TUGAS MODUL PRAKTIKUM 6
ANALISIS ALGORITMA

Irfan Satrio Nugroho

140810180003

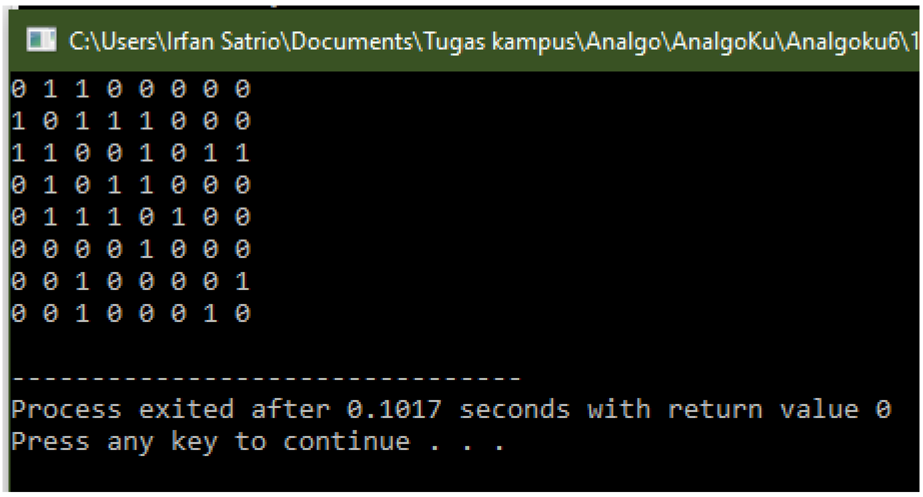
A

TUGAS ANALGO

1.

```
/*
Nama        : irfan Satrio
NPM         : 140810180003
Kelas       : A
Program     : Adjacency Matrix
*/

#include<iostream>
using namespace std;

int main()
{
     int m[8][8] =
{
     {0,1,1,0,0,0,0,0},
     {1,0,1,1,1,0,0,0},
     {1,1,0,0,1,0,1,1},
     {0,1,0,1,1,0,0,0},
     {0,1,1,1,0,1,0,0},
     {0,0,0,0,1,0,0,0},
     {0,0,1,0,0,0,0,1},
     {0,0,1,0,0,0,1,0}
};
     for(int i=0;i<8;i++)
     {
          for(int j=0; j<8;j++)
          {
               cout<<m[i][j]<<" ";
          }
```

```
            cout<<endl;
      }
}
```



```
C:\Users\Irfan Satrio\Documents\Tugas kampus\Analgo\AnalgoKu\Analgoku6\1
0 1 1 0 0 0 0 0
1 0 1 1 1 0 0 0
1 1 0 0 1 0 1 1
0 1 0 1 1 0 0 0
0 1 1 1 0 1 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 1
0 0 1 0 0 0 1 0

--------------------------------
Process exited after 0.1017 seconds with return value 0
Press any key to continue . . .
```

2.

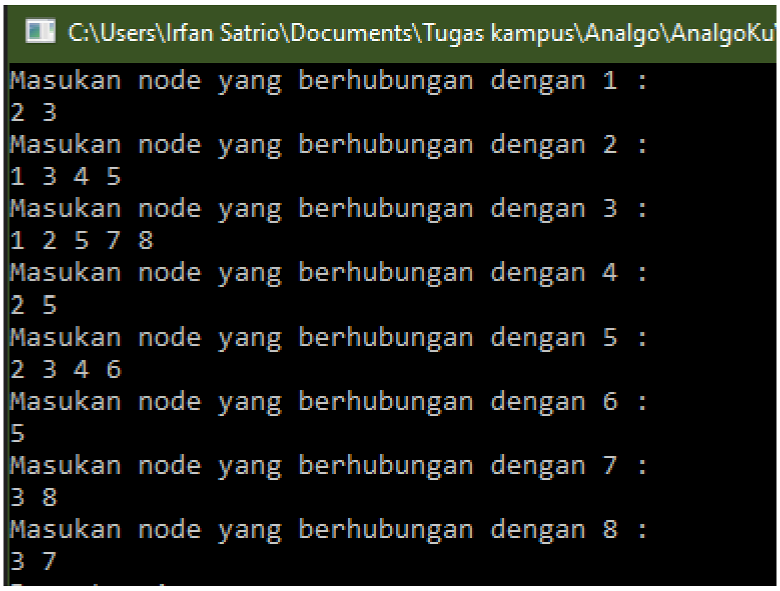
```
/*
Nama         : irfan Satrio
NPM          : 140810180003
Kelas        : A
Program      : Adjacency List
*/

#include<iostream>
#include<windows.h>
using namespace std;

struct adjacent
{
      int nodeAdj;
      adjacent* nextAdj;
};

struct elemen
{
	int node;
	elemen* next;
	adjacent* firstAdj;
};

typedef elemen* pointerNode;
typedef adjacent* pointerAdj;
typedef pointerNode list;

void createListNode(list& first)
{
	first = NULL;
}

void createNode(pointerNode& pBaru,int vertex)
{
	pBaru = new elemen;
	pBaru->node = vertex;
	pBaru->next = NULL;
	pBaru->firstAdj = NULL;
}

void createAdjacent(pointerAdj& pBaru,int vertex)
{
	pBaru = new adjacent;
	pBaru->nodeAdj = vertex;
	pBaru->nextAdj = NULL;
}

void insertAdjacent(pointerNode& curNode,pointerAdj pBaruAdj)
{
	pointerAdj last;
	if(curNode->firstAdj == NULL)
	{
```

```
            curNode->firstAdj = pBaruAdj;
      }
      else
      {
            last = curNode->firstAdj;
            while(last->nextAdj != NULL)
            {
                  last = last->nextAdj;
            }
            last->nextAdj = pBaruAdj;
      }
}

void insertElement(list& first, pointerNode pBaruNode, int size)
{
      pointerNode last;
      pointerAdj pBaruAdj;
      if(first == NULL)
      {
            first = pBaruNode;
      }
      else
      {
            last = first;
            while(last->next != NULL)
            {
                  last = last->next;
            }
            last->next = pBaruNode;
      }
      if(size>0)
      {
            cout<<"Masukan node yang berhubungan dengan "<<pBaruNode-
>node<<" : "<<endl;
```

