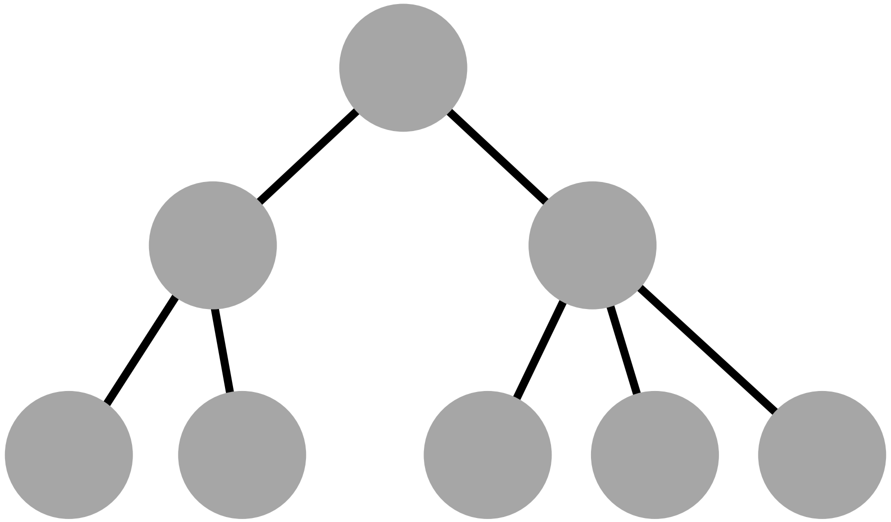
Bipartite Graph

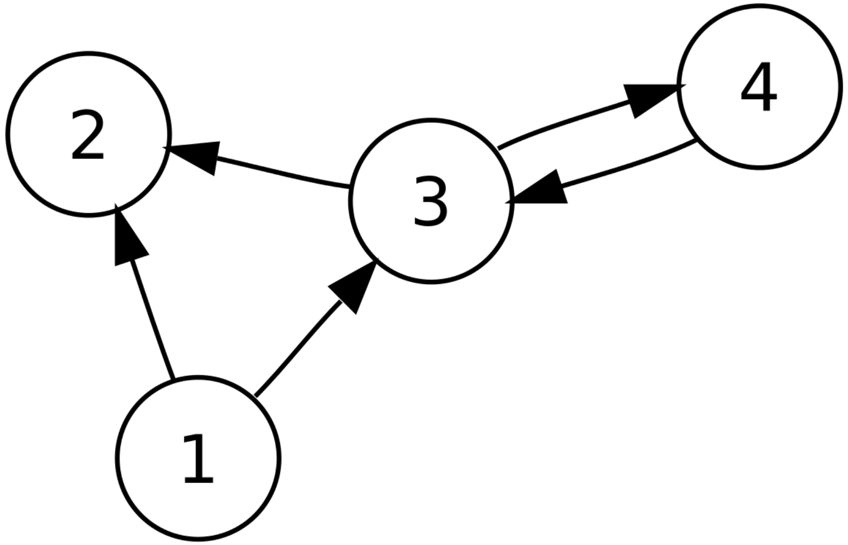
그래프

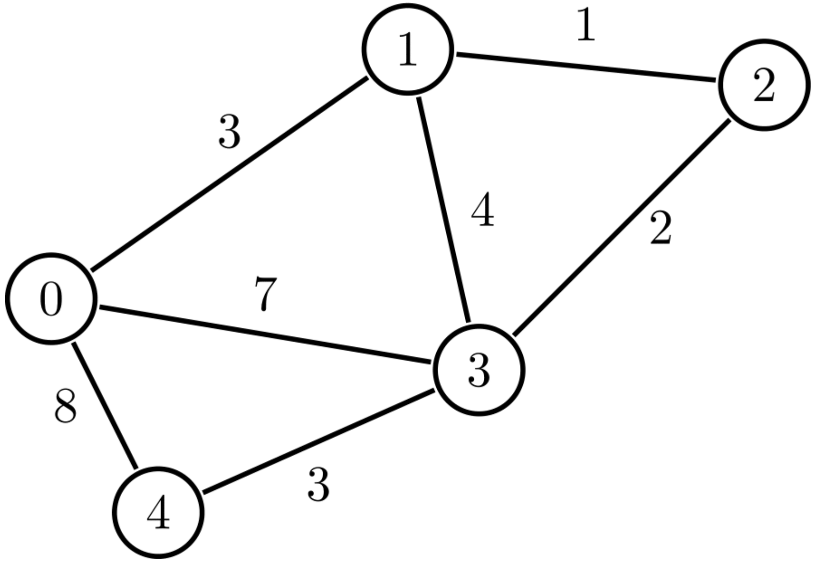
- 간선(엣지)과 정점(노드)의 집합, $G(V, E)$



그래프의 종류

- 방향성에 따라서 방향 그래프 (Direct Graph) / 무방향 그래프 (Indirect Graph)
- 정점 및 간선의 가중치 여부에 따라 가중치 그래프 (Weighted Graph)



