$  (tidak optimal).

## **B. Depth-First Search (DFS)**

*Depth-First Search* (DFS) merupakan sebuah algoritma yang melakukan penelusuran atau pencarian pada pohon, struktur pohon, atau graf. pencarian ini dimulai pada simpul akar dan menelusuri sejauh mungkin sepanjang cabang yang ada. DFS ditemukan pada abad ke-19 oleh ahli matematika dari Prancis yang bernama Charles Pierre Tremaux.

Pencarian DFS dilakukan pada satu *node* dalam setiap level dari yang paling kiri, dimana semua simpul yang telah ditelusuri dengan DFS ini dimasukkan ke dalam sebuah *stack*. Jika pada level yang paling dalam solusi belum ditemukan, maka pencarian dilanjutkan pada *node* sebelah kanan, *node* yang berada di kiri dapat dihapus dari memori (Baidari, dkk., 2012). Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses

*backtracking* (penelusuran balik untuk mendapatkan jalur yang diinginkan). Ilustrasi cara kerja DFS dapat dilihat pada Gambar 4.



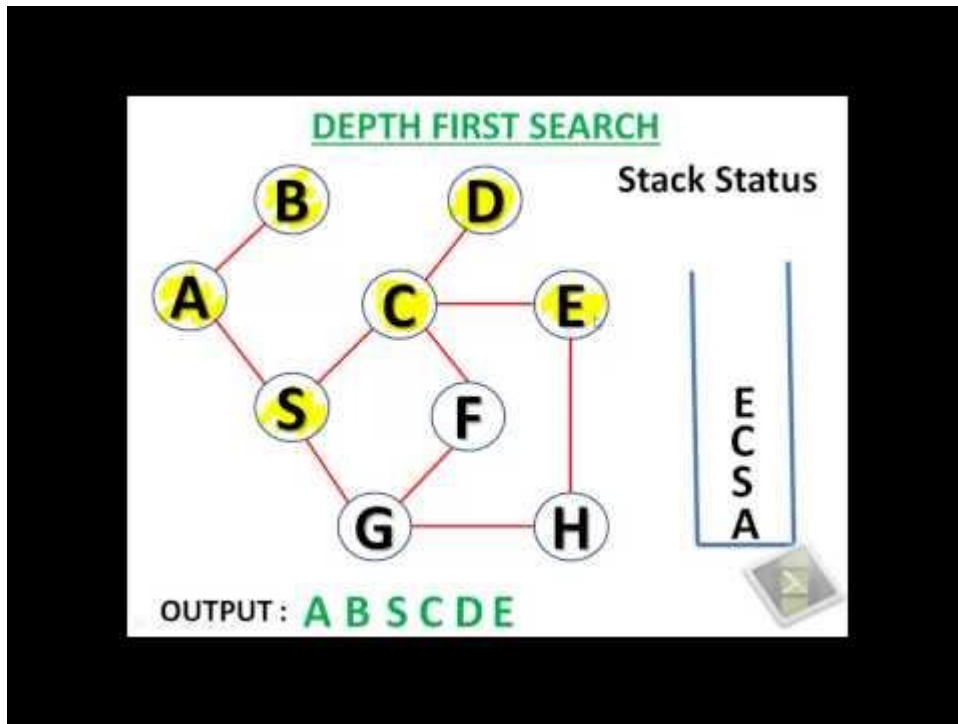
**Gambar 4. Cara Kerja DFS**

Berikut langkah-langkah cara kerja strategi pencarian DFS berdasarkan ilustrasi Gambar 4, dengan *node* awal A dan *node* tujuan F:

1. Mengunjungi *node* akar terlebih dahulu (*node* A).
2. Memeriksa apakah *node* akar mempunyai keturunan/cabang. Jika *node* tersebut memiliki keturunan (B dan C) , maka masukkan dalam *stack* dan kunjungi *node* tersebut mulai dari yang paling kiri.
3. Jika *node* yang paling kiri (*node* B mempunyai cabang D) telah dikunjungi semua, dilanjutkan mengunjungi *node* sebelahhanya (*node* E). Selanjutnya mengunjungi *node* C dan dilanjut pada *node* F sebagai cabang dari *node* C.
4. Apabila semua *node* yang bertetangga sudah dikunjungi, maka pencarian selesai dengan urutan:

A  
  B  
    D E  
  C  
    F

Untuk lebih jelasnya terkait strategi *searching* DFS dapat dilihat pada Video 2 berikut:



