Chapter 02.
고급 자료구조

핵심 유형 문제풀이

핵심 유형 문제풀이 | 다양한 문제를 접하며 코딩 테스트에 익숙해지기

강사 나동빈

Chapter 02. 고급 자료구조

핵심 유형 문제풀이

사이클 게임: https://www.acmicpc.net/problem/20040

혼자 힘으로 풀어보기

문제 제목: 사이클 게임

문제 난이도: 중(Medium)

문제 유형: 자료구조, Union-Find

추천 풀이 시간: 50분

사이클 게임: https://www.acmicpc.net/problem/20040

문제 풀이 핵심 아이디어

- 본 문제는 Union-Find 자료구조를 이용해 해결할 수 있는 문제입니다.
- 두 명의 플레이어가 차례를 반복하는 것을 “간선”으로 간주할 수 있습니다.
- 결과적으로 그래프에 간선을 하나씩 추가하면서 **사이클을 판별**하면 됩니다.
- **사이클**: 한 정점에서 출발해 출발점으로 돌아오는 것이 가능

사이클 게임: https://www.acmicpc.net/problem/20040

합 집합 찾기(Union-Find) 알고리즘

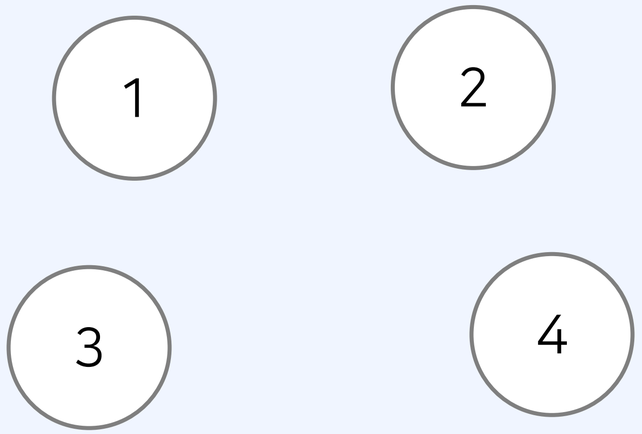
- 원소들의 “연결 여부”를 처리하는 알고리즘입니다.
- (그림 가정) 더 작은 원소를 부모로 삼도록 설정

부모 테이블

1	2	3	4
1	2	3	4

1 - 4

2 - 4



사이클 게임: https://www.acmicpc.net/problem/20040

합 집합 찾기(Union-Find) 알고리즘

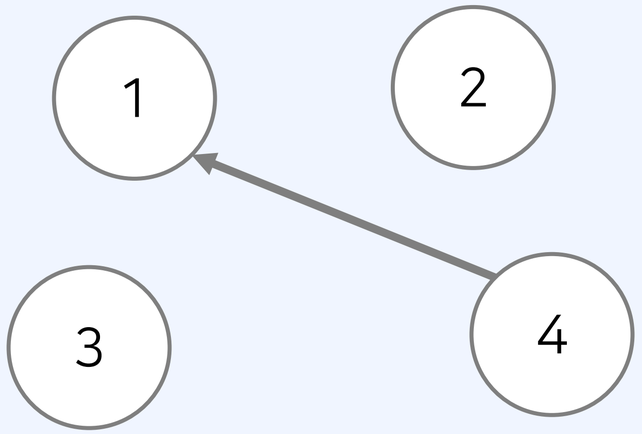
- 원소들의 “연결 여부”를 처리하는 알고리즘입니다.
- (그림 가정) 더 작은 원소를 부모로 삼도록 설정

부모 테이블

1	2	3	4
1	2	3	1

1 - 4

2 - 4



사이클 게임: https://www.acmicpc.net/problem/20040

합 집합 찾기(Union-Find) 알고리즘

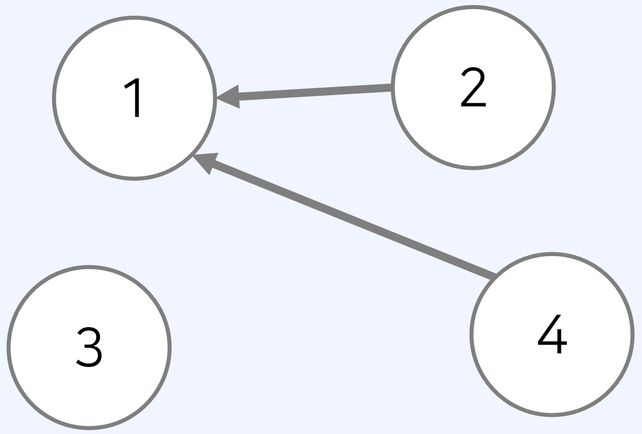
- 원소들의 “연결 여부”를 처리하는 알고리즘입니다.
- (그림 가정) 더 작은 원소를 부모로 삼도록 설정

부모 테이블

1	2	3	4
1	1	3	1

1 - 4

2 - 4



사이클 게임: https://www.acmicpc.net/problem/20040

합 집합 찾기(Union-Find) 알고리즘

- 원소들의 “연결 여부”를 처리하는 알고리즘입니다.
- (그림 가정) 더 작은 원소를 부모로 삼도록 설정

부모 테이블

1	2	3	4
1	1	3	1

결과: {1, 2, 4}, {3}

- Union-Find는 양방향 그래프에서 사이클을 판별을 위해 사용할 수 있습니다.
- Union-Find는 양방향 그래프에서 특정 두 원소의 연결성 판별을 위해 사용할 수 있습니다.

