



**T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY of ENGINEERING
DEPARTMENT of COMPUTER ENGINEERING**

GRAPH, DIJKSTRA'S ALGORITHM, BFS & DFS, IMPORTANCE VALUE

CSE 222 DATA STRUCTURES AND ALGORITHMS COURSE HOMEWORK 8 REPORT

STUDENT
Şeyda Nur DEMİR
12 10 44 042

LECTURER
Prof. Dr. Fatih Erdoğan SEVİLGİN

TEACHING ASSISTANT
Başak KARAKAŞ
M Burak KOCA

KOCAELİ, 2021

1.DESRIPTION

This homework is extra homework and will replace the homework with the lowest grade if the grade of this extra homework is greater.

PART 1: 40 pts.

Generalize the implementation of Dijkstra's algorithm in the book so that:

1. Your method should run efficiently for both ListGraph and MatrixGraph representations of the graph.
2. Suppose the edges of a graph carry several properties. For example, the properties could be distance, time, or quality. Your method should run for any specified property of the edges.
3. In Dijkstra's algorithm implementation in the book, addition operator is used to combine an edge weight with a path weight. The addition operator can be replaced with any associative operator such as addition, multiplication, or $*$ operator. Note that $*$ be a binary operator defined as $a * b = a + b - ab$. The operator is associative since $(a * b) * c = a * (b * c)$. Your method should run for any specified associative operator.

You should write only one method to implement Dijkstra's algorithm and your method should provide all the properties above.

2.REPORT


Implementation Details in Turkish :

1. ListGraph ve MatrixGraph her ikisine de uyması için, derste gösterildiği ve ders slaytlarında olduğu gibi, bir güncelleme yaptım. Bu düzeltme ile, vertes enhanced for loop ile ilerlemiyor, edge iterator ile ilerliyor. Bu sayede, her iki representasyon için de uygun hale gelmiş oluyor.

For an adjacency list representation, modify the code:

```
// Update the distances.
Iterator<Edge> edgeIter = graph.edgeIterator(u);
while (edgeIter.hasNext()) {
    Edge edge = edgeIter.next();
    int v = edge.getDest();
    if (vMinusS.contains(new Integer(v));
        double weight = edge.getWeight();
        if (dist[u] + weight < dist[v]) {
            dist[v] = dist[u] + weight;
            pred[v] = u;
        }
    }
}
```

```
Iterator<GeneralizedEdge> edgeIter = graph.edgeIterator(u); // Property 1
while (edgeIter.hasNext()) { // Property 1
    GeneralizedEdge edge = edgeIter.next(); // Property 1
    int v = edge.getDest(); // Property 1
    if (vMinusS.contains(v)) { // Property 1
```



2. Edgeler weight dışında başka özellikler de tutabilsin diye, weight verisini double arraye çevirdim, ve kaç özellik girildiği sayısını alıyorum. Örneğin sadece miktar ve ücret bilgisini tutuyorsa, weight arrayimiz 2 elemanlı oluyor, ilk elemanı miktar ikinci elemanı ücret bilgisini tutuyor, 2 özellik olduğu bilgisini de veriyoruz. Kullanırken, hangisine göre önceliklendireceğimizi seçiyoruz, örneğin 1 derse arrayin 1.indexindeki ücret değerlerini kıyaslarken, 0 dendiği takdirde arrayin 0.indexinde bulunan miktar değerlerini kıyaslar. İkinci bir örnekle pekiştirelim, örneğin verteximiz uzaklık, zaman ve kalite bilgilerini tutsun, burada özellik sayımızı 3 olarak veririz. Kıyaslamayı uzaklığa göre önceliklendirmek istersek öncelik olarak 0 değerini vermeliyiz, zaman için bu değer 1 ve kalite kıyaslaması için bu değer 2 olmalıdır.

```
public class GeneralizedEdge {  
    private int source;  
    private int dest;  
    private double[] weight;  
    public GeneralizedEdge(int source,
```

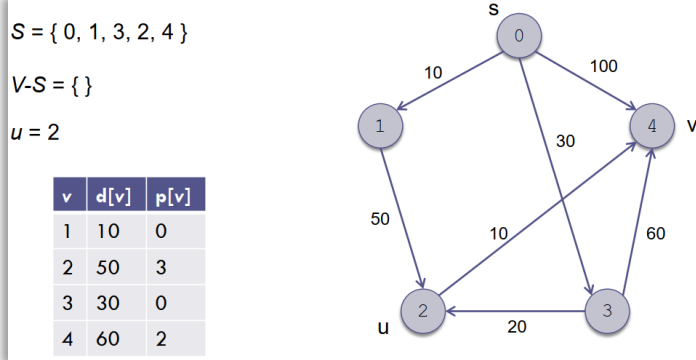
```
if (vMinusS.contains(v)) { // Property 1  
    double weight = graph.getEdge(u, v).getNthWeight(priority); // Property 2
```

3. Normalde algoritma mesafe hesabını yaparken mesafeleri toplayarak ilerler, ancak istenenlere göre bunu düzenlemek için combine şeklinde bir değişken tuttum, bu değişken 0 olduğunda hesaplamayı çarpma işlemine göre yapar, 1 olduğunda yıldız operatör işlemine göre yapar, ve bu iki değer dışında bir değer girildiğinde varsayılan işlem olarak toplama işlemi yapmaya devam eder.

```
double weight = graph.getEdge(u, v).getNthWeight(priority); // Property 2  
if (combine == 0) { // Property 3  
    if ((dist[u] * weight) < dist[v]) { // Property 3  
        dist[v] = (dist[u] * weight); // Property 3  
        pred[v] = u;  
    }  
} else if (combine == 1) { // Property 3  
    if ((dist[u] + weight - (dist[u] * weight)) < dist[v]) { // Property 3  
        dist[v] = (dist[u] + weight - (dist[u] * weight)); // Property 3  
        pred[v] = u;  
    }  
} else { // Property 3  
    if ((dist[u] + weight) < dist[v]) { // Property 3  
        dist[v] = (dist[u] + weight); // Property 3  
        pred[v] = u;  
    }  
}
```

Test :

Test ederken list graph ve matrix graph oluşturdum, tek tek hesaplarken sorunlar yaşadım ve yetiştiremedim. Senaryo olarak ders slaytlarında verilen örneği ele aldım.



```
void testPart1() {
    // 0,1 0,3 0,4
    int s1 = 0; int d1 = 1; double[] w1 = {10.0, 100.0, 5.0};
    GeneralizedEdge edge1 = new GeneralizedEdge(s1,d1,w1);
    int s2 = 0; int d2 = 3; double[] w2 = {30.0, 80.0, 5.0};
    GeneralizedEdge edge2 = new GeneralizedEdge(s2,d2,w2);
    int s3 = 0; int d3 = 4; double[] w3 = {100.0, 10.0, 5.0};
    GeneralizedEdge edge3 = new GeneralizedEdge(s3,d3,w3);
    // 1,2
    int s4 = 1; int d4 = 2; double[] w4 = {50.0, 60.0, 5.0};
    GeneralizedEdge edge4 = new GeneralizedEdge(s4,d4,w4);
    // 2,4
    int s5 = 2; int d5 = 4; double[] w5 = {10.0, 100.0, 5.0};
    GeneralizedEdge edge5 = new GeneralizedEdge(s5,d5,w5);
    // 3,2 3,4
    int s6 = 3; int d6 = 2; double[] w6 = {20.0, 90.0, 5.0};
    GeneralizedEdge edge6 = new GeneralizedEdge(s6,d6,w6);
    int s7 = 3; int d7 = 4; double[] w7 = {60.0, 50.0, 5.0};
    GeneralizedEdge edge7 = new GeneralizedEdge(s7,d7,w7);
}
```

```

int numV = 7;
int numberOfProperties = 3;

GeneralizedListGraph lgraph = new GeneralizedListGraph(numV, false, numberOfProperties);
lgraph.insert(edge1);
lgraph.insert(edge2);
lgraph.insert(edge3);
lgraph.insert(edge4);
lgraph.insert(edge5);
lgraph.insert(edge6);
lgraph.insert(edge7);

GeneralizedMatrixGraph mgraph = new GeneralizedMatrixGraph(numV, false, numberOfProperties);
mgraph.insert(edge1);
mgraph.insert(edge2);
mgraph.insert(edge3);
mgraph.insert(edge4);
mgraph.insert(edge5);
mgraph.insert(edge6);
mgraph.insert(edge7);

```

```

int start = 0; // Starts vertex 0
int[] pred = new int[numV];
double[] dist = new double[numV];
int priority = 0; // Prioritizes weight[0]
int combine = 2; // Calculates distance as adding

/*GeneralizedDijkstrasAlgorithm lsp = new GeneralizedDijkstrasAlgorithm();
lsp.generalizedDijkstrasAlgorithm(mgraph, start, pred, dist, priority, combine);
for (int i=0; i != numV; i++)
    System.out.println("Source : " + start + " Destination : " + pred[i] + " Distance : " + dist[i]);*/

GeneralizedDijkstrasAlgorithm msp = new GeneralizedDijkstrasAlgorithm();
msp.generalizedDijkstrasAlgorithm(mgraph, start, pred, dist, priority, combine);
for (int i=0; i != numV; i++)
    System.out.println("Source : " + start + " Destination : " + pred[i] + " Distance : " + dist[i]);

```

END OF THE REPORT

UPDATED : Jun 28, 2021 Monday 23:15

STUDENT

Şeyda Nur DEMİR
12 10 44 042

LECTURER

Prof. Dr. Fatih Erdoğan SEVİLGİN

TEACHING ASSISTANT

Başak KARAKAŞ
M Burak KOCA

KOCAELİ, 2021