위상정렬
강한 연결 요소

- 2111618 길수민 -

위상 정렬 - 설명

- ‘순서가 정해져있는 작업’을 차례로 수행해야 할 때
  그 순서를 결정하기 위해 사용하는 알고리즘 – Topological Sorting
- 그래프는 방향 그래프여야 하며 사이클이 존재하지 않아야 함
- DAG (Directed Acyclic Graph)에서만 적용 가능
- 여러 가지 결과가 존재 가능

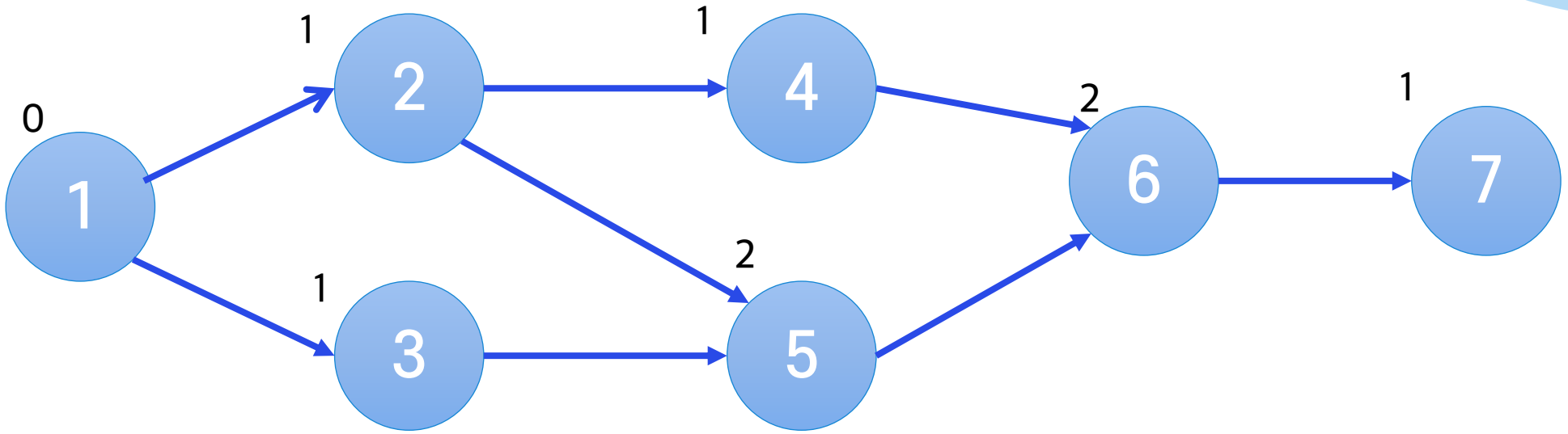
위상 정렬 - 알고리즘

1. 진입 차수가 0인 정점을 큐에 삽입 (시작점)
2. 큐에서 원소를 꺼내 연결된 모든 간선 제거
3. 간선 제거 이후 진입 차수가 0이 된 정점 큐에 삽입
4. 큐가 빌 때까지 2~3번 작업 계속해서 반복