사이클 게임: https://www.acmicpc.net/problem/20040

문제 풀이 핵심 아이디어

[알고리즘 요약]

- 평면 상의 N개의 정점이 존재하고, M개의 간선이 존재하는 것으로 이해할 수 있다.
- 어느 세 점도 일직선에 놓이지 않는다.
- 단순히 A – B, B – C, C – A라면 사이클이 형성된다는 의미로 이해할 수 있다.
- 즉, 일반적인 그래프 형태로 간주할 수 있다.
- 기본적인 양방향 그래프에서의 사이클 판별을 위해 Union-Find를 사용하면 된다.

사이클 게임: https://www.acmicpc.net/problem/20040

소스 코드


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline


# 특정 원소가 속한 집합을 찾기
def find_parent(parent, x):
    # 루트 노드가 아니라면
    if parent[x] != x:
        # 루트 노드를 찾을 때까지 재귀적으로 호출
        parent[x] = find_parent(parent, parent[x])
    return parent[x]

# 두 원소가 속한 집합을 합치기
def union_parent(parent, a, b):
    a = find_parent(parent, a)
    b = find_parent(parent, b)
    if a < b:
        parent[b] = a
    else:
        parent[a] = b
```

```python
# 정점의 개수 N과 게임(차례)의 수 M
n, m = map(int, input().split())
parent = [0] * n # 부모 테이블 초기화하기
# 부모 테이블상에서, 부모를 자기 자신으로 초기화
for i in range(n):
    parent[i] = i

cycle = False # 사이클 발생 여부
for i in range(m): # M은 합치기(Union) 연산의 수와 동일
    a, b = map(int, input().split())
    # 사이클이 발생한 경우 종료
    if find_parent(parent, a) == find_parent(parent, b):
        cycle = True
        print(i + 1) # 사이클이 처음으로 만들어진 차례의 번호
        break
    # 사이클이 발생하지 않았다면 합집합(Union) 연산 수행
    else:
        union_parent(parent, a, b)

if not cycle: # 사이클이 발생하지 않은 경우
    print(0)
```

혼자 힘으로 풀어보기

문제 제목: 연결 요소의 개수

문제 난이도: 중(Medium)

문제 유형: 자료구조, Union-Find, DFS/BFS

추천 풀이 시간: 40분

문제 풀이 핵심 아이디어

- 연결 요소의 개수를 세는 방법은 다양합니다.
- 본 문제는 Union-Find 자료구조를 이용해 해결할 수 있는 문제입니다.

문제 풀이 핵심 아이디어

[알고리즘 요약]

- 무방향 그래프에서 **연결 요소**(connected component)의 개수를 세는 문제다.
- 단순히 Union-Find **자료구조**를 사용하여 연결 요소의 개수를 셀 수 있다.
- 모든 합치기 연산을 수행한 뒤에 고유한 집합의 개수를 세면 된다.

연결 요소의 개수: https://www.acmicpc.net/problem/11724

소스 코드


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline


# 특정 원소가 속한 집합을 찾기
def find_parent(parent, x):
    # 루트 노드가 아니라면
    if parent[x] != x:
        # 루트 노드를 찾을 때까지 재귀적으로 호출
        parent[x] = find_parent(parent, parent[x])
    return parent[x]

# 두 원소가 속한 집합을 합치기
def union_parent(parent, a, b):
    a = find_parent(parent, a)
    b = find_parent(parent, b)
    if a < b:
        parent[b] = a
    else:
        parent[a] = b
```

```python
# 정점의 개수 N과 간선의 개수 M
n, m = map(int, input().split())
parent = [0] * (n + 1) # 부모 테이블 초기화하기

# 부모 테이블상에서, 부모를 자기 자신으로 초기화
for i in range(1, n + 1):
    parent[i] = i

for i in range(m): # M은 합치기(union) 연산의 수와 동일
    a, b = map(int, input().split())
    union_parent(parent, a, b) # a와 b를 연결하기

counter = set() # 고유한 집합의 수
for i in range(1, n + 1):
    # 고유한 집합 번호를 집합에 추가
    counter.add(find_parent(parent, i))
# 고유한 집합의 수 출력
print(len(counter))
```

알파벳 찾기: https://www.acmicpc.net/problem/10809

혼자 힘으로 풀어보기

문제 제목: 알파벳 찾기

문제 난이도: 하(Easy)

문제 유형: 구현, 문자열, 사전(dictionary) 자료형

추천 풀이 시간: 20분

알파벳 찾기: https://www.acmicpc.net/problem/10809

문제 풀이 핵심 아이디어

- 알파벳 a부터 z까지의 범위를 포함하는 배열을 선언한다.

1. 배열의 모든 원소의 초기값은 -1(등장하지 않음)로 설정한다.
2. 입력 문자열에 포함된 문자를 하나씩 확인하여, **각 문자가 처음 등장한 위치를 기록**한다.

소스 코드


```
import sys
input = sys.stdin.readline # 빠른 입력 함수 사용

arr = [-1] * 26 # a부터 z까지의 알파벳은 총 26개
data = input().strip() # 문자열 입력

for i in range(len(data)): # 문자를 하나씩 확인
    # 알파벳 a는 인덱스 0, z는 인덱스 25에 해당
    index = ord(data[i]) - ord('a')
    if arr[index] == -1: # 처음 등장한 알파벳이라면
        arr[index] = i # 등장한 위치 기록

for x in arr: # 각 알파벳이 처음 등장하는 위치 출력
    print(x, end=' ')
```