[제 6 주 실습]

동등 클래스

강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

Finding Equivalence Classes

문제 개요

- 입력된 동등 관계로부터 동등 클래스를 찾는다.
  - 입력:
    - 원소의 개수 (그래프의 vertex 개수)
    - 동등 관계 쌍들 (그래프에 입력되는 edge 들)
  - 출력:
    - 동등 클래스

동등 관계(relation)의 입력 [1]

S = {홍길동, 임꺽정, 김삿갓, 일지매}

R = {
<홍길동,홍길동>, <임꺽정,임꺽정>, <일지매,일지매>, <김삿갓,김삿갓>,
<홍길동,임꺽정>,<임꺽정,홍길동>,
<홍길동,일지매>, <일지매,홍길동>
<임꺽정,일지매>, <일지매,임꺽정>
}

- S를 먼저 입력 받는다.
- R은? 최소 정보만 입력 받자!
  - Reflexive pair들은 입력하지 않아도 된다
    - S를 아는 상태에서 R의 reflexive pair들은 모두 알아낼 수 있다.
  - Symmetric pair 들은 둘 중의 하나만 입력 받는다.
    - <홍길동,임꺽정>을 알면, 그 symmetric pair인 <임꺽정,홍길동>은 바로 알 수 있다.
  - Transitive pair들은? 프로그램으로 찾아낸다!

❑동등 관계(relation)의 입력 [2]

- R을 입력받기 위해서, 최소 정보만 입력 받자!
  - Reflexive pair들은 입력하지 않아도 된다.

    <홍길동,홍길동>, <임꺽정,임꺽정>, <일지매,일지매>, <김삿갓,김삿갓>,
    <홍길동,임꺽정>,<임꺽정,홍길동>,
    <홍길동,일지매>, <일지매,홍길동>,
    <임꺽정,일지매>, <일지매,임꺽정>,
  - Symmetric pair 들은 둘 중의 하나만 입력 받는다.

    <홍길동,임꺽정>,<임꺽정,홍길동>,
    <홍길동,일지매>, <일지매,홍길동>,
    <임꺽정,일지매>, <일지매,임꺽정>,
  - Transitive pair들은 프로그램으로 찾아낸다!

    <홍길동,임꺽정>,
    <홍길동,일지매>,
    <일지매,임꺽정>,

- 결국, 동등관계 R을 위해서 다음의 두 쌍만 입력 받으면 충분하다.

  <홍길동,임꺽정>, <홍길동,일지매>

동등관계 입력 방법

- 원소의 개수를 입력 받는다.
  - 원소의 개수가 N 이라면, 원소는 0부터 N-1 까지의 번호를 갖는다.
  - 원소의 개수가 0 개 이하이면, 다음과 같은 메시지를 내보내고 프로그램을 종료한다.
    - ">오류: 원소의 개수는 1 개 이상이어야 합니다."

- 그 다음 동등 관계 쌍을 차례로 입력 받는다.
  - 매번 키보드에서, 관계 쌍을 이루는 원소의 번호를 2 개씩 입력 받는다.
  - 원소의 번호는 0부터 N-1 사이의 숫자이어야 하며, 그렇지 않는 경우에는 출력 예에서와 같은 오류 메시지를 내보낸다.
  - 2 개 쌍이 아닌 하나의 원소 번호만 입력되었을 경우에도 출력 예에서와 같은 오류 메시지를 내보낸다.
  - 관계 쌍으로 (-1,-1) 이 입력되면, 관계 쌍 입력을 종료한다.

❑출력의 예

```
< 동등 클래스 찾기를 시작합니다 >
? 원소의 개수를 입력하시오: 6
? 관계 쌍을 입력하시오: 0 3
? 관계 쌍을 입력하시오: 1 2
? 관계 쌍을 입력하시오: 6 4
[오류] 입력된 관계쌍에 오류가 있습니다.
? 관계 쌍을 입력하시오: 1 1
? 관계 쌍을 입력하시오: 2 1
[오류] 입력된 관계쌍에 오류가 있습니다.
? 관계 쌍을 입력하시오: 4 2
? 관계 쌍을 입력하시오: -1 -1
>관계 쌍 입력이 종료되었습니다.
>찾아진 동등 클래스들은 다음과 같습니다.
{ 0 3 }
{ 1 2 4 }
{ 5 }
< 동등 클래스 찾기를 종료합니다 >
```

이 과제에서 필요한 객체는?

- AppController
  - AppView, AdjacencyListGraph, FindingEqvClass
- AppView
- FindEqvClass
  - AdjacencyListGraph, ArrayStack
- AdjacencyListGraph
  - Node, Edge
- ArrayStack
- Node
- Edge

AppView의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public AppView()
  - public void showStartingMsg()
  - public void showEndingMsg()
  - public void showStartingFindMsg()
  - public void showEndingInputMsg()
  - public void showNumOfElementsError()
  - public void showNumError()
  - public void showNoInputError()
  - public int inputNumOfElements()
  - public Edge inputPair()

AppControllor의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public void run()

AdjacencyListGraph의 공개 함수는?

- 사용자에게 필요한 함수
  - [문제5]에서 사용한 것을 복사하여 사용
  - public AdjacencyListGraph(int givenNumOfVertices)
  - public boolean doesVertexExist (int aVertex)
  - public boolean doesEdgeExist (Edge anEdge)
  - public int numOfVertices ()
  - public int numOfEdges ()
  - public boolean addEdge(Edge anEdge)
  - public AdjacentVerticesIterator adjacentVertircesIterator(int givenVertex)
  - public class AdjacentVerticesIterator

Node〈T〉 의 멤버 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - [문제5]에서 사용한 것을 복사하여 사용
  - public Node()
  - public Node(T givenElement)
  - public Node(T givenElement, Node givenNode)

  - public T element()
  - public void setElement(T anElement)

  - public Node next()
  - public void setNext(Node anNode)

FindEqvClass의 공개 함수는?

- 사용자에게 필요한 함수
  - public FindEqvClass(AdjacencyListGraph givenGraph)
  - public void perform()

ArrayStack〈Element〉의 공개 함수는?

- 사용자에게 필요한 함수
  - [문제5]에서 사용한 것을 복사하여 사용
  - public ArrayStack()
  - public ArrayStack(int givenMaxSize)

  - public boolean isEmpty()
  - public boolean isFull()
  - public int length()

  - public boolean push(Element anElement)
  - public Element pop()
  - public Element peek()

  - public ArrayStackIterator arrayStackIterator()
  - public class ArrayStackIterator

Edge의 공개 함수는?

- 사용자에게 필요한 함수
  - [문제5]에서 사용한 것을 복사하여 사용
  - public Edge (int givenTailVertex, int givenHeadVertex)
  - public Edge (int givenTailVertex, int givenHeadVertex, int givenCost)

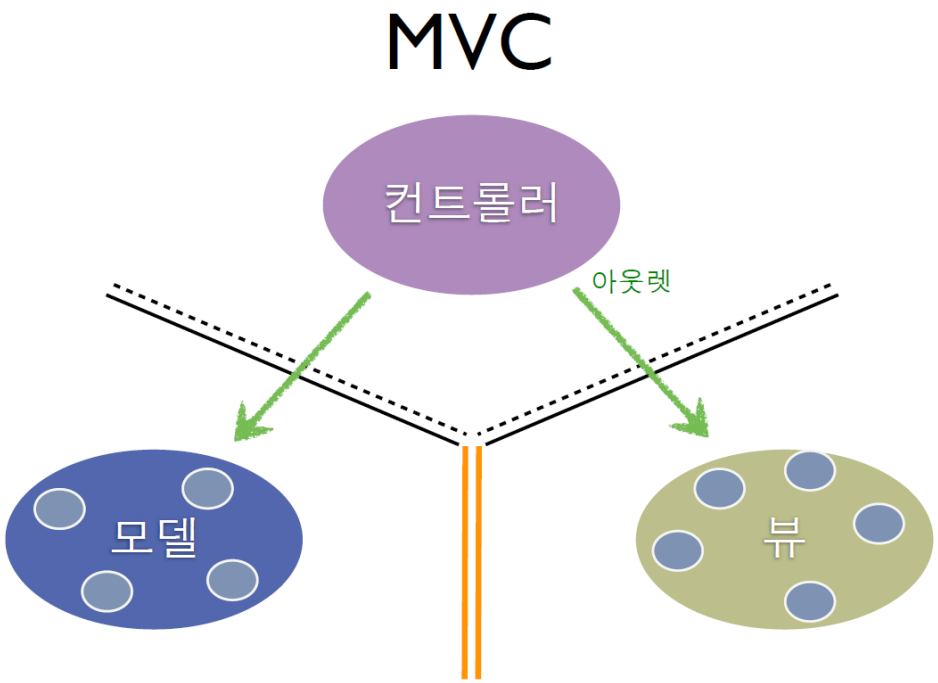
  - public void setTailVertex (int aVertex)
  - public int tailVertex ()
  - public void setHeadVertex (int aVertex)
  - public int headVertex ()
  - public void setCost (int aCost)
  - public int cost()

  - public boolean sameEdgeAs (Edge anEdge)

Class “AppControllor“

AppController

- 기존의 Class "Application"에서 input/output 기능을 제외한 것
- 핵심 기능
  - "모델"로부터 얻은 정보를 "뷰"에게 제공하여 사용자에게 보여준다.
  - "뷰"를 통해 사용자로부터 얻은 입력 정보를 "모델"에게 제공한다.
- 컨트롤러는 모델과 뷰로 직접 접근 가능
  - 컨트롤러가 모델과 뷰 객체의 소유권 보유
- 모델과 뷰는 서로 접근 불가



AppControllor– 비공개 인스턴스 변수

```
public class AppControllor {
    private AdjacencyListGraph  _graph;
    private AppView             _appview;
    private FindEqvClass        _findEqvClass ;
```

Application의 공개 함수 run()의 구현

```
public void run(){
        this._appview = new AppView();

        this._appview.showStartingMsg();
        if ( inputAndMakeGraph() ) {
            this._appview.showEndingInputMsg();
            this._findEqvClass = new FindEqvClass(this._graph);
            // 입력 받은 동등 관계로부터 동등 클래스들을 모두 찾아 출력한다.
            this._appview.showStartingFindMsg();
            this._findEqvClass.perform();
        }
        this._appview.showEndingMsg();
    }
```

Application의 Private Method

- Application의 Private Member function의 사용법
  - private boolean inputAndMakeGraph()
    - 키보드로부터 그래프를 입력 받아 그래프 객체로 저장한다.
    - 그래프가 정상적으로 입력되었으면 true, 아니면 false를 얻는다.
    - [문제5]에서 사용한 것을 수정 (이번 Graph는 방향성이 없음)
  - private boolean endOfPair(Edge anEdge)
    - 입력 받은 값이 입력의 마지막 값인지 확인한다.

Member Functions의 구현

Application의 Private Member function의 구현

private boolean inputAndMakeGraph()

```
private boolean  inputAndMakeGraph()
{
    int numOfVertices ;

    numOfVertices = this._appview.inputNumOfElements();
    if(numOfVertices == 0){
        this._appview.showNumOfElementsError();
        return false;
    }

    this._graph = new AdjacencyListGraph(numOfVertices);

    Edge anEdge = this._appview.inputPair();
    while(!endOfPair(anEdge)) {
        if(!this._graph.addEdge(anEdge))
            this._appview.showNumError();

        anEdge = this._appview.inputPair();
    }

    if(this._graph.numOfEdges() == 0){
        this._appview.showNoInputError();
        return false;
    }

    return true;
}
```

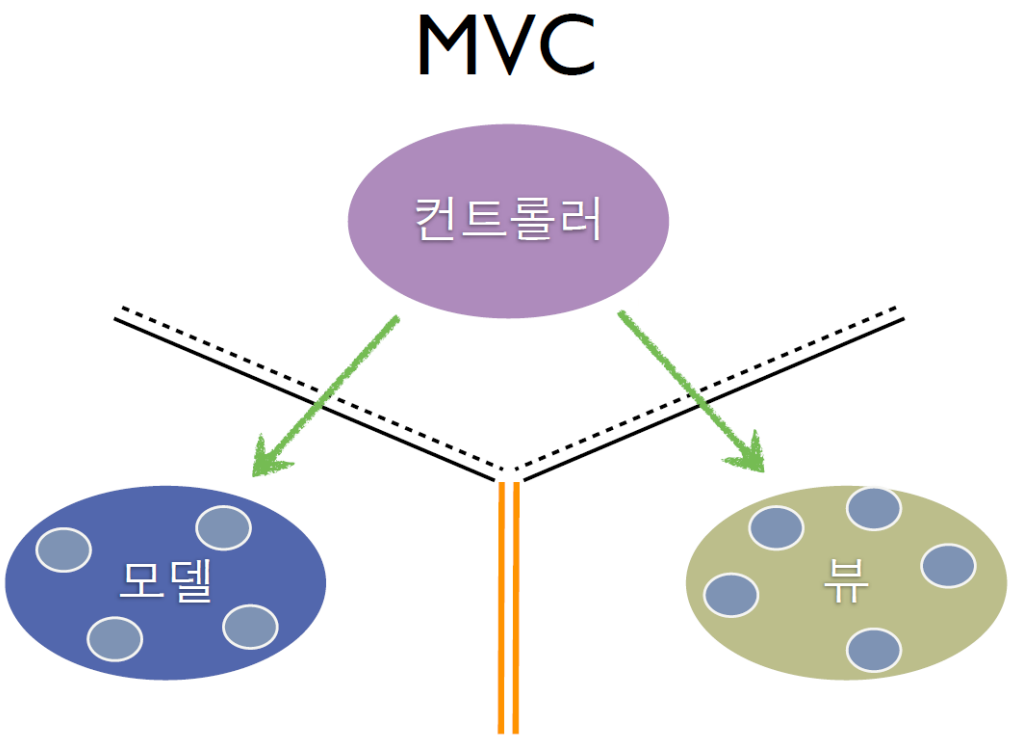
Member Functions의 구현

- Application의 Private Member function의 구현
  - private boolean endOfPair(Edge anEdge)
    - 입력 받은 anEdge의 vertex 들의 값이 -1 값이면 true를 반환

Class “AppView“

AppView

- 기존의 Class “Application”에서 입출력 부분 만을 담당하는 클래스
- 컨트롤러의 하수인
  - 사용자와의 상호작용을 위한 인터페이스



AppView – 비공개 인스턴스 변수

```
import java.util.Scanner;

public class AppView {
    private Scanner _scanner;
```

AppView의 Public Method

- Application의 Public Member function의 사용법과 구현
  - public AppView()
    - 생성자
  - public void showStartingMsg()
    - "< 동등 클래스 찾기를 시작합니다 >" 를 화면에 내보낸다.
  - public void showEndingMsg()
    - "< 동등 클래스 찾기를 종료합니다 >" 를 화면에 내보낸다.
  - public void showStartingFindMsg()
    - ">찾아진 동등 클래스들은 다음과 같습니다." 를 화면에 내보낸다.
  - public void showEndingInputMsg()
    - ">관계 쌍 입력이 종료되었습니다." 를 화면에 내보낸다.

AppView의 Public Method

- Application의 Public Member function의 사용법과 구현
  - public void showNumOfElementsError()
    - "[오류] 입력된 원소의 개수에 오류가 있습니다." 를 화면에 내보낸다.
  - public void showNumError()
    - "[오류] 입력된 관계쌍에 오류가 있습니다." 를 화면에 내보낸다.
  - public void showNoInputError()
    - "[오류] 입력된 관계쌍이 없습니다." 를 화면에 내보낸다.

AppView의 Public Method

- Application의 Public Member function의 사용법과 구현
  - public int inputNumOfElements()
    - "? 원소의 개수를 입력하시오: " 를 화면에 내보낸다.
    - 원소 개수 하나를 입력 받아 반환한다.
  - public Edge inputPair()
    - "? 관계 쌍을 입력하시오: " 를 화면에 내보낸다.
    - 관계 쌍 두개를 입력 받아 새로운 Edge를 생성하여 반환한다.

Class “AdjacencyListGraph“

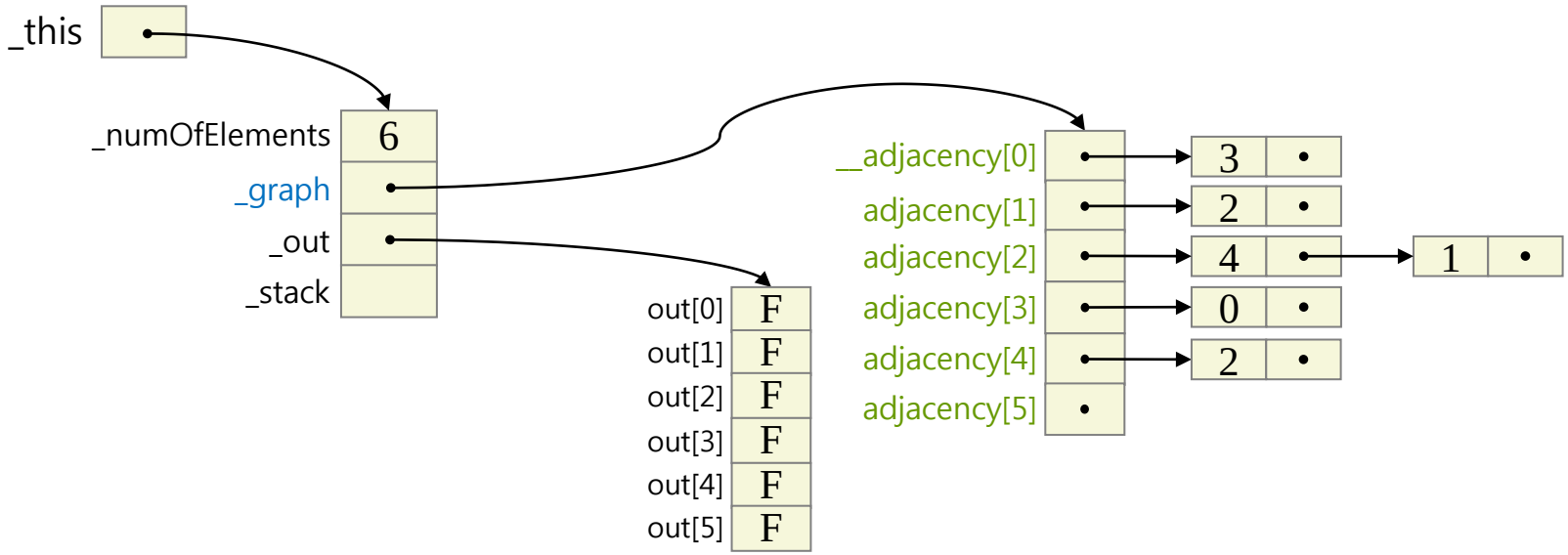
AdjacencyListGraph의 수정

- [문제5] 에서 사용한 것을 방향성 없는 그래프로 수정
  - public boolean addEdge(Edge anEdge)
    - 먼저 anEdge를 그래프에 삽입한다.
    - 방향성이 없는 Graph이므로, 입력 받은 anEdge의 headVertex를 tailVertex로, tailVertex를 headVertex로 갖는 reverseEdge를 새로 생성한다.
    - reverseEdge를 삽입한다.

Class “FindEqvClass”

FindEqvClass 비공개 인스턴스 변수

```
public class FindEqvClass {
    private int                 _numOfElements; // 원소의 개수
    private AdjacencyListGraph  _graph;
    private boolean []          _out;
    private ArrayStack<Integer> _stack;
```



FindEqvClass 의 구현

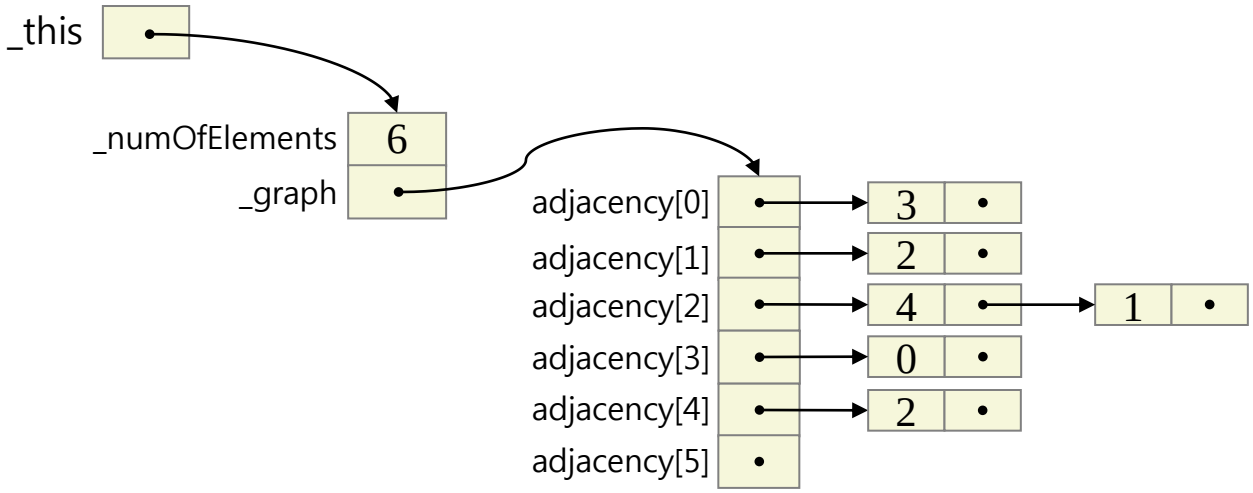
동등관계의 입력과 표현

동등 관계의 표현 [1]

- 예: 입력을 통해 주어진 동등관계
  - 원소의 집합: {0, 1, 2, 3, 4, 5}
  - 동등관계 쌍: <0,3>, <1,2>, <1,1>, <4,2>
- 원소의 집합
  - 편의상 0부터 시작하는 정수의 집합으로 나타내기로 한다.
  - 그러므로 그 개수만 알면 된다.
    - 예제의 경우 **6** 만 알면 된다.
- 동등관계 쌍
  - Reflexive pair: 따로 표현하지 않는다.
  - Symmetric pair: <a,b>가 입력되면, <b,a>도 함께 입력된 것으로 보고, 둘 다 표현한다.
  - Transitive pair: 프로그램에서 필요한 시점에 모두 찾아내며, 입력 단계에서는 표현하지 않는다.

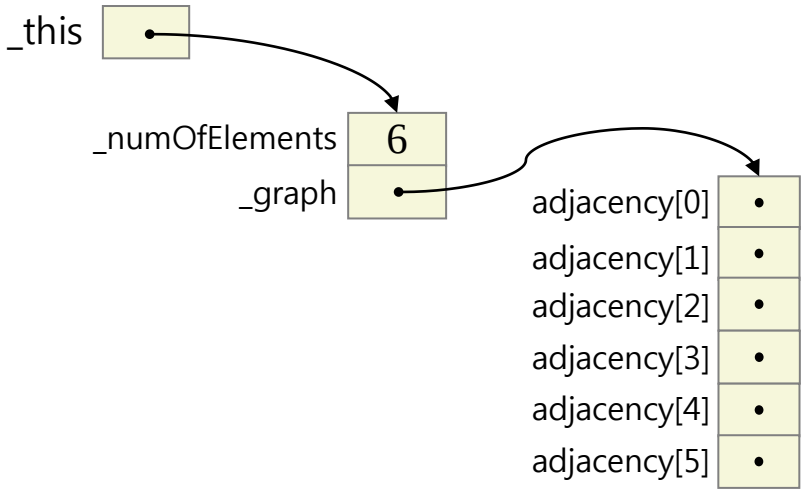
동등 관계의 표현 [2]

- 연결리스트를 사용하자.
  - 동일한 번호로 시작하는 쌍은 모두 하나의 연결리스트에 넣는다.
- 예제:
  - 입력: <0,3>, <1,2>, <1,1>, <4,2>



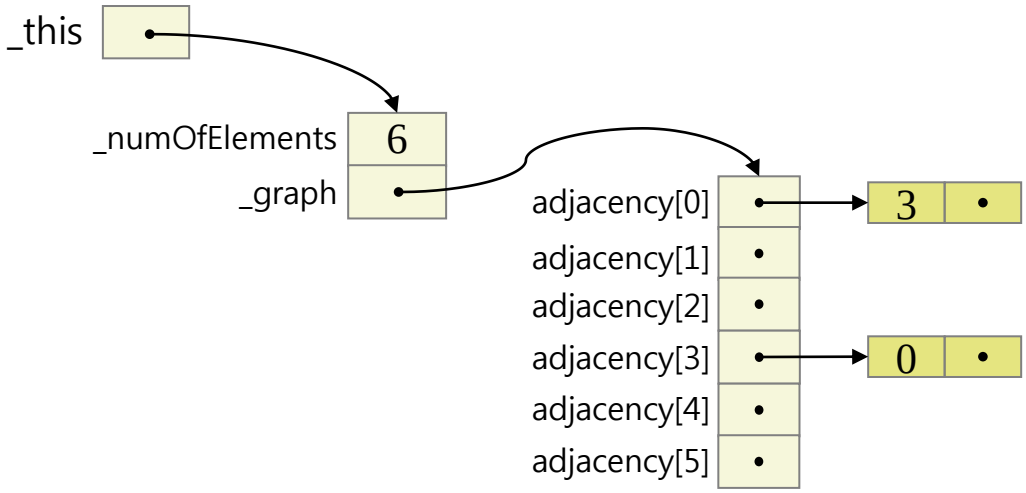
관계 쌍 입력 처리 [O]

관계 쌍 입력 시작 전 상태



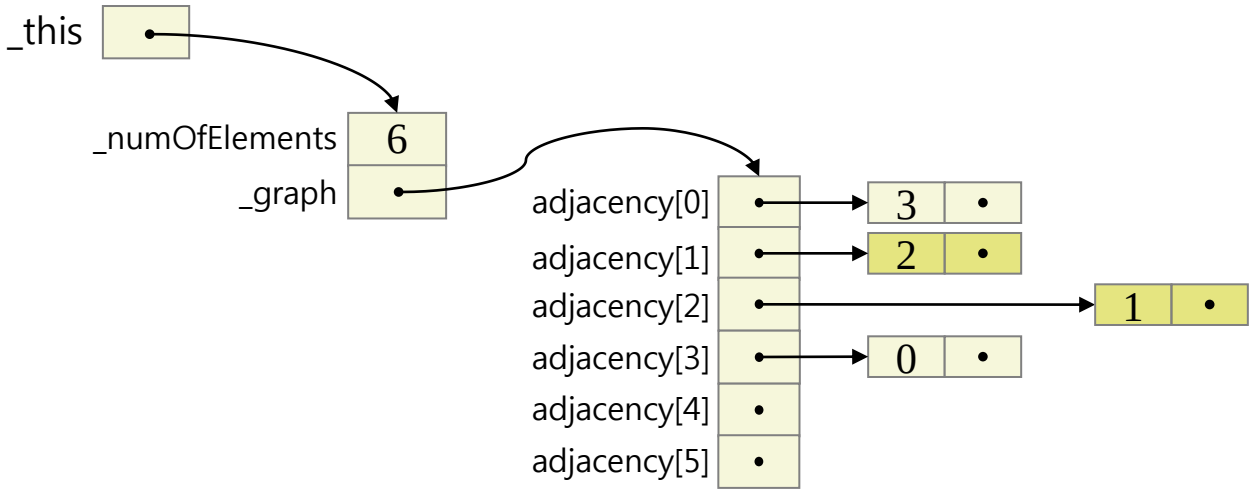
관계 쌍 입력 처리 [1]

- 관계 쌍 <0,3> 입력



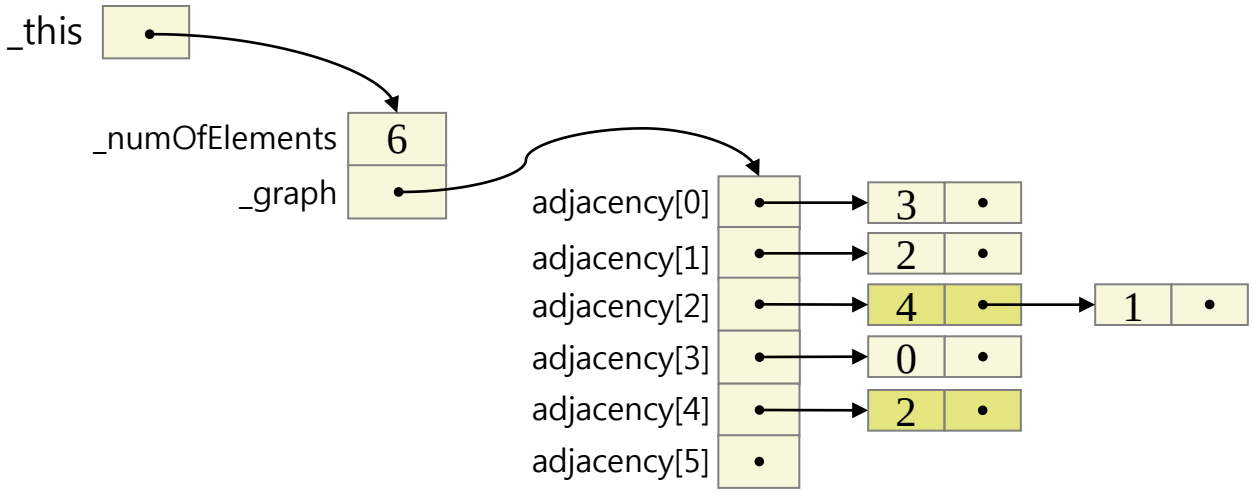
관계 쌍 입력 처리 [2]

- 관계 쌍 <1,2> 입력



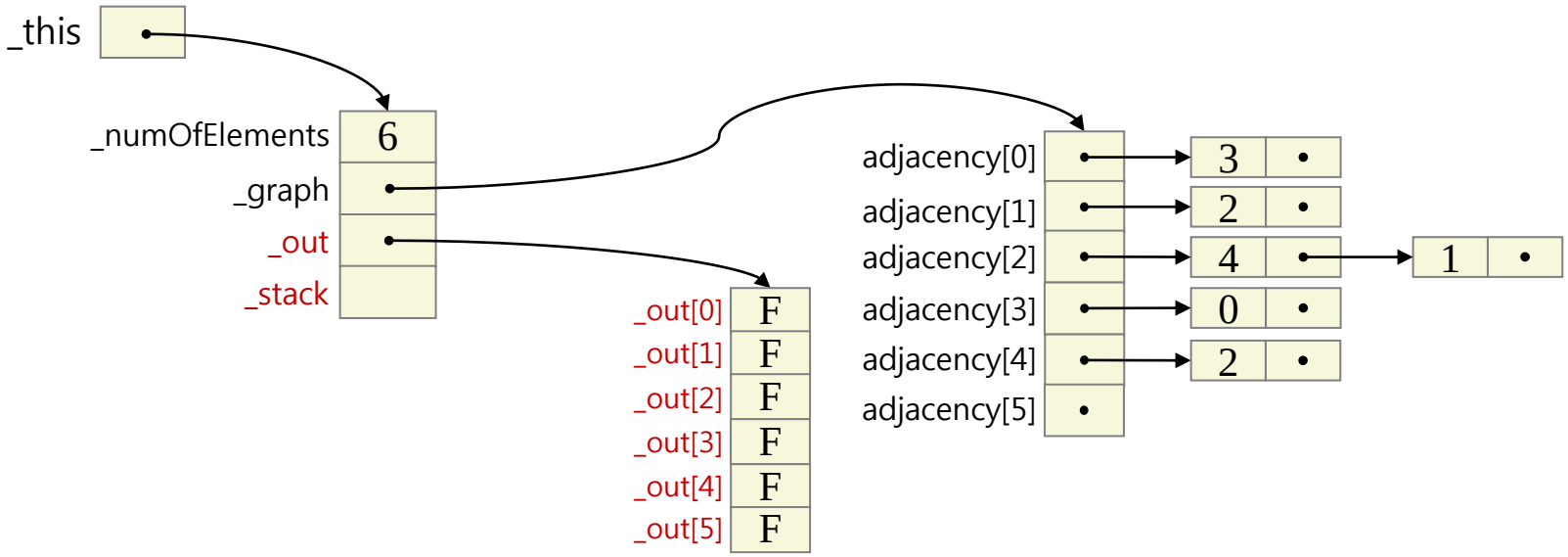
관계 쌍 입력 처리 [2]

- 관계 쌍 <1,1> 입력
  - 처리할 것 없음
- 관계 쌍 <4,2> 입력



동등 클래스 찾기 [1]

- 현재까지 동등클래스가 찾아진 원소들이 어느 것인지를 표현하자.



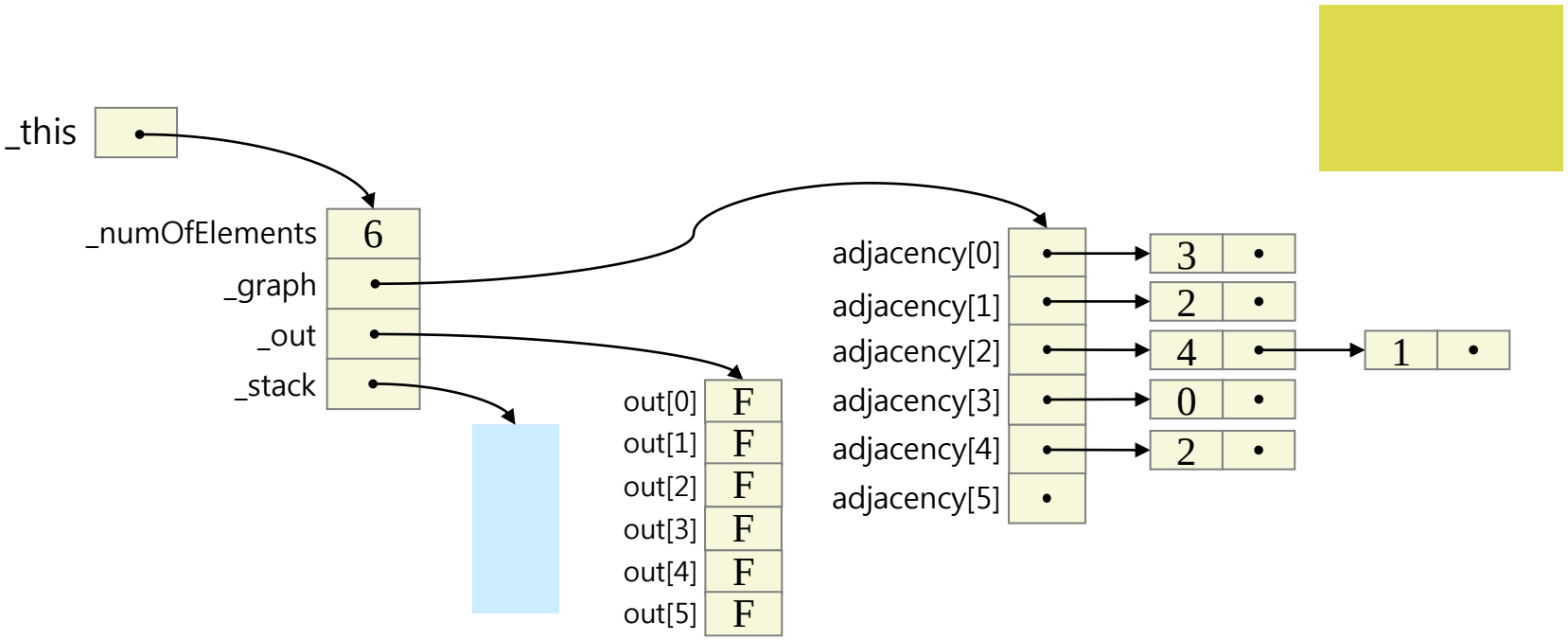
- boolean 타입의 배열 _out[]을 사용하자.
  - _out[]은 맨 처음에 모두 False 이다.
  - 매번 동등 클래스가 찾아진 원소 x에 대해, 그 _out[x] 값을 True 로 바꾼다.
  - 전부 True 가 되면 모든 원소의 동등 클래스를 찾은 것이다. 프로그램을 종료한다.

동등 클래스 찾기 [2]

- 한 원소와 동등관계 있는 모든 원소 찾기
  - Transitivity를 적용하여 동등관계가 성립하는 모든 원소를 찾아야 한다.
  - 시작 원소 x 결정하기
    - 먼저 _out[] 중에서 아직 동등 클래스가 찾아지지 않은 원소 하나를 얻는다.
  - 원소 x로부터 출발하여, s와 동등관계가 성립하는 모든 원소를 찾는다.
    - _adjacency[x]에 연결되어 있는 원소들은 모두 x와 동등 관계가 성립한다.
    - _adjacency[x]에 연결되어 있는 원소 하나 하나에 대해,
      - 그 중 하나가 y 라면, 다시 adjacency[y]에 연결되어 있는 모든 원소는 역시 x와 동등관계가 성립한다.
    - 이러한 방식으로 연결되어 있는 모든 원소는 x와 동등관계가 성립한다.
  - 스택을 이용한다.

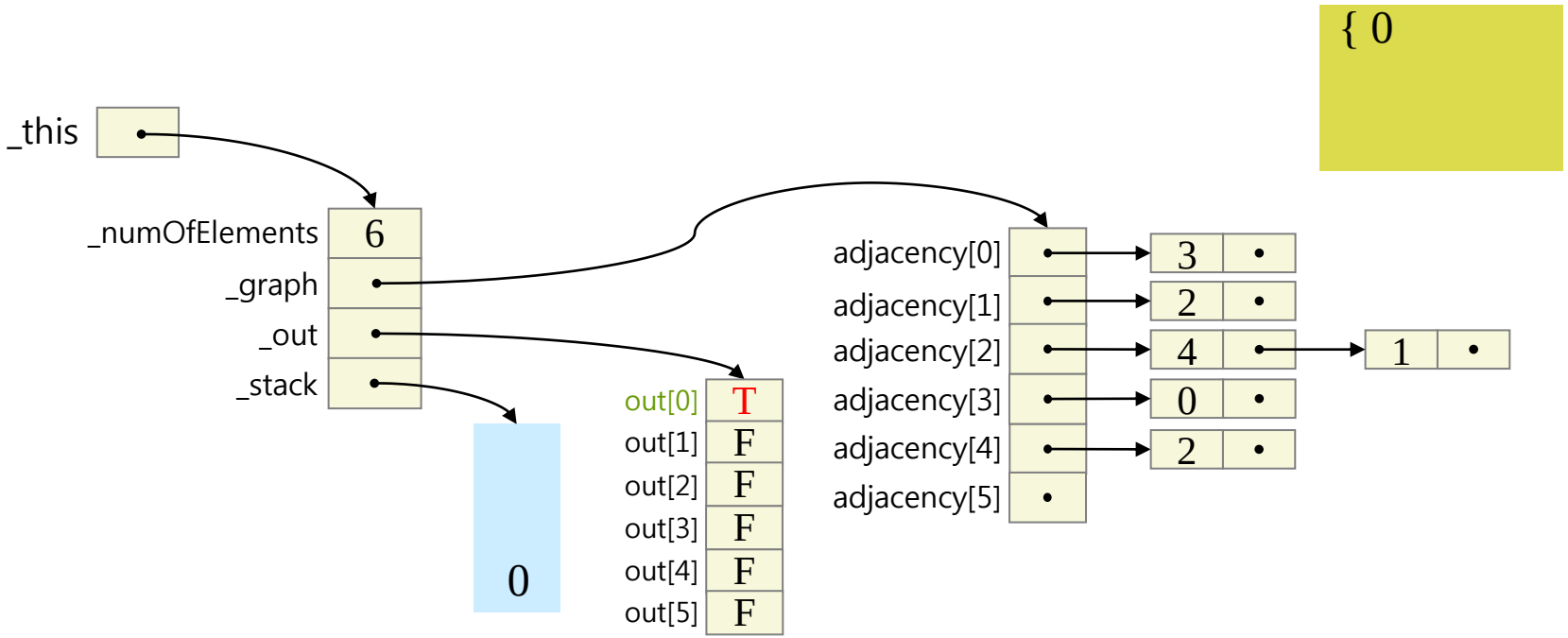
동등 클래스 찾기: 과정 [0]

- 찾기 시작 전 상태



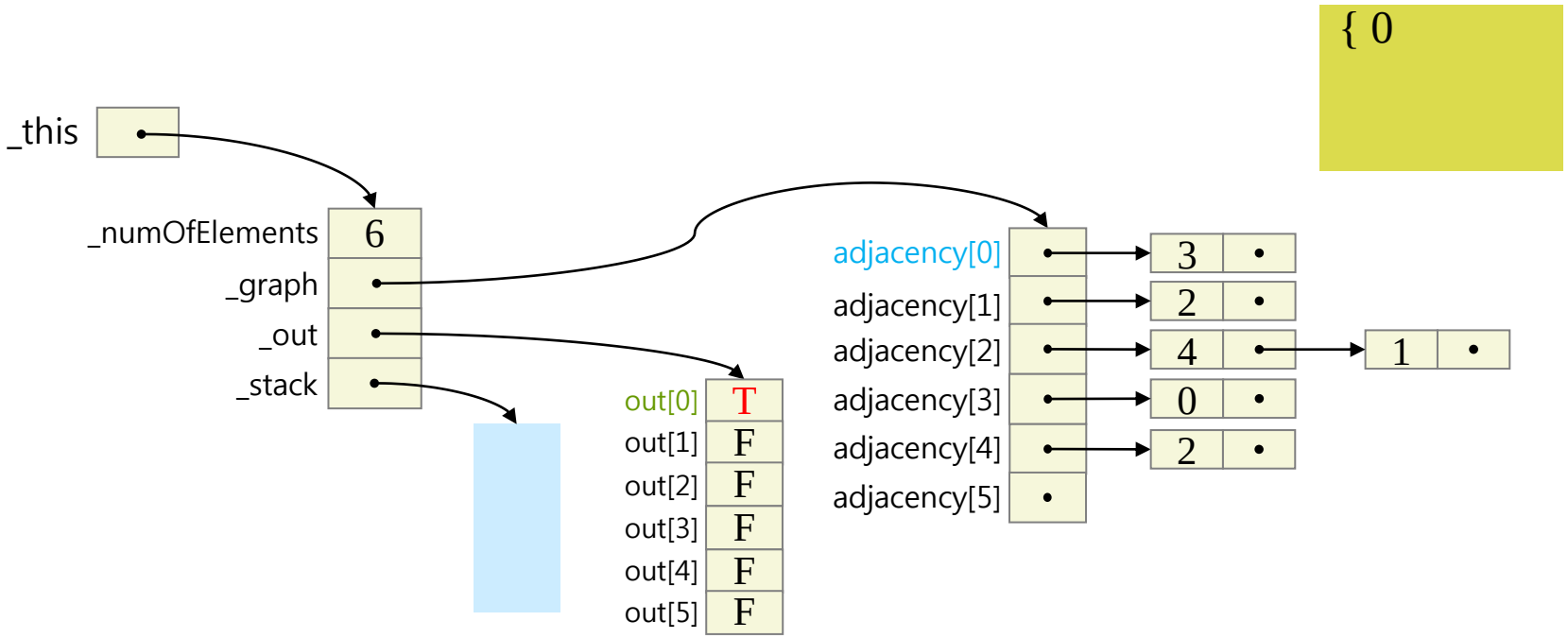
동등 클래스 찾기: 과정 [1]

- i = 0
  - out[0] 는 FALSE; 원소 0의 동등 클래스를 찾기 시작한다;
  - 원소 0 를 출력; out[0]는 TRUE로 바뀐다;
  - 원소 0 를 스택에 삽입;



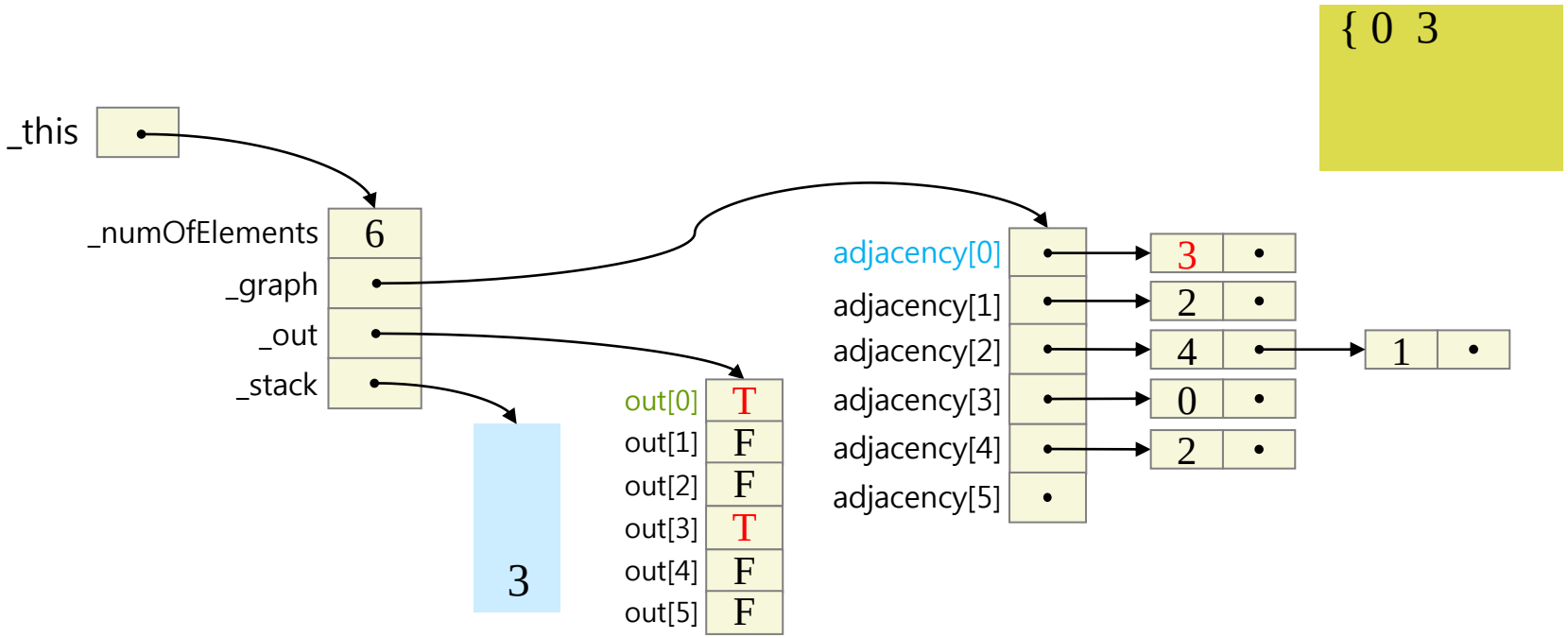
동등 클래스 찾기: 과정 [2]

- i = 0
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 0;
  - Linked list인 adjacency[0]를 scan:



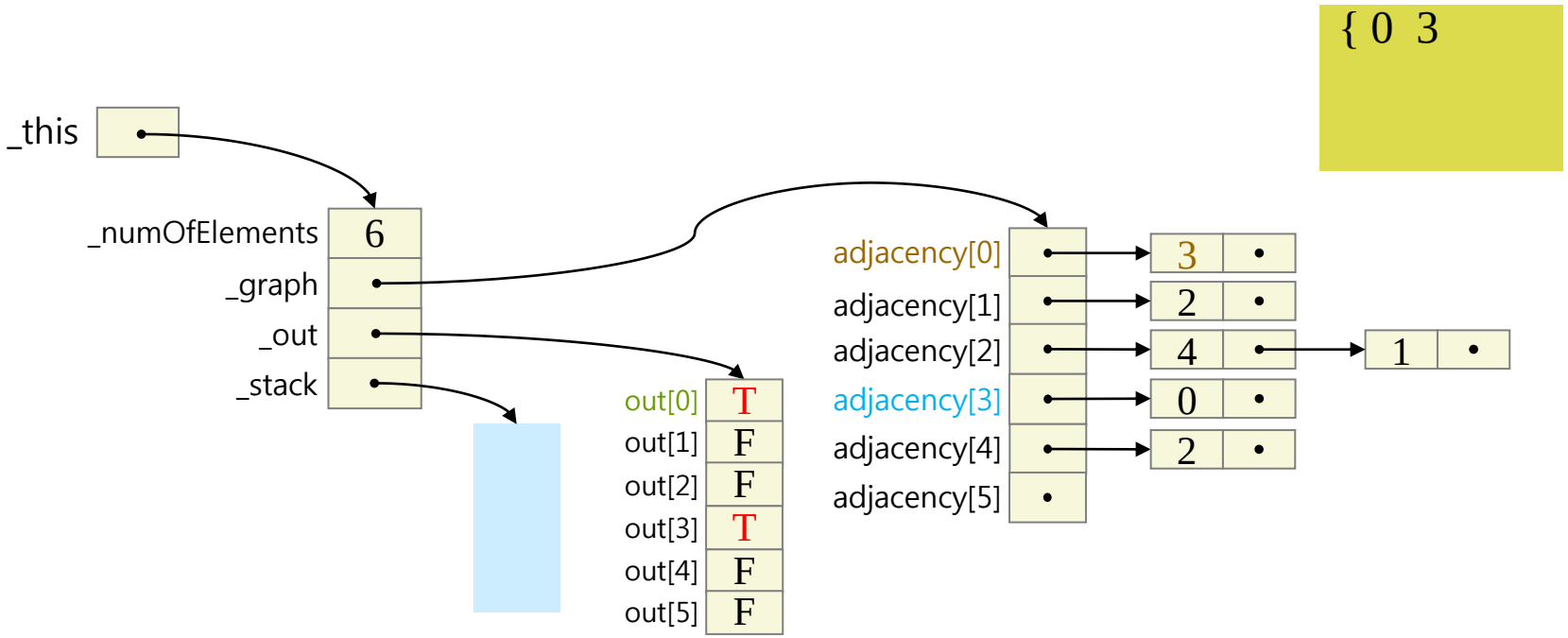
동등 클래스 찾기: 과정 [3]

- i = 0
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 0;
  - Linked list인 adjacency[0]를 scan:
    - 첫번째 원소 = 3 ;
      out[3]가 FALSE ; 원소 3 출력; out[3]는 TRUE 로; stack에 삽입;
    - 더 이상 원소가 없으므로 scan 끝;



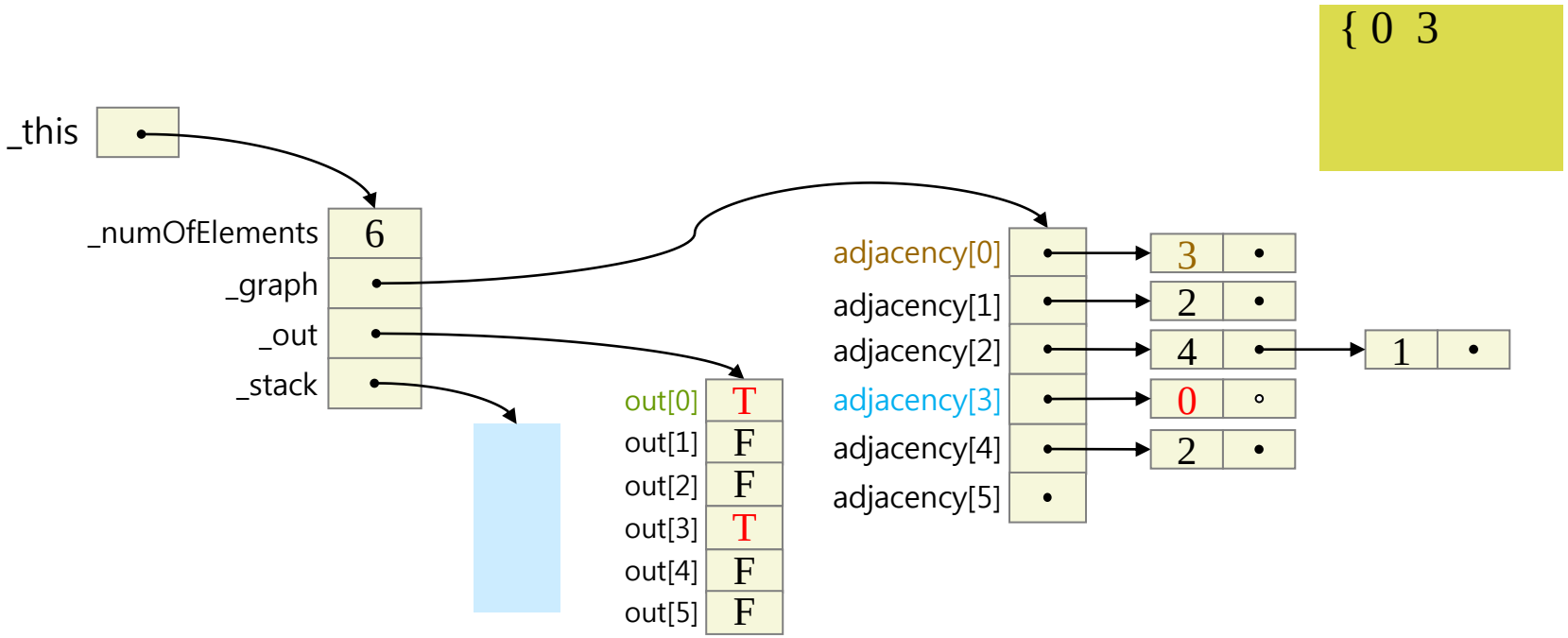
동등 클래스 찾기: 과정 [4]

- i = 0
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 3;
  - Linked list인 adjacency[3]를 scan:



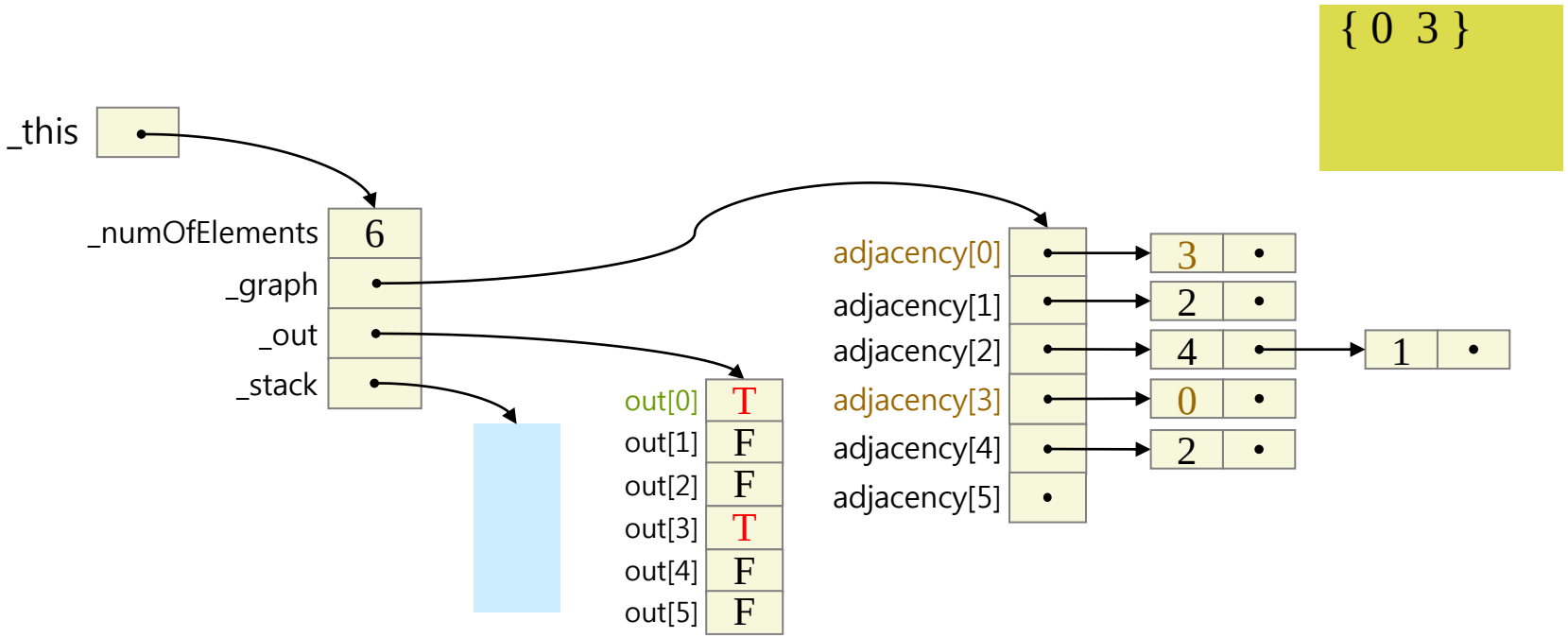
동등 클래스 찾기: 과정 [5]

- i = 0
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 3;
  - Linked list인 adjacency[3]를 scan:
    - 첫번째 원소 = 0 ;
      out[3]은 TRUE ; 이미 출력된 원소이므로 아무 일도 하지 않는다;
    - 더 이상 원소가 없으므로 scan 끝;



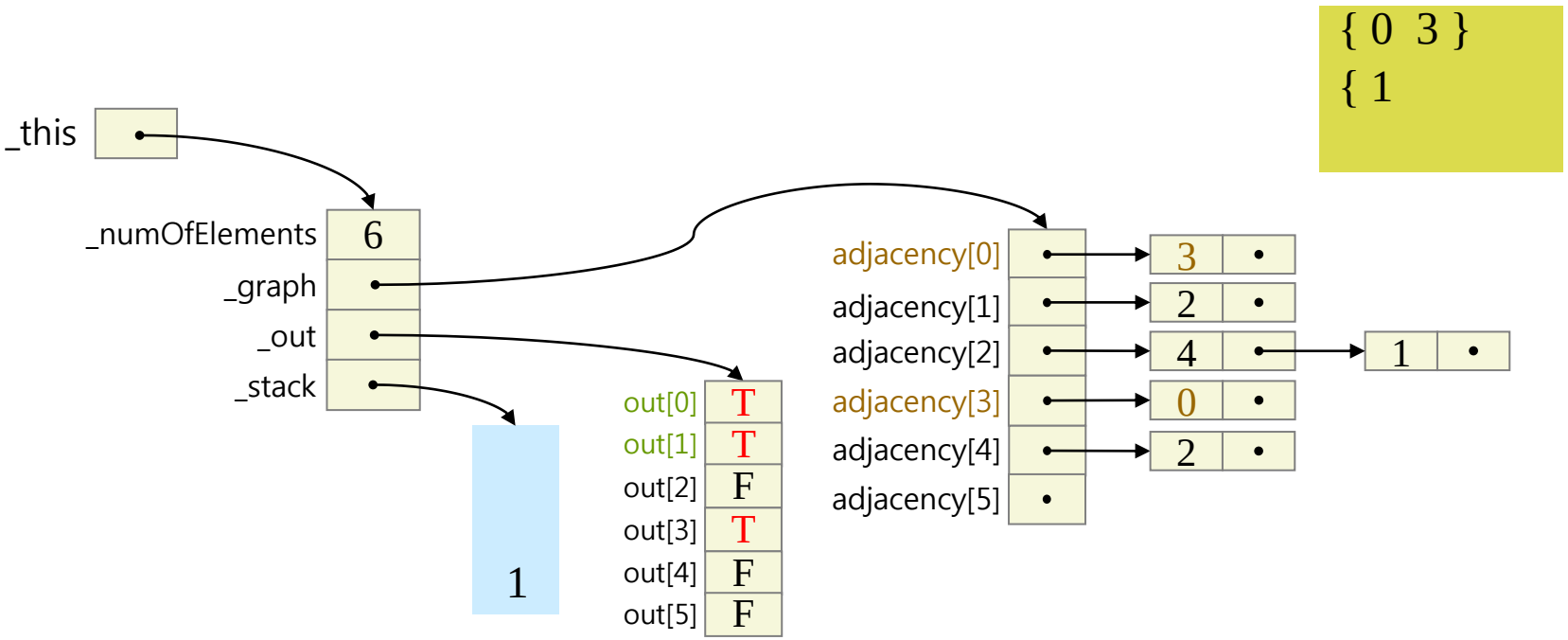
동등 클래스 찾기: 과정 [6]

- i = 0
  - 스택이 empty:
    - 원소 0의 동등 클래스를 다 찾았다;



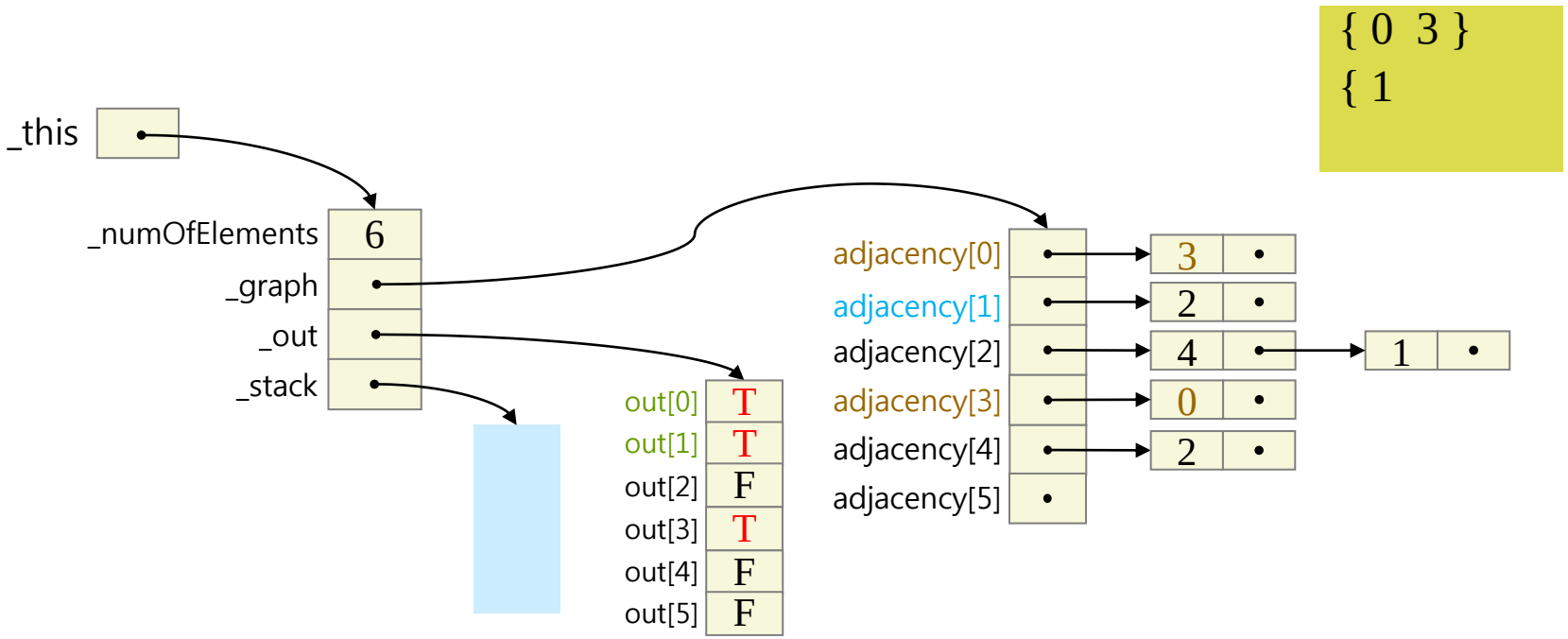
동등 클래스 찾기: 과정 [7]

- i = 1
  - out[1] 은 FALSE; 원소 1의 동등 클래스를 찾기 시작한다;
  - 원소 1 을 출력; out[1]는 TRUE로 바뀐다;
  - 원소 1 을 스택에 삽입;



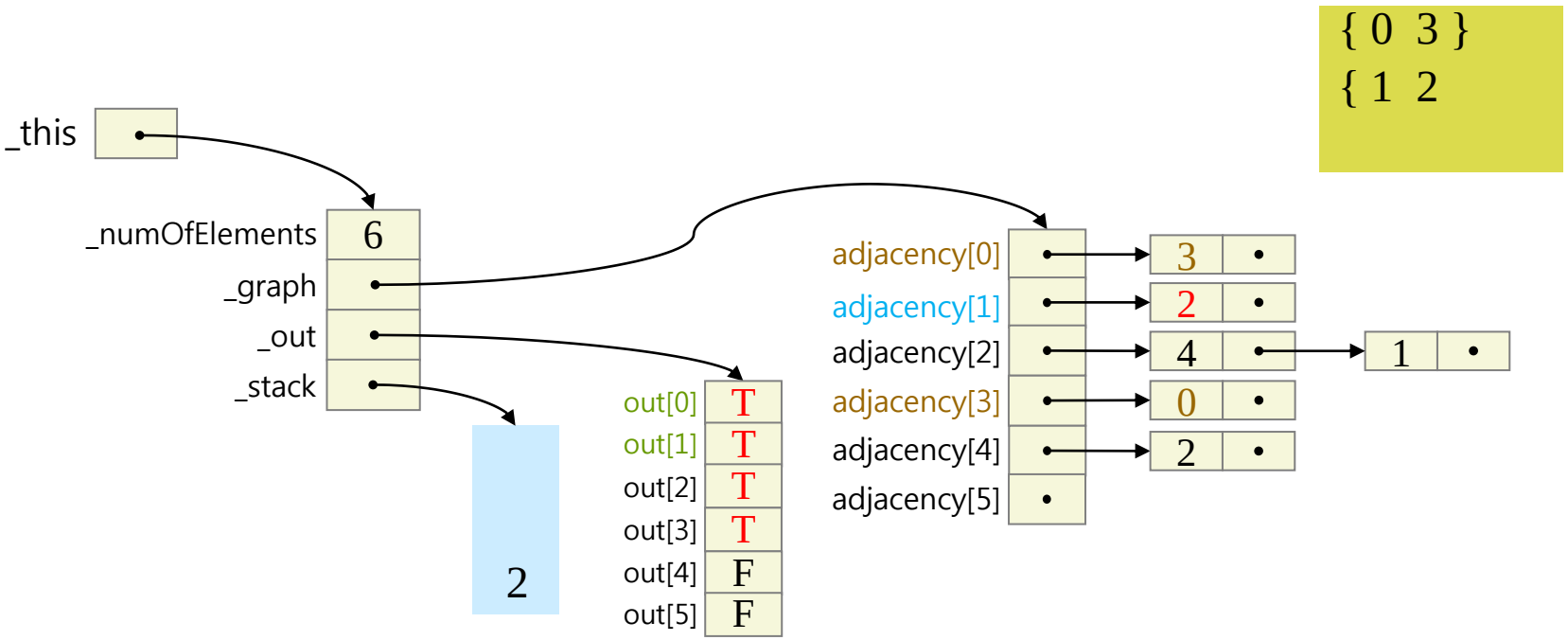
동등 클래스 찾기: 과정 [8]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 1;
  - Linked list인 adjacency[1]를 scan:



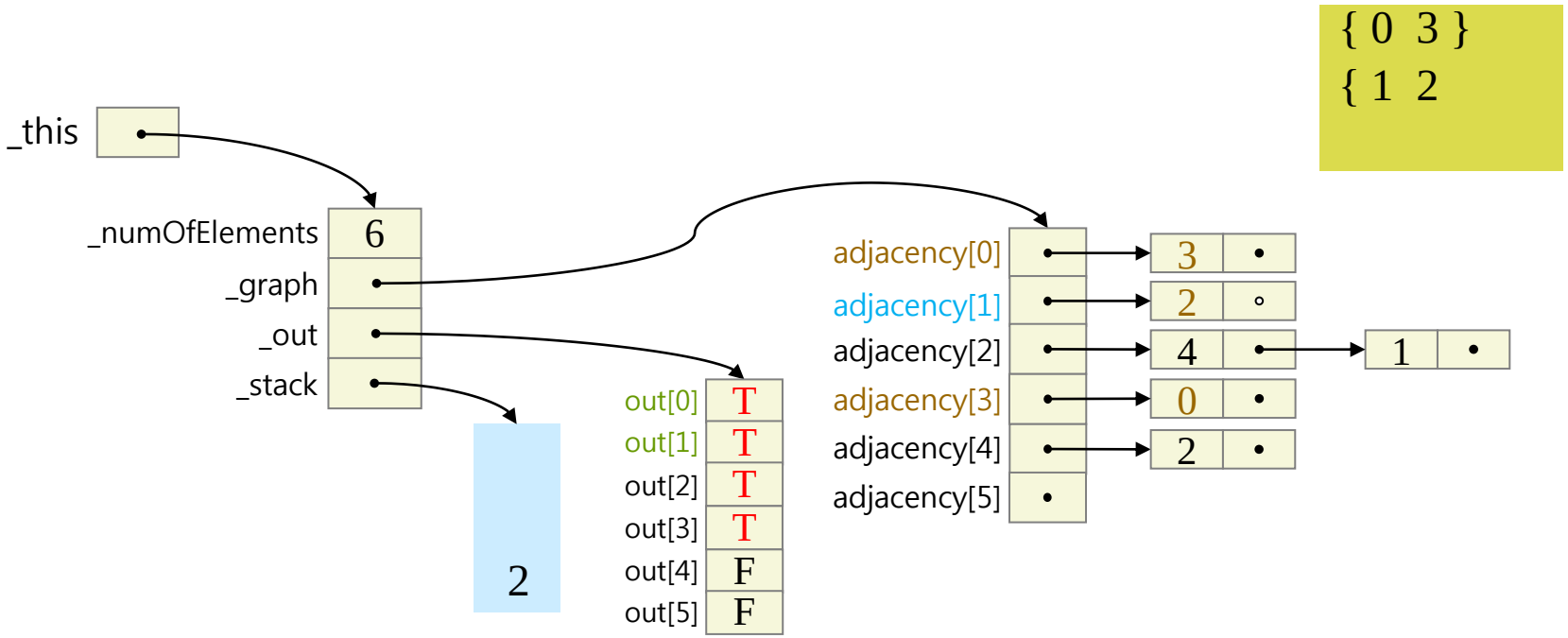
동등 클래스 찾기: 과정 [9]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 1;
  - Linked list인 adjacency[1]를 scan:
    - 첫번째 원소 = 2 ;
      out[2]는 FALSE ; 원소 2 출력; out[2]는 TRUE 로; stack에 삽입;



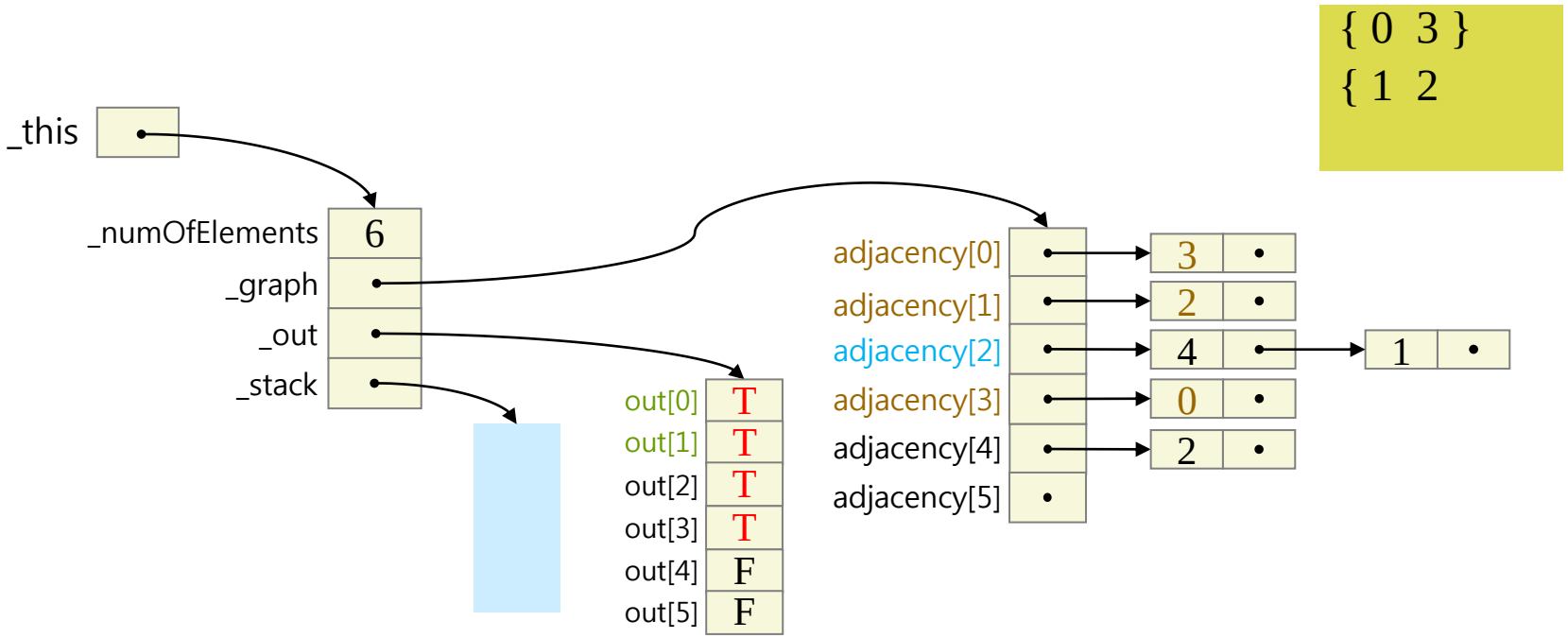
동등 클래스 찾기: 과정 [10]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 1;
  - Linked list인 adjacency[1]를 scan:
    - 첫번째 원소 = 2 ;
      out[2]는 FALSE ; 원소 2 출력; out[2]는 TRUE 로; stack에 삽입;
    - 더 이상 원소가 없으므로 scan 끝;