* 모든 원소를 방문하기 전에 큐가 빈다면 사이클 존재

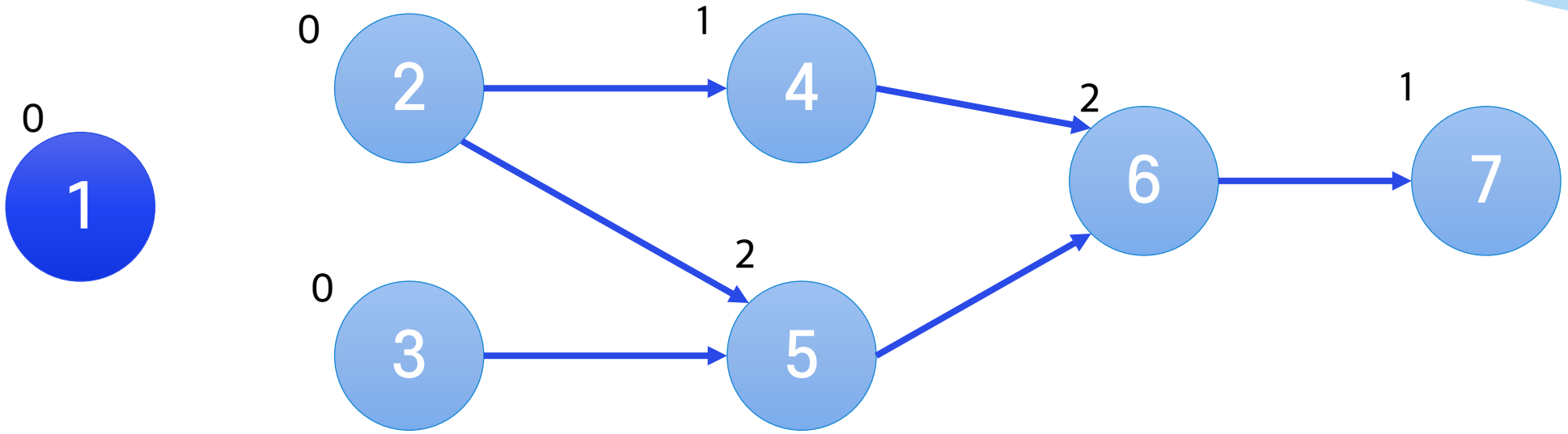
큐에서 꺼낸 순서 -> 위상정렬 결과

위상 정렬 - 예제

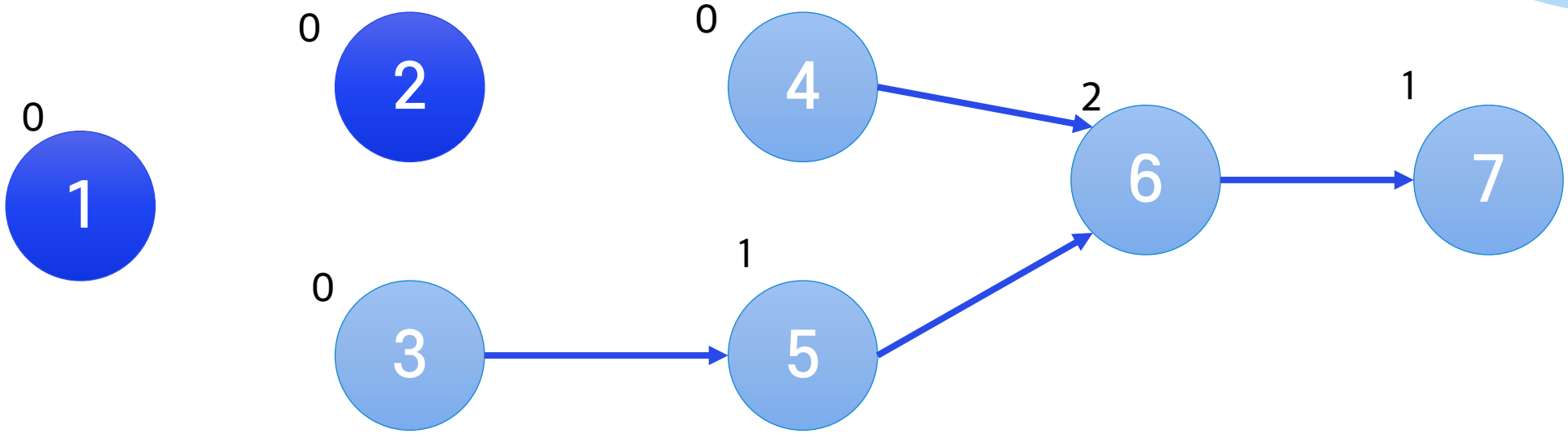


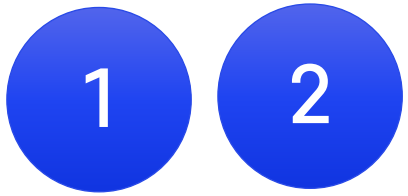
1

위상 정렬 - 예제

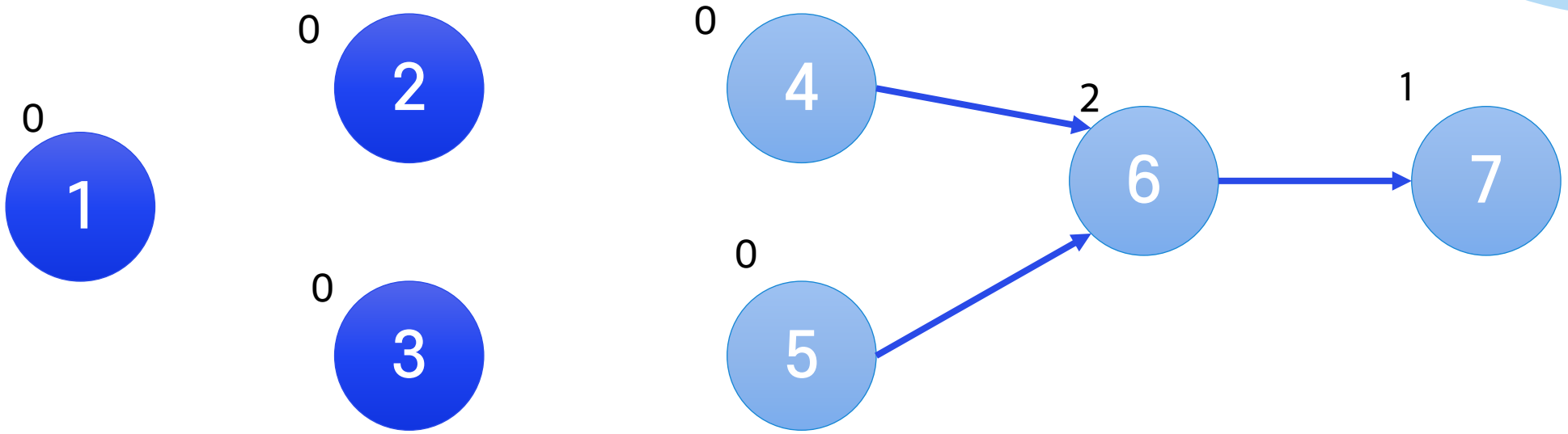


위상 정렬 - 예제

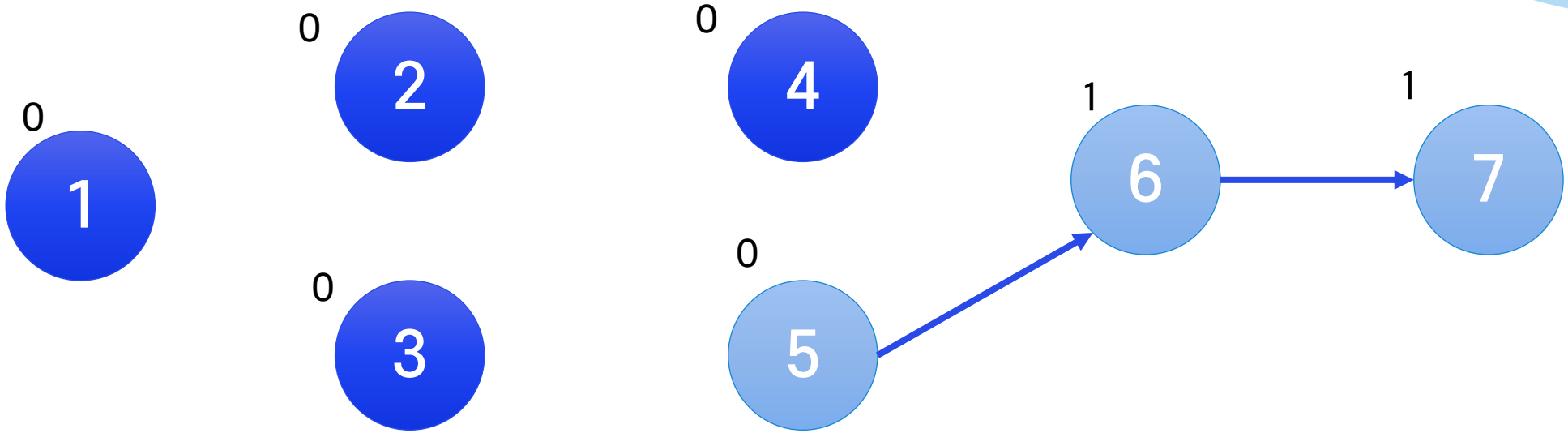


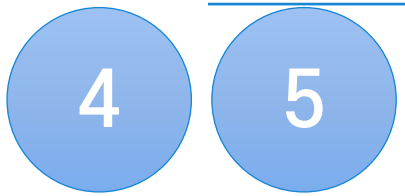


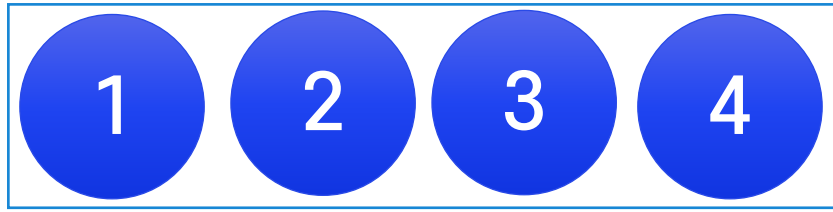
위상 정렬 - 예제



위상 정렬 - 예제

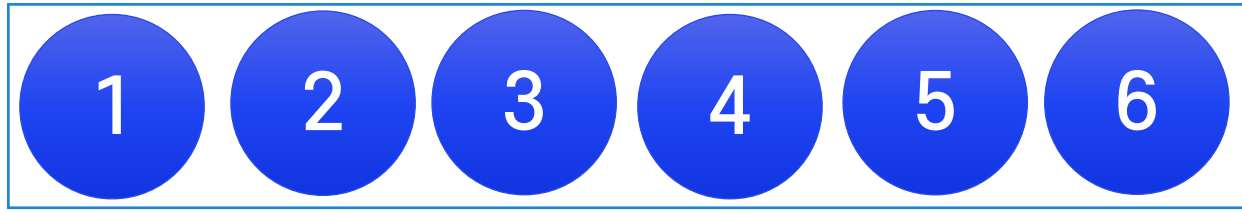




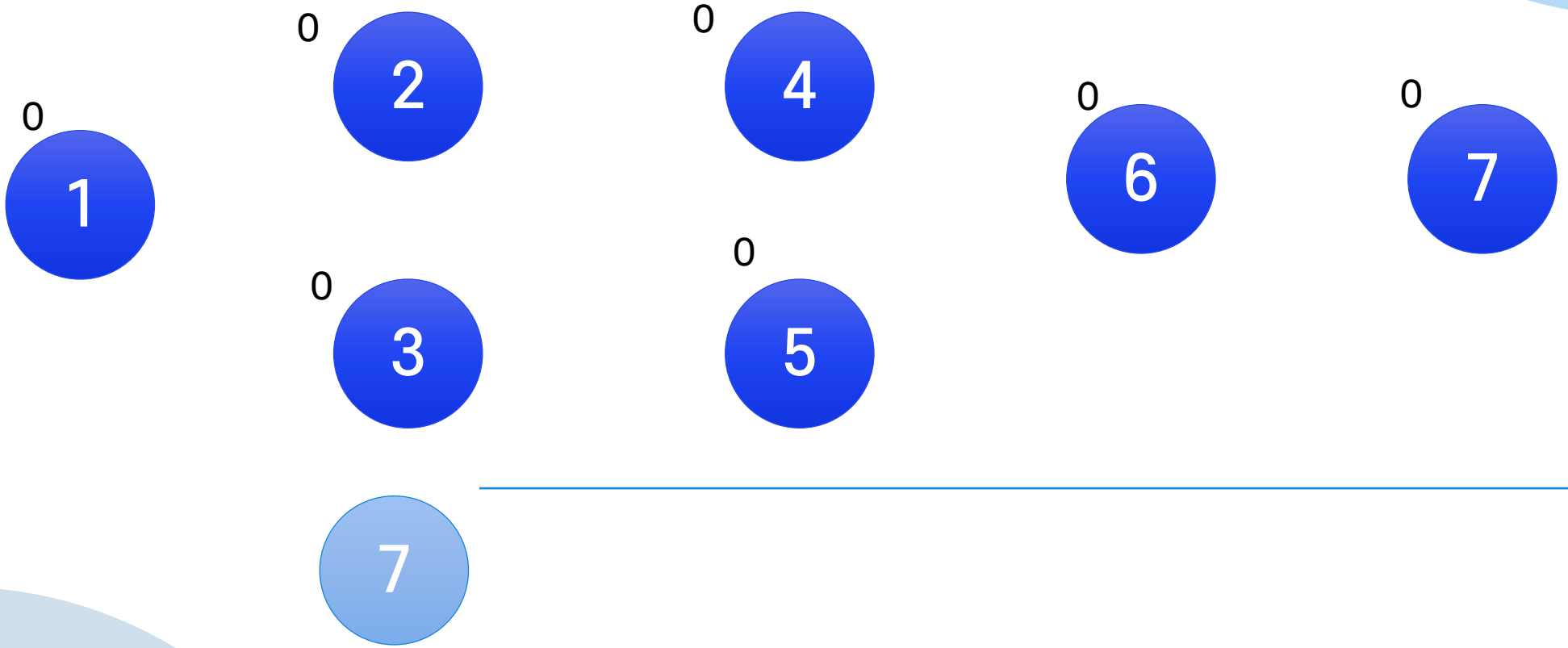


위상 정렬 - 예제

위상 정렬 - 예제



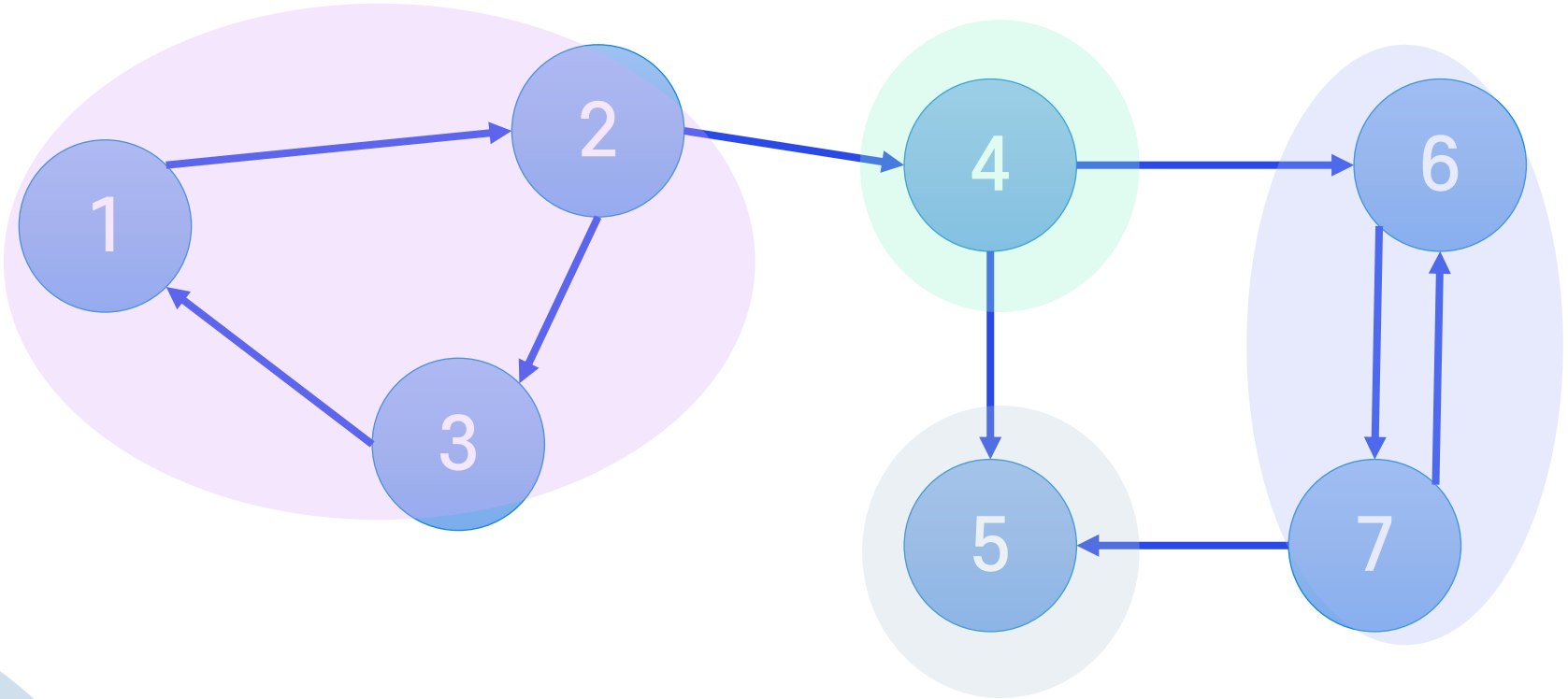
위상 정렬 - 예제



강한 연결 요소(SCC) - SCC란?

- 강한 연결 요소 (Strongly Connected Component – SCC)
- 강하게 결합된 정점 집합을 의미
- 같은 SCC에 속하는 두 정점은 서로 도달 가능
- 사이클이 발생하는 경우 무조건 SCC에 해당
- 무향 그래프→그래프 전체가 SCC
- 주로 방향 그래프에서 사용
- 코사라주 알고리즘, 타잔 알고리즘으로 구현

강한 연결 요소(SCC)



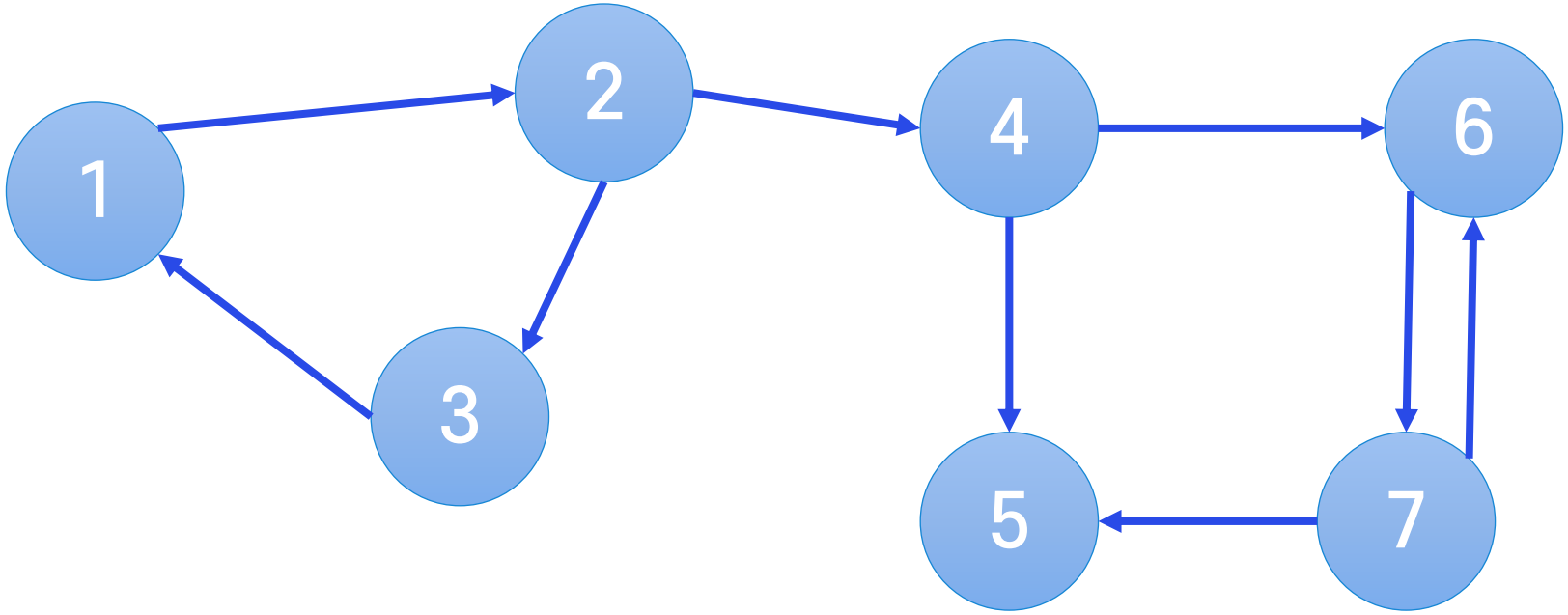
강한 연결 요소(SCC) - 코사라주

- 첫 번째 DFS
  - DFS를 하며 방문 체크
  - 인접한 정점 중 방문하지 않은 정점 방문
  - 스택에 현재 노드 삽입
- 두 번째 DFS
  - DFS를 하며 방문 체크
  - SCC의 부분 scc에 현재 노드 삽입(같은 SCC에 속하는 정점들)
  - 역방향 그래프에서 인접한 정점 중 방문하지 않은 정점을 방문
  - 더 이상 방문할 정점이 없으면 함수를 끝내고 현재 부분 scc를 추가

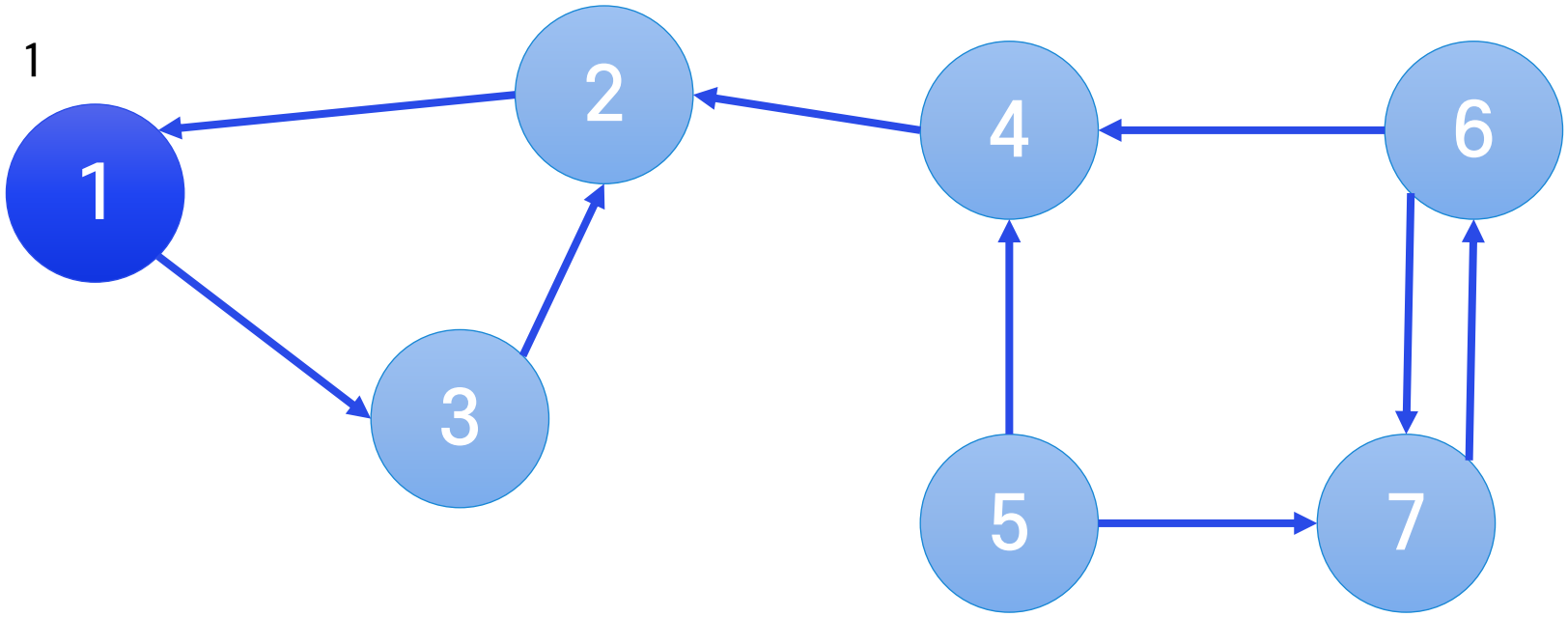
강한 연결 요소(SCC) - 코사라주

- 그래프 정보를 저장(정방향, 역방향)
- 각 정점들에 대하여 방문하지 않은 노드면 첫 번째 DFS 실행
- 스택에서 하나씩 꺼내며 방문하지 않은 노드면 두 번째 DFS 실행
- 저장된 SCC 그룹마다 정렬
- 전체 SCC 정렬 수행
- 시간복잡도 : O(V + E)