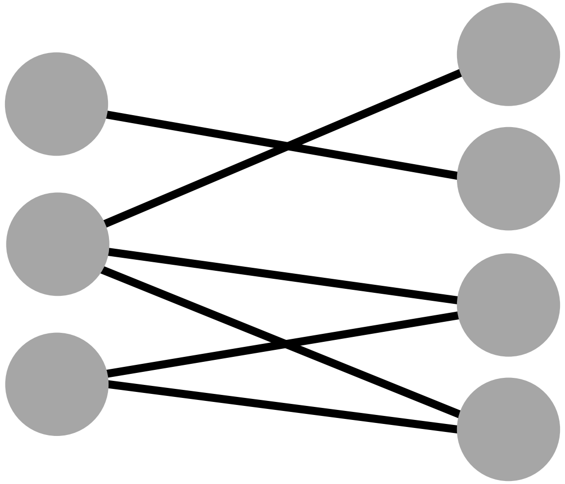


그래프의 종류

- 사이클이 없다면 Acyclic Graph
- 방향성이 있는 사이클 없는 그래프 Directed Acyclic Graph (DAG)
- 사이클이 없는 그래프 내의 모든 정점 쌍에 대해 경로가 유일한 경우 트리 Tree
- 트리의 집합 포레스트 Forest
- etc...

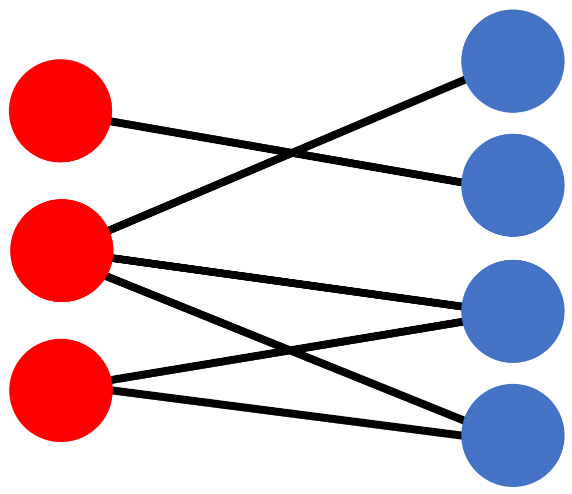
이분 그래프

- 그래프의 정점을 두 개로 나누었을 때, 모든 간선이 서로 다른 정점 집합을 잇는 경우
- $G(V, E)$에서 $A \cup B = V, A \cap B = \emptyset$인 정점 집합 A, B가 있고
- $(u, v) \in E$ 인 모든 간선에 대해서 모든 간선이 $u \in A, v \in B$ 를 만족



이분 그래프

- 정의에 의해 발생하는 성질
- 간선으로 연결된 두 정점이 서로 다른 색을 가지도록, 두 가지 색만으로 칠할 수 있다
- 사이클이 존재한다면, 해당 사이클을 이루는 정점의 개수는 짝수개
- 트리는 이분 그래프



이분 그래프

- (홀수개의 정점을 가진 사이클)이 존재하지 않는 그래프 G는 이분 그래프일까?
- 하나의 정점에서 도달할 수 없는 정점에 대해서는 독립적으로 생각해도 된다
- 그래프에 존재하는 **모든 사이클의 길이가 짝수라고 가정**하고, 임의의 정점 u를 선택
- 그래프를 다음과 같이 두 개로 나눌 수 있다 $(A \cup B = V, A \cap B = \emptyset)$
- $\boldsymbol{A}$: u로부터 최단 거리가 **짝수**인 정점들의 집합 (u는 이곳에 속한다)
- $\boldsymbol{B}$: u로부터 최단 거리가 **홀수**인 정점들의 집합

이분 그래프

- 정점 집합 A에서 임의의 정점 a_1, a_2를 고르고, (a_1, a_2) 간선이 존재한다고 하자
- $u \rightarrow a_1, a_1 \rightarrow a_2, a_2 \rightarrow u$ 경로가 각각 존재한다
- $u \rightarrow a_1, a_2 \rightarrow u$ 경로는 집합 A의 정의에 의해 짝수 길이
- (a_1, a_2)는 길이가 1인 간선이므로 $u \rightarrow a_1 \rightarrow a_2 \rightarrow u$인 홀수 사이클이 존재
- 이는 길이가 홀수인 사이클이 존재하지 않는다는 가정에 모순
- $u, v \in A, \;\; u \neq v, \; (u, v) \notin E$ 가 성립한다

이분 그래프

- 앞과 같은 방식으로 집합 B에 대해서도 $u, v \in B, \quad u \neq v, \; (u, v) \notin E$ 가 성립한다
- A의 집합 내의 두 정점에 대해서 간선이 존재하지 않고, B에 대해서도 같으므로
- (홀수 크기의 사이클)이 존재하지 않는 그래프 G는 이분 그래프이다

이분 그래프 BOJ 1707

- 주어진 그래프가 이분 그래프인지 판별하자
- 그래프를 실제로 두 가지 색으로 칠한다고 생각하고 순회
- 만약 연결되어 있지만 같은 색으로 칠해져야 한다면, 이분 그래프가 아니다

이분 그래프 BOJ 1707

- DFS에서 몇가지만 수정해 보자
- DFS에서는 이미 방문한 경우 재귀호출을 부르지 않고 종료
- **해당 정점에서 탐색했을 때 이분 그래프 여부를 반환**한다고 하자

```
void dfs(int cur){
    v[cur] = 1;
    for(auto& nxt : g[cur]){
        if (v[nxt]) continue;
        dfs(nxt);
    }
}
```

