**Nama : Meriza Corry Nurshinta**

**NIM : 2014157027**

**Mata kuliah : Kecerdasan Buatan**

**Informatika**

1. **Breadth-first search**

**PENGERTIAN ALGORITMA BFS**

*Breadth-first search* adalah algoritma yang melakukan pencarian secara melebar yang mengunjungi simpul secara *preorder* yaitu mengunjungi suatu simpul kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu. Selanjutnya, simpul yang belum dikunjungi dan bertetangga dengan simpulsimpul yang tadi dikunjungi , demikian seterusnya. Jika graf berbentuk pohon berakar, maka semua simpul pada aras *d* dikunjungi lebih dahulu sebelum simpul-simpul pad aras *d*+1.

Algoritma ini memerlukan sebuah antrian *q* untuk menyimpan simpul yang telah dikunjungi. Simpulsimpul ini diperlukan sebagai acuan untuk mengunjungi simpul-simpul yang bertetanggaan dengannya. Tiap simpul yang telah dikunjungu masuk ke dalam antrian hanya satu kali. Algoritma ini juga membutuhkan table Boolean untuk menyimpan simpul yang te lah dikunjungi sehingga tidak ada simpul yang dikunjungi lebih dari satu kali.

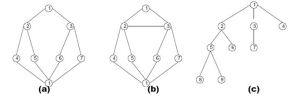
**CARA KERJA ALGORITMA BFS**

Dalam algoritma BFS, simpul anak yang telah dikunjungi disimpan dalam suatu antrian. Antrian ini digunakan untuk mengacu simpul-simpul yang bertetangga dengannya yang akan dikunjungi kemudian sesuai urutan pengantrian.

Untuk memperjelas cara kerja algoritma BFS beserta antrian yang digunakannya, berikut langkah-langkah algoritma BFS:

1. Masukkan simpul ujung (akar) ke dalam antrian
2. Ambil simpul dari awal antrian, lalu cek apakah simpul merupakan solusi
3. Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.
4. Jika simpul bukan solusi, masukkan seluruh simpul yang bertetangga dengan simpul tersebut (simpul anak) ke dalam antrian
5. Jika antrian kosong dan setiap simpul sudah dicek, pencarian selesai dan mengembalikan hasil solusi tidak ditemukan
6. Ulangi pencarian dari langkah kedua

Contohnya terlihat dibawah ini:

[](https://onbuble.files.wordpress.com/2011/05/1.jpg)

Maka penyelesaiannya adalah:

Gambar (a) BFS(1): 1, 2, 3, 4, 5, 6, 7, 1.

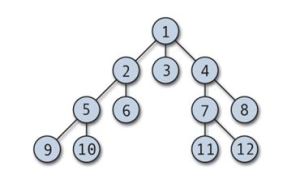
Gambar (b) BFS(1): 1, 2, 3, 4, 5, 6, 7, 1

Gambar (c) BFS(1): 1, 2, 3, 4, 5, 6, 7, 8, 9

**CONTOH PENCARIAN LINTASAN TERPENDEK DENGAN BFS**

Adapun contoh untuk mencari lintasan terpendek dengan menggukan algoritma BFS adalah sebagai berkut:

Diketahui sebuah kota, dengan memiliki inisial seperti yang ditunjukkan dibawah ini. Jarak antar kota dibentuk dengan sebuah graph terlihat dibawah:

[](https://onbuble.files.wordpress.com/2011/05/2.jpg)

Pertanyaan: sebutkan rute yang akan ditempuh untuk mencapai kota no. 8. Titik awal perjalanan adalah kota no. 1. Gunakan algoritma BFS!

Maka dengan menggunakan algoritma BFS, rute tercepat yang didapat adalah sebagai berikut:

1 – 2 – 3 – 4 – 5 – 6 – 7 – 8

Rute tersebut didapat dari pencarian secara melebar. Hal; tersebut dapat dijabarkan sebagai berikut:

–Pertama-tama, pointer menunujuk pada daun yang ada sebelah kanan, yaitu no.2 (1 – 2)

–Setelah itu, proses dilanjutkan pada tetangga no.2 yaitu no.3 (1-2-3) dan selanjutnya mengarah pada tetangga terdekat, yakni no.4 (1-2-3-4).

