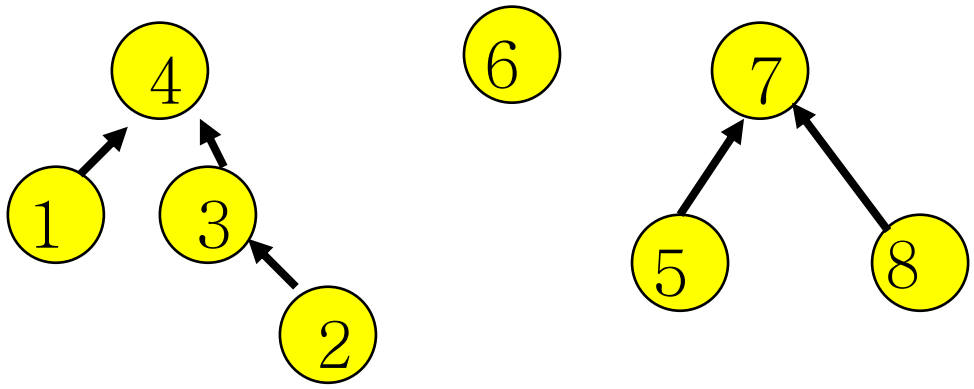
자료 구조 Lab 006 :

lab006.zip 파일 : LabTest.java lab006.java lab.in lab.out

제출

lab006.java 를 lab006_학번.java 로 변경하여 이 파일 한 개만 제출할 것.

다음은 Tree를 이용한 Disjoint Set에 관한 문제이다.
다음과 같은 Disjoint Set의 예를 살펴보자. 노드 수는 8개이다.



