

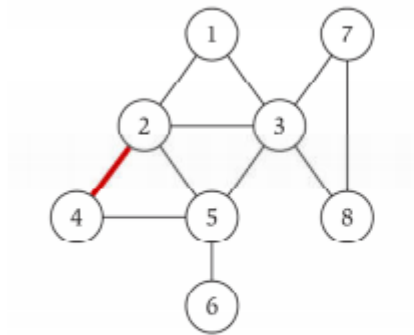
**LAPORAN PRAKTIKUM  
ANALISIS ALGORITMA**



**Dibuat oleh:**  
AHMAD IRFAN FADHOLI  
140810180034

**PROGRAM STUDI S-1 TEKNIK INFORMATIKA  
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM  
UNIVERSITAS PADJADJARAN  
2020**

1. Dengan menggunakan *undirected graph* dan *adjacency matrix* berikut, buatlah koding programnya menggunakan bahasa C++.



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

```

1.  /* Nama : Ahmad Irfan Fadholi
2.  NPM : 140810180034
3.  Kelas : B
4.  Deskripsi : Adjacency Matrix
5.  Tgl : 7 April 2020 */
6.  #include<iostream>
7.  #define NODE 8
8.  using namespace std;
9.
10. int verArray[20][20];
11. int count = 0;
12.
13. void traversal(int v) {
14.     int i, j,x,temp;
15.     cout<<"Adjacency Matrix dari graf : \n\n ";
16.     for (x = 0; x< v; x++){
17.         temp = x +1;
18.         cout<<temp<<" ";
19.     }
20.     cout<<endl;
21.     for(i = 0; i < v; i++) {
22.         temp = i +1;
23.         cout<<temp<<" ";
24.         for(j = 0; j < v; j++) {
25.             cout << verArray[i][j] << " ";
26.         }
27.         cout << endl;
28.     }
29. }
30.
31. void addEdge(int u, int v) {
32.     verArray[u][v] = 1;
33.     verArray[v][u] = 1;
34. }
35. int main() {
36.     addEdge(1, 2);
37.     addEdge(1, 3);
38.     addEdge(2, 4);
39.     addEdge(2, 5);
40.     addEdge(3, 2);
41.     addEdge(3, 8);
42.     addEdge(4, 5);
43.     addEdge(5, 6);
44.     addEdge(7, 3);

```

```
45. addEdge(8, 7);
46. traversal(NODE);
47. }
```

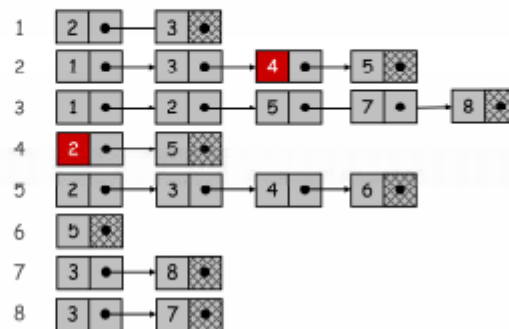
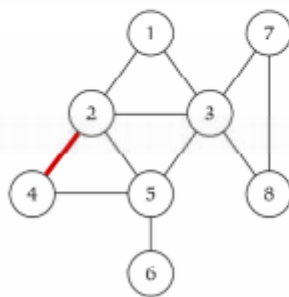
Adjacency Matrix dari graf :

```

      1 2 3 4 5 6 7 8
1  0 0 0 0 0 0 0 0
2  0 0 1 1 0 0 0 0
3  0 1 0 1 1 1 0 0
4  0 1 1 0 0 0 0 1
5  0 0 1 0 0 1 0 0
6  0 0 1 0 1 0 1 0
7  0 0 0 0 0 1 0 0
8  0 0 0 1 0 0 0 0

