자료구조설계

제 출 일	2017.10.20
과 제 번 호	05
분 반	01
학 과	컴퓨터공학과
학 번	201602038
이 름	이 미 진

1. 구현 내용 설명

1-1 코드 설명

1) public void printSets()

```
public void printSets() {
// root에 속한 원소들을 출력
        for( int i=0; i< size; i++ ) {       // 정점 탐색
                if( parent[i] <= 0 ) {      // parent[i] 가 root이면
                        System.out.print("[ROOT: "+i+"], {");       // root 출력

                        for( int j=0; j<size; j++ ) {       // 정점 확인
                                int j_2 = j;
                                if(i == j)// i 와 j 가 같으면 continue
                                        continue;

                                for( ; parent[j_2] > 0 ; ) {       // root 노드일
경우까지
                                        j_2 = parent[j_2];
                                }
                                if( i == j_2 ) {  // i 가 root 노드일 때
                                System.out.print(" "+j);  // root에 속한 원소들 출력
                                }
                        }
                        System.out.println(" }");
                        }
                }
                System.out.println();
        }
```

printSets() 메소드에선 root에 속한 원소들을 출력하는 기능을 한다.
우선 size만큼 반복하면서 정점을 탐색하고, parent[i]가 root인 조건을 넣어 예를들어 root가 1인 것을 발견하면, [ROOT: 1], { 과 같이 출력되도록 하였다.
정점 탐색 중 또 다른 정점을 탐색하고, j_2변수에는 초기 j 를 저장해둔다.
I와 j가 같으면 continue를 통해 아래의 반복 코드는 넘어간다.
for문을 사용해 parent[j_2]가 root 노드일 경우까지 반복하는데, 초기 j를 저장해두었던 j_2에 parent[j_2]를 저장한다.
만약 I가 root 노드이면, j를 출력함으로써 root에 속한 원소들을 출력한다.
반복이 종료되면 원소들의 목록을 닫아주기 위해 }를 출력하고 함수 실행이 끝난다.

2) public void weightedUnion(int I, int j)

```
public void weightedUnion(int i, int j) {
// i가 속한 집합과 j가 속한 집합을 합집합으로 만든다
        int i_2 = i;       // 현재 i 를 i_2 변수에 저장
        int j_2 = j;       // 현재 j 를 j_2 변수에 저장
        int parent_i = parent[i]; // parent[i]의 값을 parent_i에 저장
        int parent_j = parent[j]; // parent[j]의 값을 parent_j에 저장

        System.out.println("weightedUnion("+i+","+j+")");    // i와 j 출력

        for( ; parent[i] > 0; ) {  // root 노드일 경우까지
                i = parent[i];
                for( ; parent[j] > 0; ) {  // root 노드일 경우까지
                        j = parent[j];
                }
```

```
        }
        parent_i = i;
        parent_j = j;

        if( parent[parent_i] < parent[parent_j] ) {        // i 집합이 j 집합보다 작을
경우
                parent[parent_i] += parent[parent_j]; // i집합 = i 집합 + j 집합
                parent[parent_j] = parent_i;
                parent_j = parent_i;

        }
        else {   // i 집합이 j 집합보다 클 경우
                parent[parent_j] += parent[parent_i]; // j 집합 = j 집합 + i 집합
                parent[parent_i] = parent_j;
                parent_i = parent_j;
        }


        while( parent[i_2] < 0 && parent[j_2] > 0 ) {     // root 노드일 경우까지
                int tmp;
                if( parent[i_2] != parent_i ) {
                        tmp = parent[i_2];
                        parent[tmp] = collapsingFind(tmp);        // parent를
tmp로 바꿈
                        }
                        if( parent[j_2] != parent_j ) {
                                tmp = parent[j_2];
                                parent[tmp] = collapsingFind(tmp);        //
parent를 tmp로 바꿈
                        }
                }
        x}
```