–Pointer mencari teteangga no.4, namun karna tidak ada, maka pointer kembali ke kota no.2 dan masuk ke daun berikutnya, yakni no.5.

–Proses diulang hingga pointer menunjuk angka 8

**PENERAPAN BFS DALAM bahasa algoritma**

Adapun penerapan BFS pada algoritma adalah sebagai berikut:

Prosedur breadth\_first\_search

Inisialisasi : open = [start]; closed [ ]

While open = [ ] do

Begin

Hapuskan keadaan paling kiri dari keadaan open,

sebutlah keadaan itu dengan X;

Jika X merupakan tujuan then return (sukses);

Buatlah semua child dari X;

Ambillah X dan masukkan pada closed;

Eliminasilah setiap child X yang telah berada pada

open atau closed, yang akan menyebabkan loop

dalam search;

Ambillah turunan di ujung kanan open sesuai urutan

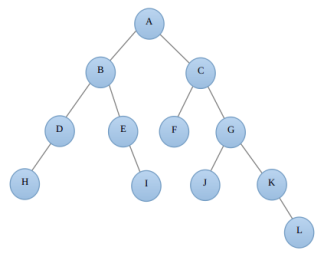
penemuan-nya;

End;

1. **DFS (Depth-First-Search)**

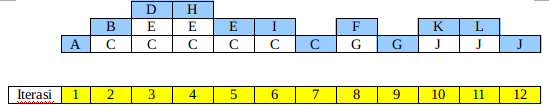
DFS (Depth-First-Search) adalah salah satu algoritma penelusuran struktur graf / pohon berdasarkan kedalaman. Simpul ditelusuri dari *root* kemudian ke salah satu simpul anaknya ( misalnya prioritas penelusuran berdasarkan anak pertama [simpul sebelah kiri] ), maka penelusuran dilakukan terus melalui simpul anak pertama dari simpul anak pertama level sebelumnya hingga mencapai level terdalam.

Setelah sampai di level terdalam, penelusuran akan kembali ke 1 level sebelumnya untuk menelusuri simpul anak kedua pada pohon biner [simpul sebelah kanan] lalu kembali ke langkah sebelumnya dengan menelusuri simpul anak pertama lagi sampai level terdalam dan seterusnya.

Jadi, jika ada pohon biner seperti gambar di bawah ini :  
[](https://saungkode.files.wordpress.com/2014/04/pohonbiner.png)  
Maka, urutan penelusurannya adalah : A – B – D – H – E – I – C – F – G – J – K – L

Dalam implementasinya DFS dapat diselesaikan dengan cara rekursif atau dengan bantuan struktur data stack. Kita akan membahas dengan cara yang menggunakan stack. Stack yang digunakan adalah stack yang isi elemennya adalah simpul pohon / tree. Bagaimana cara kerjanya ? Berikut ini adalah urutan algoritmanya :

1. Masukkan simpul *root* ke dalam tumpukan dengan push
2. Ambil dan simpan isi elemen (berupa simpul pohon) dari tumpukan teratas
3. Hapus isi stack teratas dengan prosedur pop
4. Periksa apakah simpul pohon yang disimpan tadi memiliki anak simpul
5. Jika ya, push semua anak simpul yang dibangkitkan ke dalam stack
6. Jika tumpukan kosong berhenti, tapi jika tidak kembali ke langkah dua

Jadi, untuk gambar pohon biner di atas urutan langkah dan kondisi stack-nya setiap iterasi adalah :  
[](https://saungkode.files.wordpress.com/2014/04/dfsstack.png)  
Contoh diatas menggunakan prioritas untuk memasukkan anak simpul dari sebelah kanan terlebih dahulu ke dalam stack. Sehingga, pada iterasi 2 elemen A dihapus lalu memasukkan anak simpulnya yaitu C dulu, baru B ke dalam stack. Selain itu bisa dilihat stack teratas (yang diwarna biru) pada tiap iterasi memiliki urutan A – B – D – H – E – I – C – F – G – J – K – L. Oiya, pada iterasi ke 13 itu kondisi stack sudah kosong karena ketika simpul J dibangkitkan tidak ada anak simpul yang dimasukkan ke stack.

**Algoritma DFS sebagai berikut :**

Begin

Push node awal pada stack, yang ditunjukkan dengan stack\_top

While stack tidak kosong *do*

Begin

Pop stack untuk mendapatkan stack\_top element

*If* stack\_top\_element = Tujuan

Return

Sukses dan stop

*Else* push turunan (cabang) stack\_top element ke dalam stack

End while

End

**STUDI KASUS :**

Pada suatu hari ada seorang petani yang mempunyai seekor kambing dan srigala.Pada saat itu ia baru saja panen sayuran. Karena membutuhkan uang, petani tersebut hendak menjual kambing, serigala, dan sayurannya ke pasar Johar. Untuk sampai di pasar Johar, ia harus menyeberangi sebuah sungai.

Permasalahannya adalah di sungai itu hanya tersedia satu perahu saja yang bisa memuat petani dan satu penumpang lainnya (kambing, srigala, atau sayuran). Jika ditinggalkan oleh petani tersebut, maka sayuran akan dimakan oleh kambing dan kambing akan dimakan oleh serigala

**DESKRIPSI:**

P = Petani  
Sy = Sayuran  
K = Kambing  
Sg = Serigala

**RUANG KEADAAN**  
Untuk daerah asal dan daerah seberang  
digambarkan.  
(P, Sy, K, Sg)

**KEADAAN AWAL**  
Daerah Asal = (P, Sy, K, Sg)  
Daerah seberang = (0, 0, 0, 0)

**TUJUAN**  
Daerah Asal = (0, 0, 0, 0)  
Daerah seberang = (P, Sy, K, Sg)

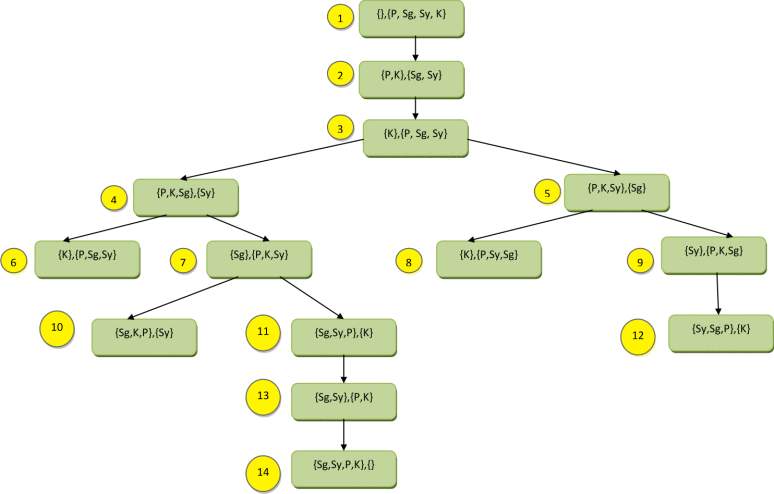
**METODE PENYELESAIAN :**

Terdapat dua metode penyelesaian yang akan dibahas pada postingan kali ini. Metode pertama adalah metode *breadth first search (BFS),*dan metode kedua adalah metode *Depth First Search (DFS)*. Berikut ini akan dijelaskan penyelesaian studi kasus diatas dengan kedua metode tersebut.