이분 그래프 BOJ 1707

- 그래프의 방문 체크를 1과 2를 통해서 관리
- 만약 이미 방문돼 있지만 같은 색으로 칠해진 경우
- 같은 그룹에 해당하지만 서로를 잇는 간선이 존재
- 이분그래프가 아님

```
bool dfs(int cur, int color){
    bool ret = true;
    v[cur] = color;
    int nxt_color = (color == 1) ? 2 : 1;
    for(auto& nxt : g[cur]){
        if (v[nxt] == color) return false;
        if (v[nxt]) continue;
        ret &= dfs(nxt, nxt_color);
    }
    return ret;
}
```

새내기와 헌내기 BOJ 17209

- 모든 사람은 다른 사람을 지목한다
- 모든 사람은 새내기 또는 헌내기 중 하나
- 어떤 사람이 새내기라면, 지목한 사람은 헌내기
- 어떤 사람이 헌내기라면, 지목한 사람은 새내기
- **위 규칙에 모순되는 입력은 없음**
- 지목한 상황 속에서 헌내기의 최대 인원 수 찾기

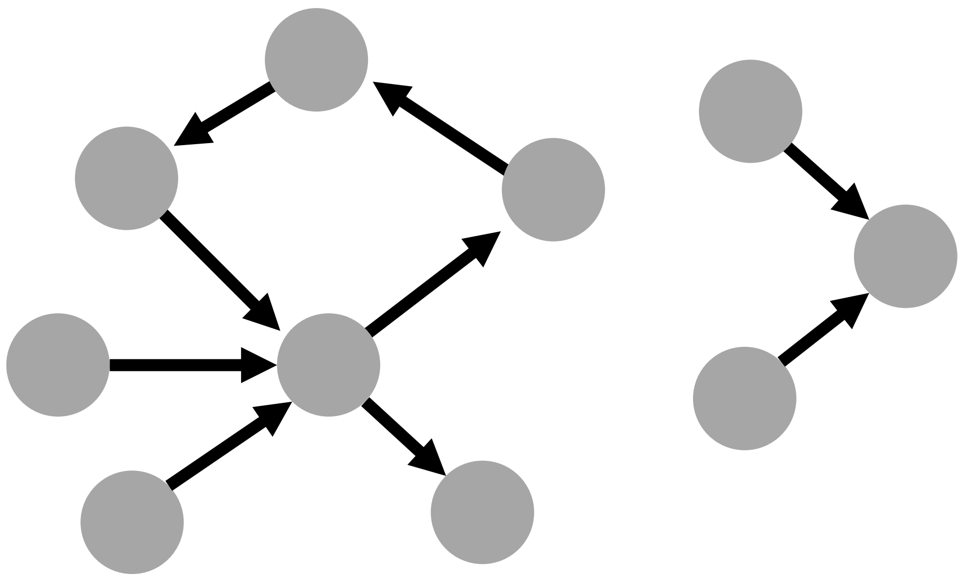
새내기와 헌내기 BOJ 17209

- 모든 사람은 다른 사람을 지목한다
- 모든 사람은 새내기 또는 헌내기 중 하나
- 어떤 사람이 새내기라면, 지목한 사람은 헌내기
- 어떤 사람이 헌내기라면, 지목한 사람은 새내기
- **위 규칙에 모순되는 입력은 없음**
- 지목한 상황 속에서 헌내기의 최대 인원 수 찾기

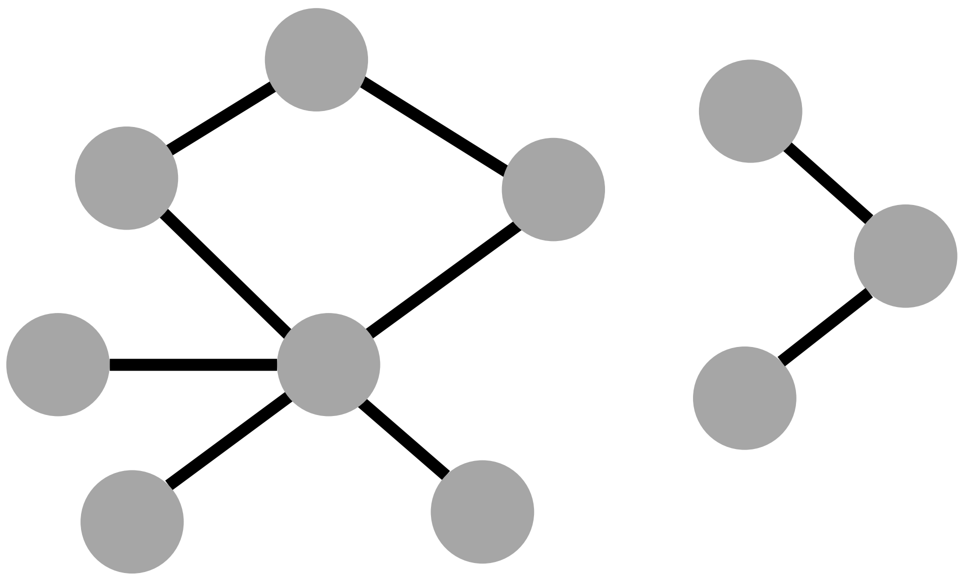
새내기와 헌내기 BOJ 17209

- 주어진 그래프는 이분 그래프인가?
- 두 그룹으로 이루어져 있음 (새내기와 헌내기)
- 반드시 새내기는 헌내기를 지목, 헌내기는 새내기를 지목
- 방향성 그래프이지만 **이분 그래프임을 활용해서 무방향 그래프처럼 생각**해도 무관
- 각 연결 컴포넌트에 대해서 두 색으로 칠했을 때, 더 많이 칠해진 색의 합을 구해나가자

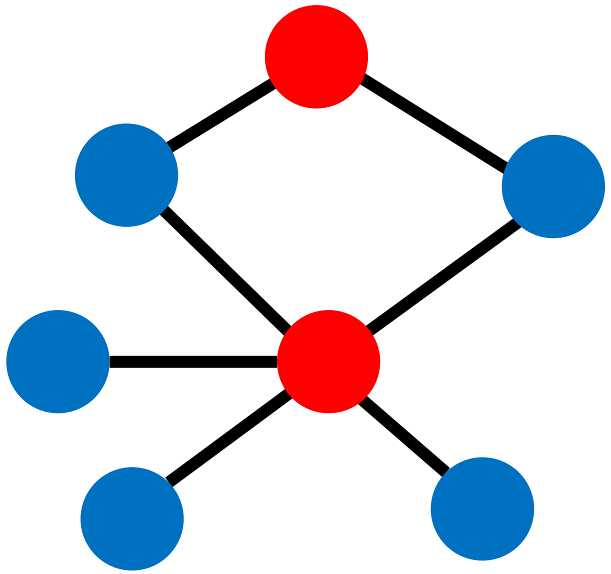
새내기와 헌내기 BOJ 17209



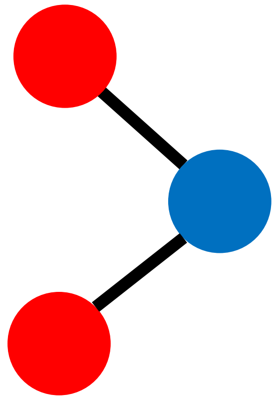
새내기와 헌내기 BOJ 17209



새내기와 헌내기 BOJ 17209



(5, 2)



(1, 2)

이분 매칭

- 아이들 N명, 과자 M종류
- 각 아이들은 원하는 과자가 존재
- 한 명의 아이는 하나의 과자만 가질 수 있음 (과자 또한 한 봉지)
- 최대 몇 명의 아이가 원하는 과자를 챙길 수 있을까?

이분 매칭

- **아이들** N명, **과자** M종류
- 각 **아이들은 원하는 과자**가 존재
- 한 명의 아이는 하나의 과자만 가질 수 있음 (과자 또한 한 봉지)
- 최대 몇 명의 아이가 원하는 과자를 챙길 수 있을까?

- **이분 그래프**

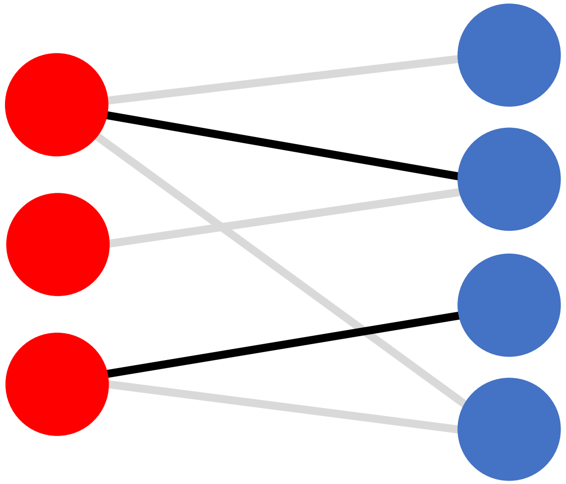
이분 매칭

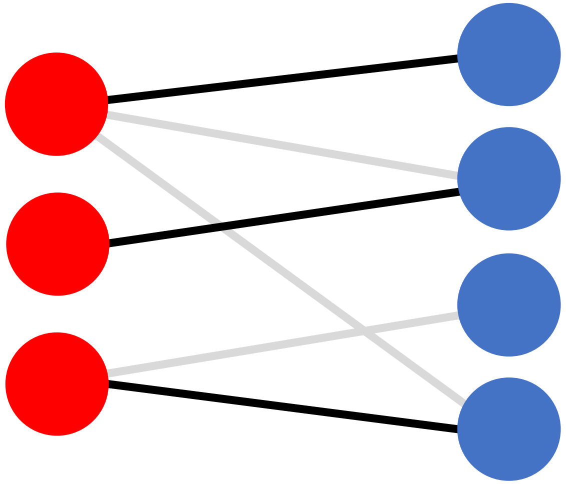
- 아이들 N명, 과자 M종류
- 각 아이들은 원하는 과자가 존재
- **한 명의 아이는 하나의 과자만 가질 수 있음** (과자 또한 한 봉지)
- **최대 몇 명의 아이가 원하는 과자를 챙길 수 있을까?**

- **매칭, 최대 매칭**

이분 매칭

- 이분 그래프에서 **두 그룹 간 최대 매칭**
- 매칭: 간선 하나를 선택 (정점도 포함), 정점은 최대 한 번만 선택되어야 함





이분 매칭

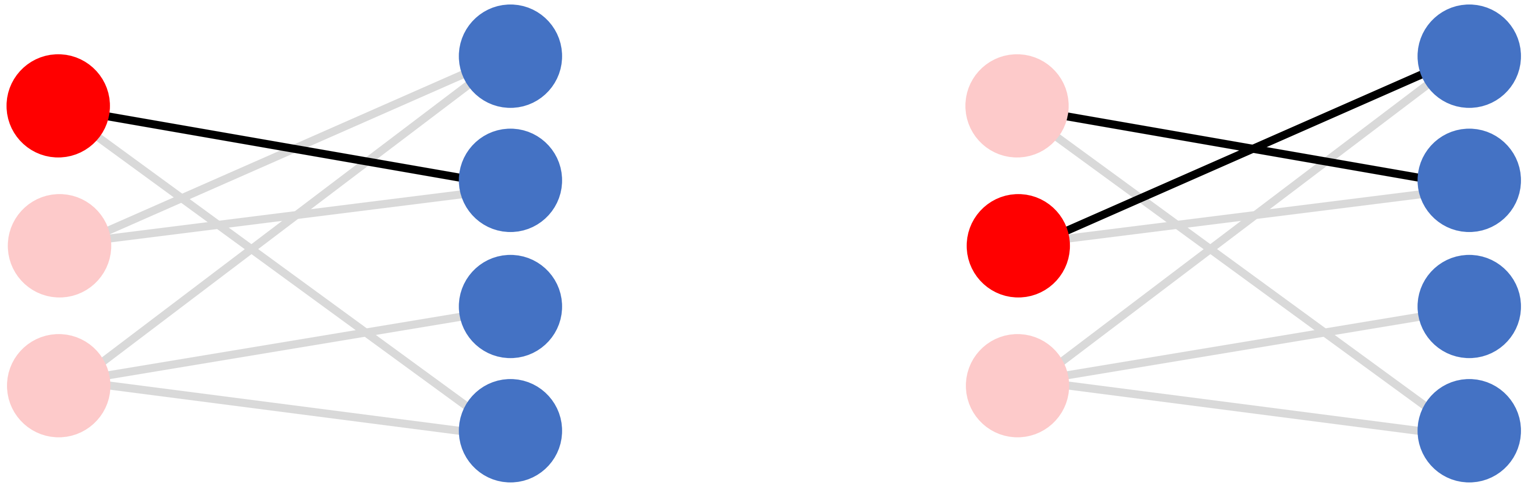
- 네트워크 플로우의 개념을 적용하면
- Edmond-Karp Algorithm: $O(VE)$
- Hopcroft-Karp Algorithm: $O\left(E\sqrt{V}\right)$
- 단순하게 이분 매칭에서 활용할 수 있는 알고리즘을 배워 보자
- $O(VE)$

이분 매칭

- **굴러온 돌이 박힌 돌 뺀다** 알고리즘(?)
- A그룹에서 매칭 후보군 B그룹을 둘러보고, 매칭되지 않은 원소가 있다면 매칭
- 만약 이미 매칭되어 있다면, 매칭된 A그룹의 원소를 다른 B그룹 원소에 매칭 시도
- 이 과정에서 여러 매칭 재시도가 재귀적으로 일어날 수 있음
- 매칭에 성공한다면 **다른 방법**으로 현재 A그룹의 원소를 매칭할 수 있다는 의미
- 모든 A그룹의 원소에 대해서 반복

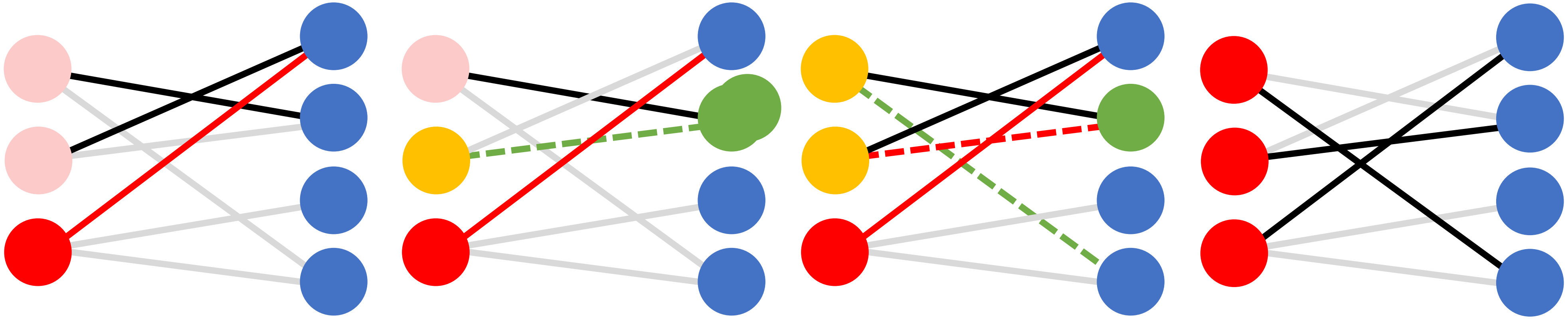
이분 매칭

- A그룹에서 매칭 후보군 B그룹을 둘러보고, 매칭되지 않은 원소가 있다면 매칭



이분 매칭

- 만약 이미 매칭되어 있다면, 매칭된 **A그룹의 원소**를 다른 **B그룹 원소**에 매칭 시도
- 이 과정에서 여러 매칭 재시도가 재귀적으로 일어날 수 있음



이분 매칭

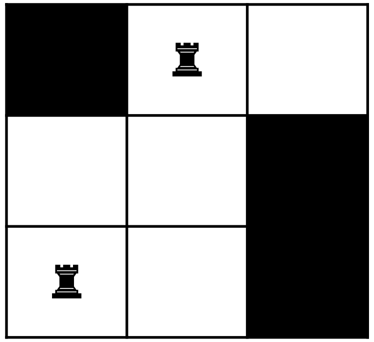
- DFS와 같이, 방문처리가 없다면 매칭 변경을 시도할 때 무한루프가 발생한다

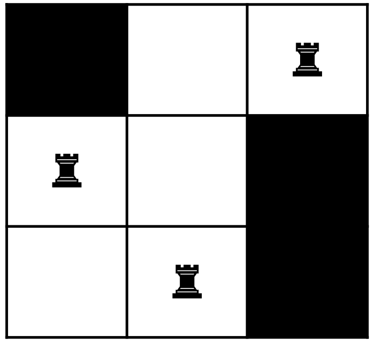
```cpp
int N, M, match[202], a, t, v[202], ans;
vector<int> g[202];

bool dfs(int a){
    if (v[a]) return false;
    v[a] = 1;
    for(auto& b : g[a]){
        if (!match[b] || dfs(match[b])) {
            match[b] = a;
            return true;
        }
    }
   return false;
}
```

```cpp
int main(){
    cin >> N >> M;
    // make graph..

    for(int i = 1; i <= N; i++) {
        memset(v, 0, sizeof(v));
        ans += dfs(i);
    }
    cout << ans << '\n';
    return 0;
}
```

룩 어택 BOJ 1574

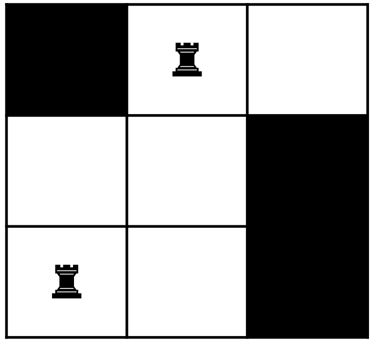
- $R \times W$ 크기의 체스판에 최대 몇 개의 룩을 공격하지 않게 놓을 수 있을까
- 룩을 놓지 못하는 칸이 존재한다

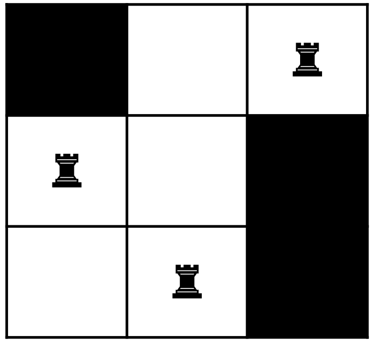




룩 어택 BOJ 1574

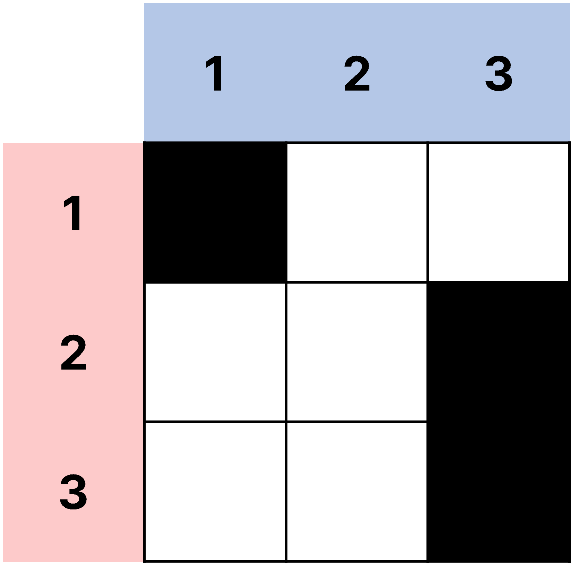
- 룩을 놓는것이 의미하는 것
- 하나의 칸에 룩을 놓게 되면, 해당 행과 열에는 다른 룩이 존재할 수 없음
- 둘 중 하나를 고정해 보자

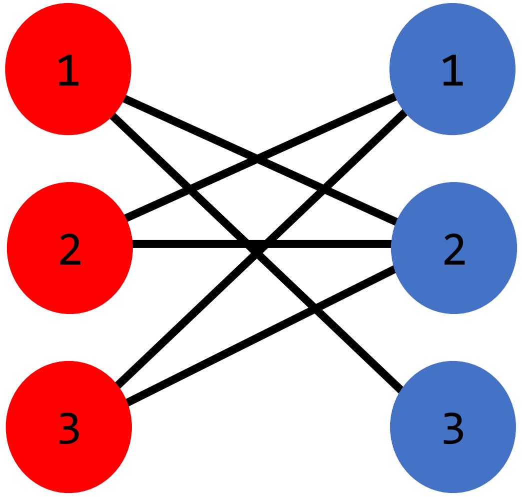




룩 어택 BOJ 1574

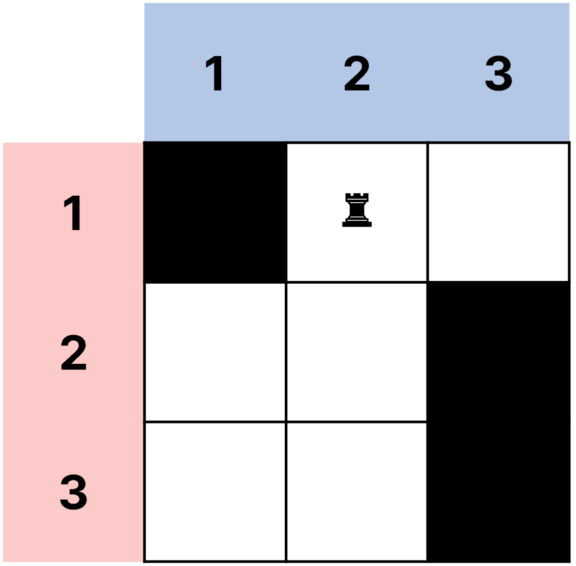
- 한 개의 행에서 한 개의 열을 고르는 문제
- 행 – 열로 구분된 이분 그래프로 모델링할 수 있다

