ACM-ICPC 상탈 사람

J

자손9319

고리즘

CC(Strongly Connected Component)

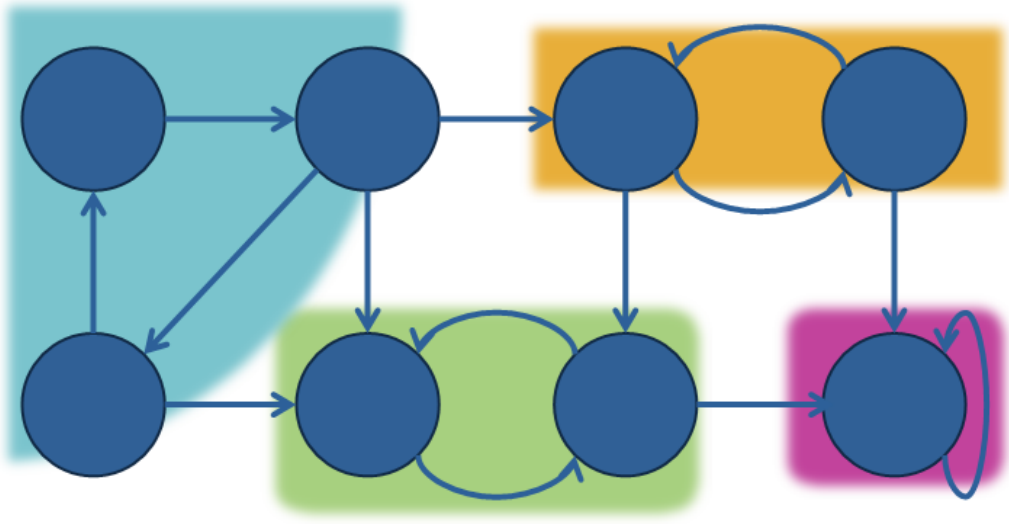
SON 자손9319 2017.01.21 20:31

향 그래프에서 어떤 그룹 X에 있는 임의의 두 정점 A,B에 대해서 항상 A->B로 가는 경로가 존재한다면 그 그룹을 **CC(Strongly Connected Component)**라 칭합니다.

정확하게 정의하자면 SCC가 되는 그룹은 항상 최대로 정의되기 때문에 다음과 같은 조건을 만족해야 합니다.

같은 SCC 내의 임의의 두 정점 A,B사이의 경로가 항상 존재한다.
서로 다른 SCC에서 뽑은 임의의 두 점 A,B 사이의 경로 A->B로 가는 경로와 B->A로 가는 경로는 동시에 존재할 수 없다.
CC 끼리는 사이클이 존재하지 않는다.)

CC를 직역하면 "강한 연결 요소" 라는 뜻이됩니다. 즉 SCC는 집합 내에서 정점들이 서로 왕복 가능한 최대 크기의 집합입니다.



음과 같은 그래프에서 색칠된 영역이 같은 정점들이 SCC를 이룹니다.

글에서는 방향 그래프가 주어졌을 때 어떤 정점들이 서로 SCC를 이루는 지 분류할 수 있는 두가지 알고리즘을 소개하겠습니다.

알고리즘 모두 **DFS를 기반으로 동작**하기 때문에 DFS(깊이우선탐색)에 익숙하지 않으신 분들은 DFS를 먼저 학습하시는 좋을것 같습니다.

선 첫번째로, **코사라주 알고리즘**을 소개해드리겠습니다.

사라주 알고리즘을 수행하기 위해서 우리는 주어지는 방향 그래프와 주어지는 방향 그래프의 방향을 모두 역으로 뒤집은 방향 그래프를 준비해야 합니다.

확히는 방향 그래프, 역방향 그래프, 스택 이렇게 세가지 컨테이너가 필요합니다.

선 정방향 그래프를 DFS를 수행하며 끝나는 순서대로 스택에 삽입해줍니다. 이때 DFS는 모든 정점에 대해서 수행되어야 합니다.

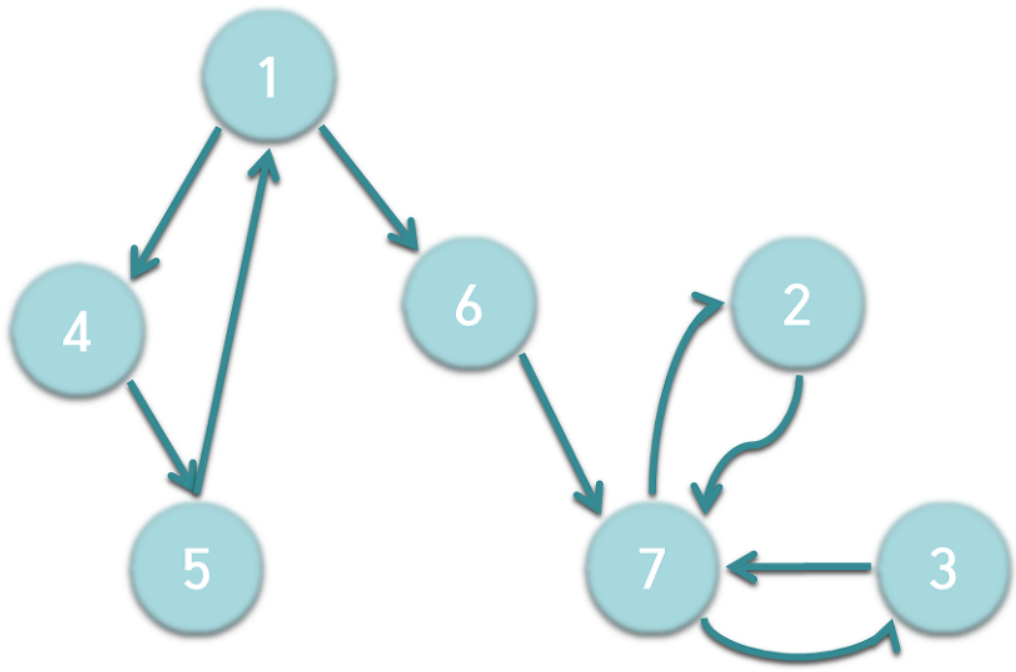
택에서 pop하는 순서대로 역방향 DFS를 수행하여 한번 수행에 탐색되는 모든 정점들을 같은 SCC로 묶습니다.

택이 빌 때 까지 이 작업을 반복하고 나면 SCC를 구할 수 있습니다.

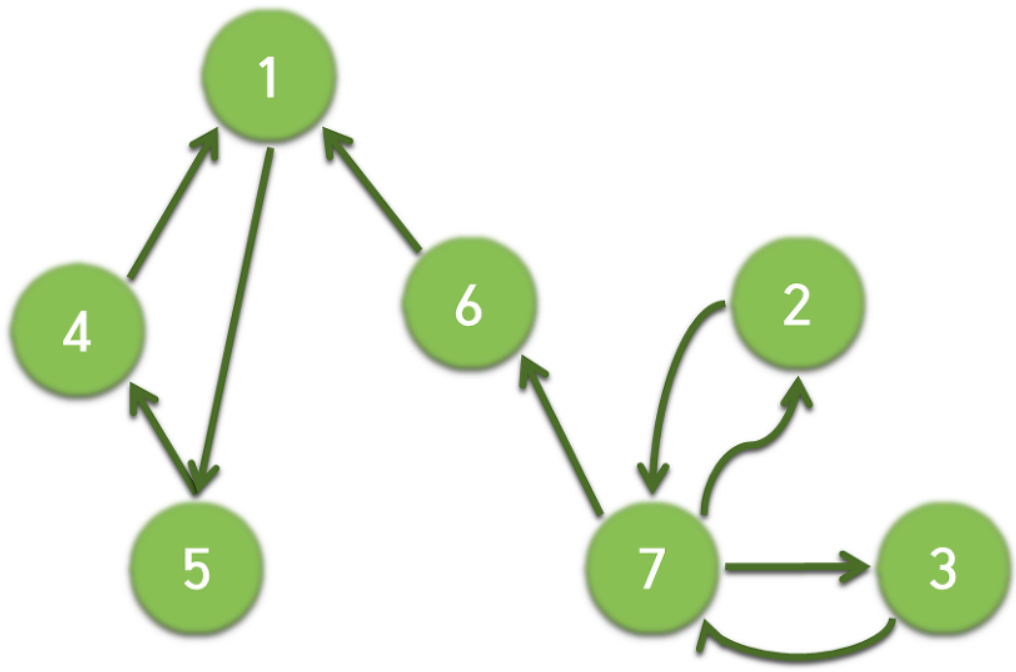
더 쉬운 이해를 위하여 그림과 함께 보겠습니다.