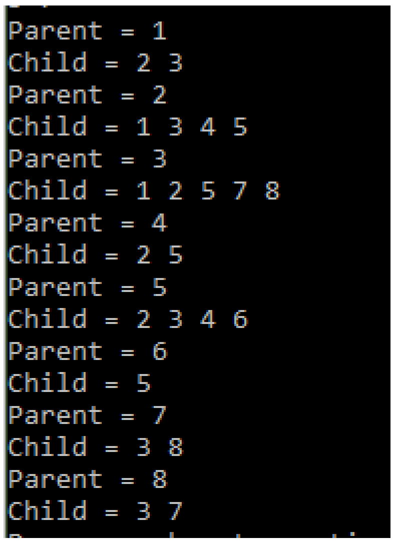
```
        }
        for(int i = 0; i < size; i++)
        {
              int vertex;
              cin>>vertex;
              createAdjacent(pBaruAdj,vertex);
              insertAdjacent(pBaruNode,pBaruAdj);
        }
  }

  void output(list first)
  {
        pointerNode pOut;
        pointerAdj pOutAdj;
        if(first == NULL)
        {
              cout<<"Tidak ada Node"<<endl;
        }
        else
        {
              pOut = first;

              while(pOut != NULL)
              {
                    cout<<"Parent = "<<pOut->node<<endl;
                    if(pOut->firstAdj == NULL)
                    {
                          cout<<"Tidak ada adjacency"<<endl;
                    }
                    else
                    {
                          pOutAdj = pOut->firstAdj;
                          cout<<"Child = ";
                          while(pOutAdj != NULL)
```

```
                        {
                              cout<<pOutAdj->nodeAdj<<" ";
                              pOutAdj = pOutAdj->nextAdj;
                        }
                  }
                  cout<<endl;
                  pOut = pOut->next;

            }
      }
}

int main()
{
      list first;
      pointerNode node;

      createListNode(first);

      createNode(node,1);
      insertElement(first,node,2);
      createNode(node,2);
      insertElement(first,node,4);
      createNode(node,3);
      insertElement(first,node,5);
      createNode(node,4);
      insertElement(first,node,2);
      createNode(node,5);
      insertElement(first,node,4);
      createNode(node,6);
      insertElement(first,node,1);
      createNode(node,7);
      insertElement(first,node,2);
      createNode(node,8);
```

```
        insertElement(first,node,2);

        output(first);

        system("pause");

}
```



```
C:\Users\Irfan Satrio\Documents\Tugas kampus\Analgo\AnalgoKu
Masukan node yang berhubungan dengan 1 :
2 3
Masukan node yang berhubungan dengan 2 :
1 3 4 5
Masukan node yang berhubungan dengan 3 :
1 2 5 7 8
Masukan node yang berhubungan dengan 4 :
2 5
Masukan node yang berhubungan dengan 5 :
2 3 4 6
Masukan node yang berhubungan dengan 6 :
5
Masukan node yang berhubungan dengan 7 :
3 8
Masukan node yang berhubungan dengan 8 :
3 7
```