**Video 2. Strategi *searching* DFS**

Kelebihan DFS adalah :

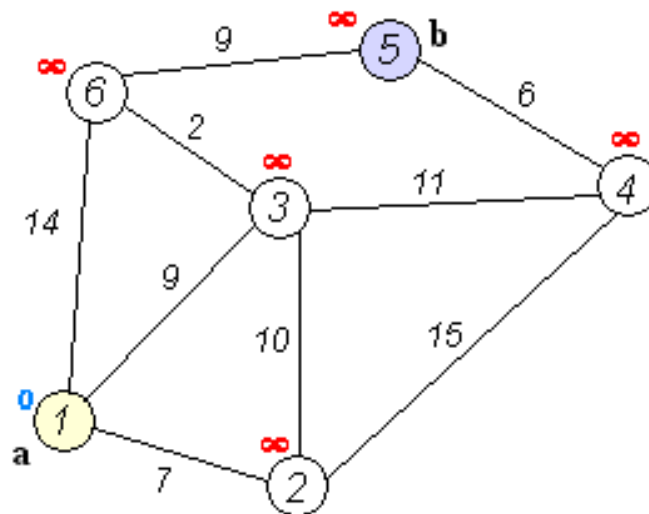
1. Penggunaan memori tidak terlalu banyak, berbeda dengan BFS yang harus menyimpan semua *node* yang pernah dibangkitkan.
2. Apabila solusi yang dicari berada pada level yang paling kiri, maka DFS tidak membutuhkan waktu yang lama dan akan menemukan secara cepat begitu sebaliknya.

Kelemahan DFS adalah :

1. Apabila pohon yang dibangkitkan mempunyai level yang dalam (tak hingga) maka tidak ada jaminan menemukan solusi (*not complete*)
2. Apabila terdapat lebih dari satu solusi yang sama tetapi berada dalam level yang berbeda, maka DFS tidak ada jaminan untuk menemukan solusi yang baik (tidak optimal).

### C. Dijkstra

Algoritma *dijkstra* merupakan salah satu bentuk algoritma yang populer untuk memecahkan persoalan yang terkait dengan masalah optimasi dan bersifat sederhana. Algoritma *dijkstra* ditemukan oleh Edger Wybe *Dijkstra*. *Dijkstra* dikenal sebagai algoritma yang mampu menyelesaikan masalah dengan rute pencarian terpendek menggunakan prinsip *greedy* (penyelesaian masalah dengan pencarian nilai maksimum) (Siswanto, 2013). Prinsip *greedy* yaitu mencari jalur terpendek dari satu *node* (titik/vertex) ke *node* lain yang searah (*directed graph*), dimulai dari *node* awal sampai pada *node* tujuan. Pada dasarnya *vertex* disimpan pada *array*, dan *edge* (bobot) dari suatu *vertex* yang disimpan pada *map* dalam *array*. Ilustrasi cara kerja *dijkstra* dapat dilihat pada Gambar 5.