강한 연결 요소(SCC) - 코사라주

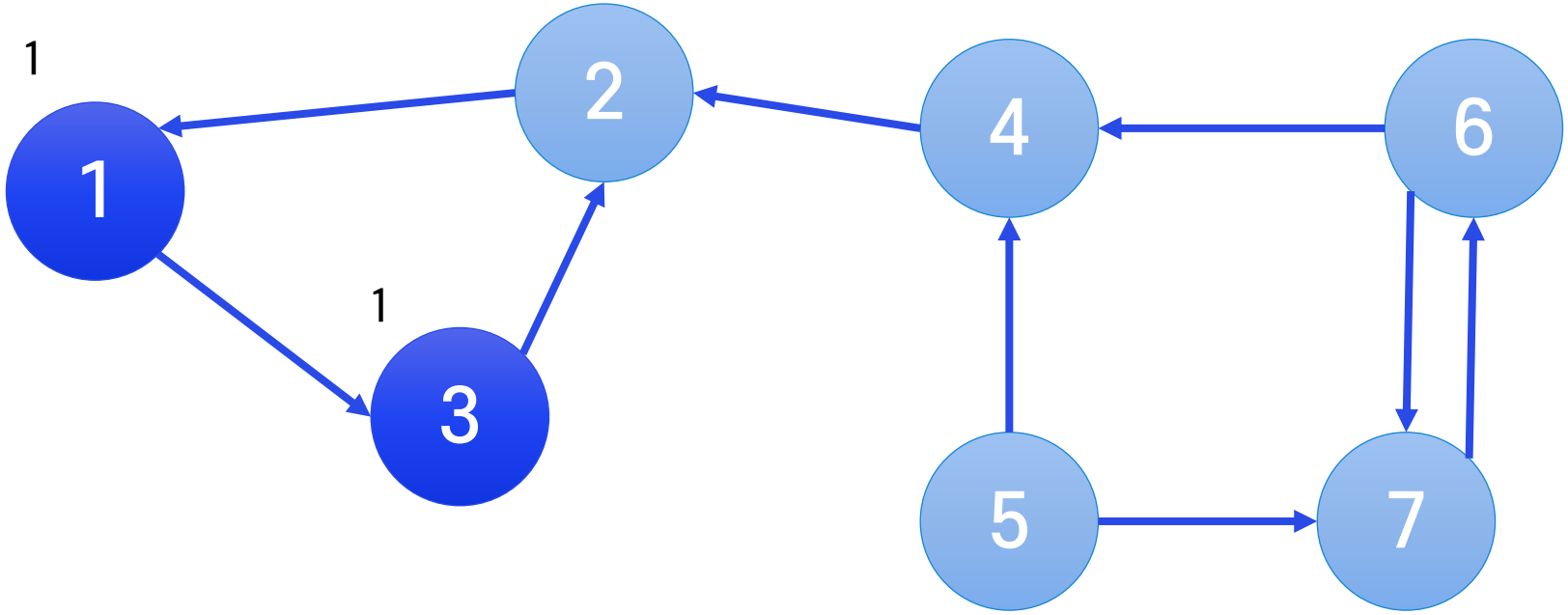


강한 연결 요소(SCC) - 코사라주



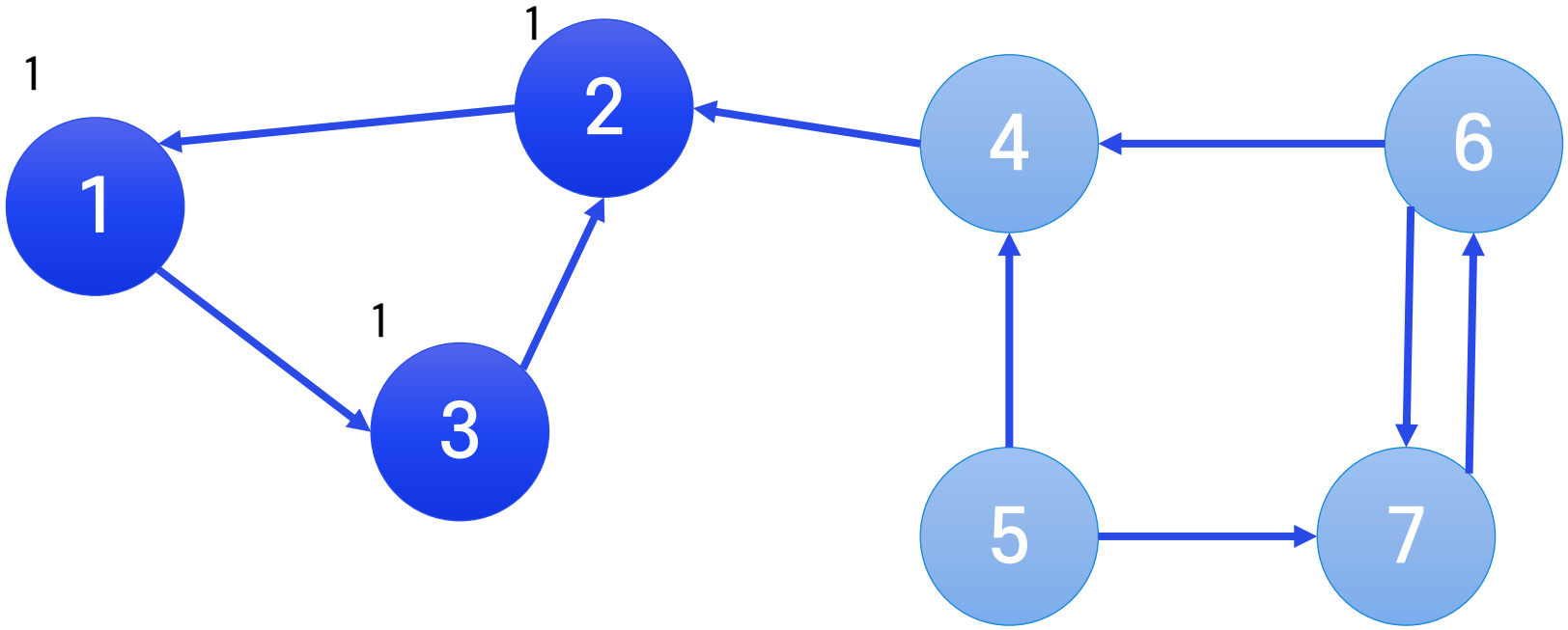
Group : 1

강한 연결 요소(SCC) - 코사라주



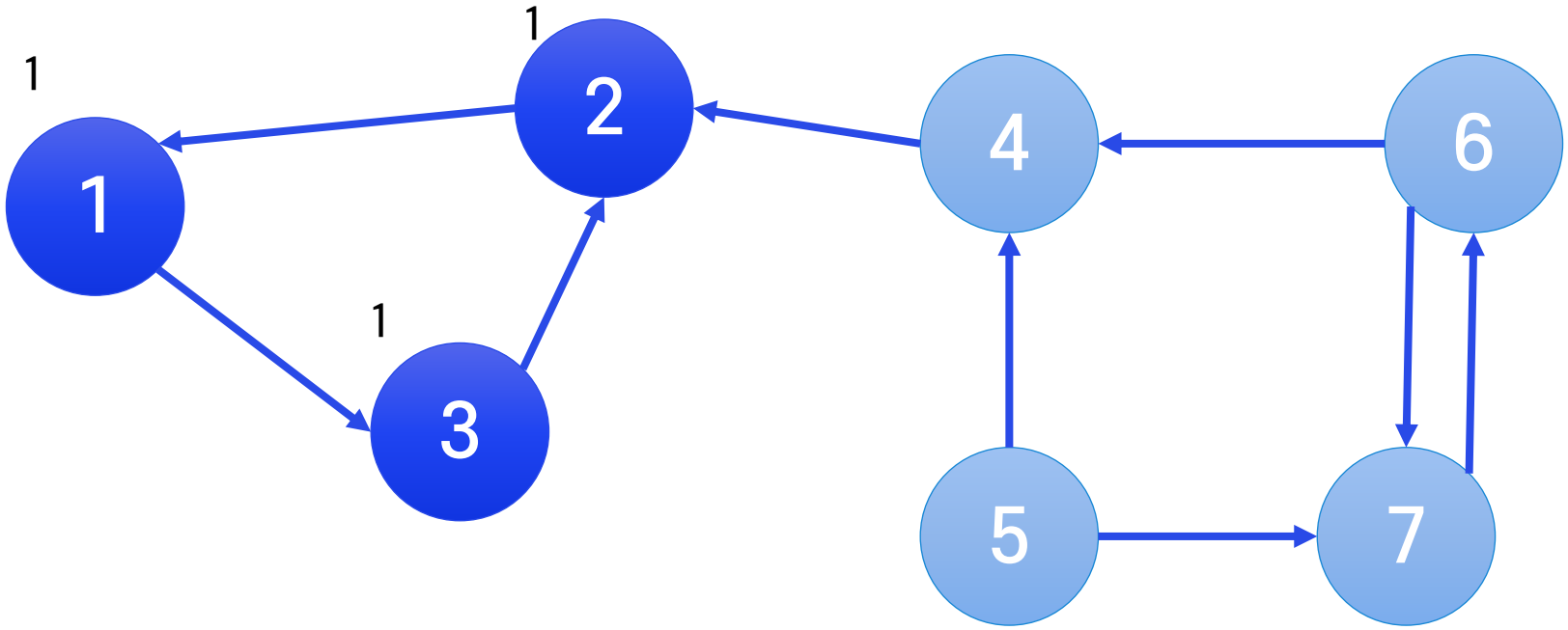
Group : 1

강한 연결 요소(SCC) - 코사라주

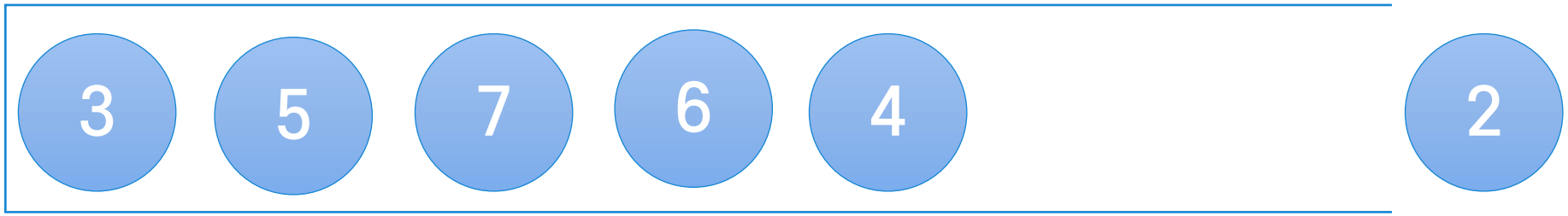


Group : 1

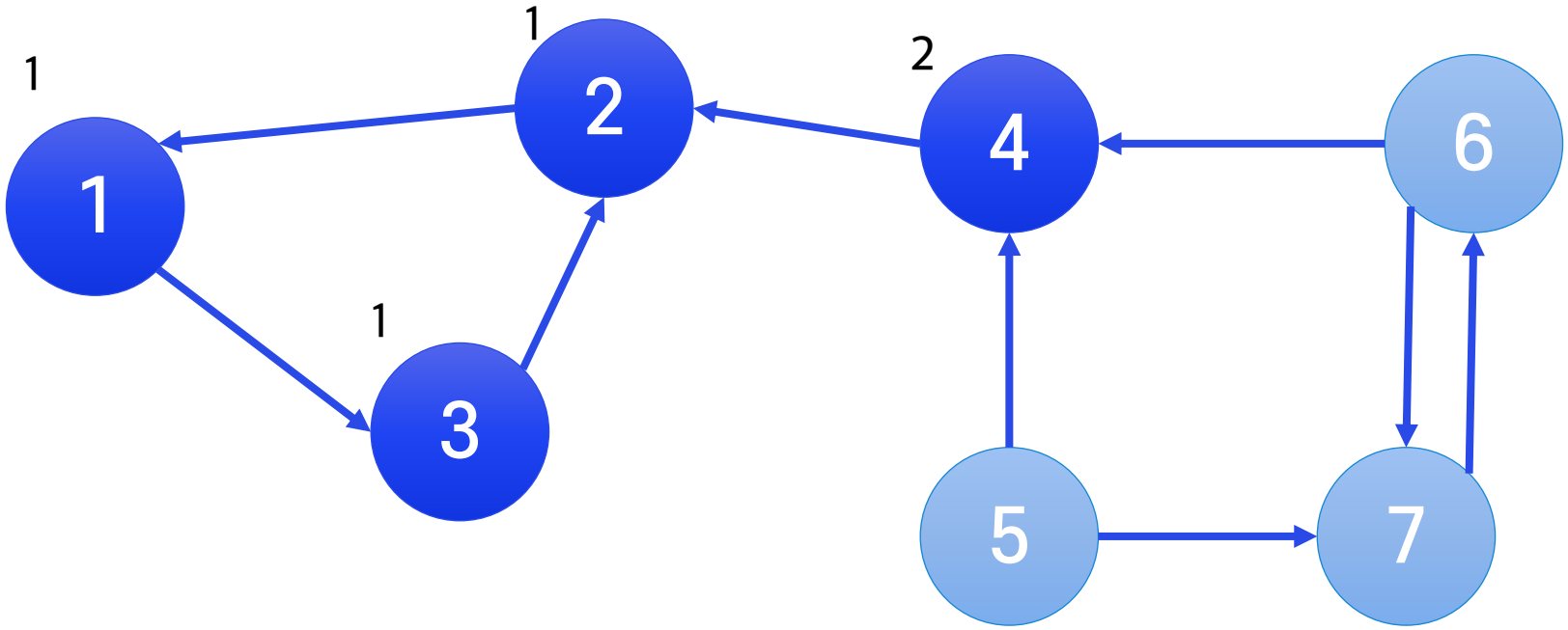
강한 연결 요소(SCC) - 코사라주



Group : 1

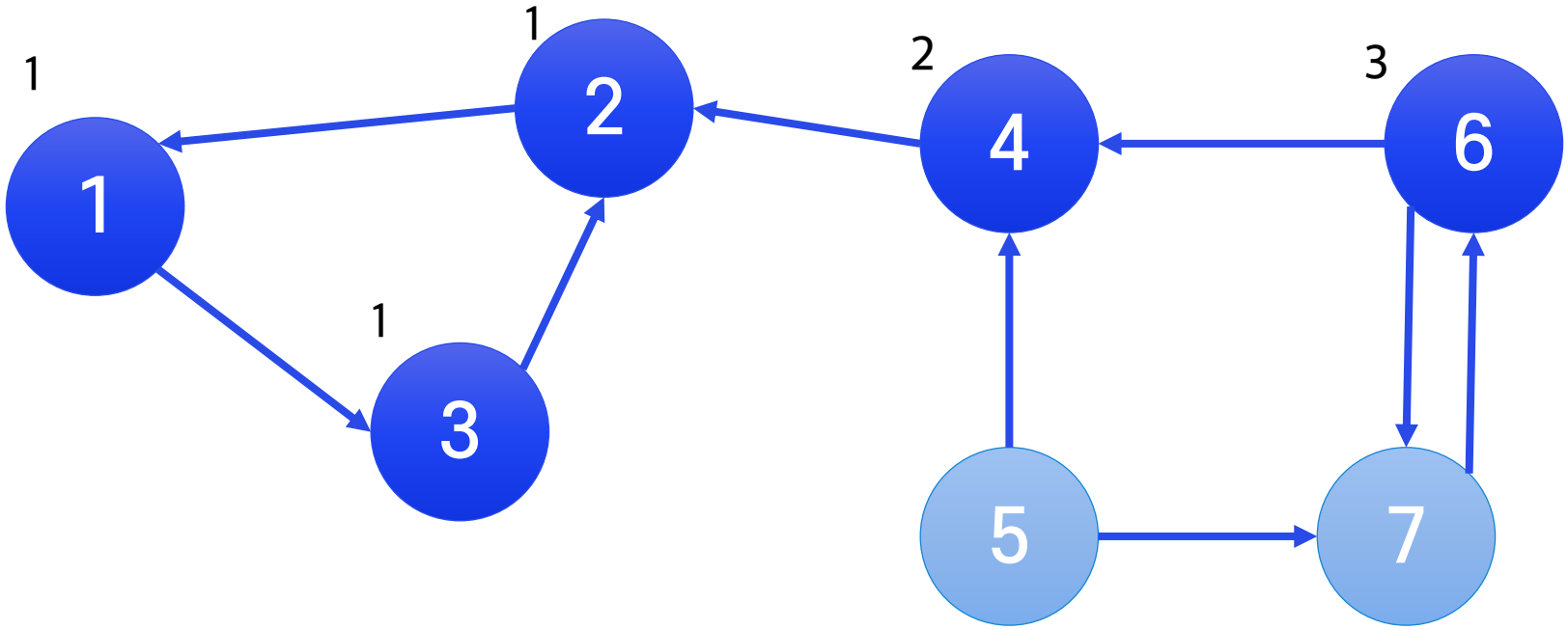


강한 연결 요소(SCC) - 코사라주



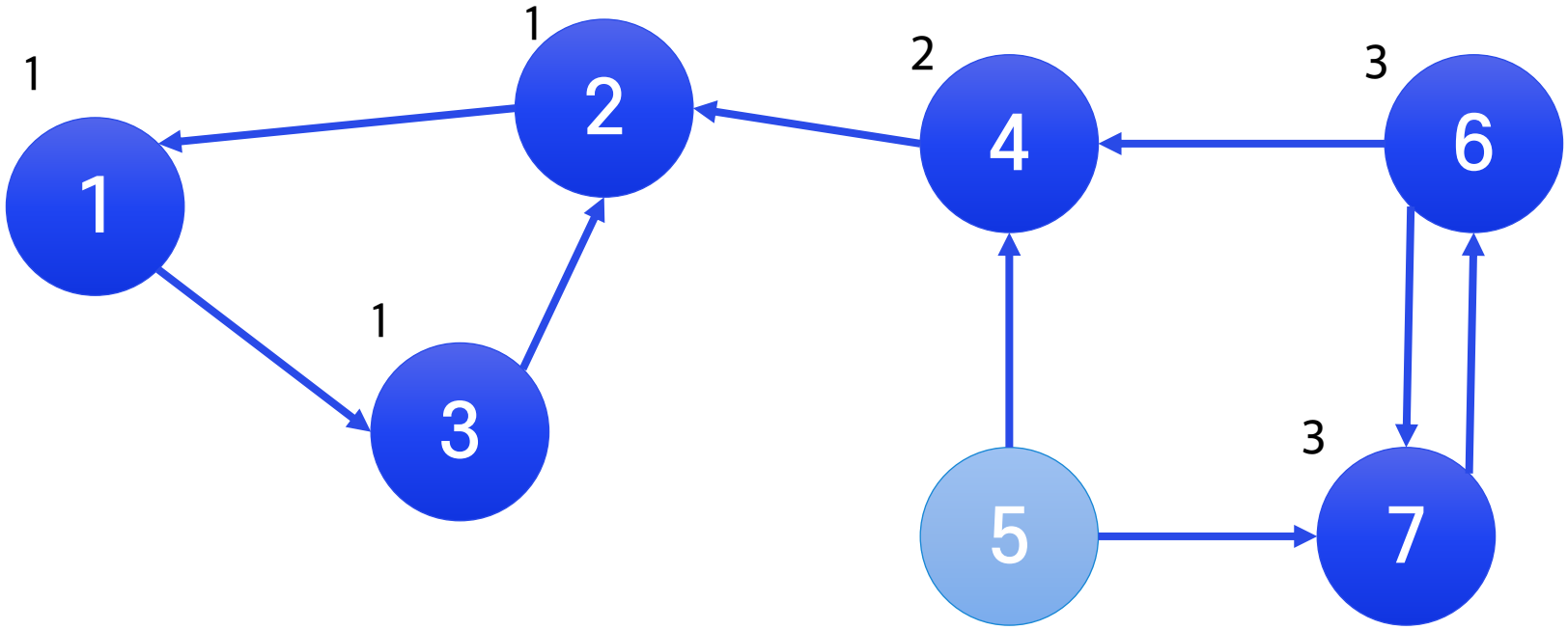
Group : 2

강한 연결 요소(SCC) - 코사라주



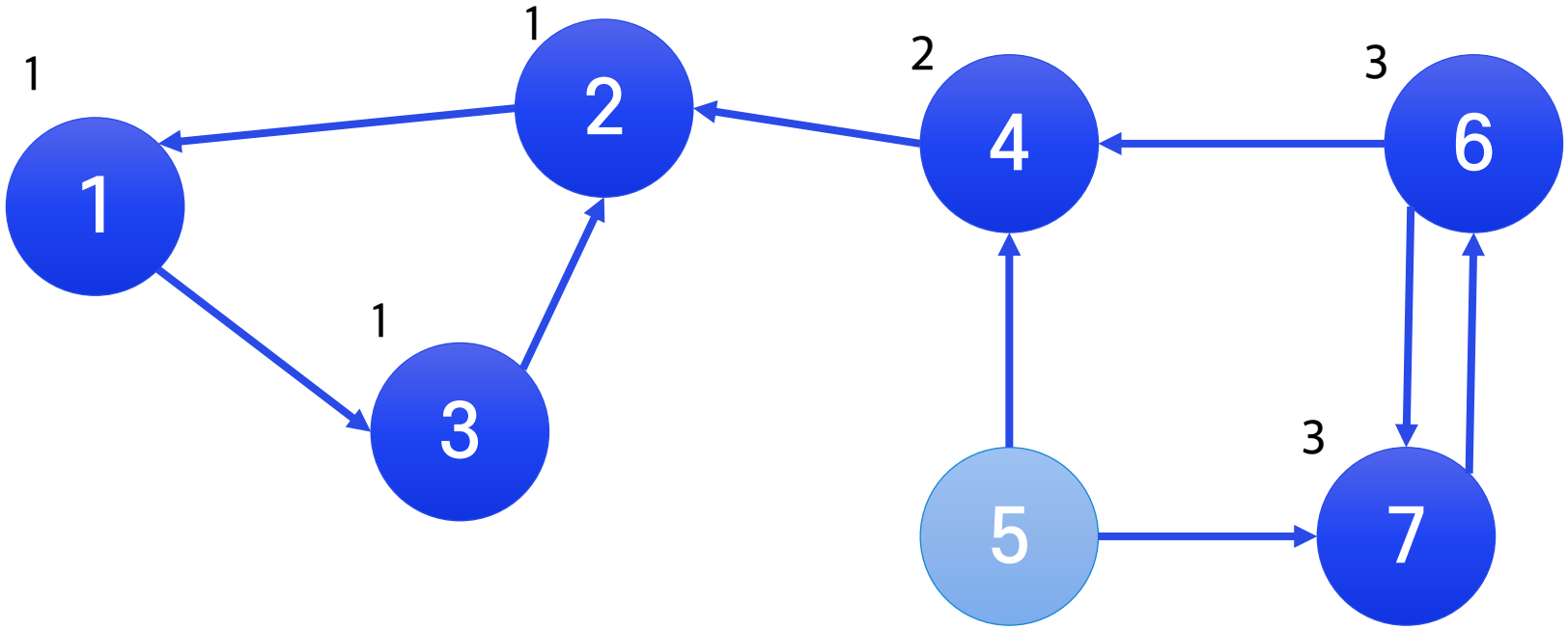
Group : 3

강한 연결 요소(SCC) - 코사라주



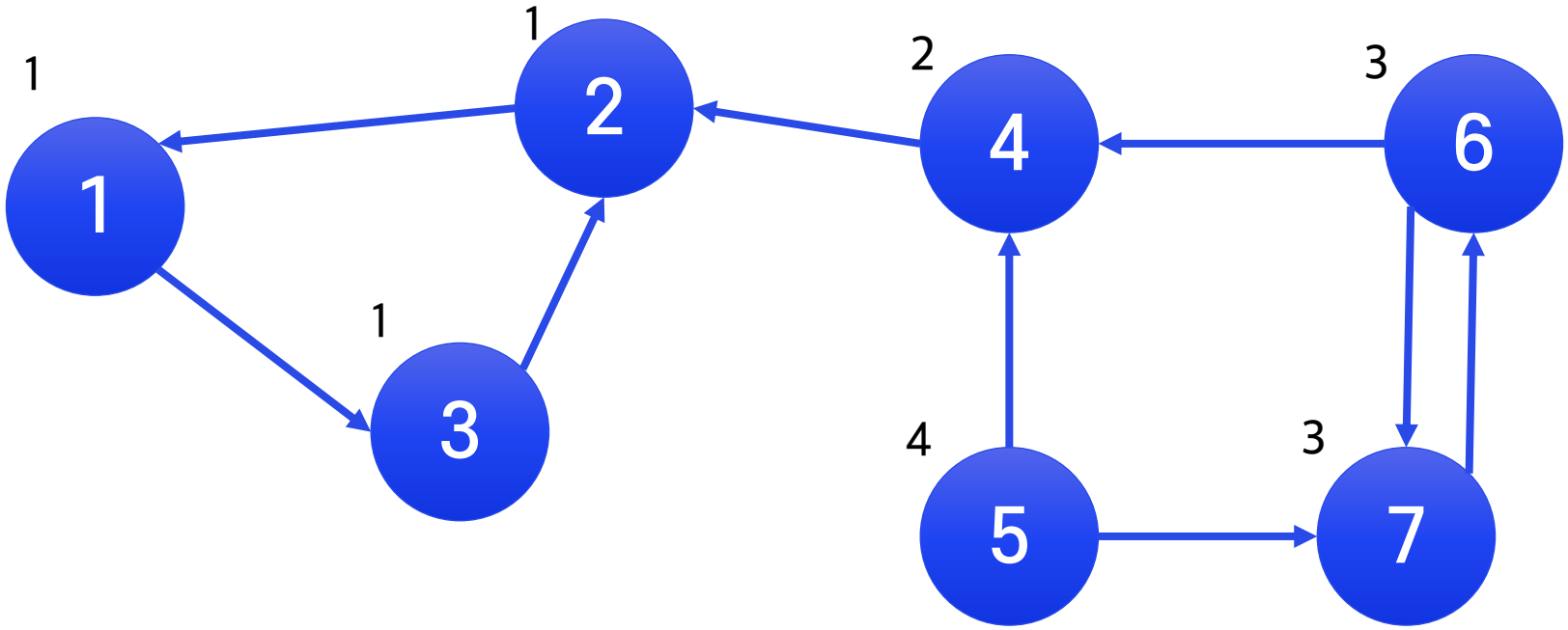
Group : 3

강한 연결 요소(SCC) - 코사라주



Group : 3

강한 연결 요소(SCC) - 코사라주

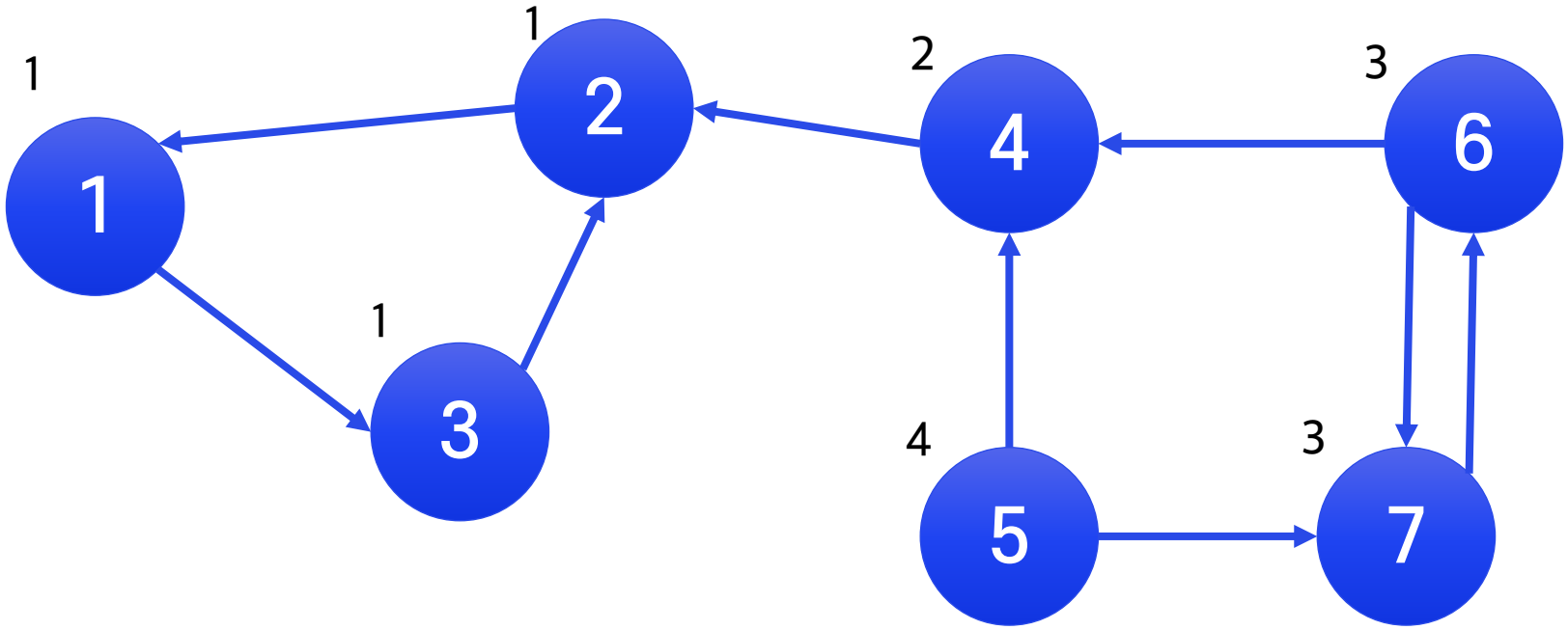


Group : 4

3

5

강한 연결 요소(SCC) - 코사라주



Group : 4

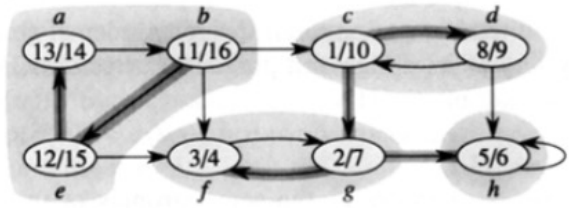
강한 연결 요소(SCC) - 코사라주

BOJ 2150

문제

방향 그래프가 주어졌을 때, 그 그래프를 SCC들로 나누는 프로그램을 작성하시오.

방향 그래프의 SCC는 우선 정점의 최대 부분집합이며, 그 부분집합에 들어있는 서로 다른 임의의 두 정점 u, v에 대해서 u에서 v로 가는 경로와 v에서 u로 가는 경로가 모두 존재하는 경우를 말한다.



예를 들어 위와 같은 그림을 보자. 이 그래프에서 SCC들은 {a, b, e}, {c, d}, {f, g}, {h} 가 있다. 물론 h에서 h로 가는 간선이 없는 경우에도 {h}는 SCC를 이룬다.

입력