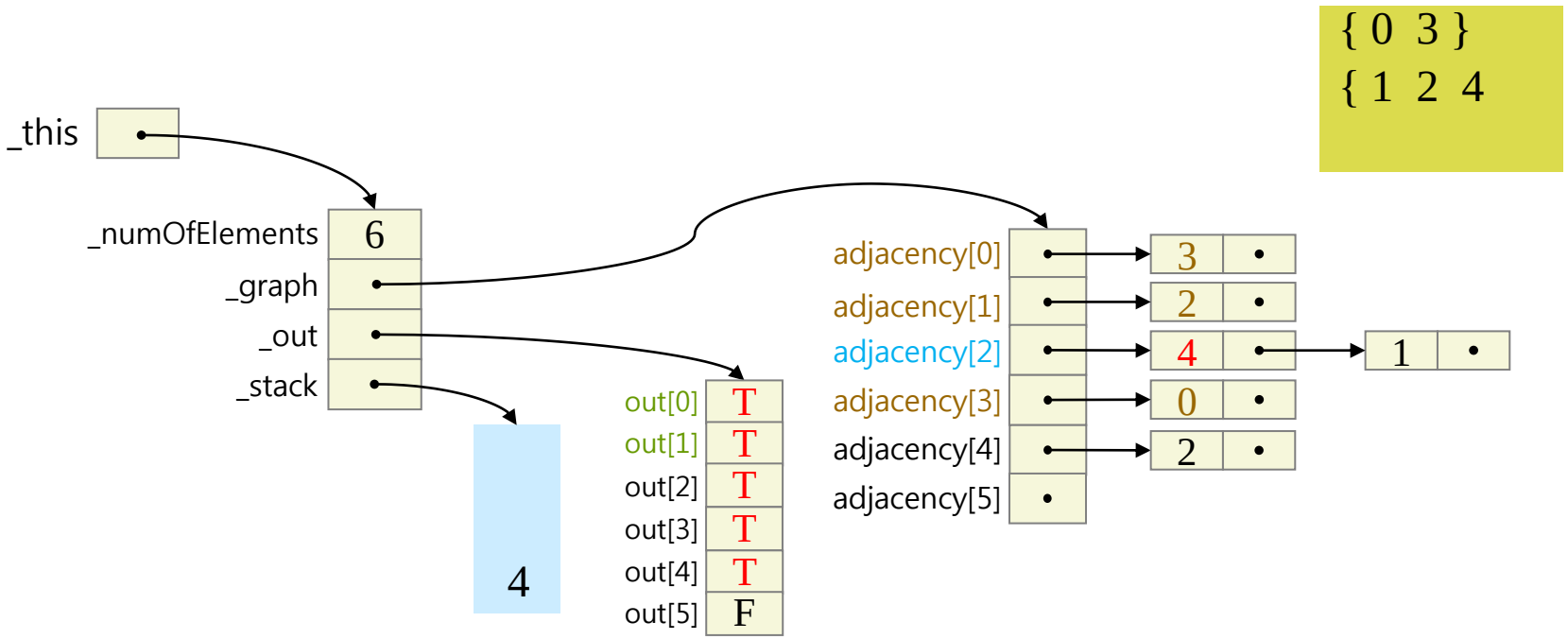
동등 클래스 찾기: 과정 [11]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 2;
  - Linked list인 adjacency[2]를 scan:



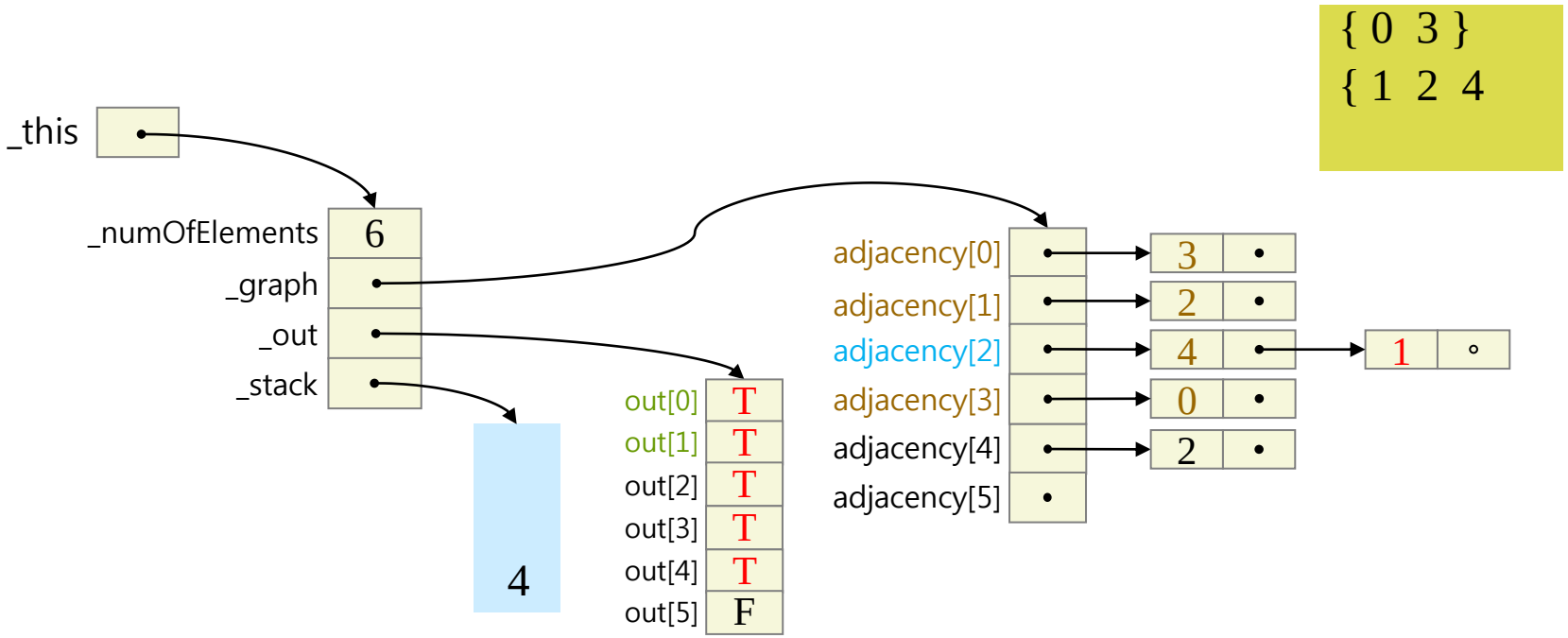
동등 클래스 찾기: 과정 [12]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 2;
  - Linked list인 adjacency[2]를 scan:
    - 첫번째 원소 = 4;
      out[4]가 FALSE ; 원소 4 출력; out[4]는 TRUE 로; stack에 삽입;



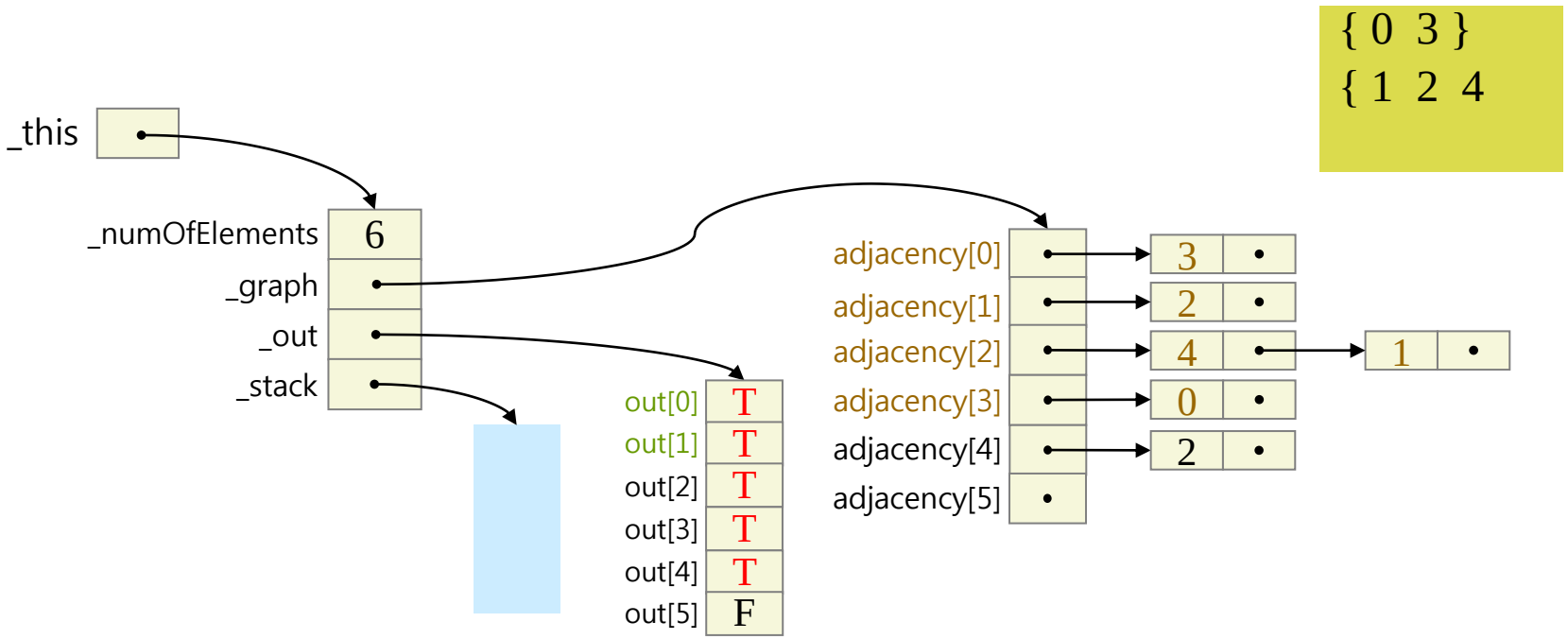
동등 클래스 찾기: 과정 [13]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 2;
  - Linked list인 adjacency[2]를 scan:
    - 두번째 원소 = 1;
      out[1]이 TRUE ; 이미 출력된 원소이므로 아무 일도 하지 않는다;
    - 더 이상 원소가 없으므로 scan 끝;



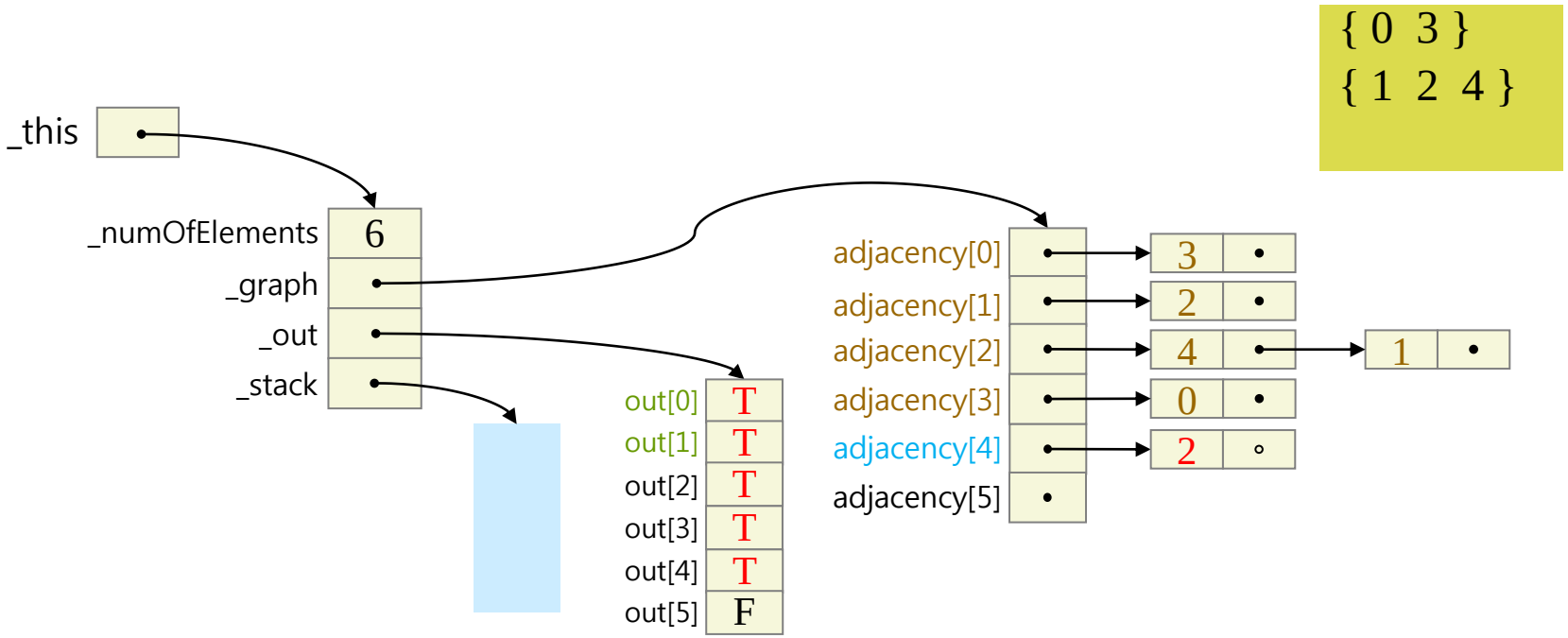
동등 클래스 찾기: 과정 [14]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 4;



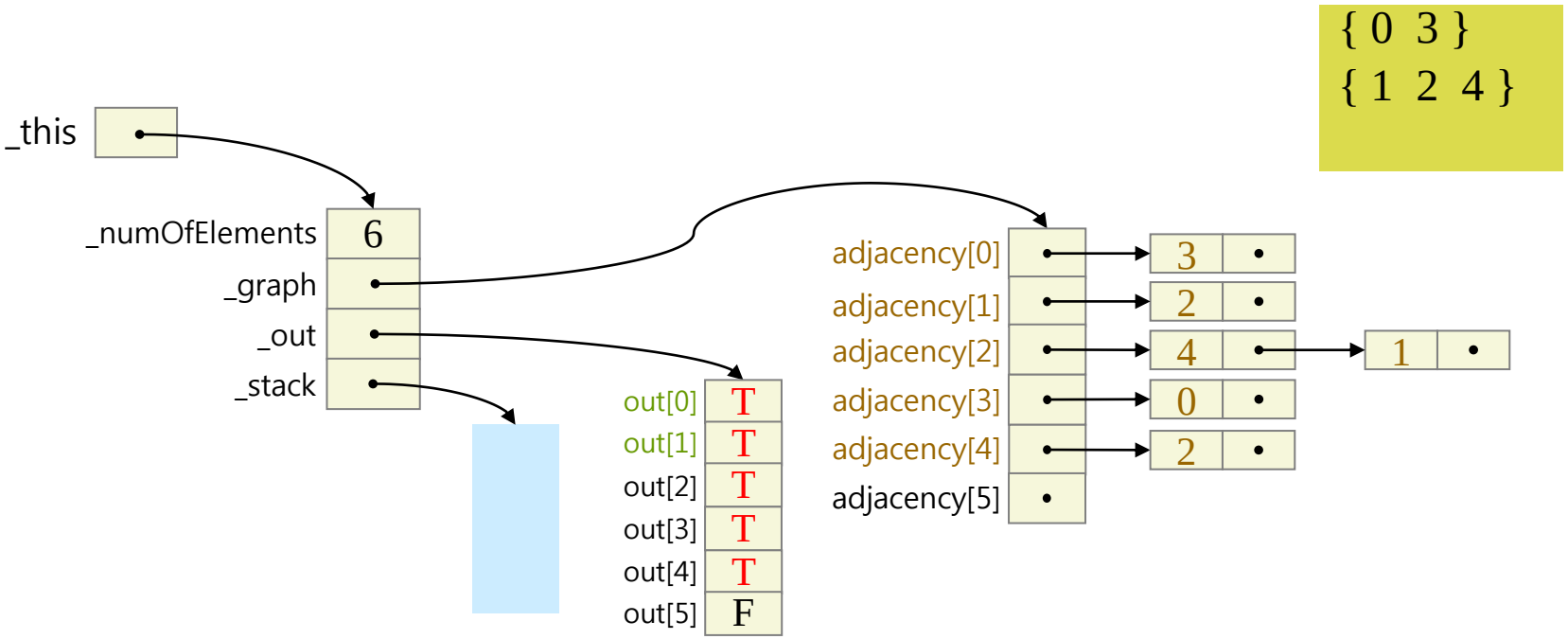
동등 클래스 찾기: 과정 [15]

- i = 1
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 4;
  - Linked list인 adjacency[4]를 scan:
    - 첫번째 원소 = 2;
      out[2]는 TRUE ; 이미 출력된 원소이므로 아무 일도 하지 않는다;
    - 더 이상 원소가 없으므로 scan 끝;



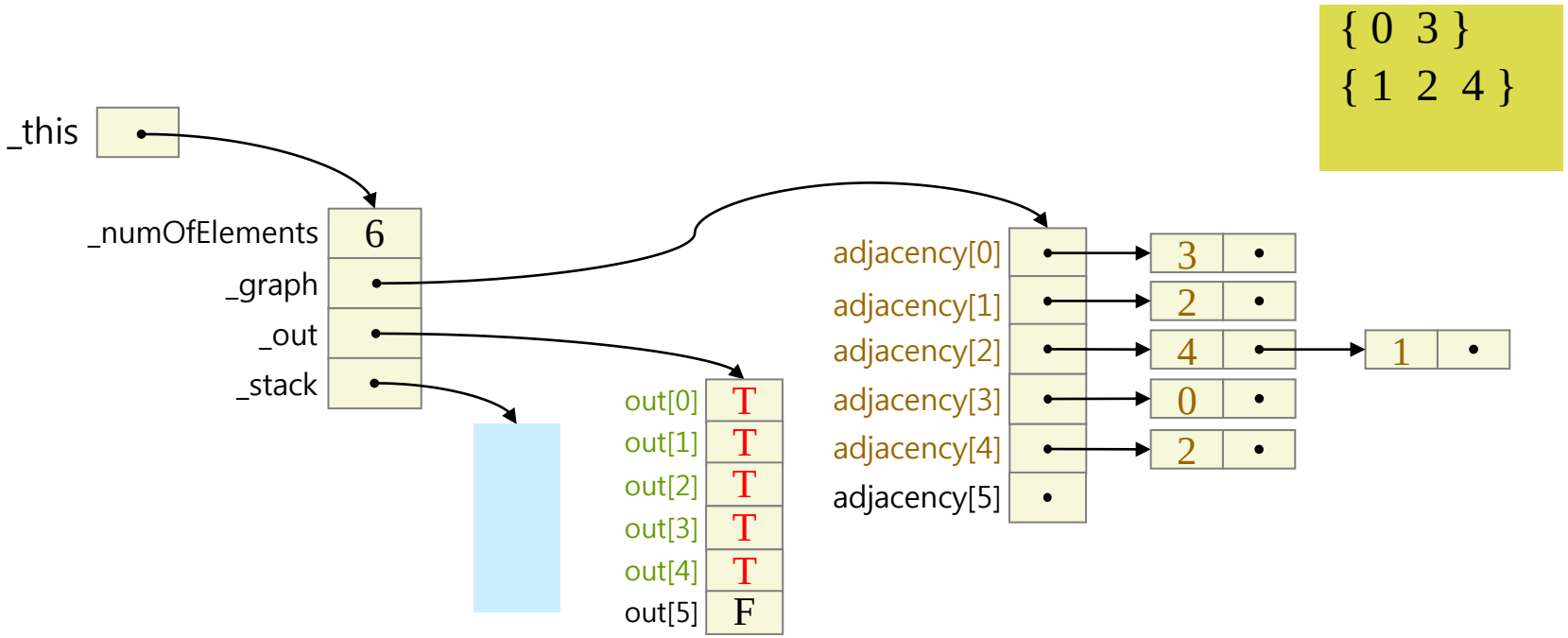
동등 클래스 찾기: 과정 [16]

- i = 1
  - 스택이 empty:
    - 원소 1 의 동등 클래스를 다 찾았다;



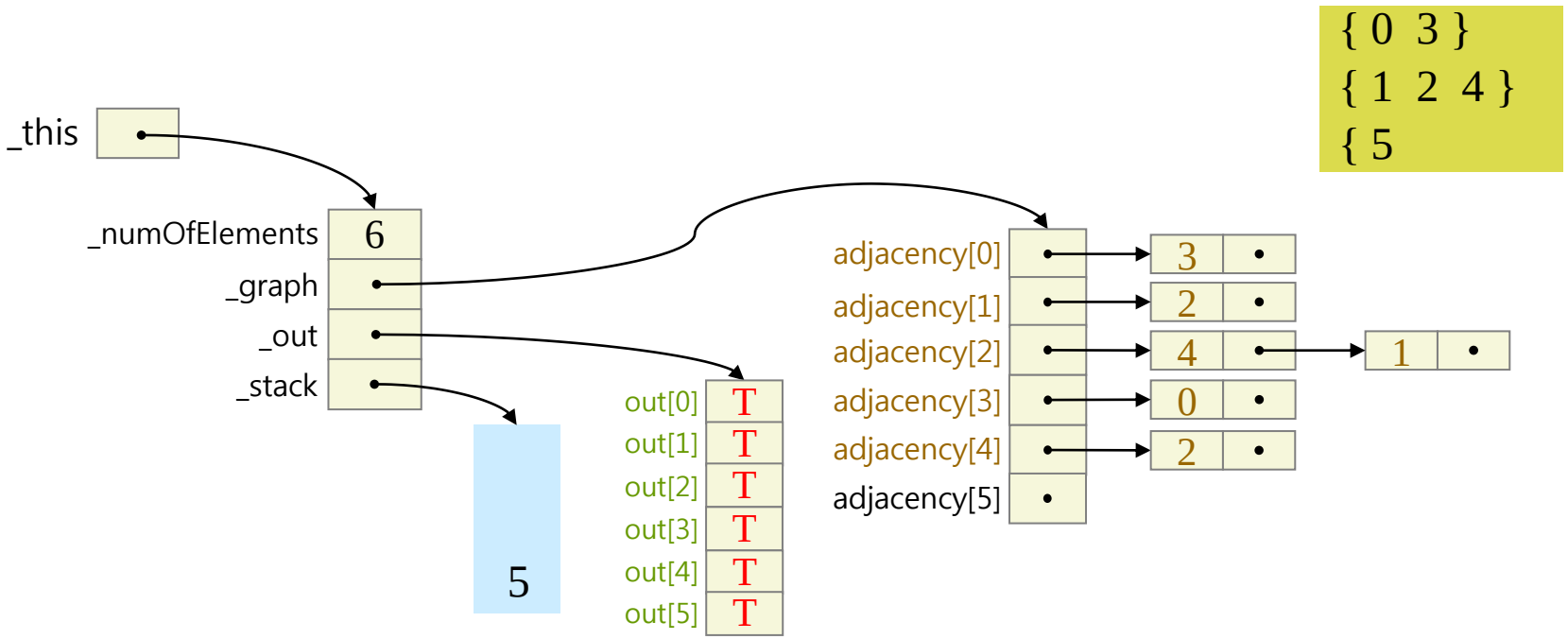
동등 클래스 찾기: 과정 [17]

- i = 2
  - out[2] 는 TRUE; 원소 2의 동등 클래스는 이미 찾았다;
- i = 3
  - out[3] 는 TRUE; 원소 3의 동등 클래스는 이미 찾았다;
- i = 4
  - out[4] 는 TRUE; 원소 4의 동등 클래스는 이미 찾았다;



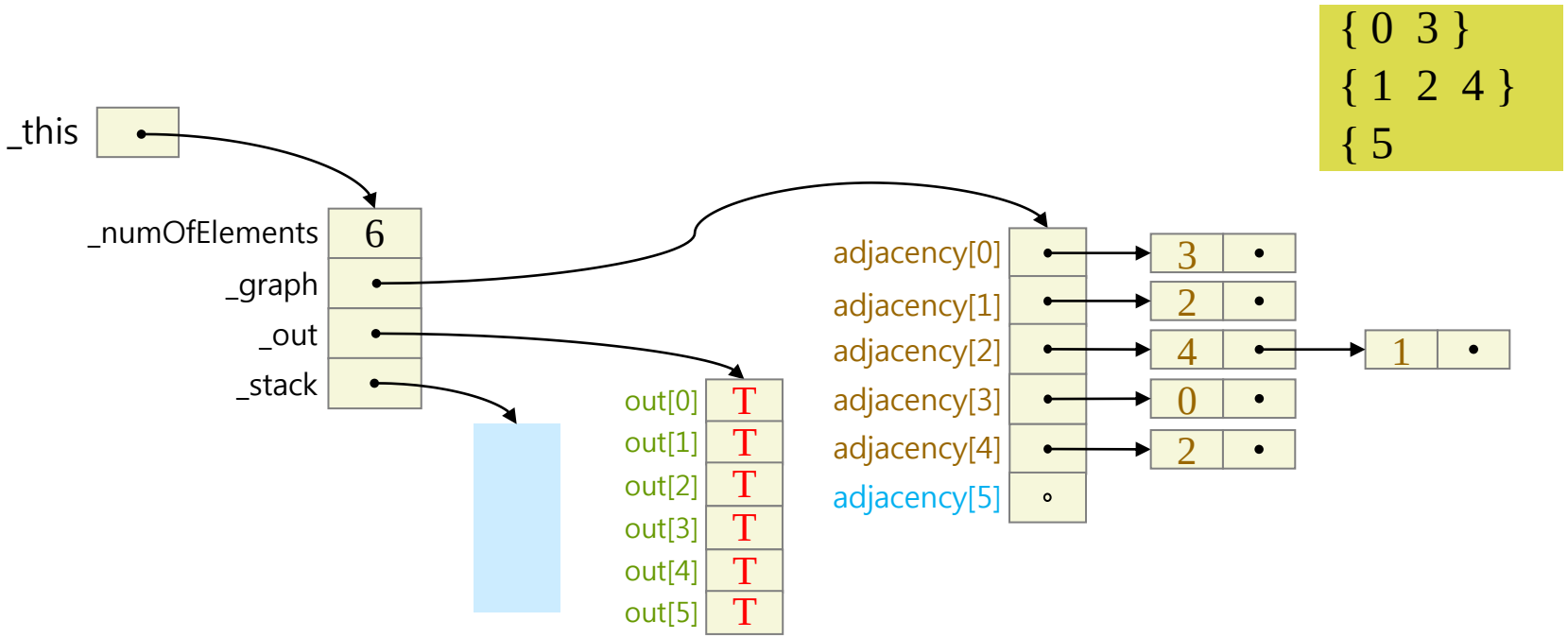
동등 클래스 찾기: 과정 [18]

- i = 5
  - out[5] 는 FALSE; 원소 5의 동등 클래스를 찾기 시작한다;
  - 원소 5 를 출력; out[5]는 TRUE로 바뀐다;
  - 원소 5 를 스택에 삽입;



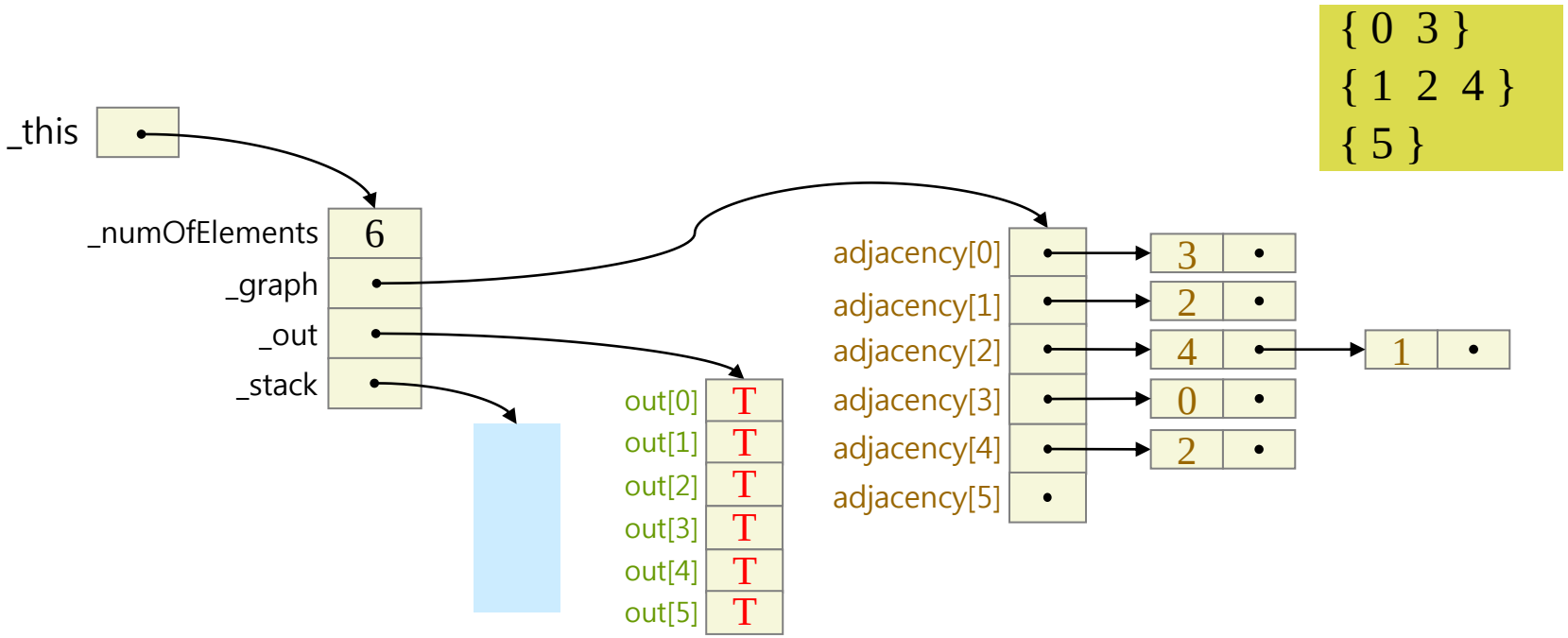
동등 클래스 찾기: 과정 [19]

- i = 5
  - 스택이 empty가 아니므로 pop:
    - poppedElement = 5;
  - Linked list인 adjacency[5]를 scan:
    - 더 이상 원소가 없으므로 scan 끝;



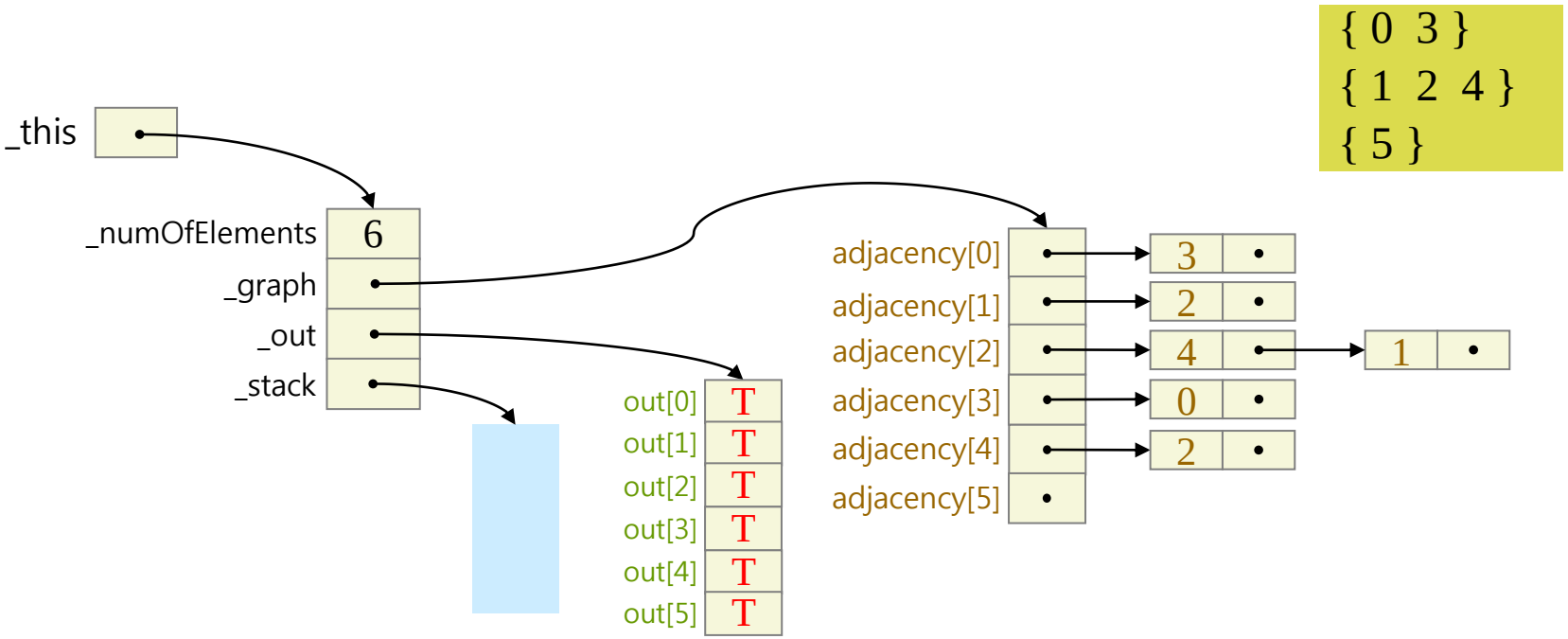
동등 클래스 찾기: 과정 [20]

- i = 5
  - 스택이 empty:
    - 원소 5 의 동등 클래스를 다 찾았다;



동등 클래스 찾기: 과정 [21]

- i = 6
  - 모든 i 에 대해 처리 완료;
  - 모든 동등 클래스를 다 찾았다;
- 프로그램 종료



FindEqvClass 의 멤버 함수는?

- FindEqvClass 의 Public Member function의 구현

```
public void perform()
{
    int   poppedElement;
    // _out[]을 전부 FALSE로 초기화
    ...

    for (int i=0; i< this._numOfElements; i++)
    {
        if (!this._out[i]) {
            /* 원소 i 는 아직 동등 클래스가 찾아지지 않았다.
             이제부터 원소 i 가 속해있는 동등 클래스를 찾는다. */
            System.out.print("{ " + i); //동등 클래스로 출력
            // _out의 i를 true로 저장 (찾은 동등 클래스이므로)
            ...
            // 동등 클래스로 출력된 원소는 스택에 삽입
            ...
            while (!this._stack.isEmpty()) { /* 스택이 empty가 아닌 동안 */
                poppedElement = // stack에서 원소를 하나 꺼내옴
                // 꺼내온 원소에 있는 Node들을 검사 -> Node들을 어떻게 검사할까?
                ...
                // linked list에 노드가 존재하는 동안(whlie)
                {
                    int element = // 저장된 다음 Node의 원소를 얻어온다.
                    if(!this._out[element]){
                        System.out.print(" " + element ); /* 동등 원소로 출력 */
                        this._out[element] = true; /* _out의 element를 true로 저장*/
                        this._stack.push(element); /*동등 클래스로 출력된 원소는 스택에 삽입*/
                    }
                } /* end while  */
            } /* end while */
            System.out.println(" }"); /* out[i]로 시작하는 동등 클래스 출력 완료 */
        } /* end if */
    } /* end for */
}
```

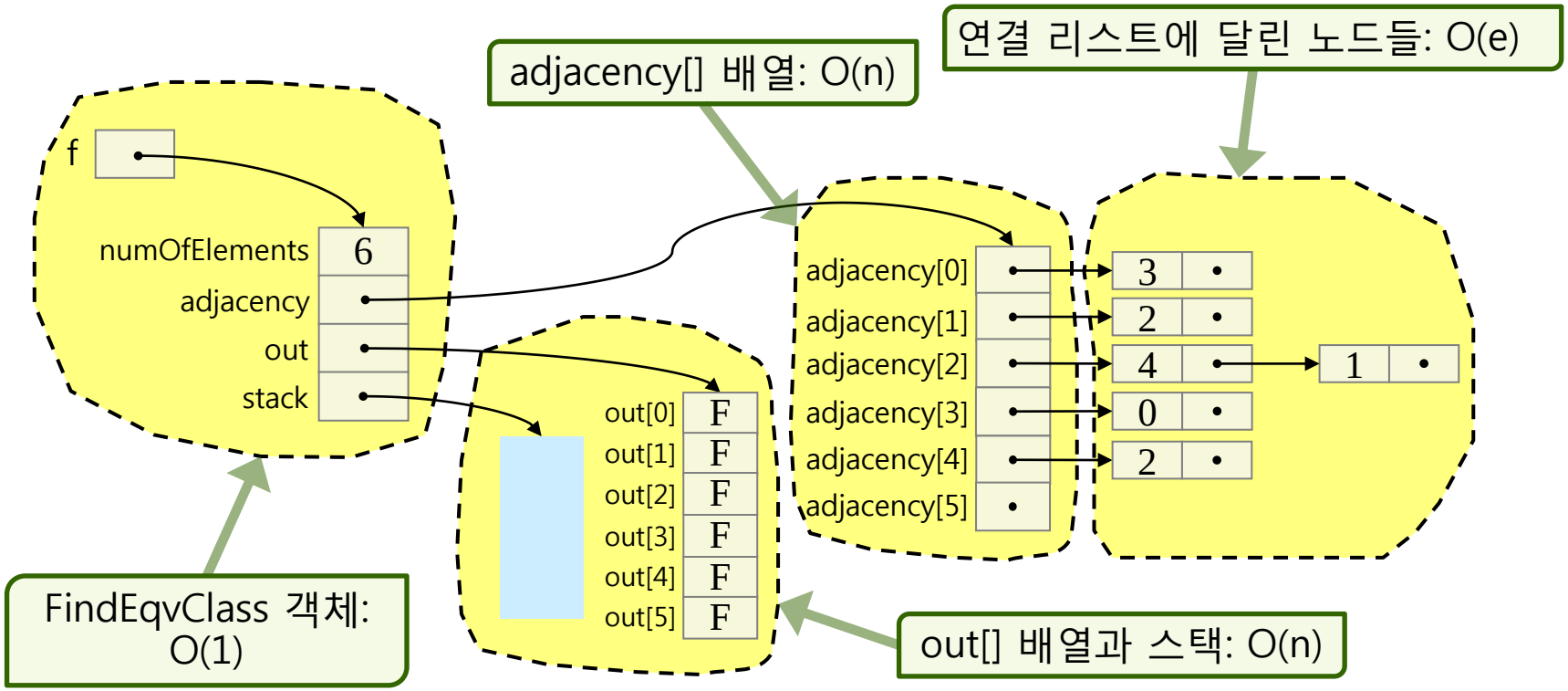
// 으로 되어 있는 주석 -> 구현!

/*으로 되어 있는 주석 -> 참고!

“동등 클래스 찾기”
복잡도 (Complexity)

공간복잡도 (Space Complexity)

- 영향을 주는 매개변수
  - n: 원소의 개수
  - e: 관계 쌍의 개수
- 공간복잡도: 필요한 메모리 크기는?
  - O(n+e)



시간 복잡도 (Time Complexity)

- 매개변수
  - n: 원소의 개수
  - e: 관계 쌍의 개수
- 주요 세부 작업별 시간 복잡도
  - 초기화
    - adjacency[] 배열: O(n)
    - out[] 배열: O(n)
  - 관계 쌍 입력 받기
    - 하나의 쌍을 입력 받아 연결리스트에 삽입하여 달아주는 일을 두 번 한다: O(e)
  - 동등 클래스 찾기
    - 모든 배열 한번씩 스캔: O(n)
    - 모든 노드 한번씩 방문: O(e)
    - 스택에서의 push/pop: O(n)
  - 객체의 소멸 (Java 환경에서는 garbage collection에 의해 실행됨)
    - 노드의 삭제: O(e)
    - 배열들(adjacency[], out[])의 삭제: O(1)
    - 스택의 삭제: O(1)
    - FindEqvClass 객체의 삭제: O(1)
- 그러므로, O(n+e)

[문제 6] 요약

- 동등관계
  - 동등 관계의 성질
  - 동등 클래스 찾기

수업 안내

- 다음 주 수요일(16일) 실습 수업은 정상적으로 진행

과제 제출

과제 제출

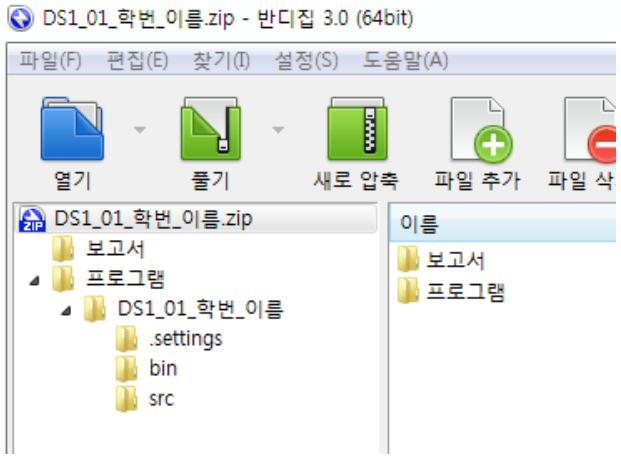
- pineai@cnu.ac.kr
  - 메일 제목 : [0X]DS2_06_학번_이름
    - 양식에 맞지 않는 메일 제목은 미제출로 간주됨
    - 앞의 0X는 분반명 (오전10시 : 00반 / 오후4시 : 01반)

- 제출 기한
  - 10월 15일(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_06_학번_이름.zip
  - 폴더의 구성
    - DS2_06_학번_이름
      - 프로그램
        - 프로젝트 폴더 / 소스
        - 메인 클래스 이름 : DS2_06_학번_이름.java
      - 보고서
        - 이곳에 보고서 문서 파일을 저장한다.
        - 입력과 실행 결과는 화면 image로 문서에 포함시킨다.
        - 문서는 pdf 파일로 만들어 제출한다.



보고서 작성 방법

- 겉장
  - 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 제외 3장 이내 작성 --------
  4. 소스코드 : 화면 capture가 아닌 소스를 붙여넣을 것

     소스는 장수 제한이 없음.

[제 6 주 실습] 끝