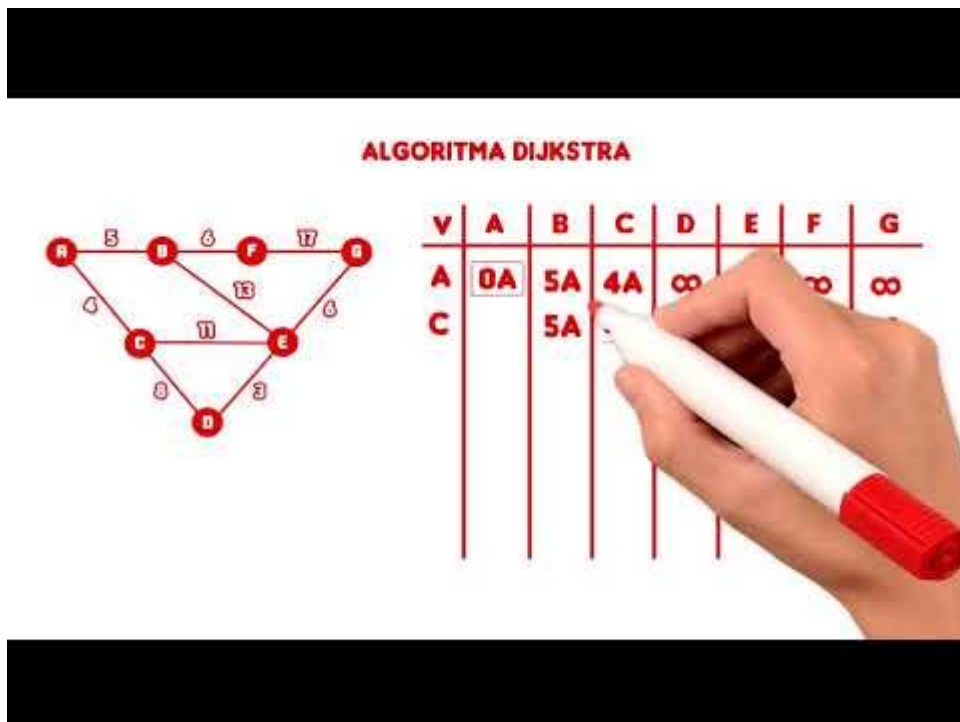
Gambar 5 Cara Kerja Dijkstra

Berikut langkah-langkah cara kerja strategi pencarian *dijkstra* berdasarkan ilustrasi Gambar 5, dengan *node* awal 1 dan *node* tujuan 5:

1. Melakukan pengambilan *edge* (garis yang berbobot positif) dengan bobot terkecil dari *node* utama (*node* 1).

2. Amati *node* mana saja yang terhubung dengan *node* pertama, kemudian jumlahkan *node* pertama dengan *node* yang terhubung langsung (*node* 2, 3, 6) dan pilihlah hasil penjumlahan bobot yang paling kecil (*node* 2)
3. Ulangi langkah 2 sesuai dengan tempat terakhir *node* dikunjungi (*node* 2), dimana *node* tersebut terhubung dengan (*node* 1, 3, 4). Pilihlah *node* baru dan memiliki jumlah bobot yang kecil (*node* 3)
4. *Node* 3 terhubung dengan (*node* 2, 4, 6), kemudian pilihlah *node* baru dan memiliki jumlah bobot yang kecil (*node* 6)
5. Selanjutnya *node* 6 terhubung dengan (*node* 1, 3, 5), karena *node* tujuan adalah 5 maka penjumlahan bobot terakhir dijumlahkan dengan bobot menuju 5.
6. Hasil pencarian menggunakan strategi pencarian *dijkstra* yaitu dengan rute **1 – 2 – 3 – 6 – 5** dengan nilai bobot **20**

Untuk lebih jelasnya terkait strategi *searching* DFS dapat dilihat pada Video 3 berikut:



**Video 3. Strategi *searching* Dijkstra**

Kelebihan *dijkstra* adalah:

1. *Dijkstra* merupakan algoritma yang digunakan untuk memetakan jalur alternatif, apabila jalur utama mengalami hambatan.
2. *Dijkstra* tidak menyelesaikan lintasan bernilai negatif dan hanya mencari bobot minimum dari satu *node* ke *node* lain yang saling berkaitan.
3. *Dijkstra* mampu menyelesaikan permasalahan rute terpendek. Elemen (bobot) dari rute tersebut berupa jarak tempuh, biaya, atau yang lainnya.
4. Pencarian dilakukan dengan mengunjungi *node* yang saling terhubung.
5. Memori yang dibutuhkan untuk mengimplementasikan algoritma *dijkstra* tidak terlalu banyak.

Kekurangan *dijkstra* adalah:

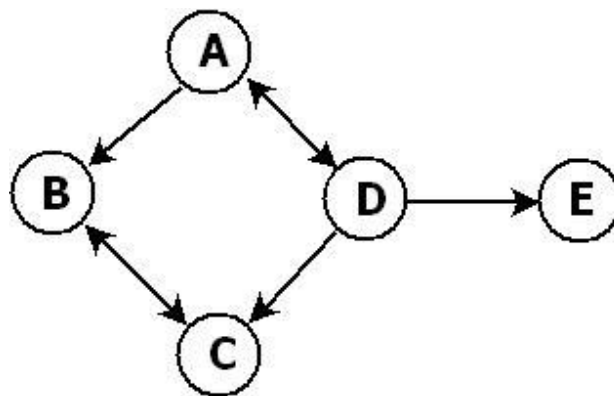
Pencarian yang dilakukan tidak terpusat, sehingga proses pencarian akan berlangsung lama dan mempunyai jumlah langkah yang cukup besar.