```

2. Dengan menggunakan *undirected graph* dan representasi *adjacency list*, buatlah koding programnya menggunakan bahasa C++.



```

1.  /*
2.  Nama : Ahmad Irfan Fadholi
3.  NPM : 140810180034
4.  Kelas : B
5.  Deskripsi : Adjacency List
6.  Tgl : 7 April 2020 */
7.  #include<iostream>
8.  #include<list>
9.  #include<iterator>
10.
11. using namespace std;
12.
13. void displayAdjList(list<int> adjList[], int g) {
14.     for(int i = 0; i<g; i++) {
15.         cout <<"Adjacency list of vertex "<< i << "\nhead";
16.         list<int> :: iterator j;
17.         for(j = adjList[i].begin(); j != adjList[i].end(); ++j) {
18.             cout <<"-> "<<*j;
19.         }
20.         cout << endl;
21.     }
22. }
23. void addEdge(list<int> adjList[], int u, int g) {
24.     adjList[u].push_back(g);
25.     adjList[g].push_back(u);
26. }
```

```

27. int main() {
28.     int g = 8;
29.     list<int> adjList[g];
30.     addEdge(adjList, 0, 1);
31.     addEdge(adjList, 0, 2);
32.     addEdge(adjList, 1, 2);
33.     addEdge(adjList, 1, 3);
34.     addEdge(adjList, 1, 4);
35.     addEdge(adjList, 2, 4);
36.     addEdge(adjList, 2, 6);
37.     addEdge(adjList, 2, 7);
38.     addEdge(adjList, 3, 4);
39.     addEdge(adjList, 4, 5);
40.     addEdge(adjList, 6, 7);
41.
42.
43.     displayAdjList(adjList, g);
44. }

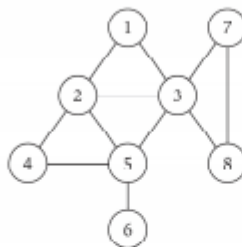
```

```

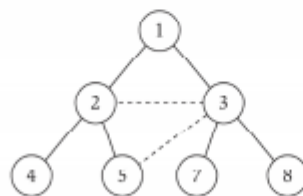
Adjacency list of vertex 0
head-> 1-> 2
Adjacency list of vertex 1
head-> 0-> 2-> 3-> 4
Adjacency list of vertex 2
head-> 0-> 1-> 4-> 6-> 7
Adjacency list of vertex 3
head-> 1-> 4
Adjacency list of vertex 4
head-> 1-> 2-> 3-> 5
Adjacency list of vertex 5
head-> 4
Adjacency list of vertex 6
head-> 2-> 7
Adjacency list of vertex 7
head-> 2-> 6

```

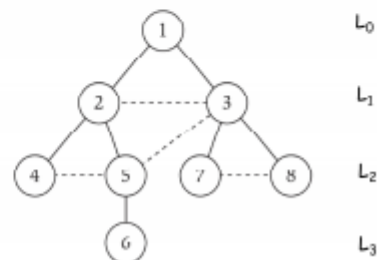
3. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan *undirected graph* sehingga menghasilkan *tree BFS*. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big- $\Theta$ !



(a)



(b)



(c)

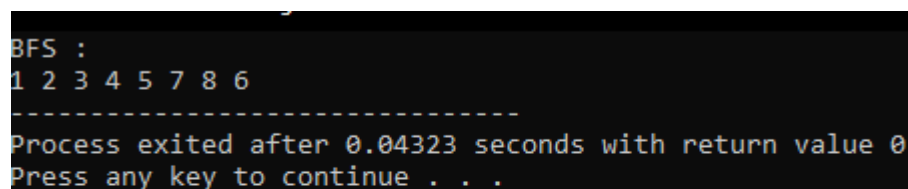
$L_0$

$L_1$

$L_2$

$L_3$

```
1. /*
2. Nama : Ahmad Irfan Fadholi
3. NPM : 140810180034
4. Kelas : B
5. Deskripsi : BFS
6. Tgl : 7 April 2020 */
7. #include <iostream>
8. using namespace std;
9. #define NODE 8
10. int main(){
11.     int adjacency[NODE][NODE] = {
12.         {0, 1, 1, 0, 0, 0, 0, 0},
13.         {1, 0, 1, 1, 1, 0, 0, 0},
14.         {1, 1, 0, 0, 1, 0, 1, 1},
15.         {0, 1, 0, 0, 1, 0, 0, 0},
16.         {0, 1, 1, 1, 0, 1, 0, 0},
17.         {0, 0, 0, 0, 1, 0, 0, 0},
18.         {0, 0, 1, 0, 0, 0, 0, 1},
19.         {0, 0, 1, 0, 0, 0, 1, 0}};
20.     bool discovered[NODE];
21.     for (int i = 0; i < NODE; i++){
22.         discovered[i] = false;
23.     }
24.     int output[NODE];
25.     discovered[0] = true;
26.     output[0] = 1;
27.     int counter = 1;
28.     for (int i = 0; i < NODE; i++){
29.         for (int j = 0; j < NODE; j++){
30.             if ((adjacency[i][j]==1) && (discovered[j] == false)){
31.                 output[counter]=j + 1;
32.                 discovered[j]=true;
33.                 counter++;
34.             }
35.         }
36.     }
37.     cout << "BFS : \n";
38.     for (int i = 0; i < NODE; i++){
39.         cout << output[i] << " ";
40.     }
```



```
BFS :
1 2 3 4 5 7 8 6
-----
Process exited after 0.04323 seconds with return value 0
Press any key to continue . . .
```