ACM-ICPC 상
탈 사람

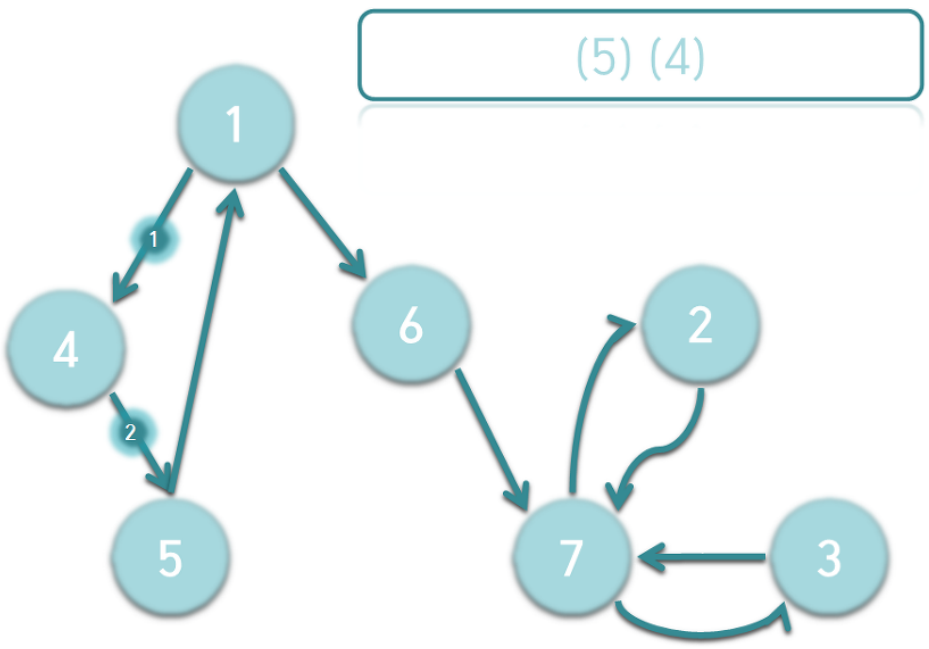
자손9319



ɜ과 같은 그래프의 SCC를 코사라주 알고리즘을 이용하여 구해보겠습니다.

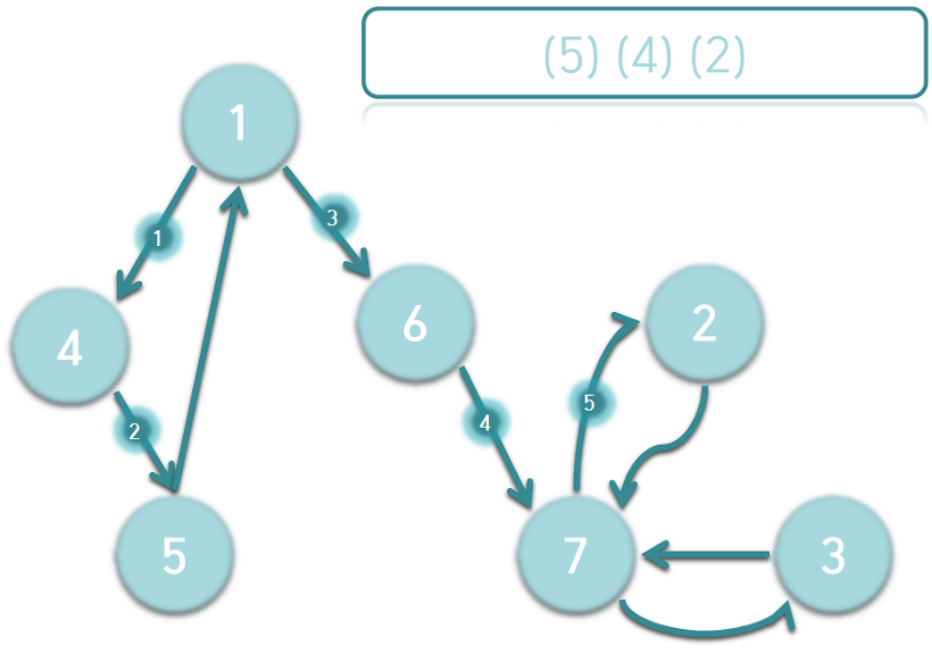


넌 그래프를 모델링하면서 이러한 역방향 그래프도 같이 모델링 해줍니다.



! 정점에서 DFS를 시작할 시 1,2번 간선을 타고 5번 정점에서 호출한 DFS가 종료되며 스택에 5가 쌓이고 그 다음 순서로
! 정점에서 호출한 DFS가 종료되며 스택에 4가 쌓이고 1번 정점으로 돌아오게 됩니다.

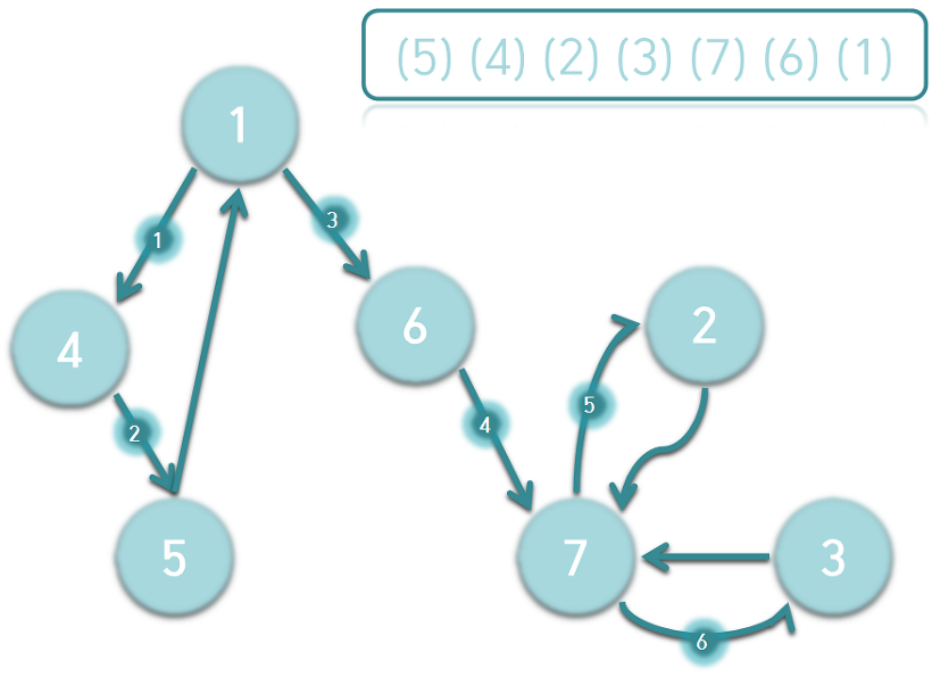
ACM-ICPC 상
탈 사람



J

자손9319

후 1번 정점에서 다시 3,4,5번 간선을 타고 2번 정점에서 호출한 DFS가 종료되며 스택에 2가 쌓이고 다시 7번 정점으로 아갑니다.