첫째 줄에 두 정수 V(1 ≤ V ≤ 10,000), E(1 ≤ E ≤ 100,000)가 주어진다. 이는 그래프가 V개의 정점과 E개의 간선으로 이루어져 있다는 의미이다. 다음 E개의 줄에는 간선에 대한 정보를 나타내는 두 정수 A, B가 주어진다. 이는 A번 정점과 B번 정점이 연결되어 있다는 의미이다. 이때 방향은 A → B가 된다.

정점은 1부터 V까지 번호가 매겨져 있다.

출력

첫째 줄에 SCC의 개수 K를 출력한다. 다음 K개의 줄에는 각 줄에 하나의 SCC에 속한 정점의 번호를 출력한다. 각 줄의 끝에는 -1을 출력하여 그 줄의 끝을 나타낸다. 각각의 SCC를 출력할 때 그 안에 속한 정점들은 오름차순으로 출력한다. 또한 여러 개의 SCC에 대해서는 그 안에 속해있는 가장 작은 정점의 정점 번호 순으로 출력한다.

예제 입력 1 복사

```
7 9
1 4
4 5
5 1
1 6
6 7
2 7
7 3
3 7
7 2
```

예제 출력 1 복사

```
3
1 4 5 -1
2 3 7 -1
6 -1
```

강한 연결 요소(SCC) -

```cpp
const int MAX = 1e4 + 7;
int N, M;
int scc_cnt, id;
vector<int> edge[MAX], revEdge[MAX];
 // 정방향, 역방향 간선
vector<int> scc[MAX];
 // scc[i] : i번째 scc에 속한 정점들
bool visited[MAX];
int group[MAX];
stack<int> st;
```

```cpp
void dfs (int node) { // 첫 번째 DFS
  visited[node] = true;
  for (auto &next : edge[node]) {
    if (!visited[next]) dfs (next);
  }
  st.push (node); // 스택에 push
}

void dfsRev (int node) { // 두 번째 DFS
  visited[node] = true;
  group[node] = scc_cnt; // 그룹 번호 저장
  scc[scc_cnt].push_back (node);
 // scc에 추가
  for (auto &next : revEdge[node]) {
    if (!visited[next]) dfsRev (next);
  }
}
```

```cpp
cin >> N >> M;
  for (int i = 0; i < M; i++) {
    int u, v;
    cin >> u >> v;
    edge[u].push_back (v); // 정방향
    revEdge[v].push_back (u); // 역방향