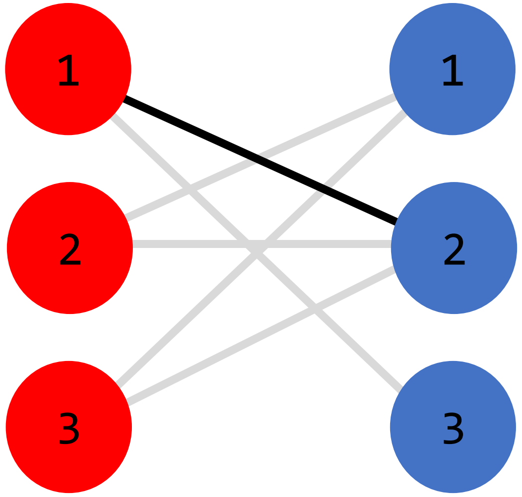




룩 어택 BOJ 1574

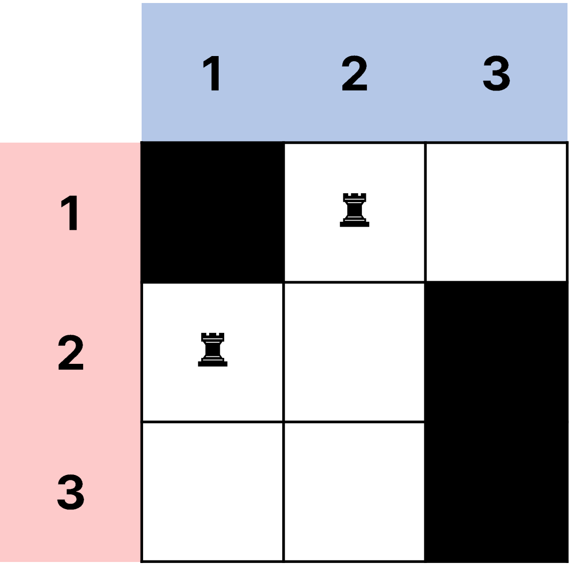
- 한 개의 행에서 한 개의 열을 고르는 문제
- 행 – 열로 구분된 이분 그래프로 모델링할 수 있다

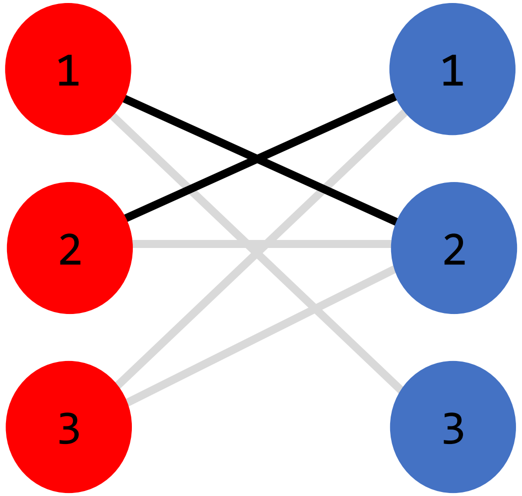




룩 어택 BOJ 1574

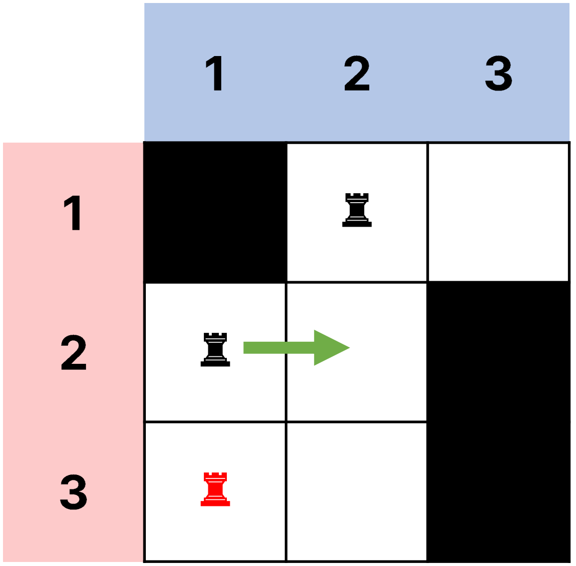
- 한 개의 행에서 한 개의 열을 고르는 문제
- 행 – 열로 구분된 이분 그래프로 모델링할 수 있다