* **BFS (Breadth First Search)**

1. Masukkan simpul akar ke dalam antrian *Q*. Jika simpul akar = simpul solusi (*goal node*), maka stop.
2. Jika *Q* kosong, tidak ada solusi. Stop.
3. Ambil simpul *v* dari kepala (*head*) antrian, bangkitkan semua anak-anaknya. Jika *v* tidak mempunyai anak lagi, kembali ke langkah 2. Tempatkan semua anak dari *v* di belakang antrian.
4. Jika suatu simpul anak dari *v* adalah simpul solusi, maka solusi telah ditemukan, kalau tidak kembali lagi ke langkah 2.

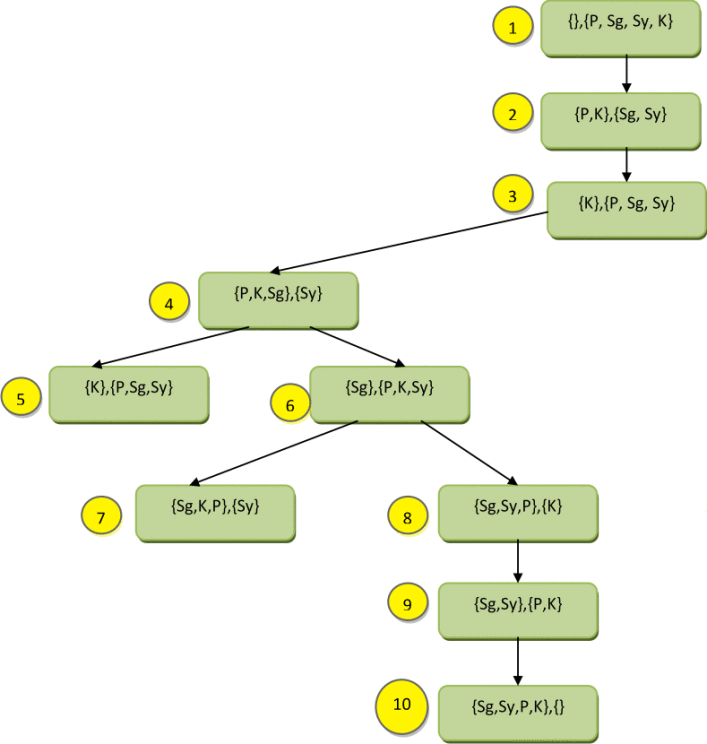
* **GAMBAR BFS :**

[](https://creactiveit.files.wordpress.com/2015/05/graphic2.png)

* **DFS (Depth First Search)**

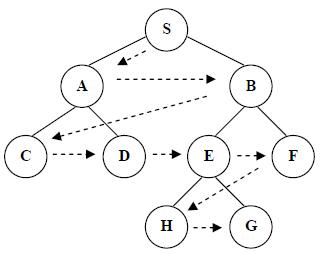
1. Masukkan simpul akar ke dalam antrian *Q*. Jika simpul akar = simpul solusi, maka stop.
2. Jika *Q* kosong, tidak ada solusi. Stop.
3. Ambil simpul *v* dari kepala (*head*) antrian. Jika kedalaman simpul *v* sama dengan batas kedalaman maksimum, kembali ke langkah 2
4. .Bangkitkan semua anak dari simpul *v*. Jika *v* tidak mempunyai anak lagi, kembali ke langkah 2. Tempatkan semua anak dari *v* di awal antrian *Q.*Jika anak dari simpul *v* adalah simpul tujuan, berarti solusi telah ditemukan, kalau tidak, kembali lagi ke langkah 2.

* **GAMBAR DFS :**

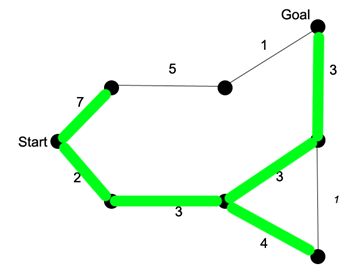
[](https://creactiveit.files.wordpress.com/2015/05/graphic1.png)

1. **Uniform Cost Search**

Uniform Cost Search adalah algoritma Seach Tree (graph) yang digunakan untuk menyelesaikan beberapa persoalan . Algoritma ini memulai pencarian dari root node, kemudian dilanjutkan ke node-node selanjutnya. Dimana node tersebut dipilih yang memilki harga (cost) terkecil dari root node. Algoritma ini merupakan modifikasi dari Bread First Search (BFS).