  }
  for (int i = 1; i <= N; i++) {
    if (!visited[i]) dfs (i); // dfs1
  }
  memset (visited, false, sizeof (visited)); // 초기화
  while (!st.empty()) {
    int st_top = st.top();
    st.pop();
    if (!visited[st_top]) {
      scc_cnt++;
      dfsRev (st_top); // dfs2
      sort (scc[scc_cnt].begin(), scc[scc_cnt].end()); // 정렬
    }
  }
  sort (scc + 1, scc + scc_cnt + 1, [](vector<int> &i, vector<int
> &j) {
    return i[0] < j[0];
  }); // 정렬
  // 출력
  cout << scc_cnt << "\n";
  for (int i = 1; i <= scc_cnt; i++) {
    for (auto& it : scc[i]) {
      cout << it << " ";
    }
    cout << "-1\n";
  }
```

BOJ 2150

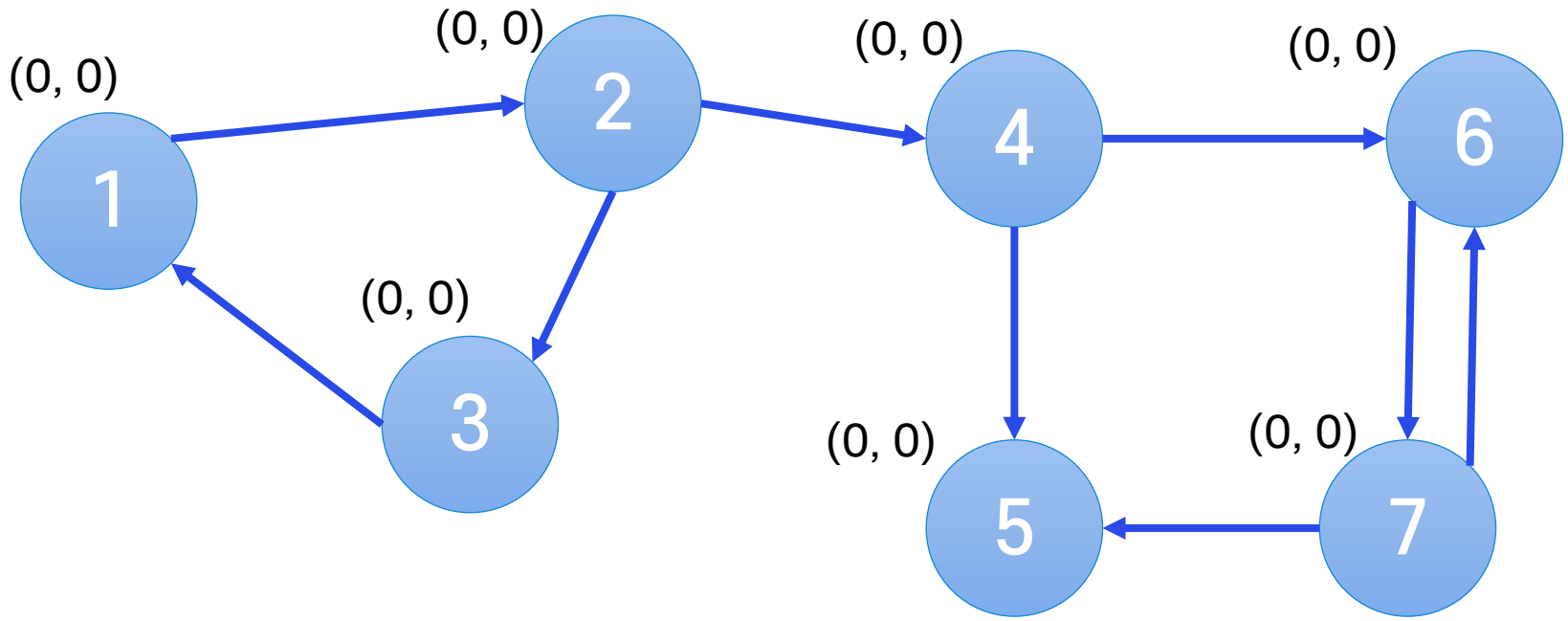
강한 연결 요소(SCC) - 타잔

- 시간 복잡도 : O(V + E)
- 타잔 알고리즘 DFS 한 번만 수행
- 기본적으로 노드마다 ID를 부여하고 parent 값을 지정
- 두 노드의 parent 값이 같다면 같은 SCC에 속함
- 이 때 parent 값은 visit이랑 같이 사용하거나 따로 id 값 저장
- 정방향 그래프에서 모든 방문하지 않은 노드들에 대해 DFS 수행

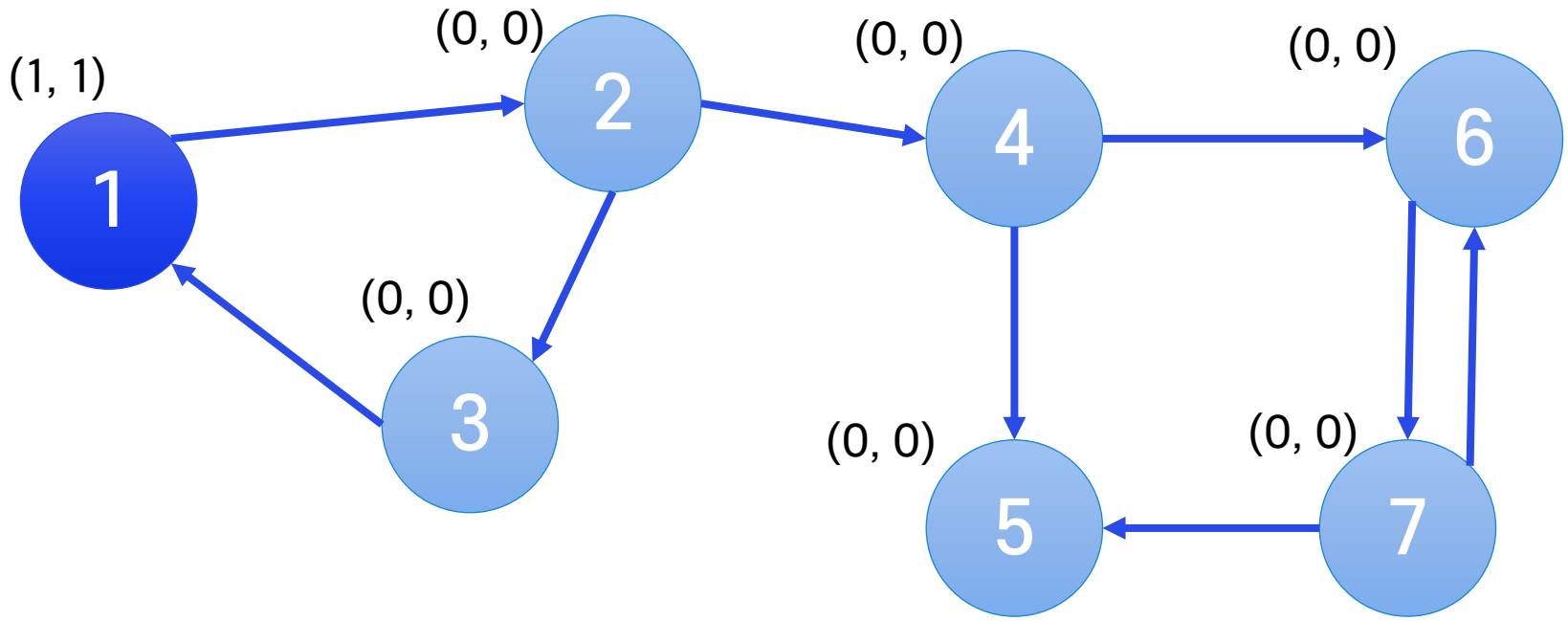
강한 연결 요소(SCC) - 타잔

- DFS
  - 현재 노드에 id 부여, 스택에 삽입
  - 인접한 노드들에 대해 다음을 수행
    - 방문하지 않은 노드라면 DFS 반환 값과 현재 parent 중 작은 값으로 parent 지정
    - 방문했고 현재 SCC에 속한다면 parent와 인접한 노드의 parent 중 작은 값으로 parent 지정
  - 만약 parent랑 자기 자신이랑 같으면 스택에서 자기 자신이 나올 때까지 pop하며 저장
  - 빼낸 노드들을 전체 SCC에 저장
  - 현재 parent 값 반환

