BFS Gezinme Algoritması

BFS Algoritmasına göre graflar üzerinde gezinme işlemi gerçekleştirmek için kuyruk yapısından yararlanılır. Kuyruk yapısını bağlı listeler ile gerçekleştirmek daha pratik olduğundan bağlı liste ile kuyruk oluşturulmuştur. Bağlı listelerde işlem yapmak için öncelikle düğüm tanımlaması yapmak gerekir. Tek yönlü bağlı listenin bir düğümünde bir sonraki düğümün adresinin ve o anki düğümün değeri tutulmalıdır. Düğüm tanımı aşağıdaki şekilde gerçekleştirilebilir. Burada elemanın parametre olarak alındığı yapılandırıcı tanımı yapılmıştır. Bu tanımda next değişkeni ataması yapılmadığı için bu değişken null değer almaktadır.

```
class Node {
    Node next;
    int deger;

    Node () {}

    Node(int eleman) {
        deger = eleman;
    }
}
```

Kuyruk yapısı için kök düğüm oluşturmak gerekmektedir. Ardından ekleme işleminde kök düğümün durumu kontrol edilir. Kök düğüm boşsa yeni düğüm kök düğüme eşitlenir. Kök düğüm dolu ise son eleman aranır. Son elemanın sonuna yeni düğüm eklemesi gerçekleştirilir. Ekleme kodu aşağıda verilmiştir.

```
Node root;

void ekle(int eleman) {
    if(root == null) {
        root = new Node(eleman);
    }else {
        Node temp = root;
        while(temp.next != null) {
            temp = temp.next;
        }

        temp.next = new Node(eleman);
    }
}
```

Kuyruktan çıkarma yapılırken ilk eleman çıkarılır. Bunun için root u bir sonraki düğüm olarak belirlemek yeterlidir. Kuyruktan çıkarma kodu aşağıda verilmiştir. root değerini kaybetmemek için değişkende tutulmuştur.

```
int cikar () {
    if(root == null) {
        System.out.println("Kuyruk boş");
        return -1;
    }else {
        int eleman = root.deger;
        root = root.next;
        return eleman;
    }
}
```

BFS'de kuyruk dolu olduğu sürece döngü devam ettirilir. Bunun için kuyruğun boş olup olmadığını bulan fonksiyonun yazılması gerekir. Bu fonksiyon aşağıda verilmiştir.

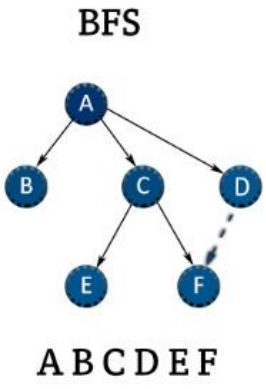
```
boolean kuyrukBosMu() {
    if(root == null) {
        return true;
    }else {
        return false;
    }
}
```

Yazmış olduğumuz kuyruk kodunun doğruluğunu test etmek için yazılan kodlar aşağıda verilmiştir.

```
void listele () {
    Node temp = root;
    while(temp!=null) {
        System.out.println(temp.deger);
        temp = temp.next;
    }
}

public static void main(String[] args) {
    Kuyruk k = new Kuyruk();
    k.cikar();

    k.ekle(5);
    k.ekle(10);

    k.listele();
    System.out.println("---" + k.cikar());
    k.listele();
}
```

BFS Algoritması için öncelikle elimizde bir graf yapısının olması gerekmektedir. Komşuluk matrisi yöntemi ile bu graf yapısı oluşturulmuştur. Oluşturulan komşuluk matrisi ve bunun graf olarak çıktısı aşağıda verilmiştir. Matris 0 = A'yı 5=F'yi temsil etmektedir.

```
int [][] komsuluk = new int[6][6];
int ziyaret [] = new int [6];
Kuyruk k = new Kuyruk();

BSFProje() {
    komsuluk[0][1] = 1; // A->B
    komsuluk[0][2] = 1; // A->C
    komsuluk[0][3] = 1; // A->D
    komsuluk[2][4] = 1; // C->E
    komsuluk[3][5] = 1; // D->F
    komsuluk[2][5] = 1; // C->F
}
```



BFS gezinmesinde kuyruk yapısı kullanılır. Kuyruk boş olmadığı sürece kuyruktan bir eleman çıkarılır ve bunun komşuları yazdırılır. Yazdırılan (işlem gören) düğümler kuyruğa eklenir. Sonraki iterasyonlarda onlar da kuyruktan çıkarılarak döngü devam ettirilir. Kuyruk boş olduğunda işlem sonlanır.

```
void BSFGez(int dugum) {
    int v = dugum;
    System.out.println(v);
    ziyaret[v] = 1;
    k.ekle(v);

    while(!k.kuyrukBosMu()) {
        //komşu bul
        for (int i = 0; i < 6; i++) {
            if(komsuluk[v][i] == 1 && ziyaret[i] == 0) {
                k.ekle(i);
                ziyaret[i] = 1;
                System.out.println(i);
            }
        }
        v = k.cikar();
    }
}
```

Yazılan kodun doğruluğunu test etmek için aşağıdaki kod parçacığı kullanılmıştır.

```
public static void main(String[] args) {
    BSFProje x = new BSFProje();

    x.BSFGez(0);
}
```