

[제 1 주 실습]

그래프의 싸이클 검사

강 지 훈

jhkang@cnu.ac.kr



□ 수업 주의사항

- 두 반이 같은 강의실을 사용하므로 본인의 소스 관리는 본인이 철저히 할 것!!
 - 검사를 통하여 Copy를 잡아낼 예정임
 - 컴퓨터 내에 소스를 남기지 말 것!

- 수업 자료
 - winslab.cnu.ac.kr
 - 왼쪽 메뉴의 [2013-autumn]의 자료구조 설계

- OFFICE TIME : Tue 13-15 (5505)

[문제 1]

그래프 : 기본기능

Adjacency Matrix

□ 구현에서의 주요 내용

■ 그래프의 표현

- Adjacency Matrix로 구현한다.

■ Union-Find 알고리즘 구현

- 새로운 edge를 추가할 때 cycle이 생기는지를 검사



□ 입력

- 키보드로부터 그래프 정보를 입력 받는다.
 - Vertex의 개수
 - ◆ Vertex는 0부터 시작하는 정수로 나타낸다.
 - Edge의 개수
 - ◆ 입력되는 그래프의 edge의 개수

출력

- 입력될 때마다 Union & Find를 통하여 cycle이 생성 되는지 확인여부 출력

- <디버깅용 출력>

입력이 완료 된 후 그래프 내용

[0] -> 1 4 5

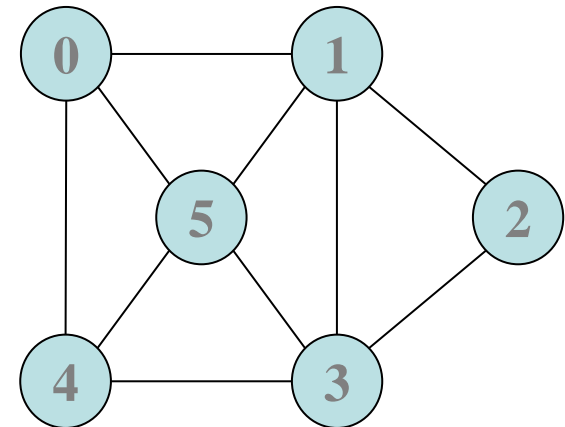
[1] -> 0 2 3 5

[2] -> 1 3

[3] -> 1 2 4 5

[4] -> 0 3 5

[5] -> 0 1 3 4



□ 출력의 예

- 그래프의 vertex수와 edge수를 입력 받아야 합니다.
- ? 그래프의 vertex 수를 입력 하시오: 6
- ? 그래프의 edge 수를 입력 하시오: 10
- 그래프의 edge를 반복하여 10 개 입력 받아야 합니다.
- 하나의 edge는 (vertex1 vertex2)의 순서로 표시됩니다.

```
? Edge를 입력하시오: 0 1
? Edge를 입력하시오: 0 4
? Edge를 입력하시오: 0 6
[Error] 입력 오류입니다.
? Edge를 입력하시오: 0 5
? Edge를 입력하시오: 1 5
? Edge를 입력하시오: 0 4
[Error] 입력 오류입니다.
? Edge를 입력하시오: 4 3
? Edge를 입력하시오: 3 4
[Error] 입력 오류입니다.
? Edge를 입력하시오: 3 5
? Edge를 입력하시오: 3 1
Edge (3, 1)는 cycle을 생성 시킵니다.
? Edge를 입력하시오: 2 1
? Edge를 입력하시오: -1 2
[Error] 입력 오류입니다.
? Edge를 입력하시오: 3 2
Edge (3, 2)는 cycle을 생성 시킵니다.
? Edge를 입력하시오: 4 5
Edge (4, 5)는 cycle을 생성 시킵니다.
```

! 입력된 그래프는 다음과 같습니다:

```
[0] -> 1 4 5
[1] -> 0 2 3 5
[2] -> 1 3
[3] -> 1 2 4 5
[4] -> 0 3 5
[5] -> 0 1 3 4
```



□사용자에게 필요한 객체는?

- AdjacencyMatrixGraph

- Edge

- PairwiseDisjointSets

- Application

AdjacencyMatrixGraph의 멤버 함수는?

■ 사용자에게 필요한 함수 (Public Functions)

- `public AdjacencyMatrixGraph(int givenNumOfVertices)`
- `public boolean doesVertexExist (int aVertex)`
- `public boolean doesEdgeExist (Edge anEdge)`
- `public int numOfVertices ()`
- `public int numOfEdges ()`
- `public boolean addEdge(Edge anEdge)`

□ Edge의 멤버 함수는?

■ 사용자에게 필요한 함수 (Public Functions)

- `public Edge(int givenTailVertex, int givenHeadVertex)`
- `public void setTailVertex(int aTailVertex)`
- `public int tailVertex()`
- `public void setHeadVertex(int aHeadVertex)`
- `public int headVertex()`

□ PairwiseDisjointSets 의 멤버 함수는?

- 사용자에게 필요한 함수 (Public Functions)
 - `public PairwiseDisjointSets(int givenMaxNumOfMembers)`
 - `public int find(int aMember)`
 - `public void union(int idOfSet1, int idOfSet2)`



□ Application의 공개 함수는?

■ 사용자에게 필요한 함수 (Public Functions)

- public void showGraph()

- public void run ()

DS2_XX_학번_이름 Class



□ DS2_01_학번_이름 Class의 구조

/* 항상 사용하는 main Class 구조

* 과제 제출시 반드시 포함 되어 있어야 함*/

```
public class DS2_01_학번_이름 {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        Application application = new Application();  
        application.run ();  
    }  
}
```

Class

"Application"



□ Application – 비공개 인스턴스 변수

```
public class Application {  
    private AdjacencyMatrixGraph _graph;  
    private PairwiseDisjointSets _pairwiseDisjointSets;  
    private Scanner _scanner;
```

□ Application의 Public Method

■ Application의 Public Member function의 사용법

● public void showGraph()

- ◆ 현재 상태의 Graph를 출력한다.

● public void run ()

- ◆ 실제 프로그램을 실행한다.

□ Application의 Private Method

■ Application의 Private Member function의 사용법

- private void inputAndMakeGraph()
 - ◆ 키보드로부터 그래프 정보를 입력 받는다.
 - ◆ 마지막에 입력된 그래프를 출력하여 보여준다.
- private void checkCycle(Edge anEdge)
 - ◆ 전달 받은 anEdge가 Cycle을 이루는지 확인한다

□ Member Functions의 구현

■ Application의 Public Member function의 구현

● public void showGraph()

- ◆ Graph의 showGraph() 함수를 호출

- ◆ 실제로 Graph의 내용을 모두 접근 하여 확인 할 수는 없기에 디버깅용 함수를 사용한다.

● public void run ()

- ◆ inputAndMakeGraph를 실행한다.

Member Functions의 구현

Application의 Private Member function의 구현

```
private void inputAndMakeGraph()
{
    int countEdges;
    int numOfVertices ;
    int numOfEdges ;

    System.out.println("- 그래프의 vertex수와 edge수를 입력 받아야 합니다.");
    System.out.print("? 그래프의 vertex 수를 입력 하시오:");
    _scanner = new Scanner(System.in);
    numOfVertices = _scanner.nextInt();
    System.out.print("? 그래프의 edge 수를 입력 하시오:");
    numOfEdges = _scanner.nextInt();

    // AdjacencyMatrixGraph 생성
    // 입력받은 numOfVertices와 numOfEdges를 이용하여 생성한다.
    this._graph = new AdjacencyMatrixGraph(numOfVertices);
    this._pairwiseDisjointSets = new PairwiseDisjointSets(numOfVertices);

    countEdges = 0;
    System.out.println("- 그래프의 edge를 반복하여 " + numOfEdges + " 개 입력 받아야 합니다.");
    System.out.println(" 하나의 edge는 (vertex1 vertex2)의 순서로 표시됩니다.");
    System.out.print("? Edge를 입력하시오: ");
    while(_scanner.hasNext()) {
        Edge anEdge = new Edge(_scanner.nextInt(), _scanner.nextInt());
        if(this._graph.addEdge(anEdge)){
            countEdges++;
            this.checkCycle(anEdge);
        }
        else
            System.out.println("[Error] 입력 오류입니다.");

        if(countEdges == numOfEdges)
            break;
        System.out.print("? Edge를 입력하시오: ");
    }

    this.showGraph();
}
```

□ Member Functions의 구현

■ Application의 Private Member function의 구현

- private void checkCycle(Edge anEdge)
 - ◆ 각 Edge의 root를 찾는다(Find)
 - ◆ 찾은 root가 같으면 "Cycle을 생성 시킵니다"를 출력한다.
 - ◆ 같지 않으면 Union을 한다.

Class "Edge"



□ 비공개 인스턴스 변수

```
public class Edge {  
    private int _tailVertex;  
    private int _headVertex;
```

□ Member Functions의 구현

■ Edge 의 Public Member function의 사용법

- public Edge(int givenTailVertex, int givenHeadVertex)
 - ◆ Edge 생성자
- public void setTailVertex(int aTailVertex)
 - ◆ _tailVertex값을 aTailVertex으로 저장한다.
- public int tailVertex()
 - ◆ _ tailVertex의 값을 확인한다.
- public void setHeadVertex(int aHeadVertex)
 - ◆ _headVertex값을 aHeadVertex로 저장한다.
- public int headVertex()
 - ◆ _ headVertex값을 확인한다.

□ Member Functions의 구현

■ Edge 의 Public Member function의 구현

- public Edge(int givenTailVertex, int givenHeadVertex)
 - ◆ givenTailVertex의 값을 this._tailVertex에 저장
 - ◆ givenHeadVertex의 값을 this._headVertex에 저장
- public void setTailVertex(int aTailVertex)
 - ◆ aTailVertex의 값을 this._tailVertex 에 저장
- public int tailVertex()
 - ◆ this._tailVertex을 반환
- public void setHeadVertex(int aHeadVertex)
 - ◆ aHeadVertex의 값을 this._headVertex에 저장
- public int headVertex()
 - ◆ this._headVertex을 반환

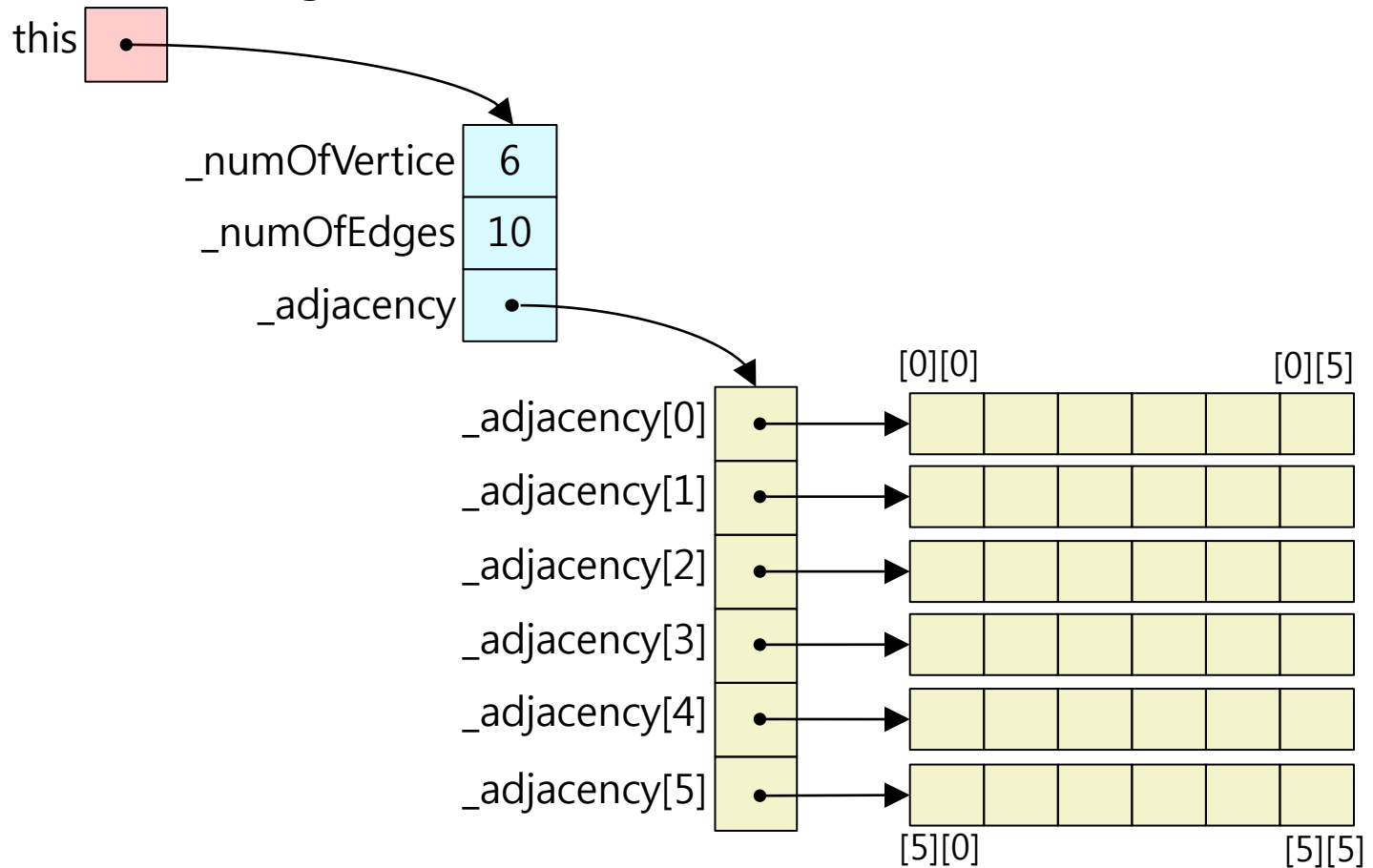
Class

"AdjacencyMatrixGraph"



비공개 인스턴스 변수

```
public class AdjacencyMatrixGraph {
    private int[][] _adjacency ;
    private int    _numOfVertices ;
    private int    _numOfEdges ;
```



Member Functions의 구현

AdjacencyMatrixGraph의 Public Member function의 사용법

- public AdjacencyMatrixGraph(int givenNumOfVertices)
 - ◆ AdjacencyMatrixGraph 생성자
- public int numOfVertices ()
 - ◆ 그래프의 vertex 개수를 얻는다.
- public int numOfEdges ()
 - ◆ 그래프의 edge 개수를 얻는다.
- public boolean doesVertexExist (int aVertex)
 - ◆ 그래프에 Vertex가 존재하는지 확인(적합한 Vertex 값인지 확인)
- public boolean doesEdgeExist (Edge anEdge)
 - ◆ 그래프에 Edge anEdge 가 존재하는지 확인
- public boolean addEdge(Edge anEdge)
 - ◆ 그래프 정보를 입력 받아 그래프를 만든다.
- public void showGraph()
 - ◆ 디버깅을 위한 그래프 내의 모든 내용을 출력하는 함수

□ Member Functions의 구현

■ AdjacencyMatrixGraph의 Public Member function의 구현

- `public AdjacencyMatrixGraph(int givenNumOfVertices)`
 - ◆ `givenNumOfVertices`를 `_numOfVertices`에 저장
 - ◆ `_numOfEdges`를 0으로 초기화
 - ◆ `_adjacency`를 `_numOfVertices * _numOfVertices` 만큼 생성
 - ◆ 생성 된 `_adjacency`를 모두 0으로 초기화
- `public int numOfVertices ()`
 - ◆ `_numOfVertices`를 반환
- `public int numOfEdges ()`
 - ◆ `_numOfEdges`를 반환

Member Functions의 구현

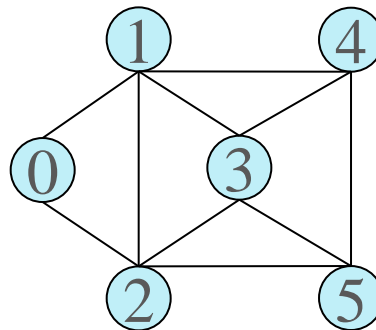
AdjacencyMatrixGraph의 Public Member function의 구현

- public boolean doesVertexExist (int aVertex)
 - ◆ aVertex의 값이 -1보다 크고 aNumOfVertices보다 작을 경우 적합한 Vertex이므로 true를 반환
- public boolean doesEdgeExist (Edge anEdge)
 - ◆ _adjacency에 anEdge.tailVertex()번째의 anEdge.headVertex()위치에 값이 이미 존재하면 true를 반환
 - ◆ _adjacency에 anEdge.headVertex()번째의 anEdge.tailVertex()위치에 값이 이미 존재하면 true를 반환
 - ◆ 값이 존재하지 않을 경우 false를 반환

Member Functions의 구현

AdjacencyMatrixGraph의 Public Member function의 구현

- public boolean addEdge(Edge anEdge)
 - ◆ 전달 받은 anEdge의 Vertex들이 적합한 지 확인(doesVertexExist)하여 존재 하지 않을 경우 false 반환
 - ◆ 전달 받은 anEdge의 Edge가 존재 하는지 확인(doesEdgeExist)하여 존재 할 경우 false 반환
 - ◆ Edge위치의 _adjacency에 값을 삽입
 - tailVertex의 headVertex번째에는 1 삽입
 - headVertex의 tailVertex번째에는 1 삽입
 - ◆ true를 반환



| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 1 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 1 | 0 | 1 |
| 5 | 0 | 0 | 1 | 1 | 1 | 0 |

□ Member Functions의 구현

■ AdjacencyMatrixGraph의 Public Member function의 구현

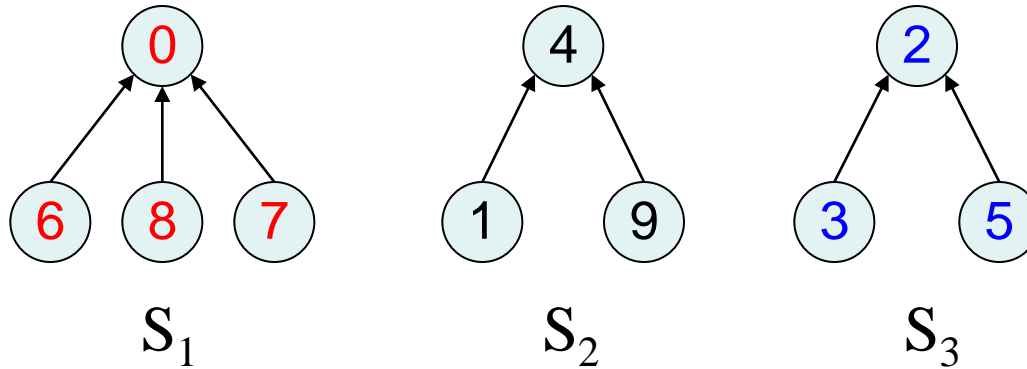
- public void showGraph()
 - ◆ 디버깅을 위한 함수
 - ◆ 모든 Vertex를 돌며 저장된 내용을 출력한다.

Class

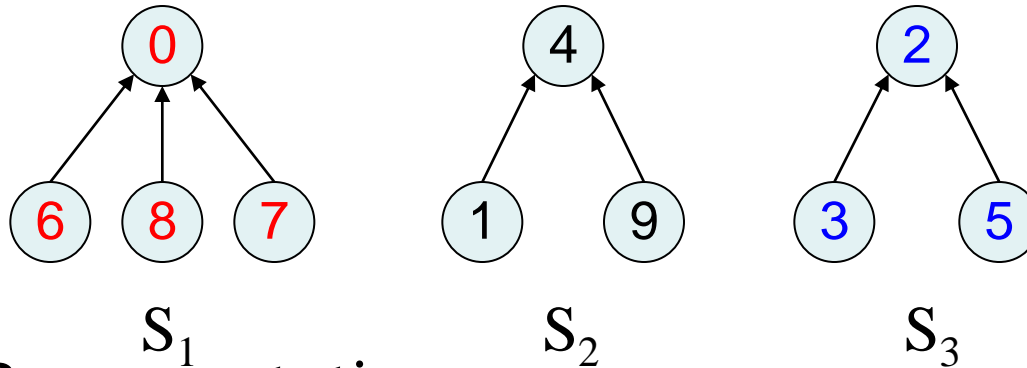
"PairwiseDisjointSets"



□ Possible Representation Model [1]



❑ Possible Representation Model [2]



■ Array Representation

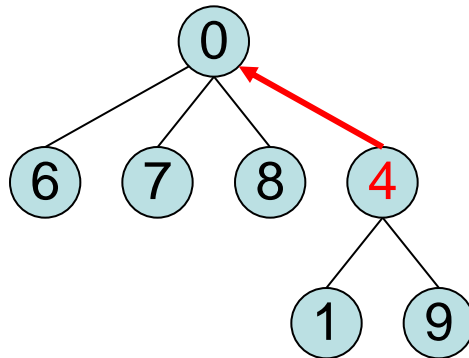
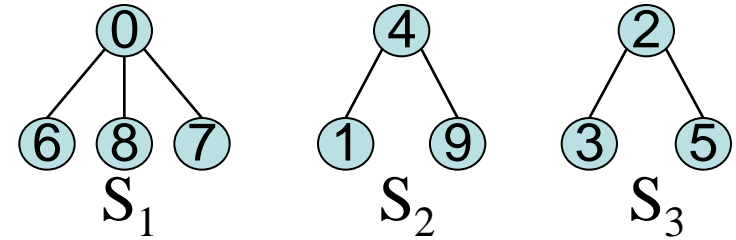
- Each element points to its parent element.
- The root has the value '-1'.
- The root label can be used as the set name.

$S_1 \rightarrow 0$ $S_2 \rightarrow 4$ $S_3 \rightarrow 2$

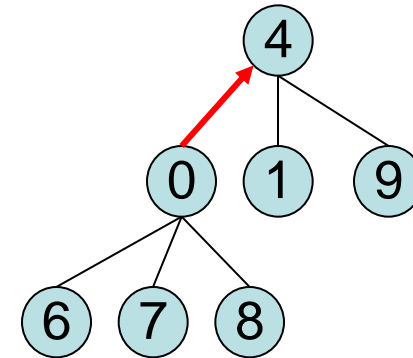
| | | | | | | | | | |
|----|---|----|---|----|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| -1 | 4 | -1 | 2 | -1 | 2 | 0 | 0 | 0 | 4 |

Union

- Union(S_1, S_2)
= { 0, 6, 7, 8, 1, 4, 9 }



or

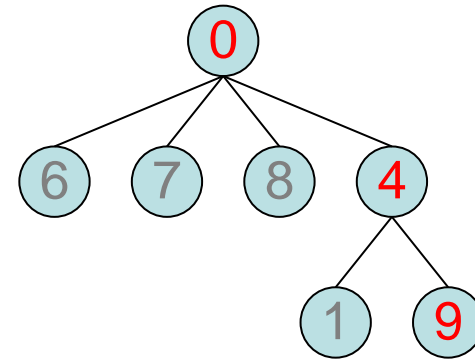


| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|---|----|---|---|---|---|---|---|---|
| -1 | 4 | -1 | 2 | 0 | 2 | 0 | 0 | 0 | 4 |

```
void union1 (int i, int j)
{
    parent[i] = j ; // or, parent[j] = i
}
```


Find

● Find (9) = 0

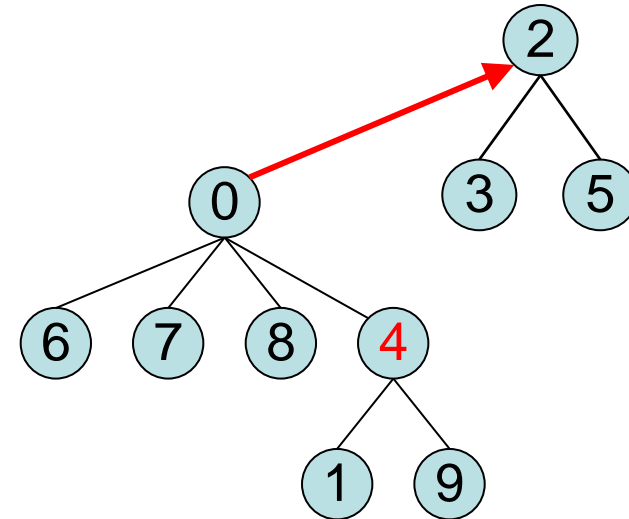
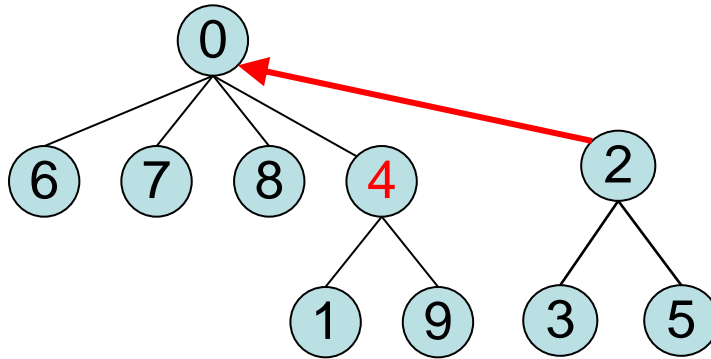


```

int find1 (int i)
{
    for ( ; parent[i] >= 0; i = parent[i] );
    return i;
}
  
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|---|----|---|---|---|---|---|---|---|
| -1 | 4 | -1 | 2 | 0 | 2 | 0 | 0 | 0 | 4 |

■ How about the tree height after Union?

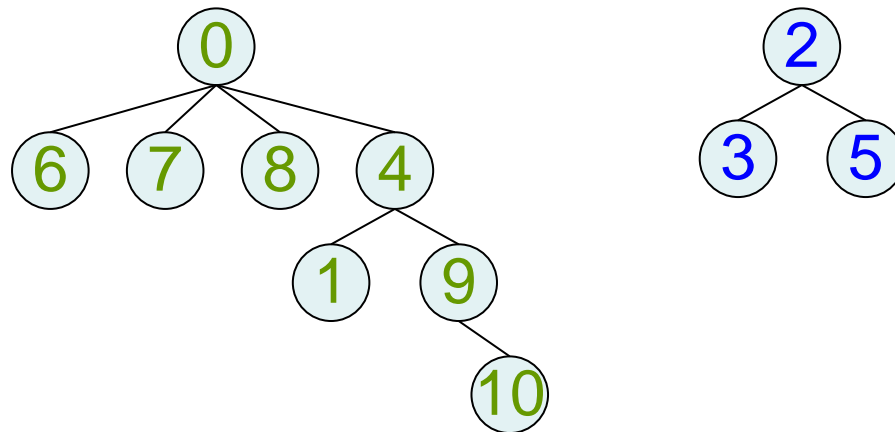


- And then, Find(1):
- Which tree is better?

Weighting Rule for union(i,j)

Example

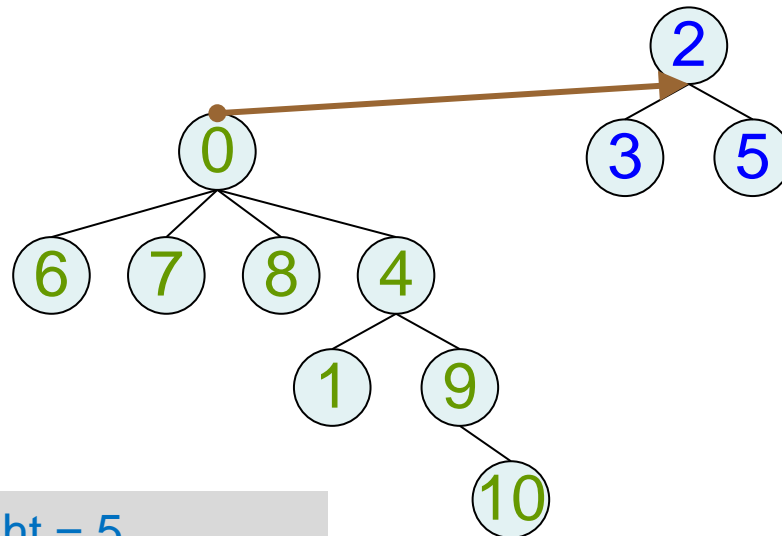
- Set(0): Height = 5, Number of elements = 8
- Set(2): Height = 2, Number of elements = 3



Weighting Rule for union(i,j)

Example

- Set(0): Height = 5, Number of elements = 8
- Set(2): Height = 2, Number of elements = 3
- If we attach Set(0) to Set(2)?



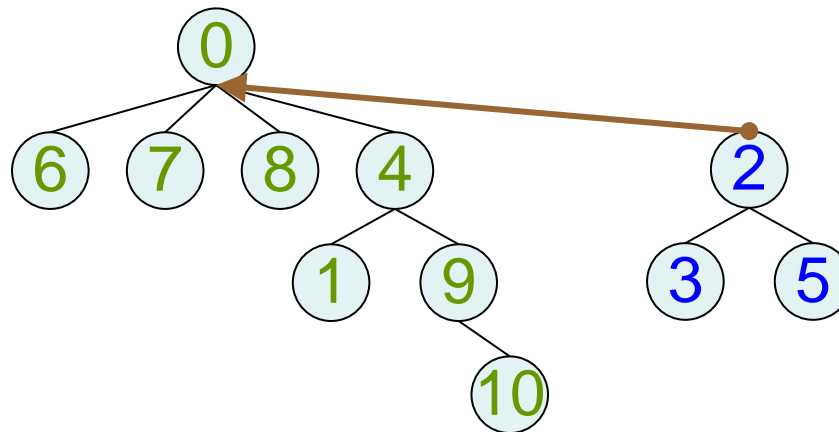
Height = 5

Number of elements = 11

Weighting Rule for union(i,j)

Example

- Set(0): Height = 5, Number of elements = 8
- Set(2): Height = 2, Number of elements = 3
- If we attach Set(2) to Set(1)?



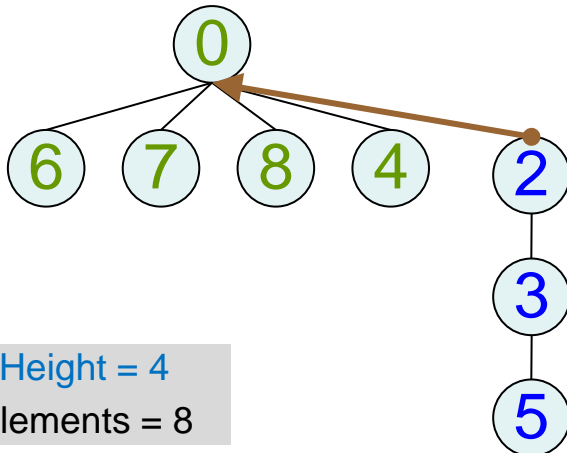
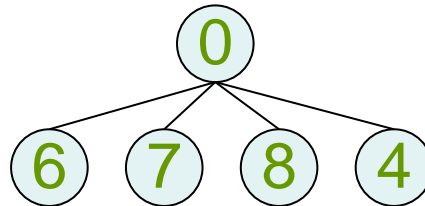
Height = 4

Number of elements = 11

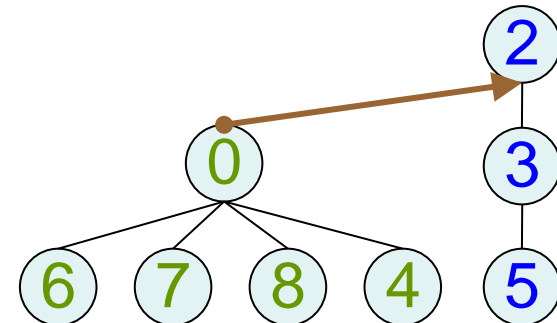
❑ Exceptional case ?

■ Unusual Example

- Set(0): Height = 2, Number of elements = 5
- Set(2): Height = 3, Number of elements = 3



Height = 4
Elements = 8



Height = 3
Elements = 8

Weighting Rule for union(i,j)

Idea

- If set(i) has more elements than set(j), then set(i) will be taller than set(j) .
- We want not to increase the height of Union of two sets.
- So, let's attach the smaller one to the bigger one.

Weighting Rule:

- If ($\#i < \#j$), then make j the parent of i.
- If ($\#i \geq \#j$), then make i the parent of j.
 - ◆ $\#i$: the number of elements in the set i.
 - ◆ $\#j$: the number of elements in the set j.

Weighting Rule for union(i,j)

Implementation

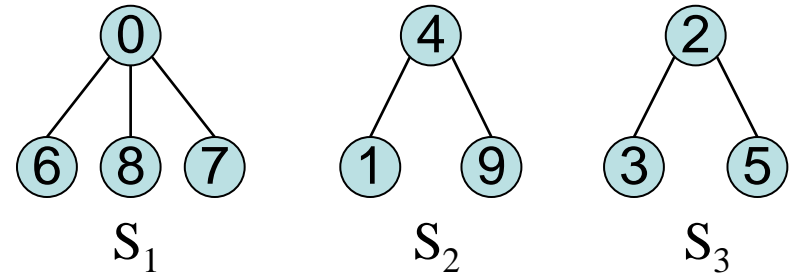
- Each root i has the negative value of the #i instead of -1.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| -4 | 4 | -3 | 2 | -3 | 2 | 0 | 0 | 0 | 4 |

```

void union2 (int i, int j)
{
    if (parent[i] >= parent[j])
        /* #i < #j */
        parent[i] = j;
    else /* #i >= #j */
        parent[j] = i;
}

```



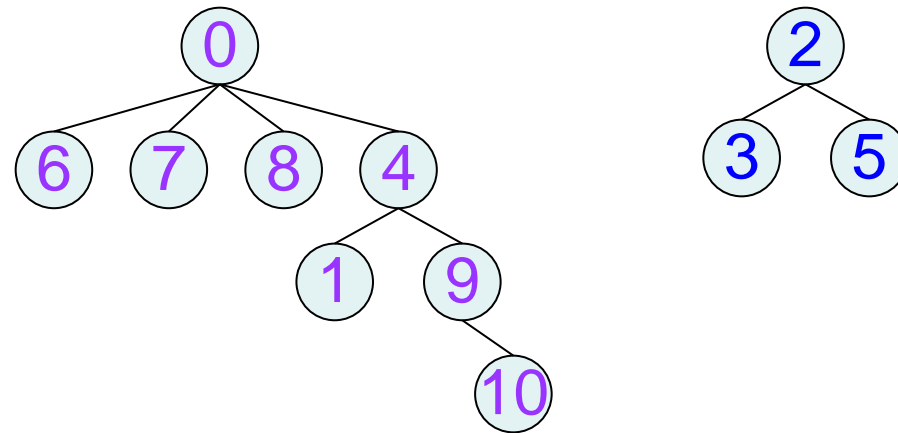
❑ Collapsing Rule for find(i)

■ Idea

- If we maintain the trees as shallow as possible, we can get efficiency during find().

■ Rule

- If j is a node in the path from i to its root, then make j a child of the root.



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|----|---|----|---|---|---|---|---|---|---|----|
| -8 | 4 | -3 | 2 | 0 | 2 | 0 | 0 | 0 | 4 | 9 |

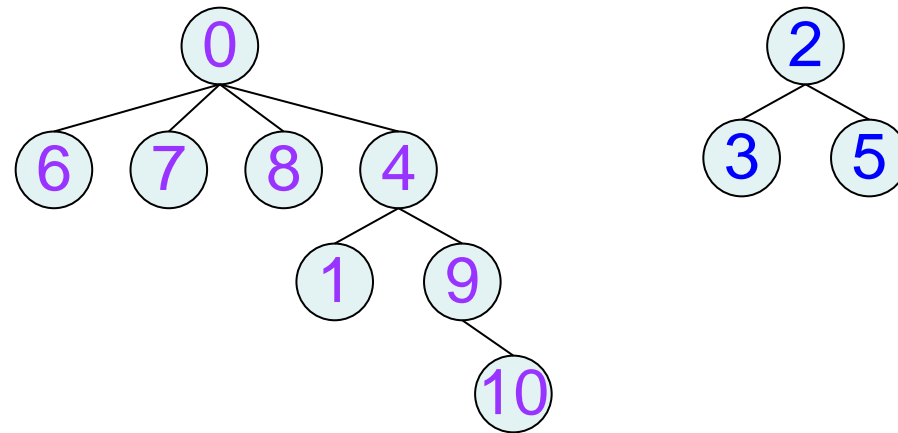
■ Collapsing Rule for find(i)

● Idea

- ◆ If we maintain the trees as shallow as possible, we can get efficiency during find().

● Rule

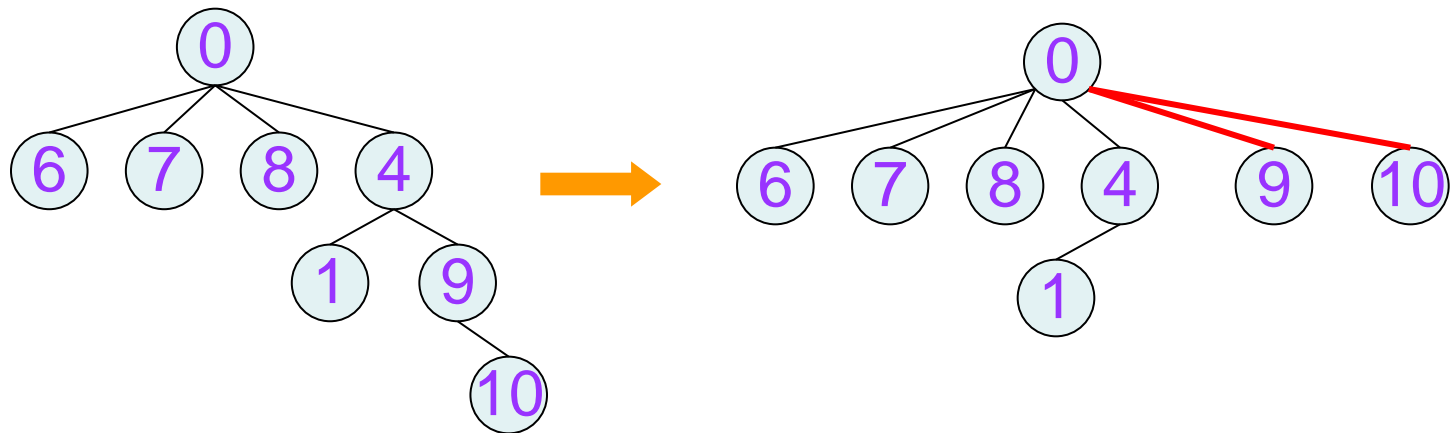
- ◆ If j is a node in the path from i to its root, then make j a child of the root.



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|----|---|----|---|---|---|---|---|---|---|----|
| -8 | 4 | -3 | 2 | 0 | 2 | 0 | 0 | 0 | 4 | 9 |

● Implementation of find2(i)

- ◆ After finding the root, we **scan again** the path from the given node i to the root and **make each node in the path to the direct child of the root**.
- ◆ Example: find2(10) = 0



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|----|---|----|---|---|---|---|---|---|---|----|
| -8 | 4 | -3 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

□ Analysis

■ Lemma

- Let $T(m, n)$ be the maximum time required to process an intermixed sequence of $m \geq n$ finds and $n - 1$ unions. Then

$$k_1 m \alpha(m, n) \leq T(m, n) \leq k_2 m \alpha(m, n)$$

for some positive constants k_1 and k_2 .

- Here, we may assume that $\alpha(m, n) = \Theta(1)$ in all practical situations.

$$\Rightarrow T(m, n) = \Theta(m \alpha(m, n)) \approx \Theta(m)$$

■ Ackerman's function

$$A(p, q) = \begin{cases} 2q & \text{if } p = 0 \\ 0 & \text{if } q = 0 \text{ \& } p \geq 1 \\ 2 & \text{if } p \geq 1 \text{ \& } q = 1 \\ A(p-1, A(p, q-1)) & \text{if } p \geq 1 \text{ \& } q \geq 2 \end{cases}$$

- For practical purposes we may assume that $A(3,4) > \log_2 n$ since $A(3,4)$ is very very big.