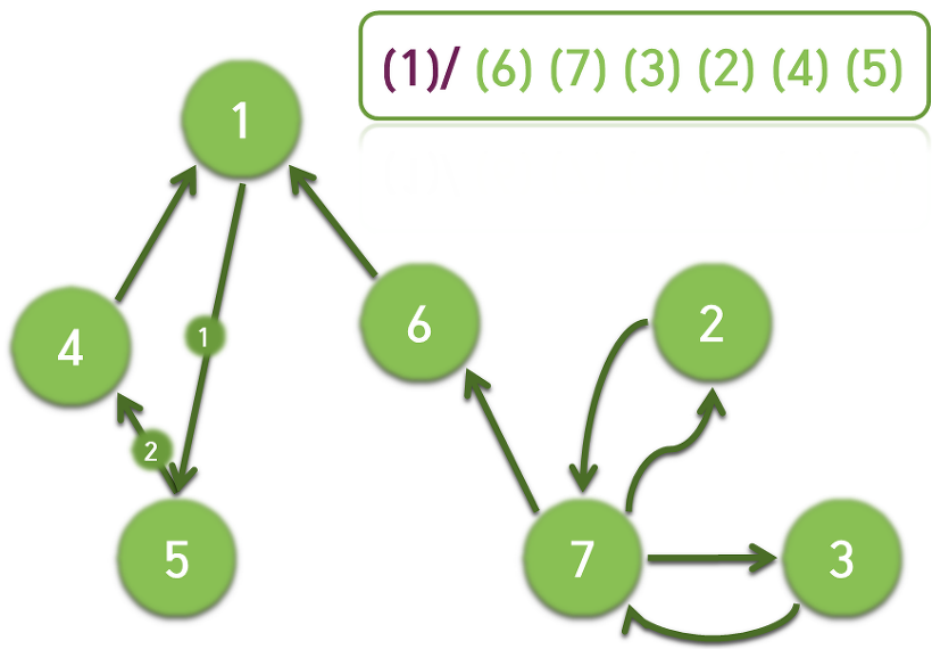


후 6번 간선을 타고 3번 정점, 7번 정점, 6번 정점, 1번 정점 순서대로 DFS가 종료되며 스택에 순서대로 쌓입니다.

든 정점을 다 방문하였으니 이제 역방향 DFS를 돌릴 차례입니다.

대 스택에는 1<-6<-7<-3<-2<-4<-5 순서대로 쌓여있고 top은 1입니다.

의를 위해 그림의 스택을 뒤집겠습니다.

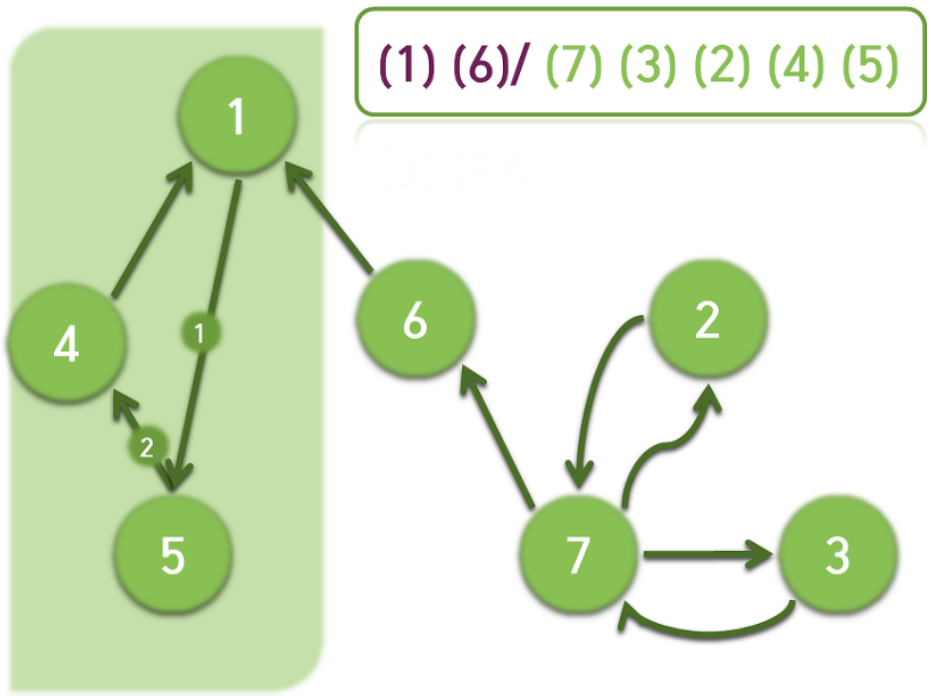


이제 스택의 top인 1번 정점에서 부터 DFS를 탐색하겠습니다.

:번 간선을 타고 4번 정점까지 도달한 뒤 DFS가 종료됩니다. 이로서 1번 4번 5번 정점이 SCC를 이루고 있다는걸 알게됩니다.

