

알고리즘

과제번호	08주차
날 짜	2018.11.01
학 번	201302395
이 름	류경빈

과제 1

Dijkstra algorithm(Minimum Priority Queue)을 사용하여 다음 graph의 최단 경로 cost를 계산하는 프로그램을 구현하라.

```
private char[] nodeName;
private int[][] w;
private int[] d;
```

- 각 노드와 vertex가 들어있는 배열을 w 이중 배열에 넣는다.
- 해당 노드의 값의 계산이 완료되었는지 검사하는 d 배열 생성

```
public void shortestPath() {
    Heap heap = new Heap(); // 힙..
    PriorityQueue Q = new PriorityQueue(); // 나머지 정점들의 큐
    ArrayList<Node> S = new ArrayList<>(); // 시작 정점으로부터 최단 경로가 이미 발견된 정점들의 집합

    makeDistance(); // distance 배열 초기화..

    for (int i = 0; i < w.length; i++){
        Node temp = new Node(i, d[i]);
        Q.insert(heap, temp);
    }

    System.out.println("dijkstra's algorithm으로 계산한 결과는 다음과 같습니다.");
    System.out.println();

    while (!Q.isEmpty(heap)){ // heap이 남아 있을때 까지..
        Node temp = Q.extract_min(heap); // 가장 작은 heap 값 추출..
        S.add(temp);
        int s = S.size()-1;
        System.out.println("=====");
        System.out.println("S[" + s + "] : d{" + nodeName[S.get(S.size() - 1).getVertex()] + "} = " + d[S.get(S.size() - 1).getVertex()]);
        System.out.println("-----");
        for (int i = 0; i < w.length; i++) {
            if (!(i == temp.getVertex())) { // 자기 자신은 빼고 계산
                if (d[i] > d[temp.getVertex()] + w[temp.getVertex()][i]
                    && w[temp.getVertex()][i] != INFINITE) { // 현재값보다 계산값이 작다면
                    d[i] = d[temp.getVertex()] + w[temp.getVertex()][i]; // 작은 값 입력
                }
            }
        }
        for (int i = 0; i < Q.size(heap); i++) { // 우선순위 큐에서 순서 출력
            System.out.print("Q[" + i + "] : " + d[
                + nodeName[Q.get(heap, i).getVertex()] + "] = "
                + Q.get(heap, i).getDist());
            if (Q.get(heap, i).getDist() != d[Q.get(heap, i).getVertex()]) { // 값이 같지 않다면
                Q.set_priority(heap, i, d[Q.get(heap, i).getVertex()]); // 우선순위 설정
                System.out.print(" -> d["
                    + nodeName[Q.get(heap, i).getVertex()] + "] = "
                    + Q.get(heap, i).getDist()); // 경로 출력
            }
            System.out.println("\n");
            Q.Build_Min_Heap(heap);
        }
    }
}
```

- shortestPath는 Dijkstra 알고리즘을 통해 최단 거리를 구해준다.
- 최소 힙 구조로 되어 있는 우선순위 Priority Queue를 만들어 해당 노드에서 가장 작은 값들을 우선적으로 나오게끔 해준다.
- 그래서 w 배열 그래프에서 최단 경로를 비교하고 이를 진행하게 되면 d 배열에 완료 되었다는 내용을 남겨 진행 상황을 파악한다.

결과 화면

```
int[][] w = {
    { 0, 10, 3, Dijkstra.INFINITE, Dijkstra.INFINITE },
    { Dijkstra.INFINITE, 0, 1, 2, Dijkstra.INFINITE },
    { Dijkstra.INFINITE, 4, 0, 8, 2 },
    { Dijkstra.INFINITE, Dijkstra.INFINITE, Dijkstra.INFINITE, 0, 7 },
    { Dijkstra.INFINITE, Dijkstra.INFINITE, Dijkstra.INFINITE, 9, 0 }
};
```

```
TestDijkstra x
/Library/Java/JavaVirtualMachines/jdk1.8.0_71.jdk/Contents/Home/bin/java ...
dijkstra's algorithm으로 계산한 결과는 다음과 같습니다.

=====
S[0] : d{A} = 0
-----
Q[0] : d[E] = 2147483647

Q[1] : d[B] = 2147483647 -> d[B] = 10
Q[2] : d[C] = 2147483647 -> d[C] = 3
Q[3] : d[D] = 2147483647

=====
S[1] : d{C} = 3
-----
Q[0] : d[B] = 10 -> d[B] = 7
Q[1] : d[E] = 2147483647 -> d[E] = 5
Q[2] : d[D] = 2147483647 -> d[D] = 11

=====
S[2] : d{E} = 5
-----
Q[0] : d[B] = 7
Q[1] : d[D] = 11

=====
S[3] : d{B} = 7
-----
Q[0] : d[D] = 11 -> d[D] = 9

=====
S[4] : d{D} = 9
----- <1 internal call>
```

```

public int howLongNumber (long number){
    return (int)((Math.log10(number)+1));
}

public int midNumber (long length){
    return (int)(Math.pow(10,(length/2)));
}

public long divisionNumber (long number, int divider){
    return number/divider;
}

public long remainderNumber (long number, int divider){
    return number%divider;
}

```

```

public long karatsubaMult (long numX, long numY){

    int x = howLongNumber(numX);
    int y = howLongNumber(numY);
    int d = x < y ? x : y;

    if (numX < numY){
        return karatsubaMult(numY, numX);
    }
    else if (x == 0 || y == 0){
        return 0;
    }
    else if (d <= THRESHOLD){
        return numX*numY;
    }

    int mid = midNumber(d);

    long x1 = divisionNumber(numX, mid);
    long x0 = remainderNumber(numX, mid);

    long y1 = divisionNumber(numY, mid);
    long y0 = remainderNumber(numY, mid);

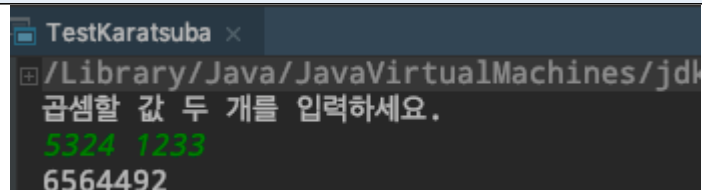
    long z2 = this.karatsubaMult(x1, y1);
    long z0 = this.karatsubaMult(x0, y0);
    long z1 = this.karatsubaMult((x0+x1), (y0+y1))-z2-z0;

    long result = (long) (z2*(Math.pow(mid,2))+z1*mid+z0);
    return result;
}

```

- 입력 된 두 값을 Karatsuba 알고리즘을 통해 divide를 진행하며 한 자리 값이 될 때까지 구해준다.
- Karatsuba 알고리즘을 재귀적으로 계속해서 최종 곱셈 연산 값을 도출한다.
- 최대 자리수는 3자리 THRESHOLD로 진행을 해주어 효율을 높여줌

구현 결과 화면



```

TestKaratsuba x
/Library/Java/JavaVirtualMachines/jdk
곱셈할 값 두 개를 입력하세요.
5324 1233
6564492

```