자료구조\_과제5\_01

Equivalence Relation

본 과제는 C++을 기반으로 작성하였습니다.



자료구조(2058)

김성열교수님

소프트웨어학과

201811182

박원준

2019.04.16

목 차

1. 문제 정의…………………………………………………………………………..p1

2. 소스코드…………………………………………………………………………….p1

3. 실행 결과………………………………………………………………………...p10

4. 참고문헌………………………………………………………………………….p13

**1. 문제 정의**

다음과 같은 정의를 만족하는 집합 A의 부분집합들이 있다.

집합 A의 부분집합들을 출력하는 프로그램을 작성해라.

프로그램의 첫 줄에서는 사용자로부터 부분집합의 총 개수에 대해서 입력 받는다. 그 후 한 번에 두 가지의 집합을 입력 받는 행위를 반복한다. 입력 받은 두집합은 서로 같은 부분집합 내에 속한다. 이 행위는 사용자가 -1 -1을 입력할 때까지 반복한다.

입력 예시는 다음과 같다.

9

0 4

5 2

3 1

5 4

8 5

이때 사용자의 입력은 Minimal Form이라 가정한다.

위의 입력은 다음을 나타낸다.

**2. 소스코드**

<Node.h>

#pragma once

class Node

{

private :

Node \* next;

Node\* before;

int key;

public:

Node();

Node(int key);

~Node();

void SetNext(Node\* next);

void SetBefore(Node\* before);

void SetKey(int key);

Node\* GetNext();

Node\* GetBefore();

int GetKey();

};

<Node.cpp>

#include "Node.h"

#include <iostream>

Node::Node()

: key(-1)

{

}

Node::Node(int key)

: key(key)

{

}

Node::~Node()

{

}

void Node::SetNext(Node \* next)

{

this->next = next;

}

void Node::SetBefore(Node \* before)

{

this->before = before;

}

void Node::SetKey(int key)

{

this->key = key;

}

Node \* Node::GetNext()

{

return next;

}

Node \* Node::GetBefore()

{

return before;

}

int Node::GetKey()

{

return key;

}

<LinkedList.h>

#pragma once

#include "Node.h"

class LinkedList

{

private :

Node\* first;

Node\* end;

Node\* current;

public:

LinkedList();

~LinkedList();

/\*void Insert(int index);

void Insert(int key, int index);\*/

void PushBack();

void PushBack(int key);

Node\* operator[](int index);

void MoveCurrentToFront();

void MoveCurrentToBack();

void SetCurrentBegin();

Node\* GetCurrent();

Node\* GetEnd();

};

<LinkedList.cpp>

#include "LinkedList.h"

LinkedList::LinkedList()

{

first = new Node();

end = new Node();

first->SetNext(end);

first->SetBefore(first);

end->SetNext(end);

end->SetBefore(first);

current = first; // current의 초기값은 first를 가르킴.

}

LinkedList::~LinkedList()

{

Node\* tmp = first;

Node\* next;

while (tmp != end)

{

next = tmp->GetNext();

delete tmp;

tmp = next;

}

delete tmp;

}

//void LinkedList::Insert(int index)

//{

// Insert(false, index);

//}

//

//void LinkedList::Insert(int key, int index)

//{

// // index번째 인덱스에 key값을 넣겠다.

// Node\* tmp = first->GetNext();

// for (int i = 0; i < index; i++)

// tmp = tmp->GetNext();

// // 현재 tmp의 next는 index번째 node.

//

// Node\* insert = new Node(key);

// insert->SetNext(tmp->GetNext());

// insert->SetBefore(tmp);

// tmp->SetNext(insert);

//}

void LinkedList::PushBack()

{

PushBack(-1);

}

/\*

\* LinkedList::PushBack(int key)

\* current 바로 다음에 push함.

\*/

void LinkedList::PushBack(int key)

{

Node\* insert = new Node(key);

insert->SetNext(current->GetNext());

current->SetNext(insert);

current = insert;

}

/\*

\* LinkedList::operator[](int index)

\* O(n)

\*/

Node\* LinkedList::operator[](int index)

{

Node\* tmp = first->GetNext();

for (int i = 0; i <= index; i++)

tmp = tmp->GetNext();

return tmp;

}

void LinkedList::MoveCurrentToFront()

{

current = current->GetBefore();

}

void LinkedList::MoveCurrentToBack()

{

current = current->GetNext();

}

void LinkedList::SetCurrentBegin()

{

current = first;

}

Node\* LinkedList::GetCurrent()

{

return current;

}

Node \* LinkedList::GetEnd()

{

return end;

}

<EquivalenceRelation.cpp>

/\*

\* Equivalence Relation

\* 박원준

\*/

#include <vector>

#include <stack>

#include <ctime>

#include "LinkedList.h"