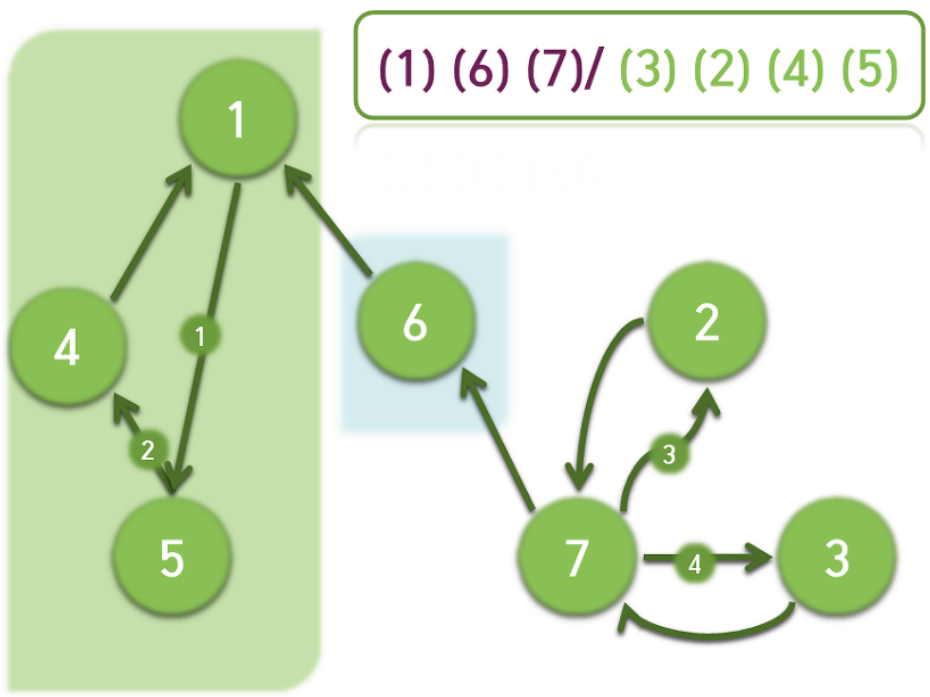
ACM-ICPC 상 탈 사람

자손9319



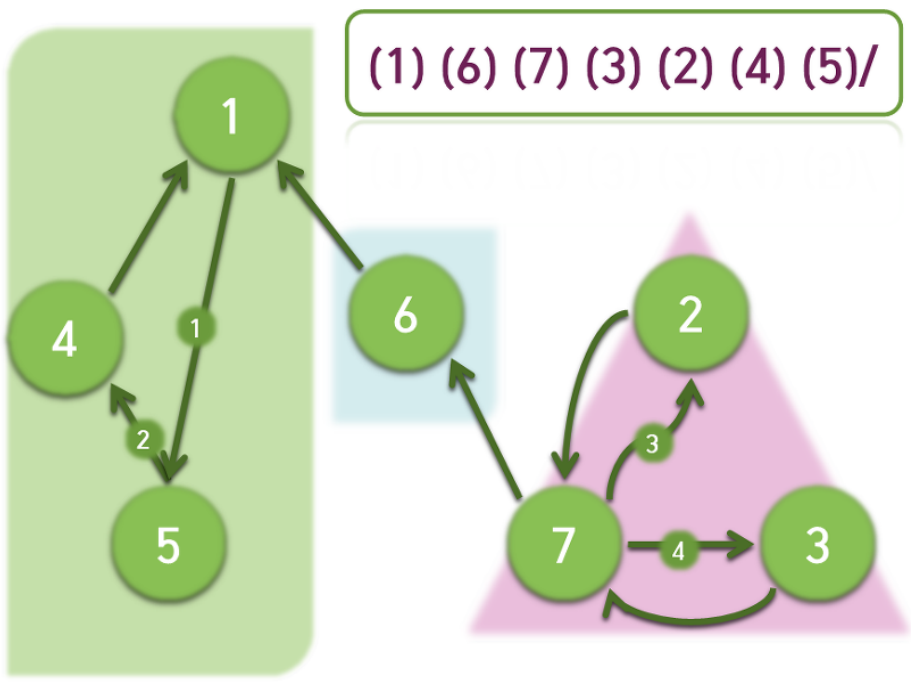
이제 스택의 top인 6번 정점에서 DFS를 탐색하겠습니다.

! 정점은 이미 visited되어 있기 때문에 6번정점에서는 더 이상 탐색할 수 있는 정점이 없으므로 DFS가 종료됩니다. 이로 6번 정점이 홀로 SCC를 이루게 됩니다.



이제 스택의 top인 7번 정점에서 DFS를 탐색하겠습니다.

! 정점은 이미 visited되어 있고 3,4번 간선을 타고 2번 3번 정점을 탐색 한 뒤 DFS가 종료됩니다. 이로서 2번 3번 7번 정 이 SCC를 이루고 있다는걸 알게됩니다.



후 스택에서 3~5번 정점들을 차례로 확인하겠지만 전부 이미 방문된 정점이므로 이대로 함수가 종료됩니다.

리는 코사라주 알고리즘을 통해 이 그래프에서 3개의 SCC를 구하게 되었습니다.

해를 돕기 위해 코드를 한번 보여드리겠습니다.

음은 BOJ 2150 Strongly Connected Component 코드입니다.

```cs
#include <cstdio>
#include <algorithm>
#include <vector>
```

```cpp
#include <stack>
#include <cstring>
#define MAX_V 10000
using namespace std;
int v, e, visited[MAX_V + 1], x, y, r;
vector<vector<int>> scc;
vector<vector<int>> vt;
vector<vector<int>> rvt;
stack<int> st;
bool cmp(vector<int> x, vector<int> y) {
    return x[0] < y[0];
}
void dfs(int v) {
    visited[v] = true;
    for (int next : vt[v]) {
        if (visited[next])
            continue;
        dfs(next);
    }
    st.push(v);
}
void func(int v, int c) {
    visited[v] = true;
    scc[c].push_back(v);
    for (int next : rvt[v]) {
        if (visited[next])
            continue;
        func(next, c);
    }
}
int main() {
    scanf("%d%d", &v, &e);
    vt.resize(v + 1);
    rvt.resize(v + 1);
    for (int i = 0; i < e; i++) {
        scanf("%d%d", &x, &y);
        vt[x].push_back(y);
        rvt[y].push_back(x);
    }
    for (int i = 1; i <= v; i++) {
        if (visited[i])
            continue;
        dfs(i);
    }
    memset(visited, 0, sizeof(visited));
    while (st.size()) {
        int here = st.top();
        st.pop();
        if (visited[here])
            continue;
        scc.resize(++r);
        func(here, r - 1);
    }
    for (int i = 0; i < r; i++)
        sort(scc[i].begin(), scc[i].end());
    sort(scc.begin(), scc.end(), cmp);
    printf("%d\n", r);
    for (int i = 0; i < r; i++) {
        for (int x : scc[i])
            printf("%d ", x);
        printf("-1\n");
    }
    return 0;
}
```

Colored by Color Scripter

터를 sort해주는 이유는 문제에서 작은 정점이 있는 scc순서대로 출력하기를 원하기 때문입니다.
의 코드에서 dfs 함수가 정방향 그래프를 위한 DFS이고 func 함수는 역방향 그래프를 위한 DFS입니다.

이제 코사라주 알고리즘으로 SCC를 구현해 보실수 있겠나요?

사라주 알고리즘은 2번의 DFS로 구현되기 때문에 시간복잡도는 DFS와 같은 O(V+E)가 됩니다.

에 두번째 방법인 **타잔 알고리즘**을 통하여 SCC를 구현해보겠습니다.

ACM-ICPC 상
탈 사람

자손9319

단 알고리즘은 DFS를 수행할 때 생성되는 DFS 트리의 간선의 정보를 이용하여 ALL DFS 한번으로 모든 SCC를 구하는 법입니다.

대 ALL DFS란 모든 정점에서 수행되는 DFS를 의미합니다.

단 알고리즘은 ALL DFS를 돌리며 Spanning Tree를 만들어 갈 때 DFS의 호출 순서에 따라서 정점을 stack에 push 합 다.

후 간선 분류를 통하여 먼저 호출 되는 정점이 더 높은 위치를 가진다고 생각할 때 가장 높이 올라갈 수 있는 정점을 찾는데 때 here->there가 교차간선이지만 아직 there가 SCC에 속하지 않는다면 discover[there] 또한 고려해 줍니다.

'S가 끝나기전에 ret과 discover[here]가 같다면 stack에서 pop하면서 here가 나올 때까지 같은 SCC로 분류합니다.

로는 이해가 어려우실 수도 있으니 코드를 첨부하겠습니다.