강한 연결 요소(SCC) - 타잔

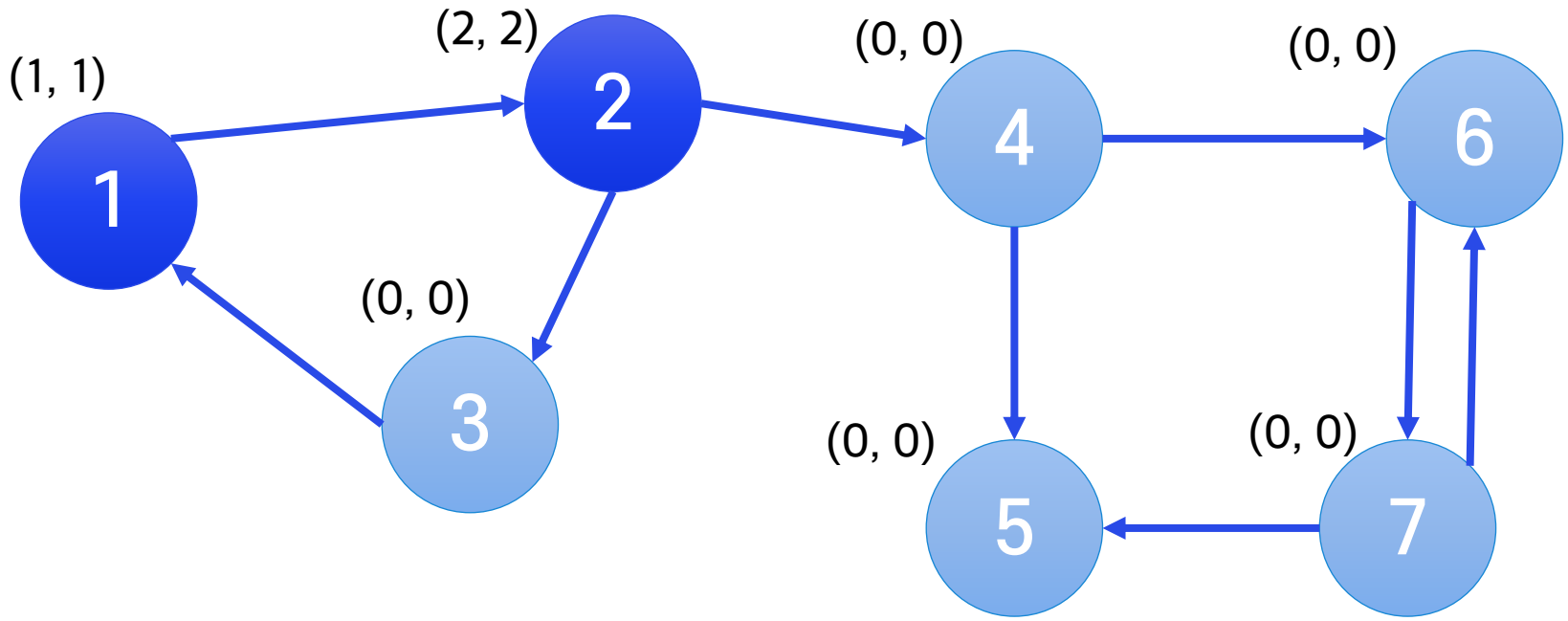


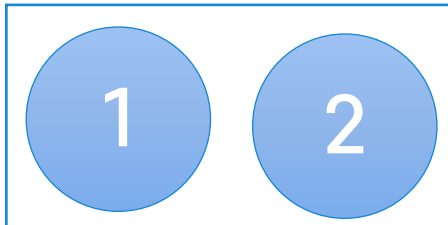
강한 연결 요소(SCC) - 타잔



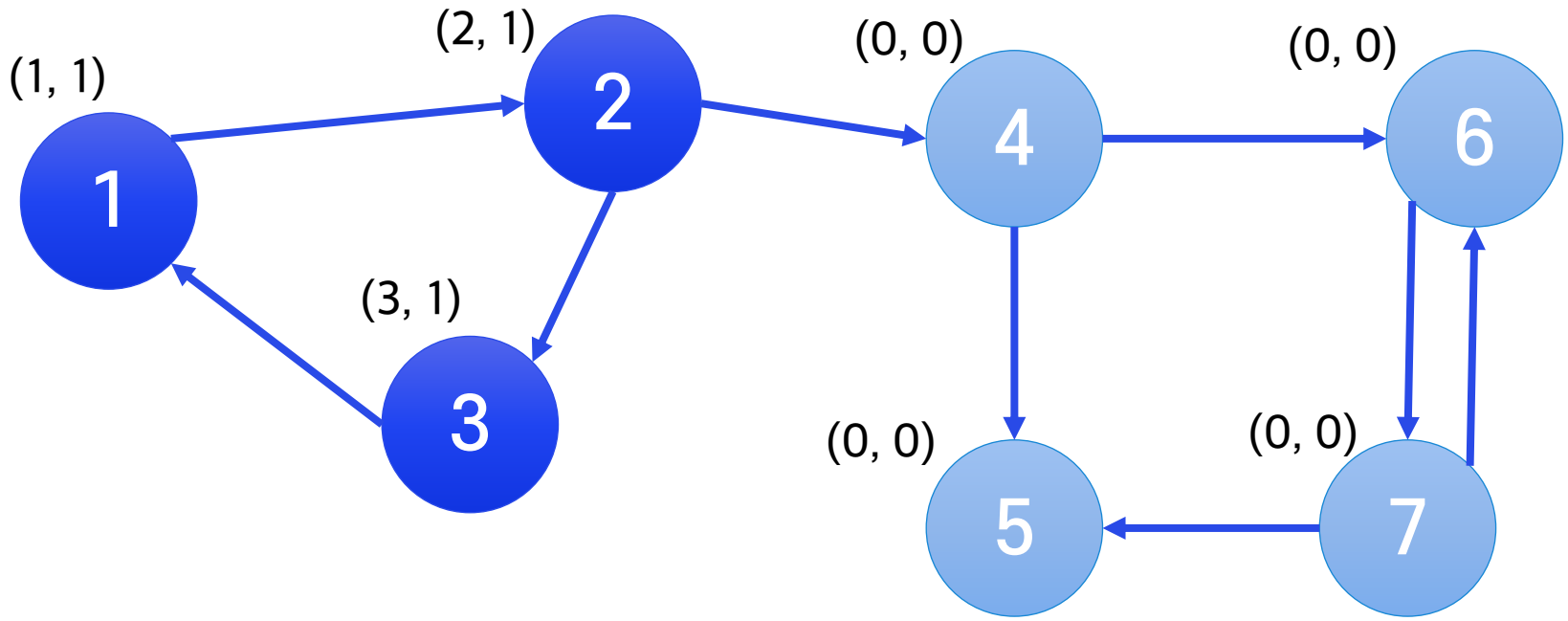
1

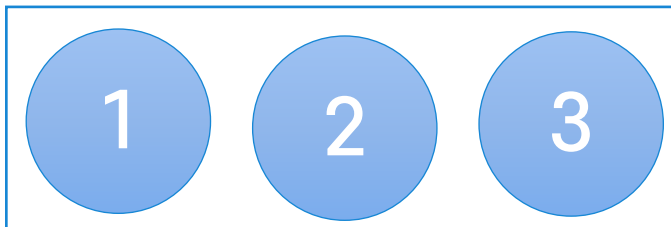
강한 연결 요소(SCC) - 타잔



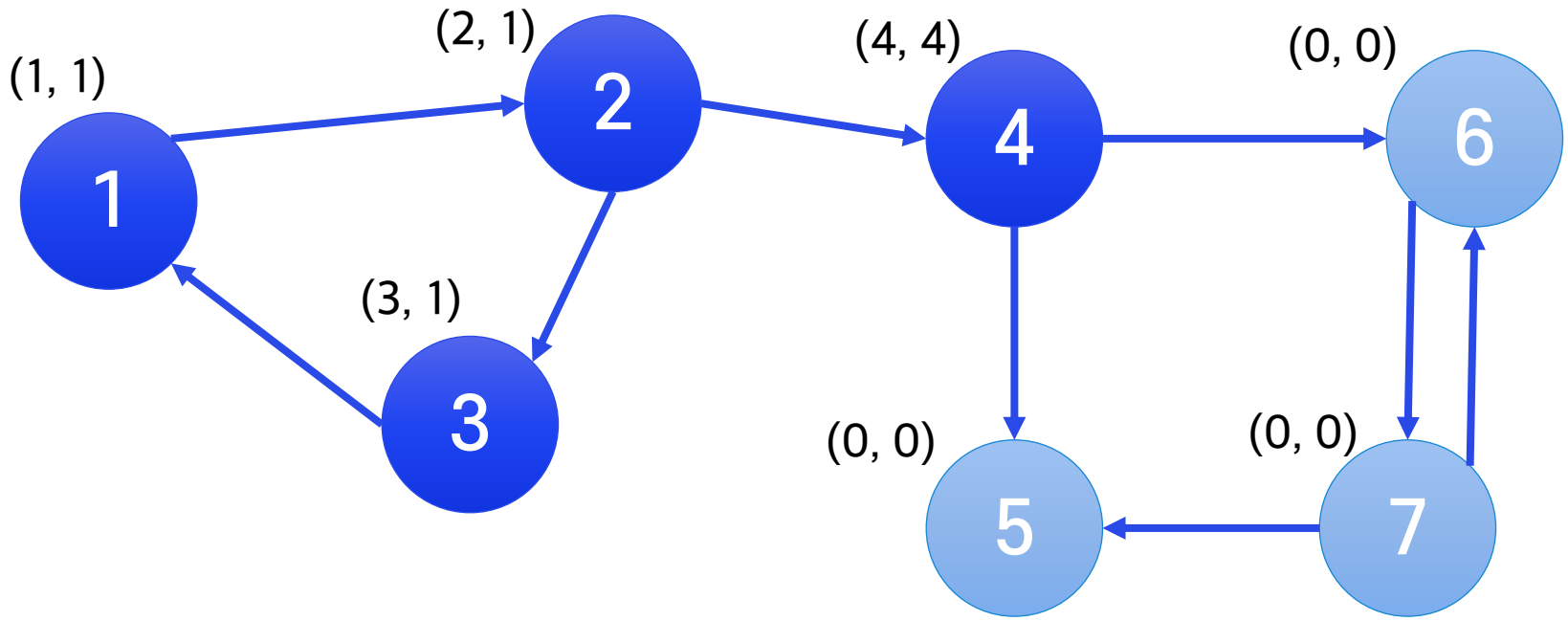


강한 연결 요소(SCC) - 타잔

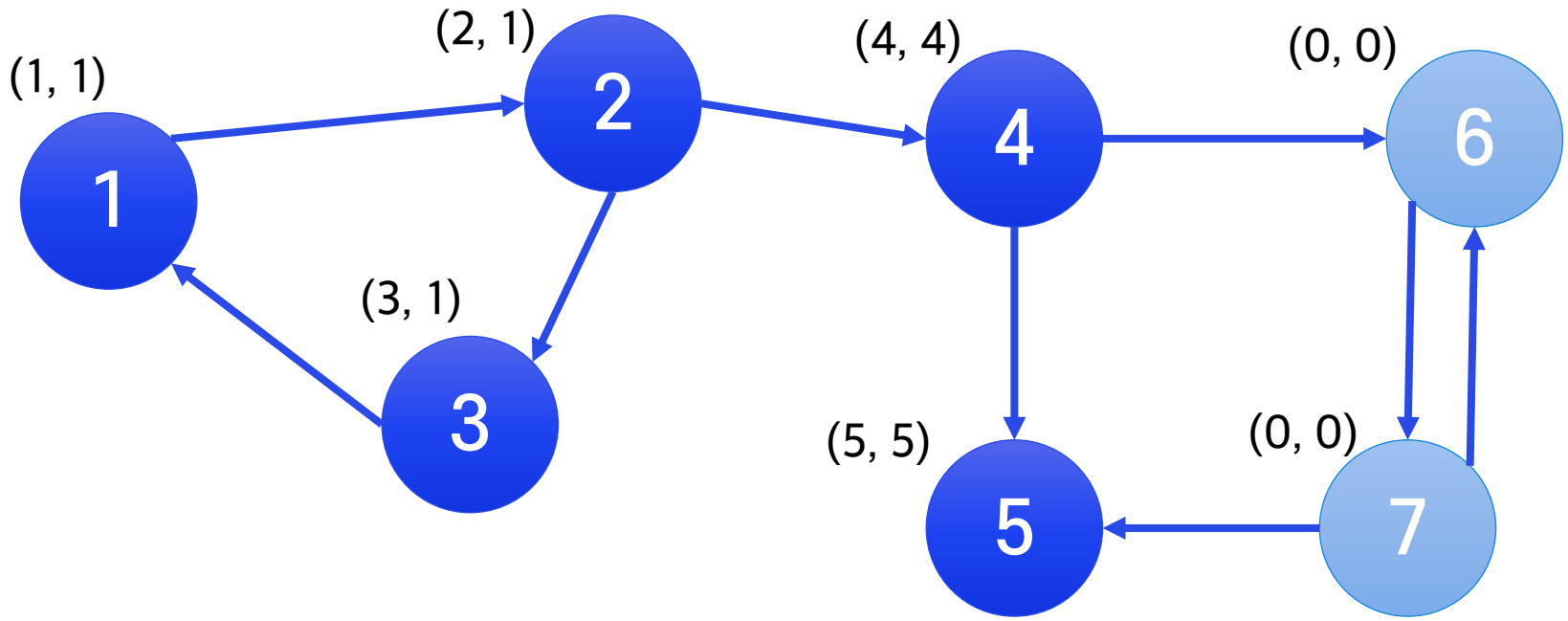


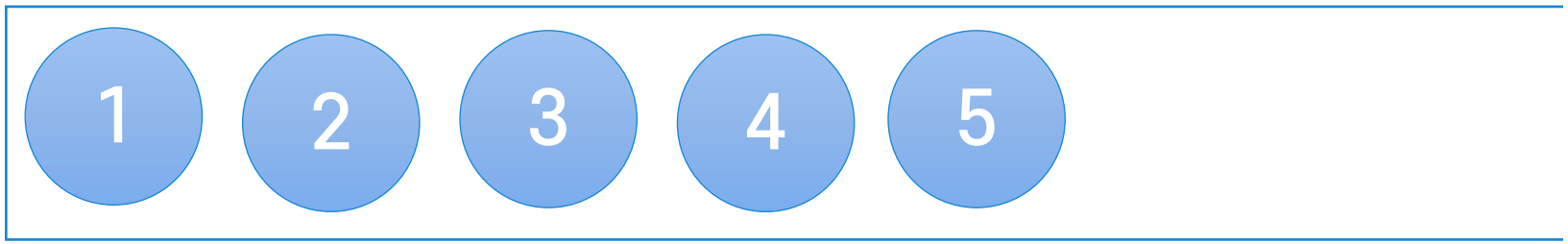


강한 연결 요소(SCC) - 타잔

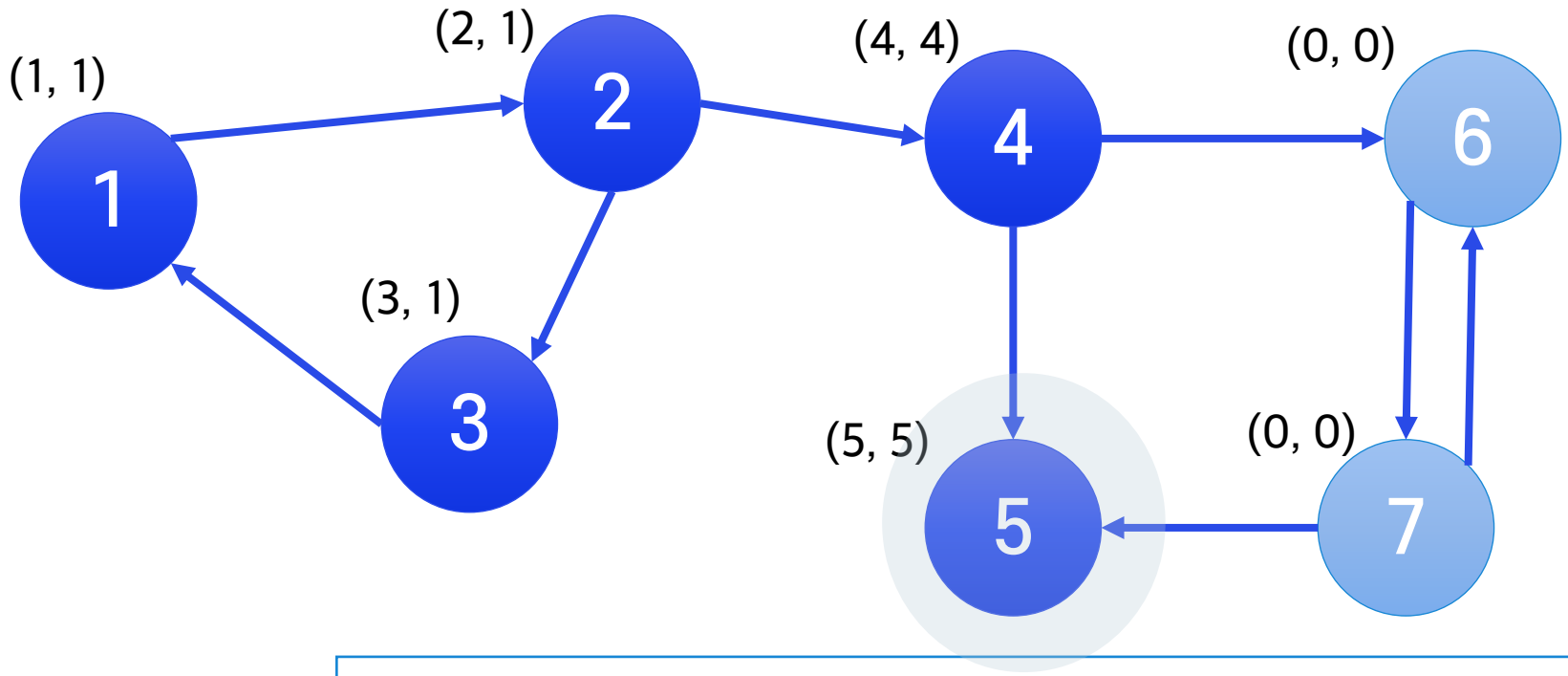


강한 연결 요소(SCC) - 타잔

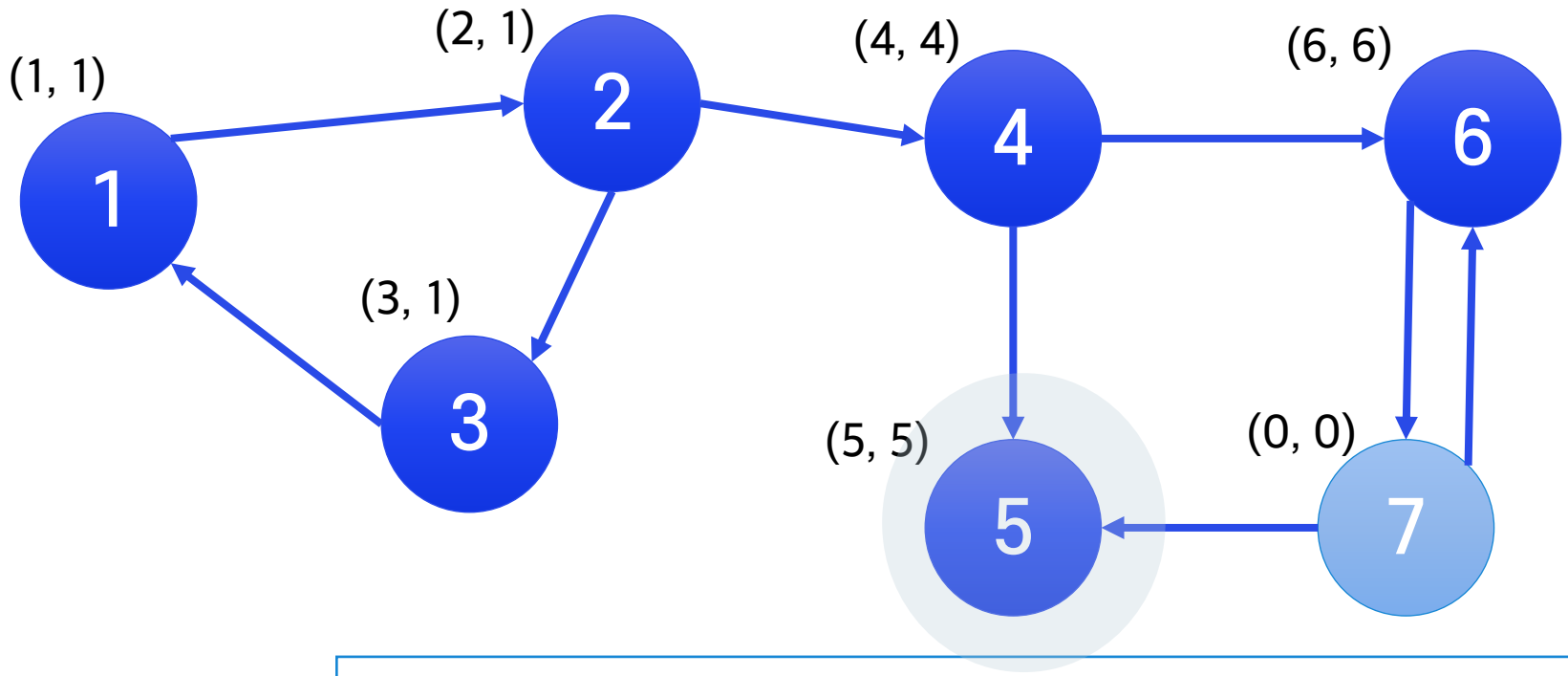




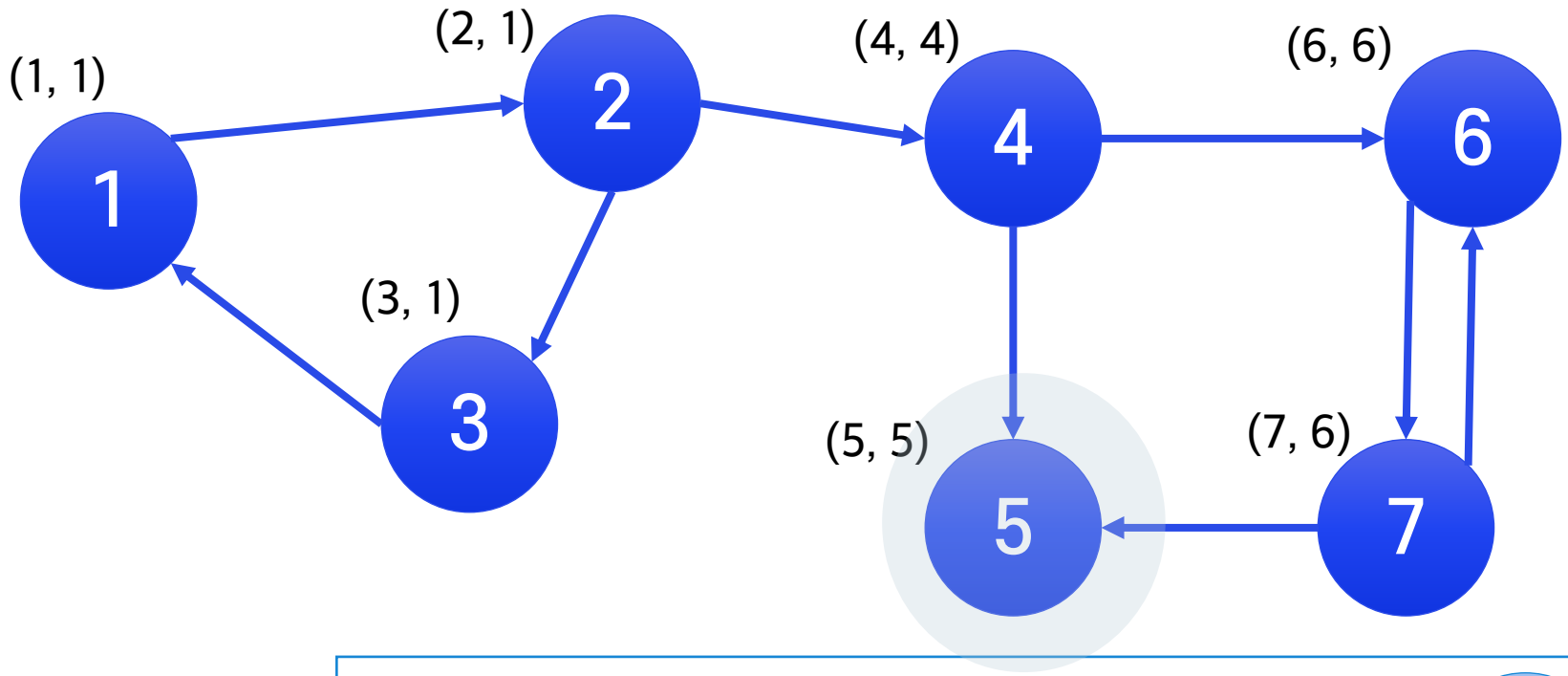
강한 연결 요소(SCC) - 타잔



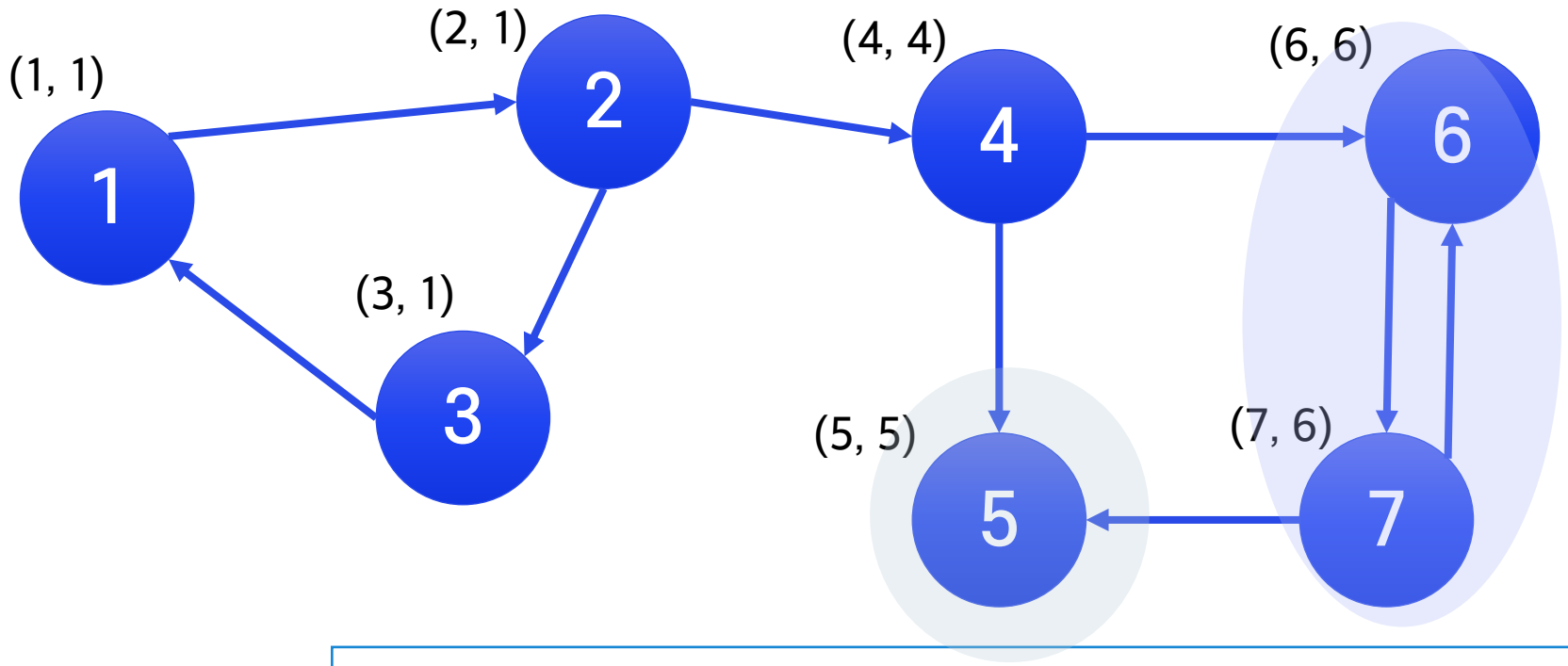
강한 연결 요소(SCC) - 타잔

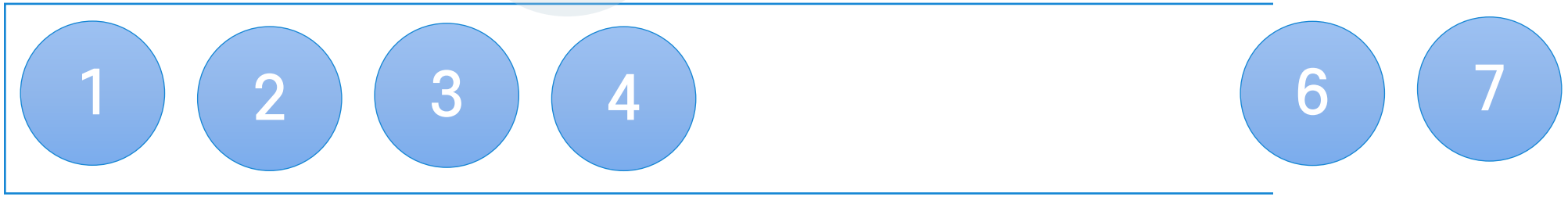


강한 연결 요소(SCC) - 타잔

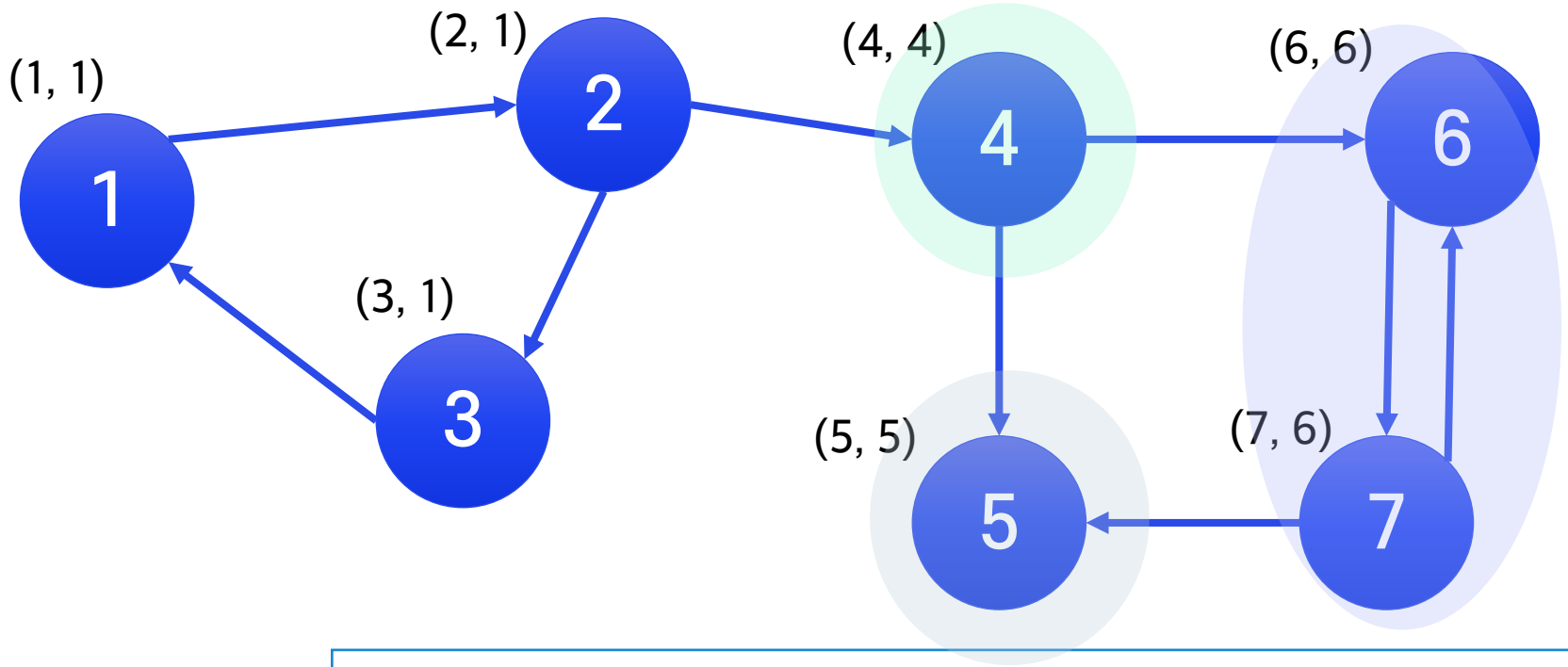


강한 연결 요소(SCC) - 타잔

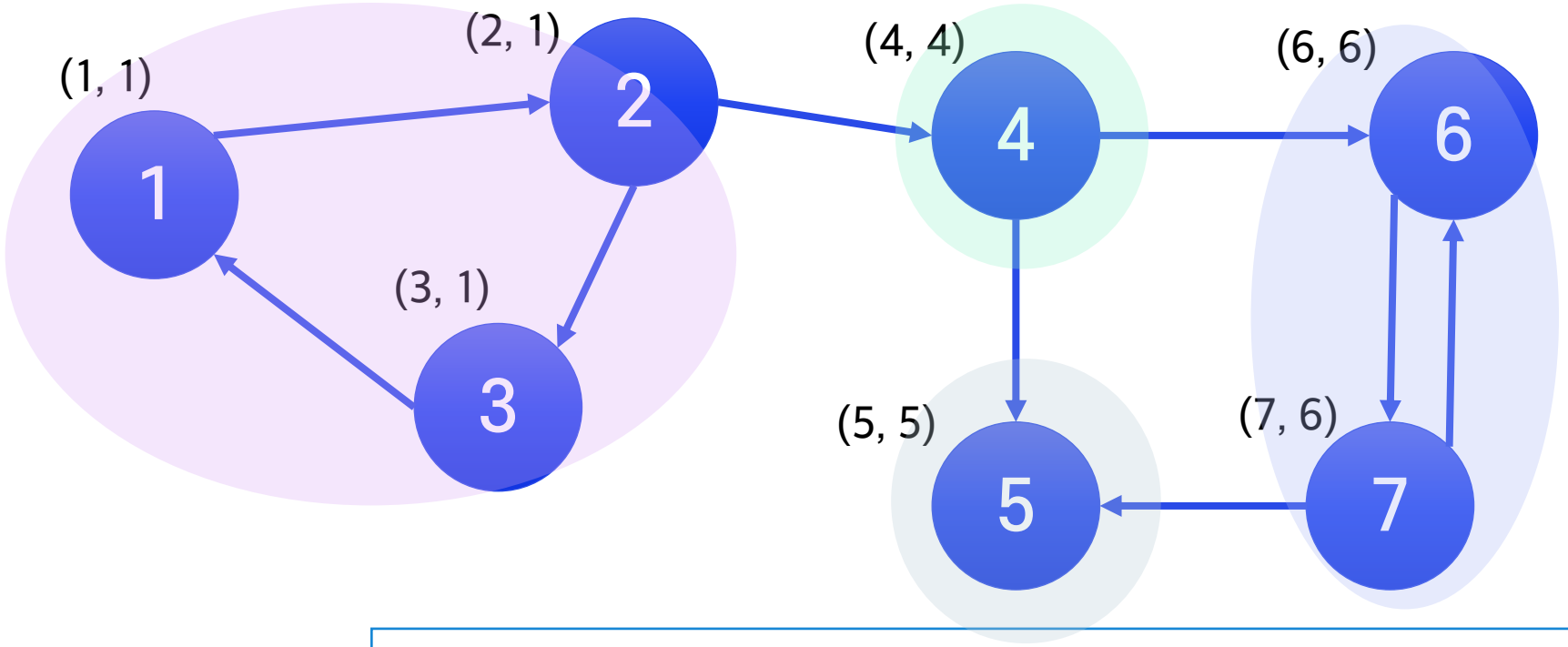




강한 연결 요소(SCC) - 타잔



강한 연결 요소(SCC) - 타잔



1 2 3

강한 연결 요소(SCC) - 타잔

```cpp
const int MAX = 1e4 + 7;
int N, M, id, scc_cnt;
vector<int> edge[MAX], scc[MAX];
bool visited[MAX], finished[MAX];
int node_id[MAX], group[MAX];
stack<int> st;
```

```cpp
int dfs (int node) {
  visited[node] = true;
  int parent = node_id[node] = ++id; // parent값
  st.push (node); // 스택에 push
  for (auto &next : edge[node]) {