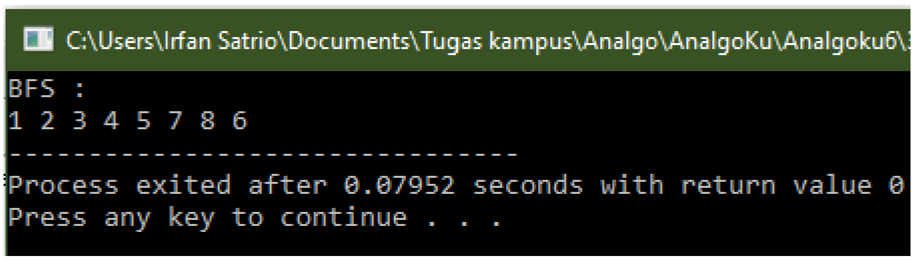


```
Parent = 1
Child = 2 3
Parent = 2
Child = 1 3 4 5
Parent = 3
Child = 1 2 5 7 8
Parent = 4
Child = 2 5
Parent = 5
Child = 2 3 4 6
Parent = 6
Child = 5
Parent = 7
Child = 3 8
Parent = 8
Child = 3 7
```

3.

```
/*
Nama        : irfan Satrio
NPM         : 140810180003
Kelas       : A
Program     : Breadth First Search
*/

#include<iostream>
using namespace std;

int main()
{
      int vertexSize = 8;
      int adjacency[8][8] =
      {
            {0,1,1,0,0,0,0,0},
            {1,0,1,1,1,0,0,0},
            {1,1,0,0,1,0,1,1},
            {0,1,0,0,1,0,0,0},
            {0,1,1,1,0,1,0,0},
            {0,0,0,0,1,0,0,0},
            {0,0,1,0,0,0,0,1},
            {0,0,1,0,0,0,1,0}
      };

      bool discovered[vertexSize];
      for(int i = 0; i < vertexSize; i++)
      {
            discovered[i] = false;
      }
```

```
        int output[vertexSize];

        //memulai inisiasi
        discovered[0] = true;
        output[0] = 1;

        int counter = 1;
        for(int i = 0; i < vertexSize; i++)
        {
                for(int j = 0; j < vertexSize; j++)
                {
                        if((adjacency[i][j] == 1)&&(discovered[j] == false))
                        {
                                output[counter] = j+1;
                                discovered[j] = true;
                                counter++;
                        }
                }
        }
        cout<<"BFS : "<<endl;
        for(int i = 0; i < vertexSize; i++)
        {
                cout<<output[i]<<" ";
        }
}
```

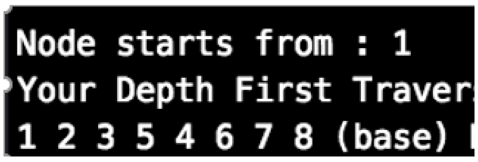


```
C:\Users\Irfan Satrio\Documents\Tugas kampus\Analgo\AnalgoKu\Analgoku6\
BFS :
1 2 3 4 5 7 8 6
--------------------------------
Process exited after 0.07952 seconds with return value 0
Press any key to continue . . .
```

4.

```
/*
Nama        : irfan Satrio
NPM         : 140810180003
Kelas       : A
Program     : Depth First Search
*/
#include<iostream>
#include<list>
using namespace std;
class Graph
{
    int V;
    list<int> *adj;
    void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);
    void addEdge(int v, int w);
    void DFS(int v);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}
```

```
void Graph::DFSUtil(int v, bool visited[])
{
    visited[v] = true;
    cout << v << " ";
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}

void Graph::DFS(int v)
{
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;
    DFSUtil(v, visited);
}

int main()
{
      int node,start;
      cout<<"Input the amount of your nodes : ";cin>>node;
    Graph g(node);
    cout<<"Instructions :"<<endl;
    cout<<"1. Enter the number of nodes from 0 to n-1"<<endl;
    cout<<"2. Enter negative numbers (such as -1) on either node input to
exit the program"<<endl;
    for(;;){
      int node1,node2;
      cout<<"Enter number between "<<0<<" to "<<node-1<<endl;
      cout<<"Input node 1 : ";cin>>node1;
      cout<<"Input node 2 : ";cin>>node2;
      if(node1>=0&&node2>=0&&node1<node&&node2<node){
            g.addEdge(node1,node2);

            cout<<endl;

            }

            else if(node1<0||node2<0)

                  break;

            else

                  cout<<"Wrong input. Please enter again"<<endl;

      }

      back:

      cout<<"\nNode starts from : ";cin>>start;

      if(start<0||start>node-1){

            cout<<"Wrong input. Please enter again"<<endl;

            goto back;

      }

    cout<<"Your Depth First Traversal (starting from vertex 
"<<start<<")"<<endl;

    g.DFS(start);

    return 0;

}
```



```
Node starts from : 1
Your Depth First Traver
1 2 3 5 4 6 7 8 (base)
```

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 $O(|V| + |E|)$.

- 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 hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan.