2019년 알고리즘

- HW 11 -

제출일자	2019.11.20.
이 름	장수훈
학 번	201402414
분 반	01

1 Matrix chain

```
public static void main(String args[]) {
    FileReader fr;
    try {
        fr = new FileReader("C:\\Users\\micke\\eclipse-workspace\\ALG\\src\\data11_matrix_chain.txt");
        BufferedReader br = new BufferedReader(fr);
        String line = br.readLine();
        int[] matrix = new int[12];

        int index = 0;

        while (line != null) {
            StringTokenizer st = new StringTokenizer(line, ",");
            while (st.hasMoreTokens()) {
                matrix[index] = Integer.parseInt(st.nextToken());
                index++;
            }
            line = br.readLine();
        }

        int input[] = new int[7];

        for (int i = 0; i < 6; i++) {
            input[i] = matrix[i * 2];
        }
        input[6] = matrix[11];

        print(MATRIX_CHAIN_ORDER(input));

    } catch (IOException e) {

    }
}
```

입력 받는 부분은 평소와 비슷하게 썼다.

l x j 행렬과 j x k 행렬로 입력을 받기 때문에 중간에 중첩되는 부분을 빼고 넣었다.

```
public static int[][] MATRIX_CHAIN_ORDER(int[] p) {
    int n = p.length - 1;
    int[][] m = new int[n + 1][n + 1];

    for (int i = 1; i <= n; i++) {
        m[i][i] = 0;
    }
    for (int l = 2; l <= n; l++) {
        for (int i = 1; i <= n - l + 1; i++) {
            int j = i + l - 1;
            m[i][j] = Integer.MAX_VALUE;
            for (int k = i; k <= j - 1; k++) {
                int q = m[i][k] + m[k + 1][j] + (p[i - 1] * p[k] * p[j]);
                if (q < m[i][j]) {
                    m[i][j] = q;
                }
            }
        }
    }

    return m;
}
```

MATRIX-CHAIN-ORDER(p)

```
1  n = p.length − 1
2  let m[1..n, 1..n]
3  for i = 1 to n
4      m[i, i] = 0
5  for l = 2 to n                    // l is the chain length
6      for i = 1 to n − l + 1
7          j = i + l − 1
8          m[i, j] = ∞
9          for k = i to j − 1
10             q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11             if q < m[i, j]
12                 m[i, j] = q
13 return m
```

행과 열이 같은 부분을 0으로 채우고

행과 열이 같지 않은부분은 q에 계산된 값을 넣어 주었다.

그리고 계산된 q가 더 작으면 해당 자리에 q를 넣어주는 방식이다.

의사코드 대로 코딩을 하였다.

결과

```
**********각 단계별로 곱셈의 수를 출력**********
     0 15750  7875  9375 11875 15125
     0     0  2625  4375  7125 10500
     0     0     0   750  2500  5375
     0     0     0     0  1000  3500
     0     0     0     0     0  5000
     0     0     0     0     0     0
****************************************
최소 곱셈의 수 : 15125
```

1 Bellman Ford

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    FileReader fr;
    try {
        fr = new FileReader("C:\\Users\\micke\\eclipse-workspace\\ALG\\src\\data11_bellman_ford_1.txt");
        BufferedReader br = new BufferedReader(fr);
        String line = br.readLine();
        int[] array1 = new int[35];
        int index = 0;
        while (line != null) {
            StringTokenizer st = new StringTokenizer(line, ",");
            while (st.hasMoreTokens()) {
                array1[index] = Integer.parseInt(st.nextToken());
                index++;
            }
            line = br.readLine();
        }
        int[] static_point = new int[5];
        for (int i = 0; i < 5; i++) {
            static_point[i] = array1[i];
        }
        int[] array2 = new int[30];
        for (int i = 5; i < 35; i++) {
            array2[i - 5] = array1[i];
        }
        int[] start_point = new int[10];
        int[] end_point = new int[10];
        int[] weight = new int[10];

        int index_count = 0;
        for (int i = 0; i < 30; i = i + 3) {
            start_point[index_count] = array2[i];
            end_point[index_count] = array2[i + 1];
            weight[index_count] = array2[i + 2];
            index_count++;
        }
        int[][] point = new int[10][3];
        for (int i = 0; i < 10; i++) {
            point[i][0] = start_point[i];
            point[i][1] = end_point[i];
            point[i][2] = weight[i];
        }
        int V = 5, E = 10;
        BellmanFord bf = new BellmanFord(V, E);
        for (int i = 0; i < 10; i++) {
            bf.edge[i].src = point[i][0];
            bf.edge[i].dest = point[i][1];
            bf.edge[i].weight = point[i][2];

        }
        bf.BellmanFord_(bf, 0);
```

Bellman Ford algorithm은 가중치가 있는 방향 그래프에서 최단 경로 문제를 푸는 알고리즘이다. input데이터의 첫줄인 정점과 시작정점, 도작정점, 가중치를 분리하여 각각 배열에 담아서 진행 하였다. 동적할당을 하지 않았기에 코드가 많이 지저분하다.
이 데이터로부터 그래프를 생성하고 0번 정점으로부터 각 정점까지의 최단 거리를 구하는게 이번 과제의 목표이다.

BELLMAN-FORD(G, w, s)
INIT-SINGLE-SOURCE(G, s)
for $i = 1$ to $|G.V| - 1$
    for each edge $(u, v) \in G.E$
        RELAX(u, v, w)
for each edge $(u, v) \in G.E$
    if $v.d > u.d + w(u, v)$
        return FALSE
return TRUE

```
void BellmanFord_(BellmanFord bf, int src) {
    int V = bf.V, E = bf.E;
    int dist[] = new int[V];
    for (int i = 0; i < V; i++) {
        dist[i] = Integer.MAX_VALUE;
    }
    dist[src] = 0;
    for (int i = 0; i < V; i++) {
        System.out.printf("----------------- Iteration %d -----------------\n", i);
        for (int j = 0; j < E; j++) {
            int u = bf.edge[j].src;
            int v = bf.edge[j].dest;
            int weight = bf.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                int temp = dist[v];

                dist[v] = dist[u] + weight;
                if (temp == Integer.MAX_VALUE) {
                    System.out.printf("Update distance of %d from inf to %d \n", v, dist[v]);
                } else {
                    System.out.printf("Update distance of %d from %d to %d \n", v, temp, dist[v]);
                }

            }
        }

        System.out.printf("iteration %d distance : [ ", i);
        for (int k = 0; k < V; k++) {
            System.out.print(dist[k] + " ");
        }
        System.out.print("]\n\n");

    }
    for (int j = 0; j < E; j++) {
        int u = bf.edge[j].src;
        int v = bf.edge[j].dest;
        int weight = bf.edge[j].weight;
        if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
            System.out.println("The bf has negative cycle");

        }

    }

}
```

시작정점을 제외한 나머지 정점에 대한 거리를 모두 int의 최대값으로 설정하고 그래프의 모든 간선에 대해 D(u,v) = D(s,u) + w(u,v)를 적용하는걸 정점의 개수에서 1뺀 횟수만큼 반복한 뒤 한번더 실행했을 경우 그래프값의 변화가 생겼을 때 음수 사이클이 존재 한다고 표시해 주었다.

결과

```
----------------- Iteration 0 -----------------
Update distance of 1 from inf to 6
Update distance of 3 from inf to 11
Update distance of 2 from inf to 7
Update distance of 4 from inf to 2
Update distance of 3 from 11 to 4
iteration 0 distance : [ 0 6 7 4 2 ]

----------------- Iteration 1 -----------------
Update distance of 1 from 6 to 2
Update distance of 4 from 2 to -2
iteration 1 distance : [ 0 2 7 4 -2 ]

----------------- Iteration 2 -----------------
iteration 2 distance : [ 0 2 7 4 -2 ]

----------------- Iteration 3 -----------------
iteration 3 distance : [ 0 2 7 4 -2 ]

----------------- Iteration 4 -----------------
iteration 4 distance : [ 0 2 7 4 -2 ]
```

```
----------------- Iteration 0 -----------------
Update distance of 1 from inf to 4
Update distance of 2 from inf to 3
Update distance of 2 from 3 to 0
Update distance of 0 from 0 to -2
iteration 0 distance : [ -2 4 0 ]

----------------- Iteration 1 -----------------
Update distance of 1 from 4 to 2
Update distance of 2 from 0 to -2
Update distance of 0 from -2 to -4
iteration 1 distance : [ -4 2 -2 ]

----------------- Iteration 2 -----------------
Update distance of 1 from 2 to 0
Update distance of 2 from -2 to -4
Update distance of 0 from -4 to -6
iteration 2 distance : [ -6 0 -4 ]

The bf has negative cycle
```