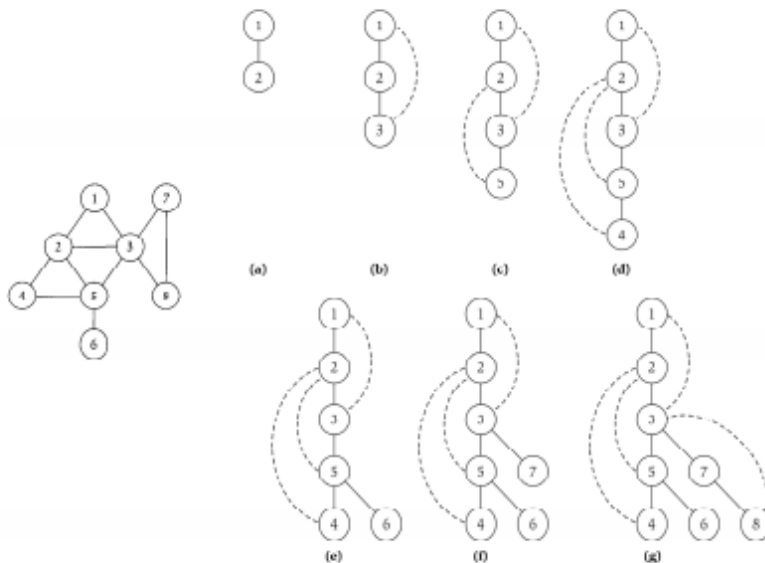
Analisis :

- BFS merupakan metode pencarian secara melebar sehingga mengunjungi node dari kiri ke kanan di level yang sama. Apabila semua node pada suatu level sudah dikunjungi semua, maka akan berpindah ke level selanjutnya. Dalam worst case BFS harus mempertimbangkan semua jalur (path) untuk semua node yang mungkin, maka nilai kompleksitas waktu dari BFS adalah  $O(|V| + |E|)$ .

- Karena Big-O dari BFS adalah  $O(V+E)$  dimana  $V$  itu jumlah vertex dan  $E$  itu adalah jumlah edges maka Big-O =  $O(n)$  dimana  $n = v + e$

Maka dari itu Big- $\Theta$  nya adalah  $\Theta(n)$ .

4. Buatlah program Depth First Search dari algoritma DFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan *undirected graph* sehingga menghasilkan *tree DFS*. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big- $\Theta$ !



```

1. /*
2. Nama : Ahmad Irfan Fadholi
3. NPM : 140810180034
4. Deskripsi : DFS
5. Tgl : 7 April 2020 */
6.
7. #include <iostream>
8. #include <list>
9.
10. using namespace std;
11.
12. class Graph{
13.     int N;
14.     list<int> *adj;
15.     void DFS(int u, bool visited[]){
16.         visited[u] = true;
17.         cout << u << " ";
18.         list<int>::iterator i;
19.         for (i = adj[u].begin(); i != adj[u].end(); i++)
20.         {
21.             if (!visited[*i])
22.             {
23.                 DFS(*i, visited);
24.             }
25.         }
26.     }
27.
28. public:
29.     Graph(int N){
30.         this->N = N;
31.         adj = new list<int>[N];
32.     }

```

```
33. void addEdge(int u, int v){
34.     adj[u].push_back(v);
35. }
36. void DFS(int u){
37.     bool *visited = new bool[N];
38.     for (int i = 0; i < N; i++){
39.         visited[i] = false;
40.     }
41.     DFS(u, visited);
42. }
43. };
44.
45. int main(){
46.     Graph g(8);
47.     g.addEdge(1, 2);
48.     g.addEdge(1, 3);
49.     g.addEdge(2, 3);
50.     g.addEdge(2, 4);
51.     g.addEdge(2, 5);
52.     g.addEdge(3, 7);
53.     g.addEdge(3, 8);
54.     g.addEdge(4, 5);
55.     g.addEdge(5, 3);
56.     g.addEdge(5, 6);
57.     g.addEdge(7, 8);
58.
59.
60.     cout << "DFS dimulai dari vertex 1\n";
61.     g.DFS(1);
62.     return 0;
63. }
```

Analisis :

- DFS merupakan metode pencarian mendalam, yang mengunjungi semua node dari yang terkiri lalu geser ke kanan hingga semua node dikunjungi. Kompleksitas ruang algoritma DFS adalah  $O(bm)$ , karena kita hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan.
- Big O  
Kompleksitas total DFS () adalah  $(V+E)$ .  
 $O(n)$  dengan  $V$  = Jumlah Verteks dan  $E$  = Jumlah Edges