$$A(3,4) = 2^{2^{2^2}} \quad \left. \vphantom{2^{2^{2^2}}} \right\} 65,536 \text{ twos}$$

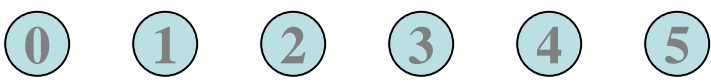
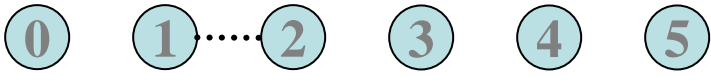
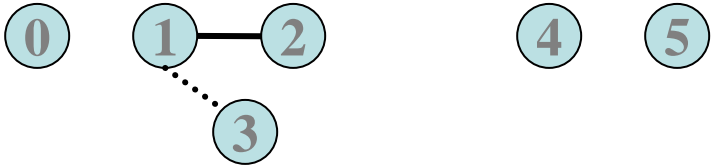
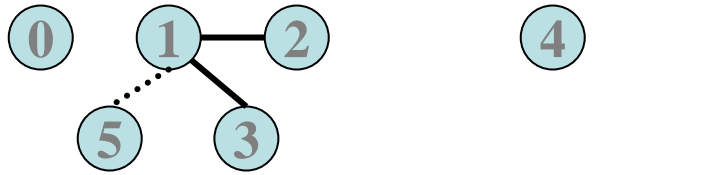
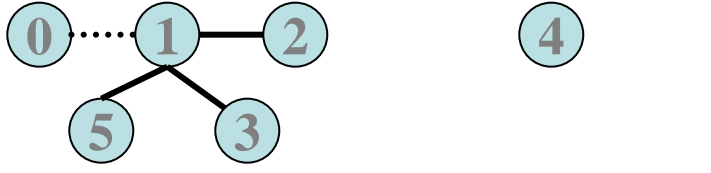
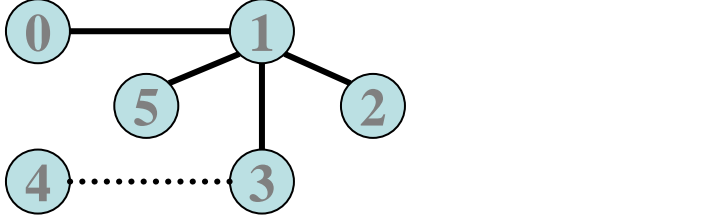
$$\alpha(m,n) = \min \left\{ z \geq 1 \mid A\left(z, 4 \left\lceil \frac{m}{n} \right\rceil\right) > \log_2 n \right\}$$

And hence, $\alpha(m,n) \leq 3$.

■ Note:

- Although $\alpha(m,n)$ is a very slowly increasing function, the complexity is *not linear* in m .

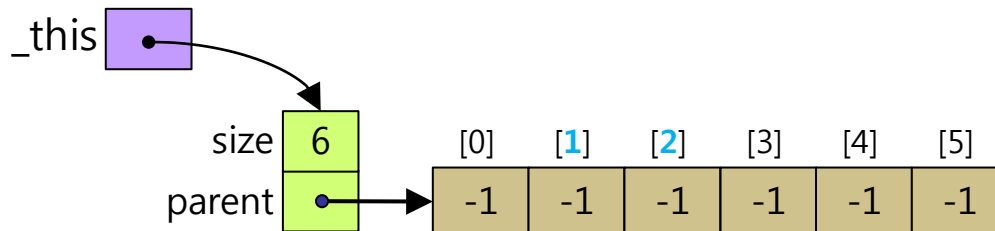


| Edge | Cost | Action | Tree Edges | Pairwise disjoint sets |
|-------|------|---------|--|-------------------------|
| - | - | - |  | {0} {1} {2} {3} {4} {5} |
| (1,2) | 5 | Add |  | {0} {1,2} {3} {4} {5} |
| (1,3) | 6 | Add |  | {0} {1,2,3} {4} {5} |
| (2,3) | 10 | Discard | | {0} {1,2,3} {4} {5} |
| (1,5) | 11 | Add |  | {0} {1,2,3,5} {4} |
| (3,5) | 14 | Discard | | {0} {1,2,3,5} {4} |
| (0,1) | 16 | Add |  | {0,1,2,3,5} {4} |
| (3,4) | 18 | Add |  | {0,1,2,3,4,5} |

□ Detecting a cycle

■ Edge: (1,2)

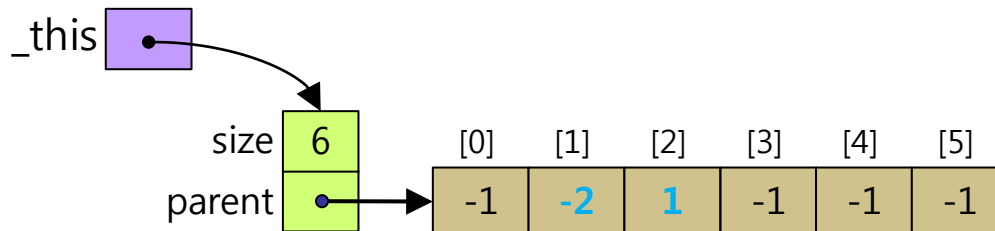
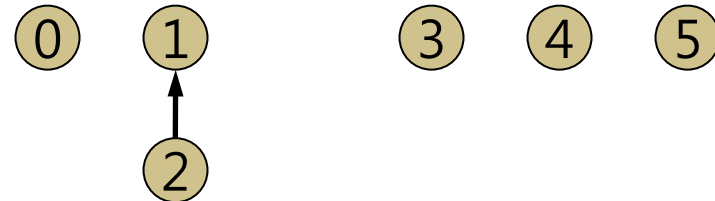
- $\text{find}(1) = 1$
- $\text{find}(2) = 2$
- So, $\text{find}(1) \neq \text{find}(2)$



□ Detecting a cycle

■ Edge: (1,2)

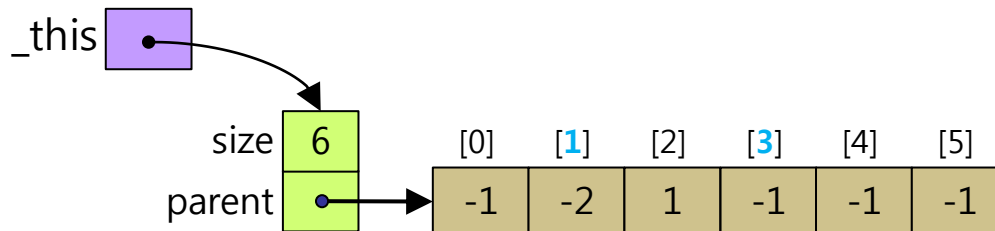
- $\text{find}(1) = 1$
- $\text{find}(2) = 2$
- So, $\text{find}(1) \neq \text{find}(2)$
- Therefore, $\text{union}(1,2)$



□ Detecting a cycle

■ Edge: (1,3)

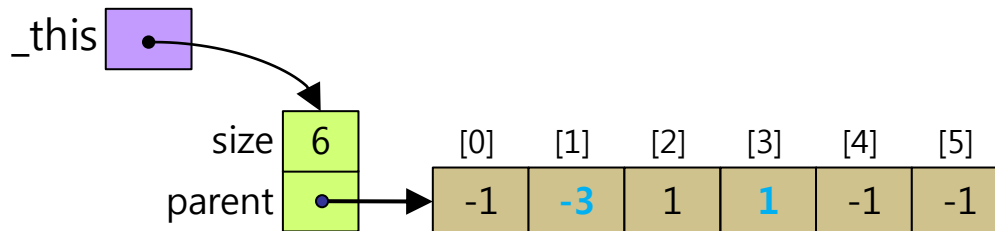
- $\text{find}(1) = 1$
- $\text{find}(3) = 3$
- So, $\text{find}(1) \neq \text{find}(3)$



□ Detecting a cycle

■ Edge: (1,3)

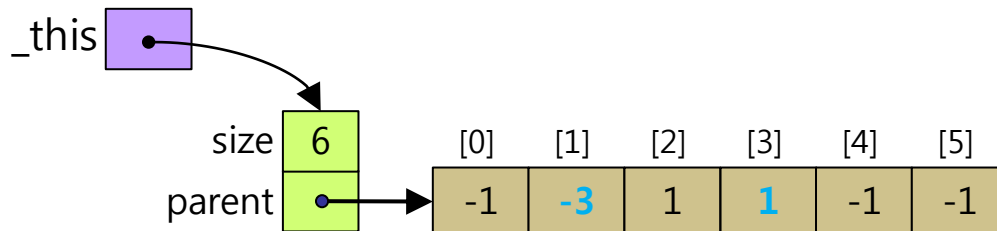
- $\text{find}(1) = 1$
- $\text{find}(3) = 3$
- So, $\text{find}(1) \neq \text{find}(3)$
- Therefore, $\text{union}(1,3)$



□ Detecting a cycle

■ Edge: (2,3)

- $\text{find}(2) = 1$
- $\text{find}(3) = 1$
- So, $\text{find}(1) == \text{find}(3)$
- Therefore, discard the edge (2,3) since (2,3) makes the cycle.



□ 비공개 인스턴스 변수

