알고리즘

- HW06 : Dijkstra , Prim-

제 출 일	2017년 11월 09일
분 반	02반
담당교수	공은배
학 과	컴퓨터공학과
학 번	201302423
이 름	신종욱

ㄱ.Dijkstra

1. 해결 방법

```
while(Q.size!=0){
int u = returnNumber(temp.ch);
for(int s =0;s<Q.size;s++) {
int v = returnNumber(Q.a[s].ch);
System.out.print("Q["+s+"] : d["+Q.a[s].ch+"] = "+d[v]);
if(d[u]+W[u][v]>0&&d[u]+W[u][v]<d[v])
{//overflow를 생각해서 0보다큰 조건을 추가하였다.
    d[v]=d[u]+W[u][v];
    Q.chagne((s+1), d[v]);
    System.out.print(" --> "+"d["+Q.a[s].ch+"] = "+d[v] );
}
System.out.println();
}

temp = Q.extract_min();
complete[compoint]=temp.ch;
System.out.print("\nS["+compoint+"] : d["+complete[compoint]+"] = "+temp.Priority+" \n\n");
compoint++;
}
```

ppt에 나와있는데로 순서를 보고 진행하였다. 최대한 비슷하게 할려고했다. d는 현재 a로부터의 떨어진 거리를 저장하는것이데 그값이 다른곳을 거쳐서 가는길보다 클경우에는 다른곳을 거쳐서 가는곳의 값으로 바꾸는식으로 구현하였다.

2. 실행 결과

```
<terminated> Dijkstra [Java Application] C:₩Program Files₩Java₩jre1.8.0_144₩bin₩javaw.exe (2017. 11. 9. 오전 12:50:38
S[0] : d[A] = 0

Q[0] : d[E] = 2147483647
Q[1] : d[B] = 2147483647 --> d[E] = 10
Q[2] : d[C] = 2147483647 --> d[B] = 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[B] = 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
```

ㄴ.Prim

1. 해결 방법

```
    while(Q.size!=0){
        int v=0;
        int s;
        int o=0;
        for(o = 0;o<name.length;o++) {
            if(chain.contains(name[o])){//이미 연결된 집합에 존재하고
        for(s =0;s<Q.size;s++) {
        v = returnNumber(Q.a[s].ch);
        if(W[o][v]<d[v])//값이 더 작을때만
        {
            d[v]=W[o][v];
            Q.chagne((s+1), d[v]);
            }//값을 변경하여준다 동시에 heapify도 진행
    }}  }
        chain.add(Q.min().ch);
        int k= returnNumber(Q.min().ch);
        int p;
        for( p=0;p<d.length;p++) {
        if(W[p][k]==Q.min().Priority&&chain.contains(name[p])) {
            break;
        }
        }
        System.out.print("w<"+name[p]+", " );
        temp = Q.extract_min();
        System.out.println(temp.ch+"> = "+temp.Priority );
    cost = cost+temp.Priority;
        }
    System.out.println("W<MST> = "+cost);
```

앞에서한 다이스트라 과제보다 어려웠다 이유는 찾은 최소가 진짜 최소값이 아닐수도있다는 점에서 무조건 집합을 써서 관리해야하기 때문이다. 이문제를 해결하기위해서 hashset을 사용하였다.
pop을 할 최소값을 hashset에 저장하고 다음에 연결될 edge를 찾을 때 hashset에 존재하는 문자열중 큐와 연결될시 가장 작은 값을 찾는 형식으로 구현하였다.
연결된 문자열 값(A에서 B로 갔다면 A값)을 찾는 형태를 만들어주기위해 코드가 추가되었다.
사실 큐가 아닌 다른 자료구조를 사용하면 더 쉽게 구현할거같다.

2. 실행 결과

```
<terminated> Prim [Java Application] C:₩Program Files₩Java₩jre1.8.0_144₩bin₩javaw.exe (2017. 11. 9. 오전 12:58:42)
w< , a> = 0
w<a, b> = 4
w<a, h> = 8
w<h, g> = 1
w<g, f> = 2
w<f, c> = 4
w<c, i> = 2
w<c, d> = 7
w<d, e> = 9
W<MST> = 37
```

3. 느낀 점

우선 순위큐말고 다른 자료구조로 구현해본 기억이 있어서 큐를 사용하여도 쉽게 할줄 알았는데 해보니까 생각보다 어려운점이 많았다 자바의 내장기능의 위대함을 느꼈다.
그래도 이런식으로 구현해보는것도 좋은 경험인거 같다.