    if (!visited[next]) // 한 번도 방문하지 않은 경우
      parent = min (parent, dfs (next));
    else if (!finished[next]) // 아직 끝나지는 않은 노드
      parent = min (parent, node_id[next]);
  }
  if (parent == node_id[node]) { // 자기 자신이 parent
    scc_cnt++;
    while (true) {
      int top = st.top();
      st.pop();
      scc[scc_cnt].push_back (top); // scc에 추가
      group[top] = scc_cnt;
      finished[top] = true; // 끝났음을 표시
      if (top == node) break; // 자기 자신일 때까지 pop
    }
    sort (scc[scc_cnt].begin(), scc[scc_cnt].end());
  }
  return parent;
}
```

BOJ 2150

강한 연결 요소(SCC) - 타잔

```
cin >> N >> M;
  for (int i = 0; i < M; i++) {
    int u, v;
    cin >> u >> v;
    edge[u].push_back (v);
  }
  for (int i = 1; i <= N; i++) {
    if (!visited[i]) dfs (i);
  }
  sort (scc + 1, scc + scc_cnt + 1, [](vector<int> &
i, vector<int> &j) {
    return i[0] < j[0];
  });
  cout << scc_cnt << "\n";
  for (int i = 1; i <= scc_cnt; i++) {
    for (auto& it : scc[i]) {
      cout << it << " ";
    }
    cout << "-1\n";
  }
```

BOJ 2150

SCC + 위상 정렬

- 우선 SCC를 수행하여 group 정보를 저장
- DAG를 만들기 위해 새로운 간선 벡터 scc_edge 사용
- 주어진 모든 간선에 대해 탐색
  - 두 노드가 다른 그룹이라면 간선 추가(group[u] -> group[v])
  - Indegree 배열 정보 추가(indegree[group[v]]++)
- 위상정렬 수행

SCC + 위상 정렬