Dalam implementasi algoritma ini , melibatkan semua node yang berhubungan dengan root node, dan meletakannya dalam priority queue untuk mencapai node tujuan. Dimana node – node yang dipilih merupakan node yang berharga terkecil.Ilustrasi jalannya algoritma Uniform Cost Search dapat digambarkan sebagai berikut :



Seperti tampak pada gambar, initial state terletak pada node start, kemudian untuk mencapai node berikutnya, algoritma ini memilih jalur yang memilki harga terkecil diantara dua node di depannya. Begitu seterusnya, dilakukan pengecekan node yang memilki harga terkecil hingga sampai pada goal state.

Uniform Cost Search menggunakan priority queue (antrian prioritas). Algoritma menggunakan antrian prioritas ini adalah sebagai berikut:

**Masukkan root ke dalam antrian**

**While antrian not empty**

**Pilih elemen dengan prioritas maksimum dari antrian (dalam hal ini cost terkecil)**

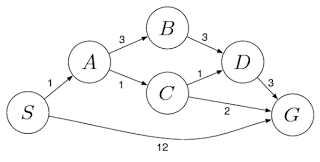
**If prioritas sama, jalur dipilih secara alphabetis terkecil**

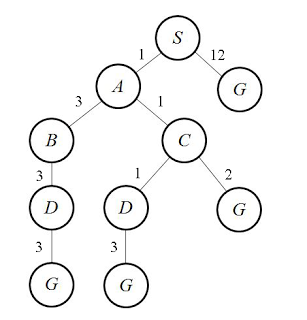
**If jalur sampai ke goal state, print jalur and exit**

**Else**

**Masukkan semua children dari elemen yang dipilih, dengan cost kumulatif sebagai prioritas (cost terkecil memiliki prioritas terbesar)**

Contoh penggunaan algoritma diatas pada pohon pencarian. Algoritma ditulis setiap iterai. Setiap elemen dari antrian prioritas ditulis [jalur, cost kumulatif].

[](http://1.bp.blogspot.com/-eTjJkb2lLis/Vf_zJ_mLuDI/AAAAAAAAAU8/c-4NwTYFpj0/s1600/searchproblem.png)

[](http://4.bp.blogspot.com/-jz8m0FaJ6lk/Vf_zIwl6y0I/AAAAAAAAAU4/cbXocb7DsYY/s1600/searchtree.png)

Initialisasi : { [ S , 0 ] }  
Iterasi 1 : { [ S->A , 1 ] , [ S->G , 12 ] }  
Iterasi 2 : { [ S->A->C , 2 ] , [ S->A->B , 4 ] , [ S->G , 12] }  
Iterasi 3 : { [ S->A->C->D , 3 ] , [ S->A->B , 4 ] , [ S->A->C->G , 4 ] , [ S->G , 12 ] }  
Iterasi 4 : { [ S->A->B , 4 ] , [ S->A->C->G , 4 ] , [ S->A->C->D->G , 6 ] , [ S->G , 12 ] }  
Iterasi 5 : { [ S->A->C->G , 4 ] , [ S->A->C->D->G , 6 ] , [ S->A->B->D , 7 ] , [ S->G , 12 ] }  
Iterasi 6 mengeluarkan output final yaitu S->A->C->G  
  
Di setiap iterasinya, algoritma ini tidak pernah melanjutkan node yang memiliki cost lebih besar dari cost kumulatif yang terkecil.  Elemen di antrian prioritas memiliki cost yang hampir sama dalam satu iterasi, karena itu disebut Uniform Cost Search. Di contoh diatas mungkin keseragaman cost tidak terlihat, tetapi jika digunakan pada graf yang lebih besar akan terlihat.