ㅡ는 위에 코사라주 알고리즘때 첨부한 BOJ 2150을 타잔 알고리즘으로 구현한 version입니다.

```cs
#include <cstdio>
#include <algorithm>
#include <vector>
#include <stack>
#include <cstring>
#define MAX_V 10000
using namespace std;
int v, e, discover[MAX_V + 1], x, y, r, c, scc[MAX_V + 1];
stack<int> st;
vector<vector<int>> vt;
vector<vector<int>> res;
bool cmp(vector<int> a, vector<int> b) {
    return a[0] < b[0];
}
int dfs(int here) {
    discover[here] = c++;
    int ret = discover[here];
    st.push(here);
    for (int there : vt[here]) {
        if (discover[there] == -1)
            ret = min(ret, dfs(there));
        else if (scc[there] == -1)
            ret = min(ret, discover[there]);
    }
    if (ret == discover[here]) {
        vector<int> tmp;
        while (1) {
            int t = st.top();
            st.pop();
            scc[t] = r;
            tmp.push_back(t);
            if (t == here)break;
        }
        sort(tmp.begin(), tmp.end());
        res.push_back(tmp);
        r++;
    }
    return ret;
}
int main() {
    scanf("%d%d", &v, &e);
    vt.resize(v + 1);
    for (int i = 0; i < e; i++) {
        scanf("%d%d", &x, &y);
        vt[x].push_back(y);
    }
    memset(discover, -1, sizeof(discover));
    memset(scc, -1, sizeof(scc));
    for (int i = 1; i <= v; i++) {
        if (discover[i] == -1)
            dfs(i);
    }
    sort(res.begin(), res.end(), cmp);
```

```
    printf("%d\n", r);
    for (int i = 0; i < r; i++) {
        for (auto h : res[i])
            printf("%d ", h);
        printf("-1\n");
    }
    return 0;
}
```

Colored by Color Scripter

ACM-ICPC 상탈 사람

J

자손9319

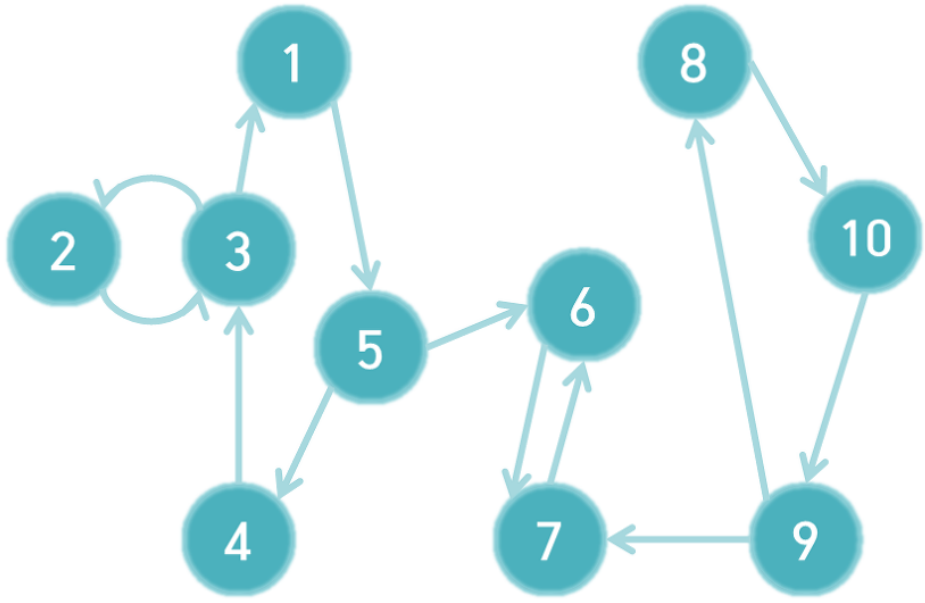
더 호출되는 정점을 높은 위치라고 생각할 때

;cover되지 않은 정점이나 discover는 됬지만 아직 scc에 속하지 않는 정점들 중에서 자신이 탐색 할 수 있는 정점 중 가 높은 위치를 return 받습니다.

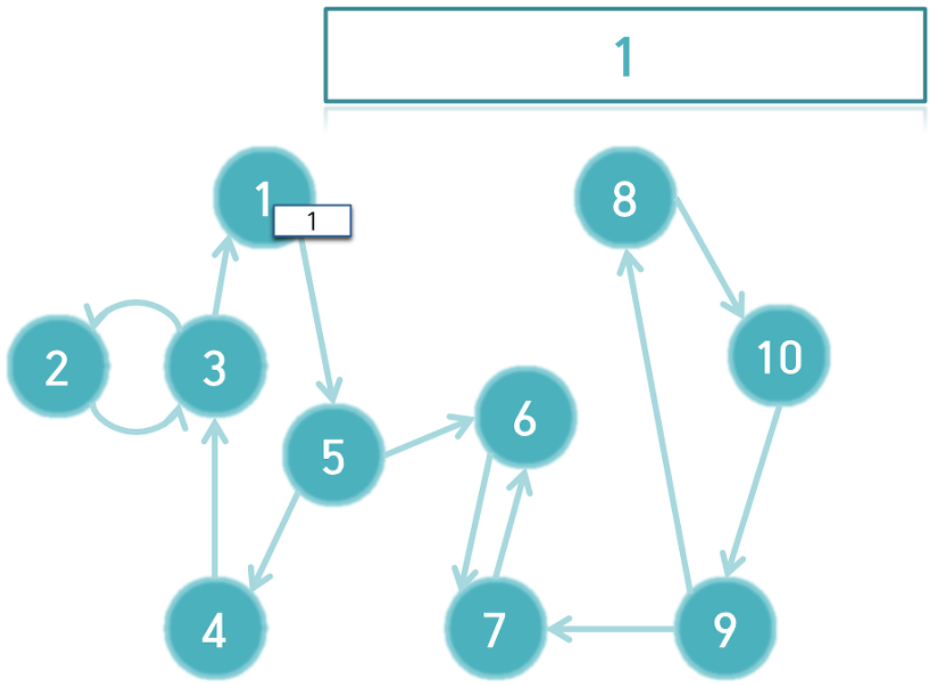
return 값이 자기에게 할당된 discover 값과 같다면 이 정점이 나올 때 까지 stack을 pop하며 나오는 모든 정점들을 :C로 묶습니다.

단 알고리즘은 위상 정렬을 이용한 방법으로 생성되는 SCC들은 위상정렬의 역순으로 생성됩니다.

쉬운 이해를 위하여 타잔 알고리즘도 그림과 함께 진행해보겠습니다.



음과 같은 그래프를 타잔 알고리즘을 통해 SCC를 분류 해보겠습니다.

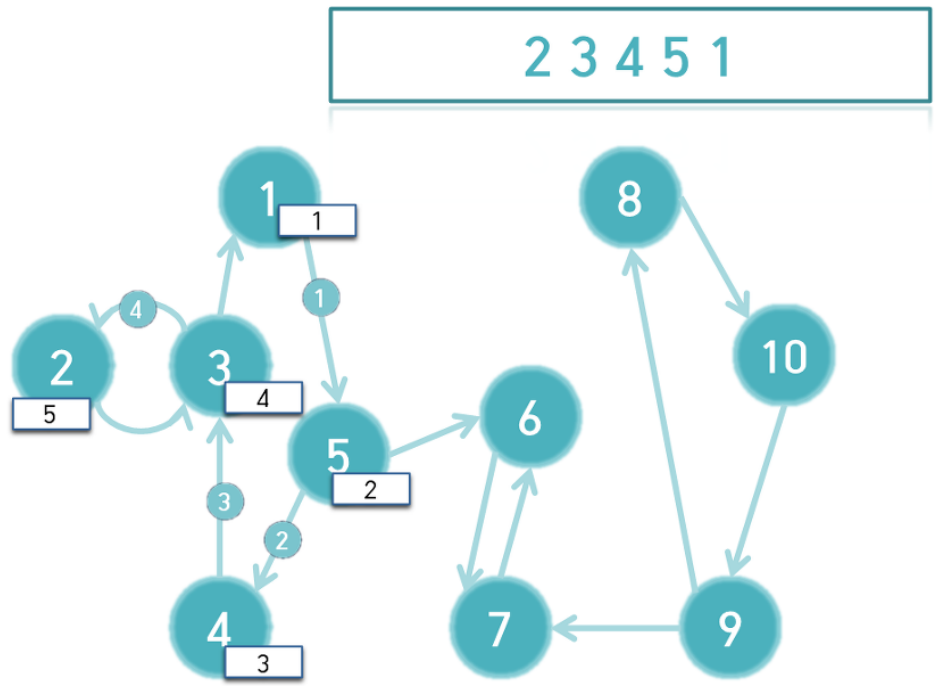


넌 1번 정점부터 DFS를 탐색하겠습니다. 스택에 1을 삽입한 뒤 1번 정점에 discover 번호를 1로 매겨줍니다.