parent[]=

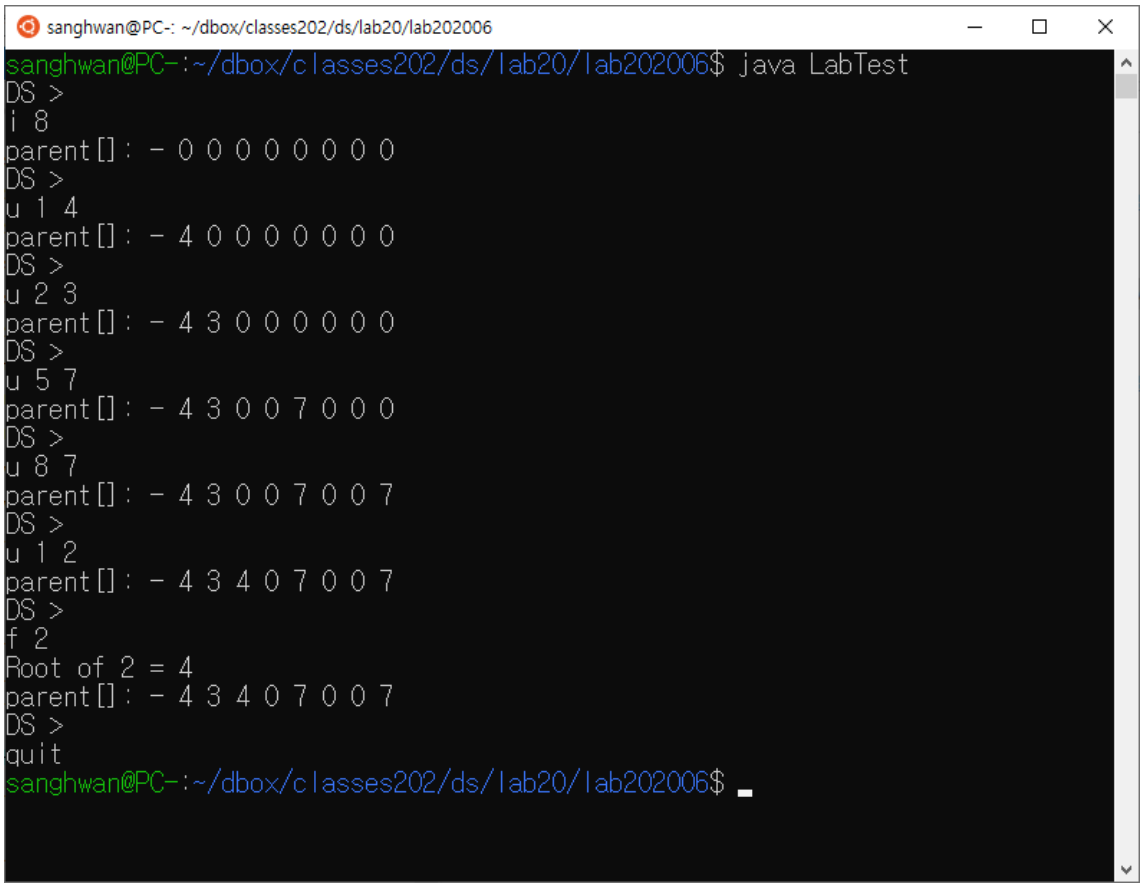
-	4	3	4	0	7	0	0	7
0	1	2	3	4	5	6	7	8

weight[]=

-	-	-	-	4	-	1	3	-
0	1	2	3	4	5	6	7	8

parent[] 배열은 각 원소의 부모 노드를 저장하고 있다. weight[] 배열은 해당 루트 노드가 속한 집합의 원소의 수를 저장하고 있다.. 노드는 1번부터 n 번까지만 사용된다.

이러한 상황에서 사용자는 Union 과 Find 를 수행하게 된다. 수행 예는 다음과 같다.



사용자가 사용하는 명령어의 syntax는 다음과 같다. main() 함수에 정의되어 있다.

- i : i noe

  정수인 noe를 입력 받는다. noe는 전체 원소의 수를 의미한다. 이 명령이 수행되면 parent배열과 weight 배열이 초기화 된다.

- u : u i j

  Union 명령이다. 원소 i와 원소 j를 입력 받아 이 두 원소가 속한 집합을 합하여 합집합을 형성한다. 만약 i와 j가 같은 집합에 속하면 에러 메시지를 출력한다. 여기서 **i와 j는 루트 노드는 아닐 수도 있다.**

- f : f i

  Find 명령이다. 원소 i를 입력 받아 이 원소가 속한 집합의 root 원소를 출력하여 보여준다.

구현이 필요한 부분은 다음 세 함수이다. 구체적인 알고리즘도 함께 설명한다.

- Void InitSet(int n)

  이 함수는 Disjoint Set을 초기화 하는 역할을 한다. 파라미터로 주어진 값이 전체 노드의 수이다. 이 값을 numofelements 변수에 assign 하고, **parent** 배열과

weight 배열을 초기화 한다. 초기 상태는 모든 노드가 각각 하나의 트리를 형성한다는 점에 착안하면 쉽게 초기화 할 수 있다. 주어진 원소의 수보다 1크게 배열을 잡아야 함에 주의해야 한다.

- int SimpleFind(int i);
  원소 i를 입력 받아 이 원소가 속한 집합의 root 원소를 찾아서 return 한다. 알고리즘은 parent[] 배열에 정의된 부모 노드에 대한 포인터를 따라가서 루트 노드가 나오면 멈추는 것이다. 그리고 찾은 루트 노드를 return 한다. 어떤 원소가 루트 일 경우 위 예에서 보는 바와 같이 parent[]의 해당 루트 원소의 값이 0이다.
- bool Union(int i, int j);
  이 함소는 원소 i와 j가 속한 두 집합을 합치는 역할을 한다. 우선 i와 j가 속하는 집합의 root를 찾는다. 이 작업은 Find() 함수를 이용해서 할 수 있다. 그 다음에 두 집합을 합치는데 이 때 **weight[]가 작은 집합이 weight[]가 큰 집합의 루트의 서브트리가 되도록 하여 합집합을 구성한다**. 즉 이른바 Weight Rule을 따른다. 합치고 나면 이 합집합의 루트 노드의 weight를 변경한다. 이 새로운 weight는 두 집합의 weight를 합한 값이다.
  Weight가 같은 경우에는 root의 번호가 작은 트리가 root의 번호가 큰 트리의 sub tree가 되도록 한다.
  만약 i와 j 가 같은 집합에 속해 있으면 바로 false를 return 한다. 그렇지 않은 경우 작업을 끝내고 true를 return 한다.

프로그램 결과 테스트

$ **diff aaa lab.out**

또는

$ **diff** -i --strip-trailing-cr -w **aaa lab.out**