```
public class PairwiseDisjointSets {  
    private int _size;  
    private int[] _parent;
```

□ Member Functions의 구현

■ PairwiseDisjointSets 의 Public Member function의 사용법

- `public PairwiseDisjointSets(int givenMaxNumOfMembers)`
 - ◆ PairwiseDisjointSets 생성자
- `public int find(int aMember)`
 - ◆ 주어진 member 가 어느 집합에 속해 있는지, 그 집합의 ID를 얻는다.
- `public void union(int idOfSet1, int idOfSet2)`
 - ◆ 주어진 두 집합 id1과 id2를 하나의 집합으로 만든다.

□ Member Functions의 구현

■ PairwiseDisjointSets 의 Public Member function의 구현

- public PairwiseDisjointSets(int givenMaxNumOfMembers)
 - ◆ givenMaxNumOfMembers를 this._size에 저장
 - ◆ this._parent를 this._size만큼 생성
 - ◆ 생성된 this._parent를 모두 -1로 초기화
- public int find(int aMember)
 - ◆ Collapsing rule을 적용한 find를 작성한다.
- public void union(int idOfSet1, int idOfSet2)
 - ◆ idOfSet1과 idOfSet2는 서로 다른 set으로 가정한다.
 - UNION을 하기 전에 반드시 Find를 실행하여 두 집합이 서로 다르다는 것을 확인해야 한다.

□[문제 1] 요약

- Matrix를 이용하여 Graph를 구현해본다.
- Union & Find 알고리즘을 이해하고 실제 적용하여 입력받는 Edge가 Cycle을 이루는 지 확인 하여 본다.
- 확인 해본 내용은 보고서에 포함이 되어야 한다.

과제 제출

□ 과제 제출

■ pineai@cnu.ac.kr

- 메일 제목 : [0X]DS2_01_학번_이름
 - ◆ 양식에 맞지 않는 메일 제목은 미제출로 간주됨
 - ◆ 앞의 0X는 분반명 (오전10시 : 00반 / 오후4시 : 01반)

■ 제출 기한

- 9월 3일(화) 23시59분까지
- 시간 내 제출 엄수
- 제출을 하지 않을 경우 0점 처리하고, 숙제를 50% 이상 제출하지 않으면 F 학점 처리하며, 2번 이상 제출하지 않으면 A 학점을 받을 수 없다.

□ 과제 제출

■ 파일 이름 작명 방법

● DS2_01_학번_이름.zip

● 폴더의 구성

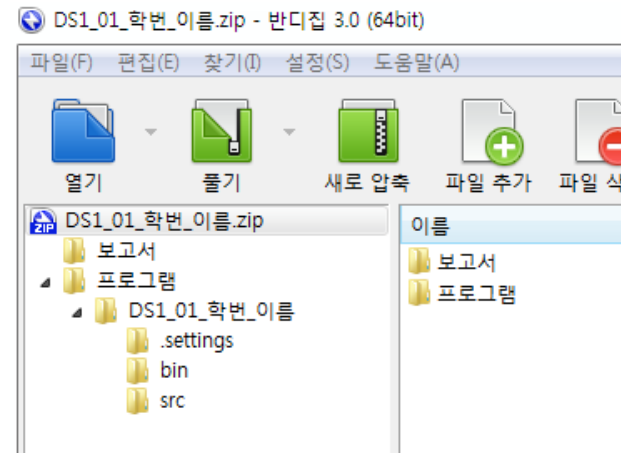
◆ DS2_01_학번_이름

■ 프로그램

- 프로젝트 폴더 / 소스
- 메인 클래스 이름 : DS2_01_학번_이름.java

■ 보고서

- 이곳에 보고서 문서 파일을 저장한다.
- 입력과 실행 결과는 화면 image로 문서에 포함시킨다.
- 문서는 pdf 파일로 만들어 제출한다.



□ 보고서 작성 방법

■ 표지

- 제목: 자료구조 실습 보고서
- [제xx주] 숙제명
- 제출일
- 학번/이름

■ 내용

1. 프로그램 설명서

1. 주요 알고리즘 /자료구조 /기타
2. 함수 설명서
3. 종합 설명서 : 프로그램 사용방법 등을 기술

2. 구현 후 느낀 점 : 요약의 내용을 포함하여 작성한다.

3. 실행 결과 분석

1. 입력과 출력 (화면 capture : 실습예시와 다른 예제로 할 것)
2. 결과 분석

----- 표지 제외한 3번까지의 내용을 A4 세 장 내외의 분량으로 작성 할 것 -----

4. 소스코드 : 화면 capture가 아닌 소스를 붙여넣을 것 소스는 장수 제한이 없음.

[제 1 주 실습] 끝