ACM-ICPC 상 탈 사람

J

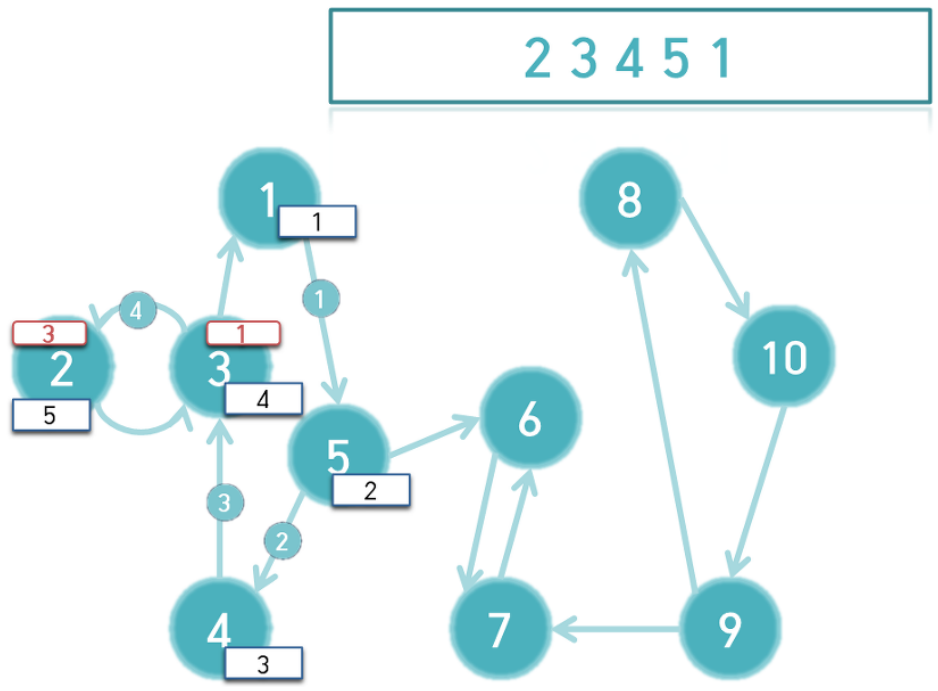
자손9319



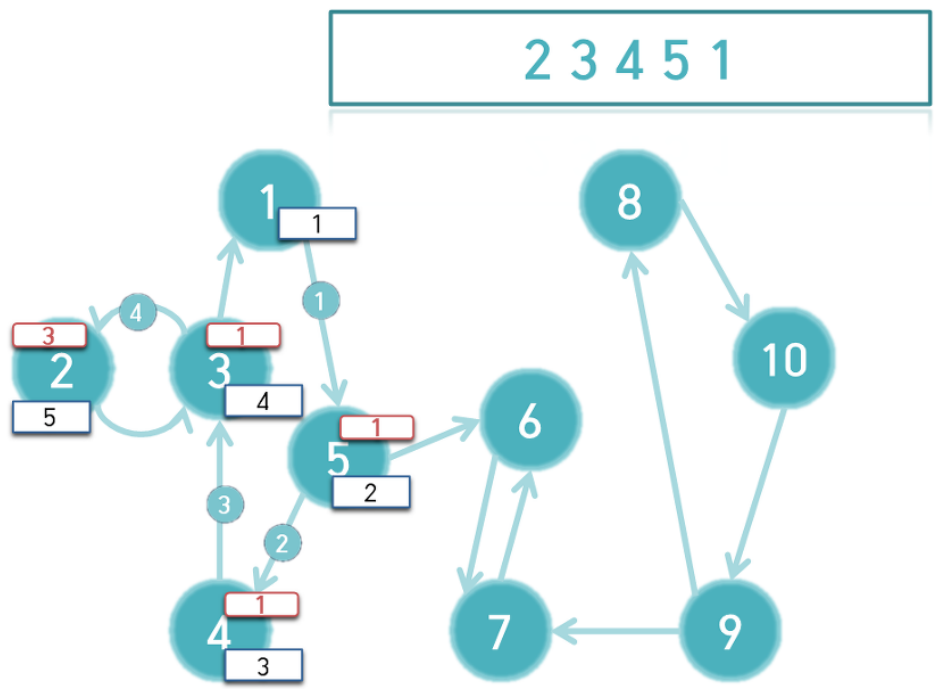
후 1,2,3,4번 간선을 통하여 순서대로 5 4 3 2 번 정점이 stack에 삽입되며 각각 탐색순서대로 discover 번호를 할당받습
다.

때 2번 정점은 더 이상 탐색할 정점이 없지만 이미 탐색된 3번 정점이 아직 SCC에 속하지 않으므로 2번 정점에서 시작 된
`S는 3번 정점의 discover 번호를 return 합니다.

데 그림에서 discover 값은 파란 박스에 해당 정점에서 호출한 DFS의 return 값은 빨간 박스에 표시하겠습니다.

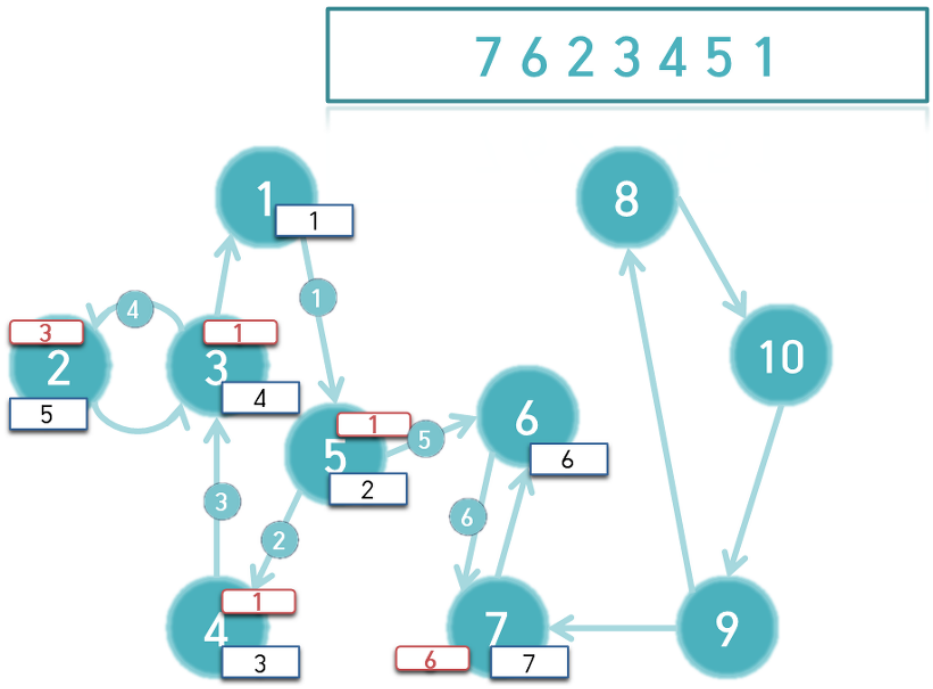


데 2번 정점에서 3을 return받은 3번 정점은 1번 정점이 탐색됬지만 SCC에 속하지 않으므로 가장 작은 discover 값인 1
정점의 discover 값을 return 합니다.