### 3. Pemrograman Searching pada Google Colaboratory

Penggunaan *google colaboratory* pada penerapan algoritma *searching* mampu mempercepat komputasi dengan fitur GPU yang terdapat di dalamnya. Pada dasarnya *google colaboratory* menggunakan bahasa pemrograman *python* untuk menjalankan perintah dari *user*. Penerapan algoritma *searching* pada *google colaboratory* berbeda-beda sesuai dengan strategi pencarian yang dilakukan. Penyelesaian masalah pada Gambar 7 dapat diselesaikan dengan menggunakan algoritma BFS dan DFS. Adapun langkah pertama yang harus dilakukan yaitu dengan menyiapkan



*graph* awal untuk memudahkan dalam mengasumsikan urutan *node*. *Graph* merupakan kumpulan dari busur dan simpul yang dinyatakan dalam  $G = (V, A)$ . Sebuah *graph* ada yang hanya terdiri dari satu simpul, *graph* belum tentu berhubungan dengan busur, *graph* mungkin mempunyai simpul yang tak terhubung dengan simpul yang lain, *graph* memungkinkan semua simpulnya berhubungan. Gambar 6 merupakan salah satu contoh *graph* yang saling berhubungan.



**Gambar 6. Penerapan BFS dan DFS**

Kode program yang digunakan untuk merepresentasikan hubungan setiap *node* pada BFS dan DFS adalah:

```
+ Code + Text      Connect | Editing | ^
graph = {
  'A' : ['B', 'C'],
  'B' : ['A', 'C'],
  'C' : ['B', 'D'],
  'D' : ['A', 'C', 'E'],
  'E' : ['D']
}
```

### **Menjalankan fungsi BFS pada Google Colaboratory:**

Pada BFS, setelah menuliskan hubungan setiap *node* selanjutnya data dan antrian yang telah dibuat disimpan menggunakan kode program berikut:

```
Visited = []
Queue = []
```

Untuk menjalankan fungsi BFS yaitu dengan menggunakan kode:

```
def bfs (visited, graph, node):
    visited.append(node)
    queue.append(node)
```

Untuk memasukkan antrian paling depan ke dalam variabel jalur dengan menggunakan kode program:

```
while queue:
    s = queue.pop(0)
    print (s, end = " ")
```

Selanjutnya, untuk melakukan pengecekan semua cabang dari *state* yaitu dengan menggunakan kode program:

```
for neighbour in graph[s]:
    if neighbour not in visited:
```

Apabila jalur cabang tidak dikunjungi maka digunakan kode program berikut untuk mengupdate antrian dengan jalur baru :

```
visited.append(neighbour)
queue.append(neighbour)
```

Jalankan kode berikut untuk mendapatkan jalur yang dikunjungi menggunakan BFS:

```
bfs(visited, graph, node awal yang ingin
dikunjungi)
```

Apabila *node* awal yang ingin dikunjungi adalah B maka kode yang dijalankan dan *output* BFS yang dihasilkan sebagai berikut:

```
bfs(visited, graph, 'B')
```

```
B A C D E
```

### Menjalankan fungsi DFS pada Google Colaboratory:

Pada DFS setelah menuliskan hubungan setiap *node*, data akan disimpan dan dikunjungi dengan menggunakan kode program:

```
visited = set ()
```

Langkah selanjutnya menuliskan Fungsi DFS menggunakan kode program:

```
def dfs (visited, graph, node):
```

Jika solusi tidak ditemukan pada *node* yang dikunjungi, maka akan dilanjutkan pada *node* berikutnya (*node* tatangga) dan terus berulang sampai solusi ditemukan dengan menggunakan kode program berikut:

```
if node not in visited:
    print (node)
    visited.add(node)
    for neighbour in graph[node]:
        dfs(visited, graph, neighbour)
```

Jalankan kode berikut untuk mendapatkan jalur yang dikunjungi menggunakan DFS:

```
dfs(visited, graph, node awal yang ingin dikunjungi)
```

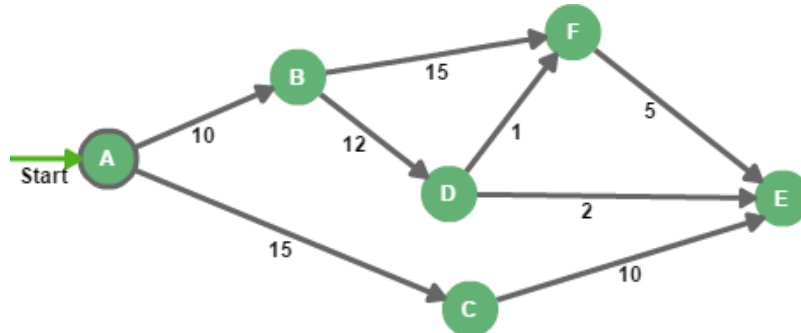
Misal *node* awal yang ingin dikunjungi adalah B maka kode yang dijalankan dan *output* DFS yang dihasilkan sebagai berikut:

```
dfs(visited, graph, 'B')
```

```
B
A
C
D
E
```