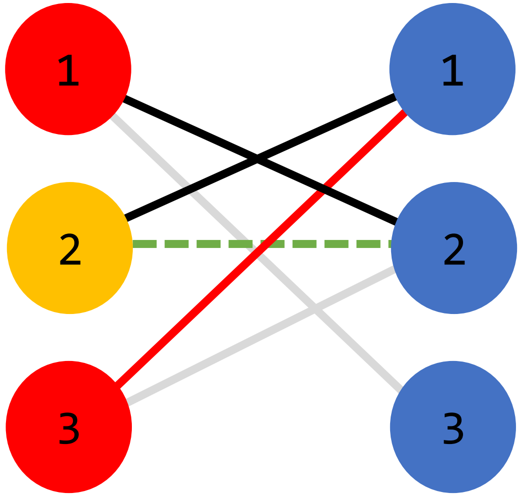




룩 어택 BOJ 1574

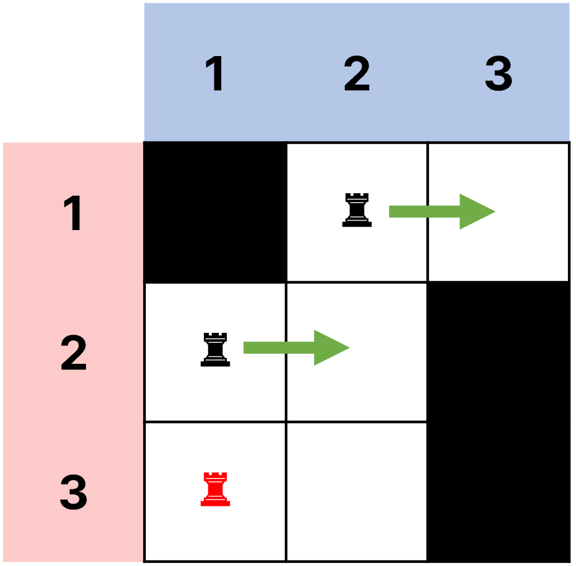
- 한 개의 행에서 한 개의 열을 고르는 문제
- 행 – 열로 구분된 이분 그래프로 모델링할 수 있다

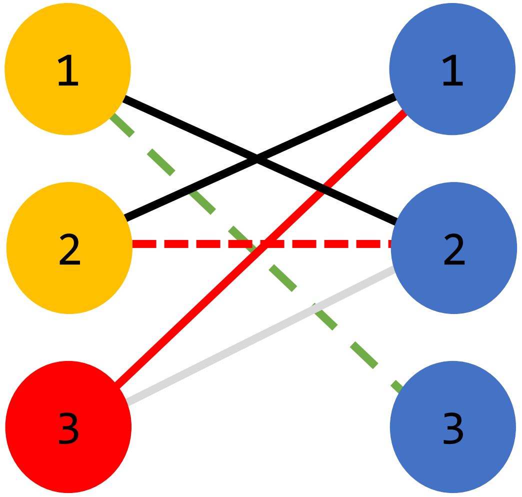




룩 어택 BOJ 1574

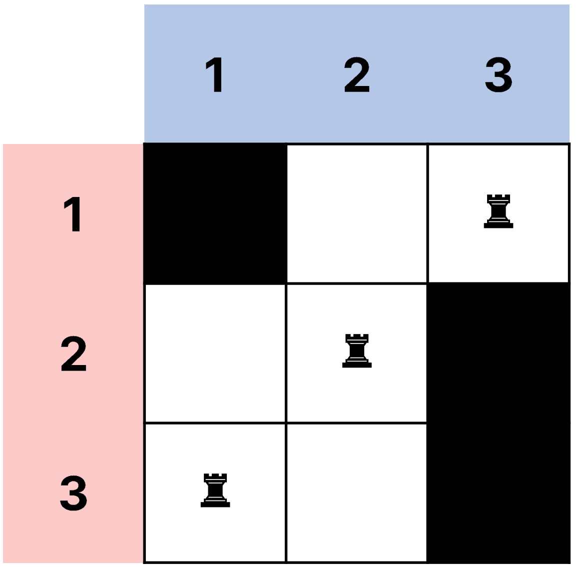
- 한 개의 행에서 한 개의 열을 고르는 문제
- 행 – 열로 구분된 이분 그래프로 모델링할 수 있다

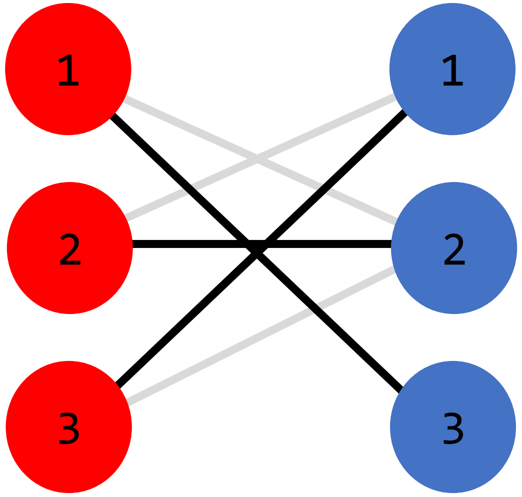




룩 어택 BOJ 1574

- 한 개의 행에서 한 개의 열을 고르는 문제
- 행 – 열로 구분된 이분 그래프로 모델링할 수 있다





연습 문제

1707 : 이분 그래프
12893 : 적의 적
17209 : 새내기와 헌내기
28102 : 단순한 그래프와 이상한 쿼리 *

2188 : 축사 배정
11375 - 11378 : 열혈강호 시리즈
11376 : 열혈강호 2
1574 : 룩 어택
28090 : 특별한 한붓그리기