후 4번 정점과 5번 정점의 return 값은 3번으로부터 return 받은 1이 가장 작은 값이므로 1로 갱신됩니다.
데 5번 정점에서 나머지 간선들을 탐색하겠습니다.

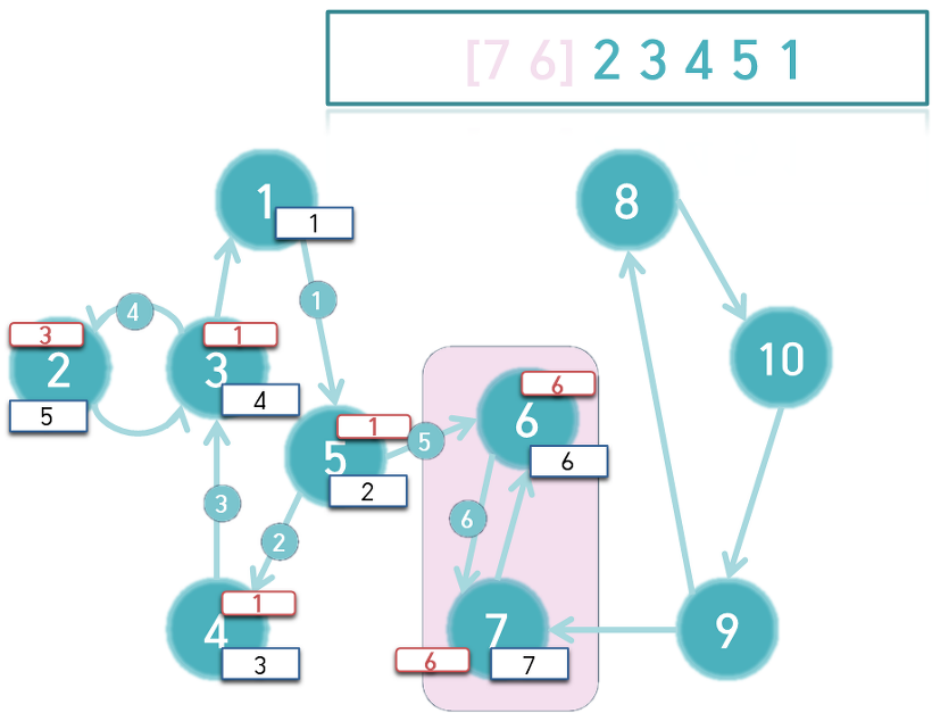
ACM-ICPC 상 탈 사람



자손9319

5,6번 간선을 통하여 스택에 순서대로 6 7번 정점이 쌓이고 discover값을 할당 받습니다.

후 6번 정점에서 7번 정점을 볼 때 아직 SCC에 속하지 않으므로 return 값이 6으로 갱신된 후 6번 정점으로 돌아갑니다.

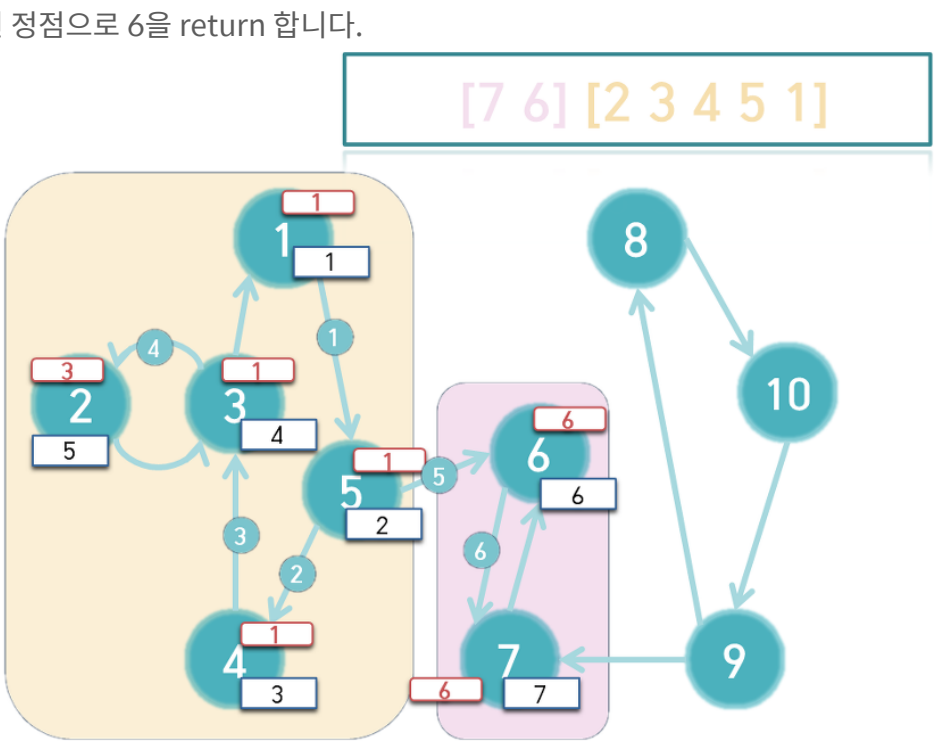


이제 6번 정점은 더이상 탐색 가능한 정점이 없기 때문에 return 값은 6이 됩니다.

! 정점에서의 return 값과 6번 정점의 discover 값이 6으로 같기 때문에 stack에서 6번 정점이 나올때 까지 pop되는 모 정점들이 SCC를 이루게 됩니다.

의 그림에서는 6번 정점과 7번 정점이 SCC를 이루게 되는군요.

이제 6번 정점은 5번 정점으로 6을 return 합니다.



이제 5번 정점은 6을 return 받았지만 1이 더 작은 값이므로 1을 return값으로 가진 뒤 자신을 호출한 1번 정점으로 1을 :urn 합니다.

! 정점에서 더 탐색할 정점이 없으므로 1을 return 값으로 가집니다. 이때 1번 정점에서 discover값과 return 값이 같으 로 1번 정점이 나올때 까지 pop되는 모든 정점들은 SCC를 이룹니다.

ACM-ICPC 상 탈 사람

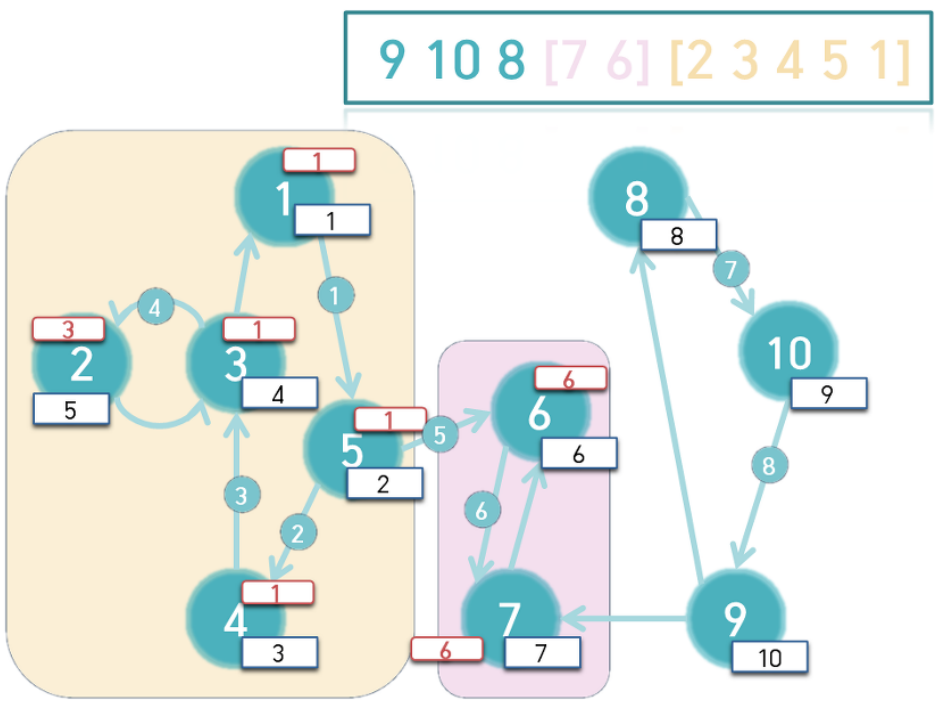
J

자손9319

의 그림과 같이 1번 2번 3번 4번 5번 정점들이 SCC를 이루게 됩니다.