### Menjalankan fungsi *Dijkstra* pada *Google Colaboratory*:

Penerapan strategi *searching* menggunakan *dijkstra* dapat dilihat pada Gambar 7 berikut:



Gambar 7. Penerapan *Dijkstra*

Untuk merepresentasikan hubungan setiap *node*, berikut kode yang digunakan beserta nilai bobot setiap *node*:

```
graph = {'a':{'b':10,'c':15},  
        'b':{'d':12,'f':15},  
        'c':{'e':10},  
        'd':{'e':2,'f':1},  
        'e':{'c':10,'d':2,'f':5},  
        'f':{'e':5}}
```

Maksud dari program tersebut yaitu, *node* A berhubungan dengan *node* B dengan bobot 10, dan berhubungan dengan C dengan bobot 15. Begitu seterusnya dengan *node* yang lainnya. Kemudian jalankan fungsi *Dijkstra* dengan menggunakan kode berikut:

```
def dijkstra(graph,start,goal):
```

Untuk mencatat bobot yang digunakan agar mencapai *node* yang diinginkan menggunakan kode berikut:

```
    shortest_distance = {}  
    predecessor = {}
```

Sedangkan kode program yang digunakan untuk melacak jalur yang menuju ke simpul sebagai berikut:

```

unseenNodes = graph
infinity = 9999999
path = []

```

Menetapkan 0 sebagai nilai untuk mewakili *node* sumber:

```

for node in unseenNodes:
    shortest_distance[node] = infinity
shortest_distance[start] = 0

```

Perulangan akan terus berjalan sampai semua *node* telah ditelusuri.

`minNode = None` digunakan untuk menentukan jarak minimum.

Seperti pada kode program berikut:

```

while unseenNodes:
    minNode = None
    for node in unseenNodes:
        if minNode is None:
            minNode = node
        elif shortest_distance[node] < shortest_distance[minNode]:
            minNode = node

```

Selanjutnya mengambil nilai bobot yang paling kecil untuk dilewati,

menuju *node* tujuan dengan menggunakan kode program berikut:

```

for childNode, weight in graph[minNode].items():
    if weight + shortest_distance[minNode] < shortest_distance[childNode]:
        shortest_distance[childNode] = weight + shortest_distance[minNode]
        predecessor[childNode] = minNode
unseenNodes.pop(minNode)

```

*Node* yang dikunjungi hanya *node* baru, sehingga tidak mengulangi *node* yang pernah dilewati agar mencapai *node* tujuan. Kemudian dilakukan penelusuran jalur yang sudah dilewati dan menghitung total bobot yang terakumulasi dengan kode program:

```

currentNode = goal
while currentNode != start:
    try:
        path.insert(0,currentNode)
        currentNode = predecessor[currentNode]
    except KeyError:
        print('Path not reachable')
        break
path.insert(0,start)

```

Selanjutnya kode program berikut digunakan apabila jumlah bobot tidak terbatas dan *node* tujuan belum tercapai.

```

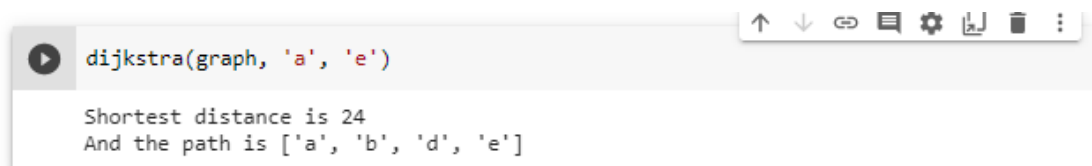
if shortest_distance[goal] != infinity:
    print('Shortest distance is ' + str(shortest_distance[goal]))
    print('And the path is ' + str(path))

```

Jalankan kode berikut untuk mendapatkan jalur yang dikunjungi menggunakan *dijkstra*:

```
dfs(graph, node awal yang ingin dikunjungi)
```

Misal *node* yang ingin dikunjungi dimulai dari A menuju E maka kode yang dijalankan dan *output dijkstra* yang dihasilkan sebagai berikut:



```

dijkstra(graph, 'a', 'e')

```

Shortest distance is 24  
And the path is ['a', 'b', 'd', 'e']

Penjabaran mengenai pemrograman *searching* menggunakan *google colaboratory* lebih lanjut dapat diakses melalui jobsheet yang terdapat pada *e-collab classroom*. Pada jobsheet tersebut terdapat pemrograman yang terstruktur untuk menggambarkan algoritma *searching* (BFS, DFS, dan *Dijkstra*) sehingga tahapan pada setiap algoritmanya dapat diterapkan.