weightedUnion(int I,int j) 메소드는 I가 속한 집합과 j가 속한 집합을 합집합으로 만들어주는 메소드이다. I와 j의 초기값을 각각 I_2와 j_2에 저장하고, parent[i],parent[j]를 각각 parent_i,parent_j 변수에 저장한다.
우선, weightedUnion(i,j) 형식으로 I와 j를 출력해주어, 어떤 집합에 대한 것인지 명시하도록 하였다.
for문을 통해 parent[i] parent[j]가 root 노드일 경우까지 반복하면서 parent[i]는 I에 저장, parent[j]는 j에 저장한다.
반복이 끝나면 parent_i에 I를, parent_j에 j를 저장해둔다.
I와 j로 값이 바뀐 parent_i와 parent_j를 통해 합집합을 만드는 과정을 수행한다.
parent[parent_i] (i 집합) 이 parent[parent_j] (j 집합)보다 작을 경우, parent[parent_i] (i 집합) 에 parent[parent_j] (j 집합)을 더해 parent[parent_i] 값에 저장한다. 그럼 I 집합에는 I집합 + j집합을 한 값이 들어가게 된다.
그 후에 root를 재설정하는 과정을 거친다.
반대의 경우도 I와 j만 서로 변경하여 수행한다.
합집합으로 만드는 과정이 끝나면 parent[i_2] 와 parent[j_2]가 root 노드일 경우까지, collapsingFind() 메소드를 호출해 원하는 곳에 속한 집합의 root를 찾는다.
parent[i_2]가 parent_i(초기 parent[i] 값)와 같지 않으면, 임시 저장 변수인 tmp에 parent[i_2]를 저장하고, parent[parent[i_2]] 에는 collapsingFind(tmp)를 수행한 결과를 저장한다. j의 경우도 동일하게 수행한다.

3) public int collapsingFind(int I)

```
public int collapsingFind(int i) {
        // i가 속한 집합의 root를 찾는다.
        for( ; parent[i] > 0 ;) {
                i = parent[i];    // 자식에서 부모로 이동
        }
        return i;          // i = root 노드
        }
```

collapsingFind() 메소드는 I가 속한 집합의 root를 찾는 메소드이다.
parent[i]가 root 노드일 경우까지 반복하면서 I에 parent[i]를 저장하여 자식에서 부모로 이동하는 과정을 수행한다.
I를 return 해주면서 I가 root 노드가 된다.

1-2 Find & Union / WeightedUnion & CollapsingFind 비교

1) find는 특정 원소에 대한 root를 구하는 연산이고, union은 2개의 집합에 대한 합집합을 구하는 연산이다.

2) weightedUnion은 두 집합 a,b가 있을 때, a 집합의 크기가 b 집합보다 크면, b를 a의 자식으로 만들고,
CollapsingFind는 root로 가는 경로 상의 모든 원소를 root의 자식으로 만드는 것이다.

2. 실행 결과 화면

```
<terminated> TestSets [Java Application] C:₩Program Files₩Java₩jre
weightedUnion(0,1)
[ROOT: 1], { 0 }
[ROOT: 2], { }
[ROOT: 3], { }
[ROOT: 4], { }
[ROOT: 5], { }
[ROOT: 6], { }
[ROOT: 7], { }

weightedUnion(2,3)
[ROOT: 1], { 0 }
[ROOT: 3], { 2 }
[ROOT: 4], { }
[ROOT: 5], { }
[ROOT: 6], { }
[ROOT: 7], { }

weightedUnion(4,5)
[ROOT: 1], { 0 }
[ROOT: 3], { 2 }
[ROOT: 5], { 4 }
[ROOT: 6], { }
[ROOT: 7], { }

weightedUnion(6,7)
[ROOT: 1], { 0 }
[ROOT: 3], { 2 }
[ROOT: 5], { 4 }
[ROOT: 7], { 6 }

weightedUnion(0,3)
[ROOT: 3], { 0 1 2 }
[ROOT: 5], { 4 }
[ROOT: 7], { 6 }

weightedUnion(4,7)
[ROOT: 3], { 0 1 2 }
[ROOT: 7], { 4 5 6 }

weightedUnion(0,7)
[ROOT: 7], { 0 1 2 3 4 5 6 }
```