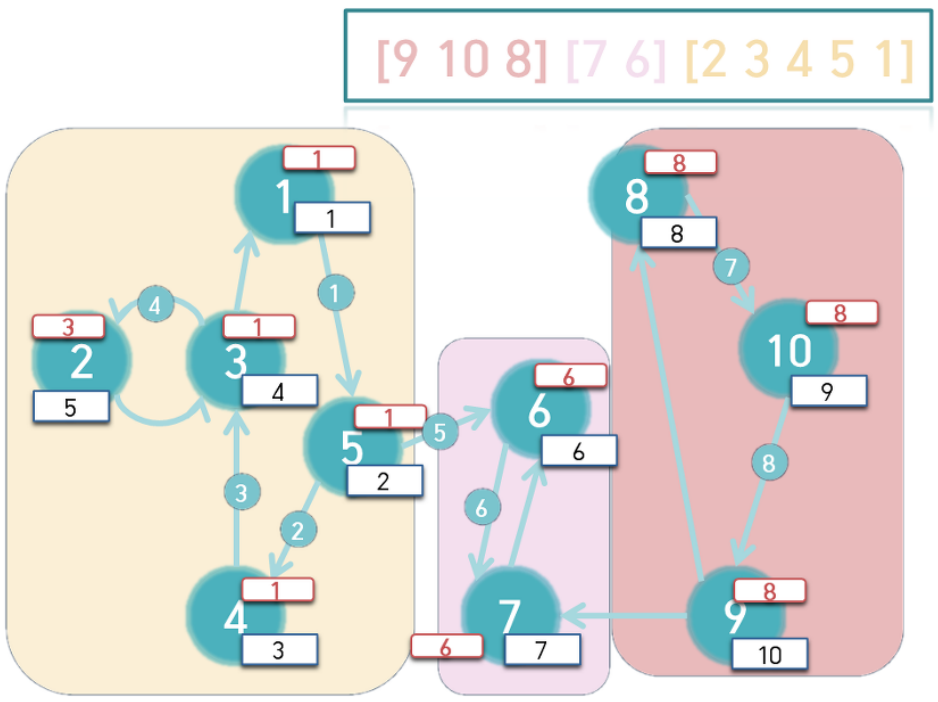
이제 1번 정점에서 호출한 DFS가 종료되었습니다.

리는 ALL DFS를 돌려야 하므로 남은 정점들 중 8번 정점에서 DFS를 시작해보겠습니다.



리는 8번 정점의 discover 값을 8으로 할당해준 후 7,8번 간선을 통하여 10,9번 정점을 탐색하면서 discover값을 할당해었습니다.

데 9번에서 시작하는 간선은 두가지가 있는데 7번 정점의 경우 discover되었는데 SCC에 이미 속하므로 제외하게 되고 8정점의 discover값이 더 작으므로 return 값으로 8을 가지게 됩니다.



후 8번 정점에서 return 된 값으로 10번 정점의 return 값이 채워지고 8번 정점의 return 값은 결국 8이됩니다.

! 정점의 discover값과 return 값이 같으므로 stack에서 8번 정점이 나올때 까지 나오는 정점들을 하나의 SCC로 묶어니다.

의 그림과 같이 8번 9번 10번 정점들이 SCC를 이루게 됩니다.

후 8번 정점에서 호출된 DFS가 종료되고 모든 정점들에 대한 탐색이 이루어 졌으니 모든 정점에 대한 SCC 분류가 끝났습다.

단 알고리즘의 시간복잡도는 DFS 1번이기 때문에 DFS와 같은 O(V+E)입니다.

우리는 이제 SCC를 구할 수 있는 두가지 알고리즘에 대해 알게 되었습니다.

나라주 알고리즘의 경우 구현이 좀 더 손쉽고 외우기 편한 반면에 DFS를 2번 돌리고 역방향 간선으로 이루어진 그래프 구현해야 하기 때문에 메모리를 조금 더 사용합니다.

단 알고리즘의 경우 코사라주 알고리즘에 비해 이해나 구현이 조금 어려울 수 있으나 DFS 한번으로 SCC를 구해낼 수 있니다.

실 시간복잡도는 둘다 O(V+E)기 때문에 무엇을 선택하는지는 취향차이인것 같습니다.

컬 SCC를 이용하여 무엇을 할 수 있을까요?

:C를 한 정점으로 볼 경우 SCC컴포넌트에 대한 사이클이 존재하지 않기 때문에 다이나믹 프로그래밍등이 가능해질수도 고

우에 따라 SCC를 이용하여 여러 문제를 해결해야 할 일 있습니다.

2-SAT 문제를 해결하기 위한 알고리즘에 SCC가 사용되므로 알아두면 요긴하게 쓰일 두 알고리즘에 대해서 공부해봤습다.

이제 SCC를 이용하여 여러가지 문제들을 풀어봅시다.

ACM-ICPC 상탈 사람

J

자손9319

:고

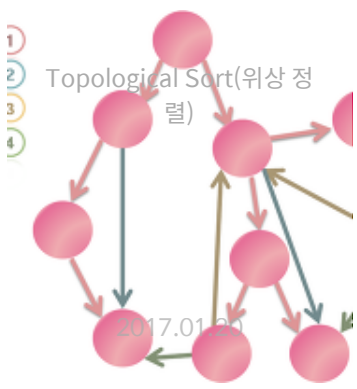
알고리즘' 카테고리의 다른 글

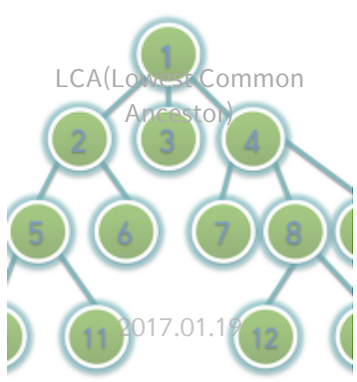
자료구조]트라이(Trie) (0)	2017.02.04
단절점(Articulation Point)와 단절선(Bridge) (0)	2017.01.31
최장 증가 수열] LIS(Longest Increasing Subsequence) (4)	2017.01.25
CC(Strongly Connected Component) (0)	2017.01.21
opological Sort(위상 정렬) (6)	2017.01.20
CA(Lowest Common Ancestor) (0)	2017.01.19

g dfs, SCC, 코사라주 알고리즘, 타잔 알고리즘

알고리즘' Related Articles

more





Comments

Name Password

여러분의 소중한 댓글을 입력해주세요

Secret Send

Prev 1 ... 75 76 77 78 79 80 81 82 83 ... 159 Next

ACM-ICPC 상
탈 사람

J

자손9319