Derinlik Öncelikli Arama Algoritması

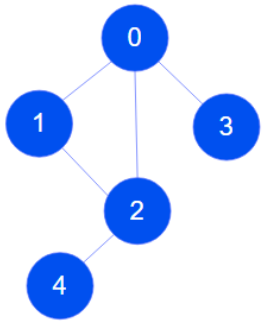
Algoritmanın amacı, döngülerden kaçınırken her bir köşeyi ziyaret edilmiş olarak işaretlemektir.

DFS algoritması aşağıdaki gibi çalışır:

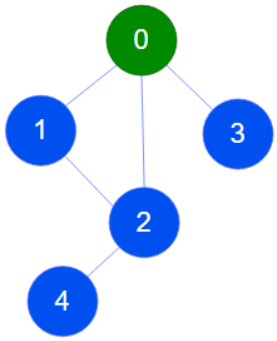
1. Grafiğin köşelerinden herhangi birini bir yığının üzerine koyarak başlayın.
2. Yığının en üstteki öğesini alın ve ziyaret edilenler listesine ekleyin.
3. Bu köşenin bitişik düğümlerinin bir listesini oluşturun. Ziyaret edilenler listesinde olmayanları yığının en üstüne ekleyin.
4. Yığın boşalana kadar 2. ve 3. adımları tekrarlamaya devam edin.

DFS Algoritmasının Uygulanma Alanları: Yolu bulmak için , bir grafikteki döngüleri algılamak için kullanılır.

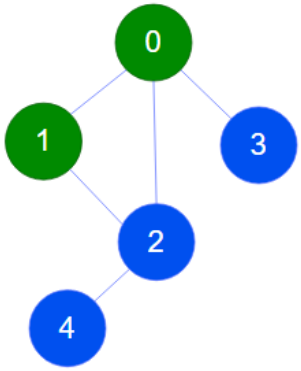
Örnek Problem:



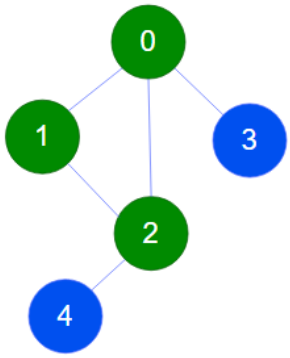
					Ziyaret Edilen
					Stack



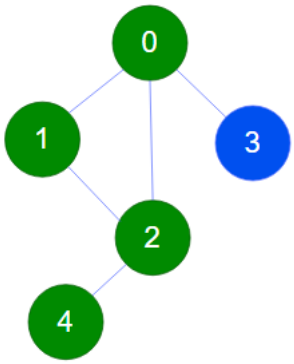
0					Ziyaret Edilen
1	2	3			Stack



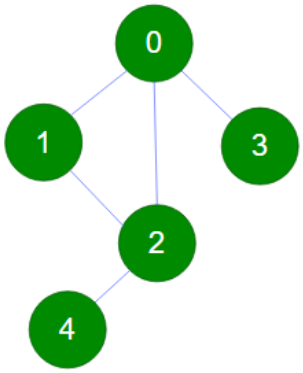
0	1				Ziyaret Edilen
2	3				Stack



0	1	2			Ziyaret Edilen
4	3				Stack

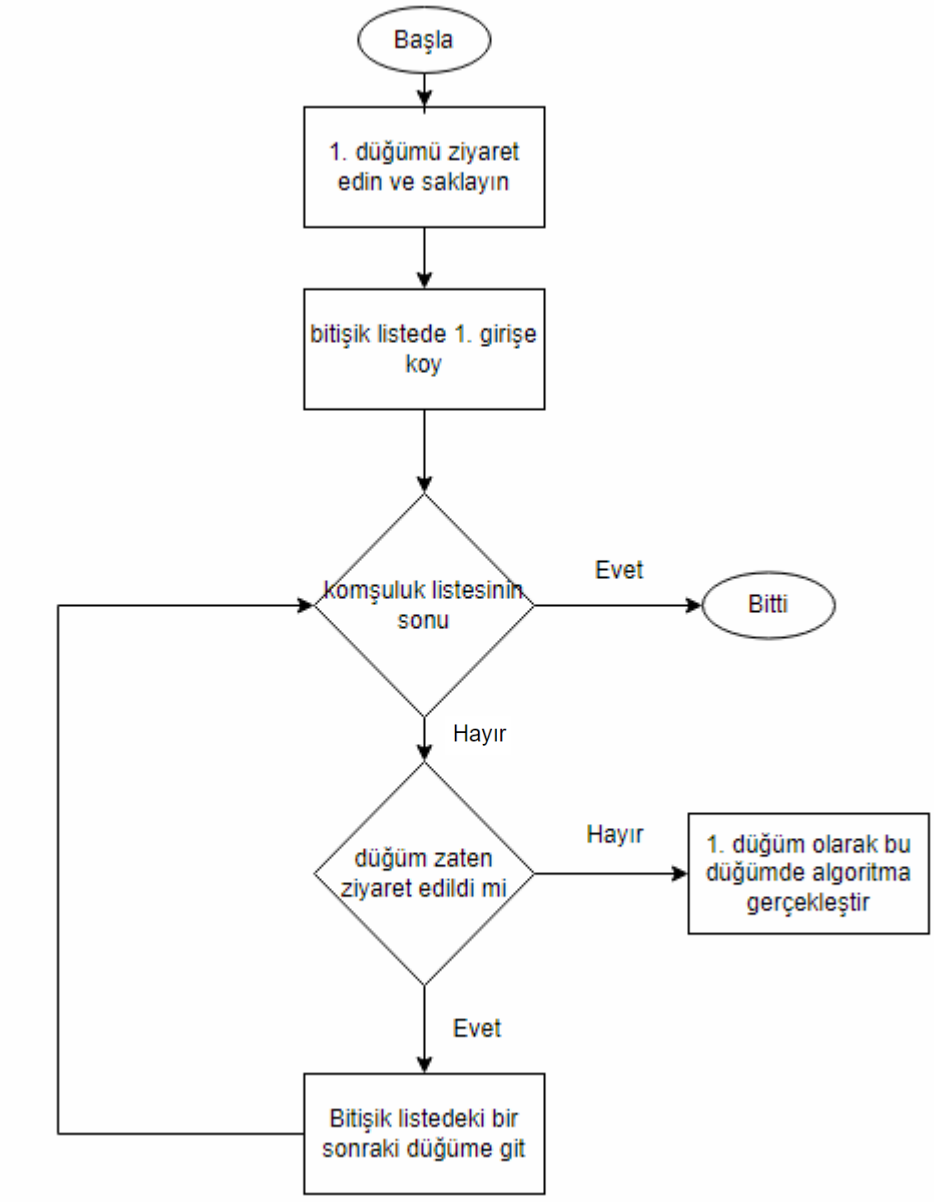


0	1	2	4		Ziyaret Edilen
3					Stack



0	1	2	4	3	Ziyaret Edilen
					Stack

Çözümün Akış Diyagramı:



```
#include <bits/stdc++.h>
using namespace std;

class Graph {
public:
   map<int, bool> visited;
   map<int, list<int> > adj;

   void addEdge(int v, int w);

   void DFS(int v);
};

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

void Graph::DFS(int v)
{
   visited[v] = true;
   cout << v << " ";

   list<int>::iterator i;
   for (i = adj[v].begin(); i != adj[v].end(); ++i)
      if (!visited[*i])
         DFS(*i);
}

int main()
{
   // Yukaridaki diyagramda verilen bir grafi olusturun
   Graph g;
   g.addEdge(0, 1);
   g.addEdge(0, 2);
   g.addEdge(1, 2);
   g.addEdge(2, 0);
   g.addEdge(2, 3);
   g.addEdge(3, 3);
   g.addEdge(3, 4);
   g.addEdge(4, 4);


   cout << "Derinlik Ilk Geçis asagidadir" "[tepe noktasindan baslayarak 0]\n";
   g.DFS(0);

   return 0;
}
```