
**YILDIZ TEKNİK ÜNİVERSİTESİ**
**ELEKTRİK-ELEKTRONİK FAKÜLTESİ**
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**VERİ YAPILARI ALGORİTMALAR**

# Proje

Muhammet Ali Şen - 20011701

**DFS Algoritması Yardımıyla Labirent Sorusu Çözümü**

DFS, (Derin öncelikli arama) algoritması graflarda ve ağaçlarda arama işlemini gerçekleştiren algoritmalardan bir tanesidir.

Arama işleminin ağaçlar üzerinde daha başarılı sonuçlar verdiği ortaya konulduğu için ağaçlar ve benzer şekilde graflar üzerinde de çeşitli arama algoritmaları geliştirilmiştir. Bunlardan en yaygın olanlarından bir tanesi DFS'dir.

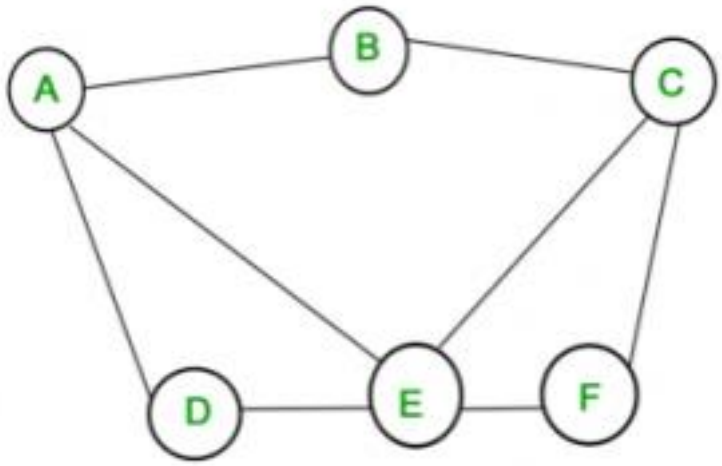
DFS veri yapısı olarak genellikle STACK yani yığın kullanır. Bunun sebebi derin öncelikli arama yaparken graf üzerindeki node'ların sırayla işleme alınması ve ilk girenin ilk çıkması prensibinden gelmektedir. Yani bir başlangıç noktası belirleyip bu noktadan (node) komşu node'lara doğru gezilir ancak tek seferde sadece bir adet komşuya bakılır. Bu sayede bakılan her komşu node yığına eklenir ve bir önceki node yığından çıkar.

DFS kodlanırken stack kullanmadan da kodlama yapılabilir. Ancak literatürde DFS'nin genellikle veri yapısı olarak yukarıda izah edilmeye çalıştığı sebeple Stack (yığın) kullandığını söyleyebiliriz.

DFS'de bir başlangıç düğümü (node) seçilir. Buna root node denilmektedir. Daha sonra bu düğümden gidilebilecek komşu düğümlere tek tek gidilir. Tüm düğümler gezilmiş olana dek bu işlem devam eder. Yani gidilen her bir düğümün komşuları ziyaret edilerek rekürsif bir şekilde gezilir. Dikkat edilecek husus gezilen düğüm tekrar ziyaret edilmemelidir. Bir düğüm sadece bir kez ziyaret edilmeli ve tüm düğümlerin ziyaretleri bittiğinde tekrar başlangıç düğümüne gelinmelidir.

Derin öncelikli arama algoritması olduğu için amacımız mümkün olduğunca derine inmektir. Binary Tree gibi veya kuralları olan diğer ağaçlar gibi bir kurala göre şekillenmeyen ağaç veya graflarda kullanımı yaygın bir algoritma olacaktır.

**DFS algoritmasının çalışma mantığı:**




**Resim-1**

Pek çok kaynakta yer alan yukarıdaki resimden örneklendirecek olursak öncelikle kendimize bir başlangıç düğümü belirlememiz gerekiyor. Bir kural olmasa da BST'lerden alışkın olduğumuz şekilde sol taraf veya sola yakın bir taraftan başlangıç düğümü seçmekle başlayabiliriz. Şimdi A düğümünü root düğüm olarak belirleyelim ve bu düğümden tek seferde gidilebilecek komşu ve onların da komşularını ziyaret edelim. Bunu yığın veri yapısına göre anlatmaya çalışacağım. (LIFO)

1- A dan başla

2- A'nın komşularına bak B, D ve E doğrudan gidilebilecek komşu düğümlerdir.

3- B'ye ilerle. Yığına A'yı ekle  push(A) B düğümündeyken yığına push(B) girer.

4- B düğümünden gidilebilecek komşulara baktığımızda C düğümü olduğu görüyoruz. push(C).

5- C düğümünden E ve F düğümlerine gidilebiliyor. F düğümüne gidelim. push(F)

6- F düğümünden E düğümüne gidelim. push(E).

7- E düğümünden daha önce ziyaret edildiği için C ve A düğümlerine gitmeyip sadece D düğümüne gidelim. push(D)

8- D düğümünden gidilmemiş komşu düğüm kalmadığı için tekrar başlangıç noktamıza dönelim. Böylece tüm düğümleri bir kez ziyaret edecek şekilde gezmiş olduk.

Öncelikle her bir komşu ziyaretinde push ile düğümleri yığına dahil ediyoruz. Burada LIFO yani Last In First Out bir başka deyişle yığın veri yapısının özelliği olan son giren ilk çıkar mantığı ile push ile eleman aldıkça pop ile elemanları dışarı çıkarmak gerekiyor. Yani F düğümü girdiğinde bir önceki adımdan kalma son giren konumundaki C düğümü yığından çıkar. Bu şekilde yığın kodlandığında her bir adımda girenler ve çıkanlar olacaktır. Algoritmada başlangıç düğümü farklı seçildiğinde gidilecek yolların sırası değişebilir ancak sonucu değişmeyecektir.

DFS algoritmasının karmaşıklığını inceleyecek olursak her bir düğümü en az 1 kez gezdiği için O(N) olacaktır.

**Projede Dikkat Edilmesi Gerekenler**

Projedeki okunan maze.txt dosyası discord üzerinden asistan hocalarımızın gönderdiği dosyadır. İlk okuma işlemi buffer üzerinden yapılmış ve bu sayede satır ve sütun bilgileri alınmıştır. Akabinde buffer üzerinde karakter karakter bakılarak matris tablomuza

duvarlar (0),
yollar (1),
başlangıç(2),
bitiş(3),
bitişe giden yol (4),
elma-puan(5)

olarak işlenmiştir. Aynı anda maze tablomuza da duvarlar olduğu gibi, yollar boşluk, başlangıç (s), bitiş (f), bitişe giden yol(.), elma-puan(#) olarak işaretlenmiştir. Elma-puan yerleri boşluk-yollarının bir diziye atılıp oradan rastgele 20 adet seçilmesi ile yapılmıştır. Son gösterimde yollar (.) eğerki elma-puan(#) üzerinden geçerse rahatlıkla görülecektir.

Problem toplamda 173 adımlık bir yol çıkararak çözülmüştür. DFS algoritması labirent üzerinde;

1.yukarı,

2.sola,

3.sağa

4.aşağı şeklinde ilerlemiştir.

Her çalıştırmada elmaları rastgele bir şekilde dağıttığı için puan sürekli değişmektedir.

Milisaniyelik farklar ile ekrana yazdırma işlemi çok fazla sonuç verdiği için şimdilik yapılmamıştır.

Youtube linki: https://youtu.be/lKgd35hJA_c