#include <iostream>

using namespace std;

int N; // Number of Set. Use like const number in code.

bool\*\* link; // Use in EquivalenceRelationArray()

vector<LinkedList> linked\_list(N); // Use in EquivalenceRelationLinkedList()

bool\* cache; // Use in EquivalenceRelationArray() & EquivalenceRelationLinkedList()

vector<int> part; // Use in EquivalenceRElationArray() & EquivalenceRelationLinkedList()

stack<int> partition\_array; // Use in EquivalenceRElationArray() & EquivalenceRelationLinkedList()

vector<vector<int>> answer; // Use in EquivalenceRElationArray() & EquivalenceRelationLinkedList()

clock\_t EquivalenceRelationArray();

void dfs(const int& i);

clock\_t EquivalenceRelationLinkedList();

void dfs\_LinkedList(const int& i);

void Print();

int main(void)

{

clock\_t by\_array = EquivalenceRelationArray();

//by\_array /= CLOCKS\_PER\_SEC;

clock\_t by\_linkedlist = EquivalenceRelationLinkedList();

//by\_linkedlist /= CLOCKS\_PER\_SEC;

cout << endl;

cout << "시간(by\_array) : " << by\_array << endl;

cout << "시간(by\_linkedlist) : " << by\_linkedlist << endl;

return 0;

}

clock\_t EquivalenceRelationArray()

{

cout << "BY\_ARRAY" << endl;

cin >> N;

link = new bool\*[N]; // []로 동적할당 해주면 메모리만 할당됨. ()로 동적할당 해주면 배열 첫번째 방이 해당 값으로 초기화도 됨.

for (int i = 0; i < N; i++)

{

link[i] = new bool[N];

fill(link[i], link[i] + N, false);

}

cache = new bool[N];

fill(cache, cache + N, false);

// 하위 while문. 입력 받음. 집합의 연결

int a, b;

while(true)

{

cin >> a >> b;

if (a == -1 && b == -1) break; // -1 -1 입력시 입력 종료.

link[a][b] = link[b][a] = true;

}

clock\_t begin, end;

begin = clock();

// 탐색.

for (int i = 0; i < N; i++)

{

dfs(i);

while (!partition\_array.empty())

{

part.push\_back(partition\_array.top());

partition\_array.pop();

}

if (!part.empty())

{

answer.push\_back(part);

part.clear();

}

}

end = clock();

// 출력

Print();

answer.clear();

for (int i = 0; i < N; i++)

delete link[i];

delete[]link;

delete[]cache;

return end - begin;

}

void dfs(const int& i)

{

// 하위 if문. 탈출조건. 이미 검사한 구간일 경우 검사하지 않음

if (cache[i]) return;

bool flag = true;

// 하위 for문. 검사

for (int j = 0; j < N; j++)

{

if (link[i][j])

{

if (cache[j]) continue;

cache[i] = true;

dfs(j);

if (flag)

{

partition\_array.push(i);

flag = false;

}

}

}

if (!cache[i])

{

partition\_array.push(i);

cache[i] = true;

}

return;

}

clock\_t EquivalenceRelationLinkedList()

{

cout << "BY\_LINKEDLIST" << endl;

cin >> N; // N가지의 입력을 받을 것임

cache = new bool[N];

fill(cache, cache + N, false);

// link 2차원 배열 초기화.

linked\_list = vector<LinkedList>(N);

// 하위 while문. 파티션 생성.

int a, b;

while (true)

{

cin >> a >> b;

if (a == -1 && b == -1) break;

// linked\_list[a][b]에 b값을 저장해야지. 그래서 b 값을 타고 linked\_list[b][?]값에 탐색하는거야. 그곳이 비었으면 탐색이 끝난거고.

linked\_list[a].PushBack(b); linked\_list[b].PushBack(a);

}

for (int i = 0; i < N; i++)

{

linked\_list[i].SetCurrentBegin();

linked\_list[i].MoveCurrentToBack(); // 기본적으로 current node는 end를 가리킴.

}

clock\_t begin, end;

begin = clock();

// 탐색

for (int i = 0; i < N; i++)

{

dfs\_LinkedList(i);

while (!partition\_array.empty())

{

part.push\_back(partition\_array.top());

partition\_array.pop();

}

if (!part.empty())

{

answer.push\_back(part);

part.clear();

}

}

end = clock();

// 출력

Print();

answer.clear();

delete[]cache;

return end - begin;

}

void dfs\_LinkedList(const int& i)

{

if (cache[i]) return;

bool flag = true;

while (linked\_list[i].GetCurrent() != linked\_list[i].GetEnd())

{

int j = linked\_list[i].GetCurrent()->GetKey();

if (cache[j])

{

linked\_list[i].MoveCurrentToBack();

continue;