```cpp
// Make SCC Graph
    for (int i = 1; i <= N; i++) {
        for (auto& it : adj[i]) {
            if (node_scc[it] != node_scc[i]) {
                scc_adj[node_scc[i]].push_back
(node_scc[it]);
                scc_indegree[node_scc[it]]++;
            }
        }
    }
    // 시작점 초기화
    queue<int> q;
    for (int i = 1; i <= scc_cnt; i++) {
        if (!scc_indegree[i]) {
            dp[i] = scc_size[i];
            come_from[i] = i;
            q.push (i);
        }
    }
```

```cpp
// Toplogy Sort
    while (!q.empty()) {
        int here = q.front();
        q.pop();

        if (here == node_scc[K]) break;

        for (auto& it : scc_adj[here]) {
            if (dp[it] < dp[here] + scc_size[it]) {
                come_from[it] = here;
                dp[it] = dp[here] + scc_size[it];
            }
        if (--scc_indegree[it] == 0) q.push (it);
        }
    }
```

BOJ 19649

추천 문제

- 위상정렬
  - BOJ 1516 게임 개발(G3)
- 강한 연결 요소
  - BOJ 4196 도미노(P4)
  - BOJ 3977 축구 전술(P4)
- 강한 연결 요소 + 위상 정렬
  - BOJ 2152 여행 계획 세우기(P3)
  - BOJ 4013 ATM(P2)

추천 문제

* Hint

2152

- connected 배열로 S에서 갈 수 있는 그룹 체크
- dp[next] = max (dp[next], dp[cur] + groupSize[next]) 이용

4013

- 2152번 이용
- DP 점화식 수정


THANK
YOU