## Rangkuman Materi

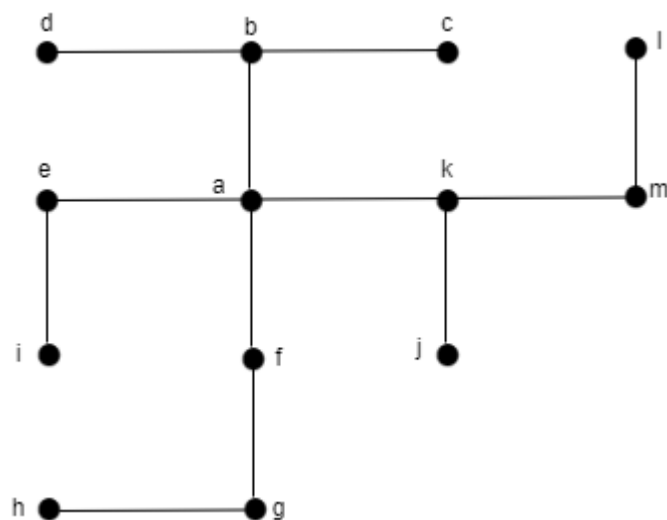
1. Algoritma *searching* dapat digunakan untuk menyelesaikan permasalahan dalam pengambilan suatu keputusan,
2. *Searching* merupakan algoritma pencarian yang mendasar dalam pemrograman.
3. Metode yang sering digunakan dalam *searching* yaitu *sequential search (linear search)*, *binary search*, dan *interpolation search*
4. Proses pencarian dilakukan dengan menggunakan tiga strategi yaitu *Depth-First Search (DFS)*, *Breadth-First Search (BFS)*, dan *Dijkstra*.
5. Strategi DFS menyimpan data dengan jumlah memori yang kecil, sedangkan pada BFS menyimpan data dalam jumlah memori yang cukup besar. Kedua strategi *searching* tersebut belum optimal, karena membutuhkan waktu lama untuk menemukan suatu *node* yang paling dalam,
6. *Dijkstra* merupakan strategi pencarian yang mampu menemukan jalur terpendek dengan menggunakan prinsip *greedy* yaitu dengan mencari nilai maksimum.



## Materi Pengayaan

1. Penerapan metode *searching* BFS dan DFS pada sistem maze/labirin sederhana.

Pada algoritma DFS, pencarian dilakukan pada satu *node* dalam setiap level dan dimulai dari yang paling kiri. Apabila pada level yang paling dalam solusi belum ditemukan, maka pencarian dilanjutkan pada *node* sebelah kanan. *Node* yang berada di bagian kiri dapat dihapus dari memori, sedangkan pada algoritma BFS menyimpan semua *node* dalam satu antrian sehingga membutuhkan memori yang cukup besar. Penggunaan algoritma BFS dapat menemukan solusi yang paling baik. Misalnya suatu ruang keadaan masalah ditunjukkan seperti Gambar 8, dengan ketentuan *node* awal berada di huruf **a** dan *node* akhir huruf **l**:

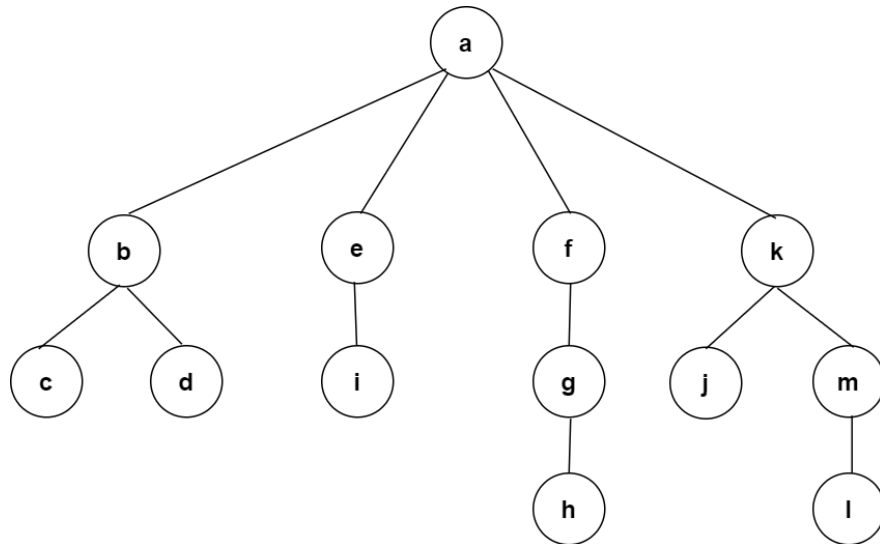


Gambar 8. Studi Kasus *Searching*



Langkah yang dilakukan untuk mendapatkan solusi terbaik dengan menggunakan metode *searching* BFS dan DFS adalah sebagai berikut:

1. Mengubah *node* di atas menjadi sebuah *graph*:



**Gambar 9. Graph Studi Kasus Searching**

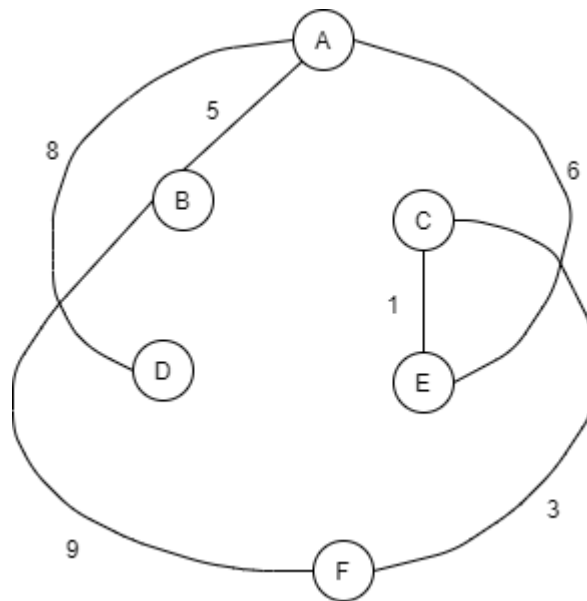
2. Menentukan jalur terpendek antar *node* tersebut menggunakan Teknik pencarian BFS dan DFS.

BFS = **a - b - e - f - k - c - a - i - g - j - m - h - l**

DFS = **a**

**b**  
**c d**  
**e**  
**i**  
**f**  
**g**  
**h**  
**k**  
**j m**  
**l**

2. Penerapan metode *searching dijkstra* pada rute perjalanan. *Dijkstra* dikenal sebagai suatu algoritma yang mampu menemukan solusi dengan rute pencarian terpendek menggunakan prinsip *greedy*. Contoh masalah pada algoritam *dijkstra* pada Gambar 10.



**Gambar 10. Rute dari desa A ke desa E**

Berdasarkan Gambar 10, cara untuk menentukan *shortest route* dari desa A menuju desa F menggunakan algoritma *dijkstra* adalah sebagai berikut:

Initialisasi

Vertex	Known	Cost	Path
A	F	$\infty$	-1
B	F	$\infty$	-1
C	F	$\infty$	-1
D	F	$\infty$	-1
E	F	$\infty$	-1
F	F	$\infty$	-1

Dari A

Vertex	Known	Cost	Path	
A	T	0	-1	
B	F	5	A	$0+5 < \infty$
C	F	$\infty$	-1	
D	F	8	A	$0+8 < \infty$
E	F	6	A	$0+6 < \infty$
F	F	$\infty$	-1	

Dari B

Vertex	Known	Cost	Path	
A	T	0	-1	$5+5 < 0$
B	T	5	A	
C	F	$\infty$	-1	
D	F	8	A	
E	F	6	A	$0+6 < \infty$
F	F	14	B	$5+9 < \infty$

Dari E

Vertex	Known	Cost	Path	
A	T	0	-1	$8+6 < 0$
B	T	5	A	
C	F	7	E	$6+1 < \infty$
D	F	8	0	
E	F	6	0	
F	F	14	B	

Dari C

Vertex	Known	Cost	Path
A	T	0	-1
B	T	5	A
C	F	7	E
D	F	8	A
E	F	6	A
F	F	10	C

$$7+1 < 6$$

$$7+3 < 14$$

Dari D

Vertex	Known	Cost	Path
A	T	0	-1
B	T	5	A
C	F	7	E
D	F	8	A
E	F	6	A
F	F	10	C

$$8+8 < 0$$

$$10+9 < 5$$

$$10+3 < 7$$

Sehingga diperoleh rute tersingkat dari desa A menuju desa F yaitu A – E – C – F dengan bobot 10.

# 3

## EVALUASI

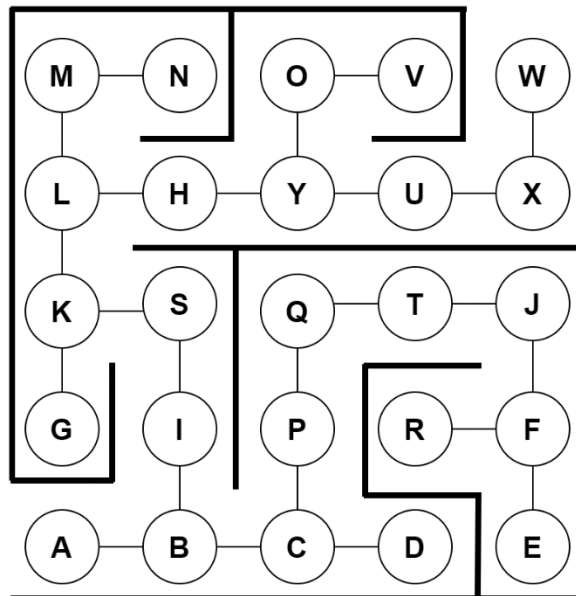
**Tes Individu**

**Referensi**

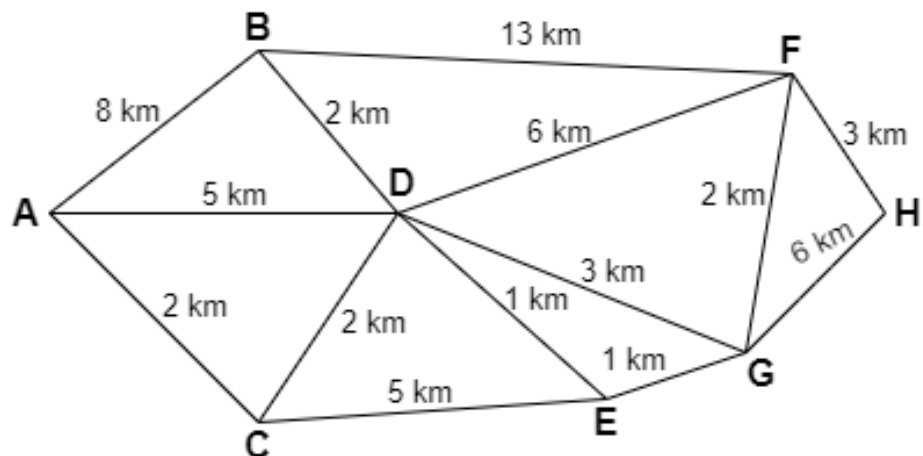


### Tes Individu

1. Tentukan solusi permasalahan maze/labirin berikut dengan menggunakan algoritma *searching* BFS dan DFS



2. Tentukan rute terpendek dari kota A menuju kota H dan analisislah jarak yang ditempuh untuk menuju kota H tersebut!





## Referensi

- Baidari, I., Roogi, R. and Shinde, S. 2012. Algorithmic Approach to Eccentricities, Diameters and Radii of *Graphs* using DFS. International Journal of Computer Applications, 54(18), 1-4, (Online), (<https://www.ijcaonline.org/archives/volume54/>)
- Siswanto. 2013. Pencarian Rute Terpendek Algoritma Dijkstra dari Setiap Titik yang Telah Dibangun. Yogyakarta.
- Sitorus, Lamhot. 2015. Algoritma dan Pemrograman. Yogyakarta: Andi, (Online), (<https://books.google.co.id/books?id>)
- Suryadi, S. 2014. Perancangan Aplikasi Pencarian File Dengan Menggunakan Metode Best First Search. Sumatra Utara: Universitas Labuhan Batu, (Online), (<https://garuda.ristekbrin.go.id/documents/detail/>)
- Suyanto. 2014. Artificial Intelligence (Revisi Kedua). Bandung: Penerbit Informatika.