}

cache[i] = true;

dfs\_LinkedList(j);

if (flag)

{

partition\_array.push(i);

flag = false;

}

linked\_list[i].MoveCurrentToBack();

}

if (!cache[i])

{

partition\_array.push(i);

cache[i] = true;

}

return;

}

void Print()

{

cout << endl << "Set A = " << endl;

for (int i = 0; i < answer.size(); i++)

{

cout << "A" << i + 1 << " : ";

for (int j = 0; j < answer[i].size(); j++)

cout << answer[i][j] << " ";

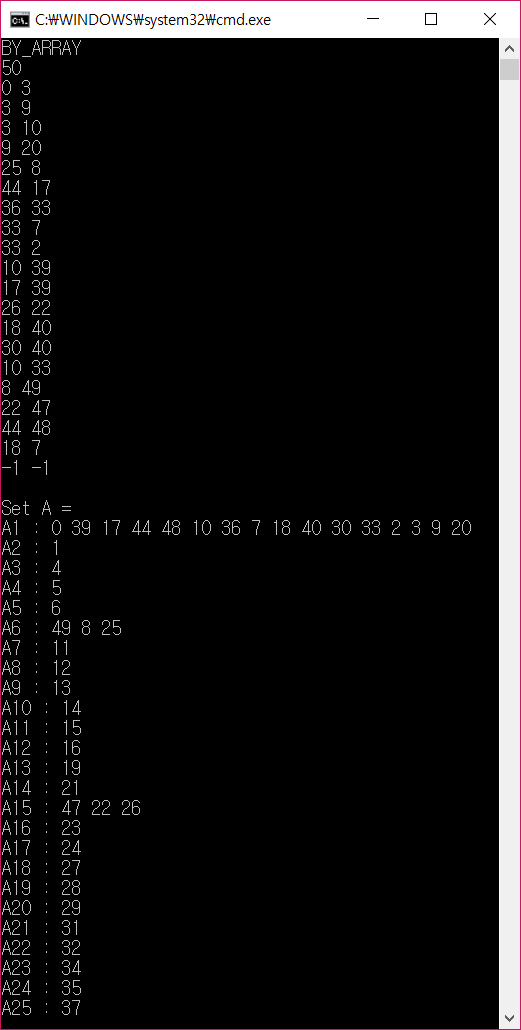
cout << endl;

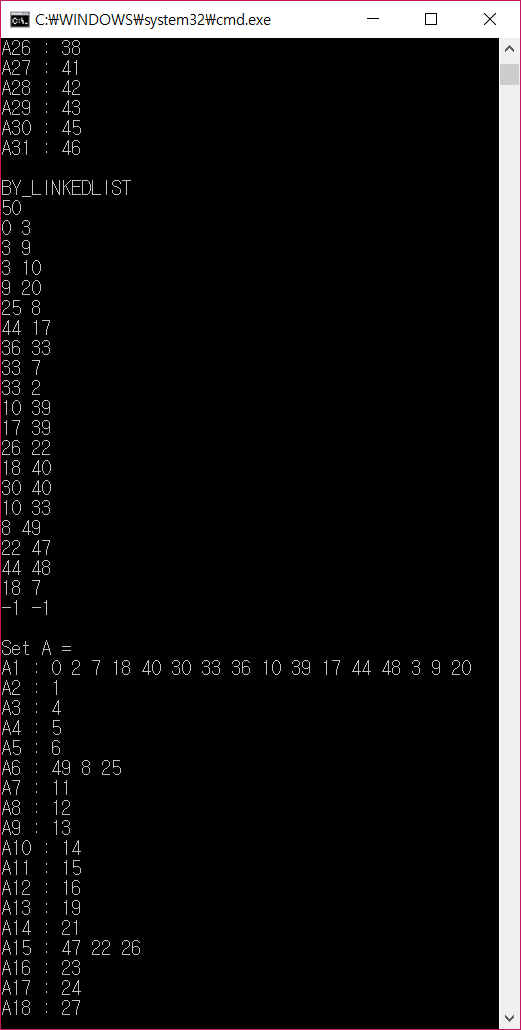
}

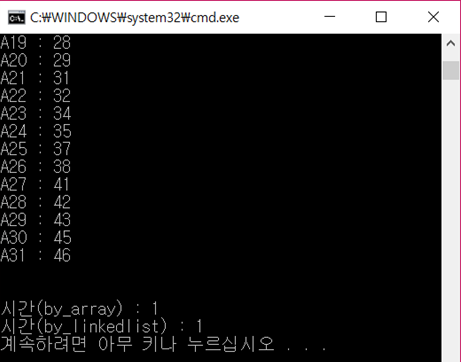
cout << endl;

}

**3. 실행 결과**







**4. 참고 문헌**

.