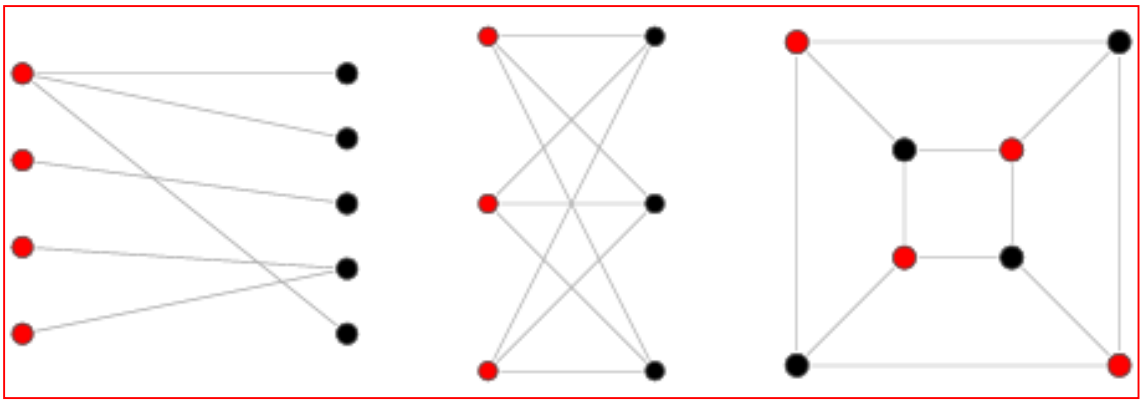
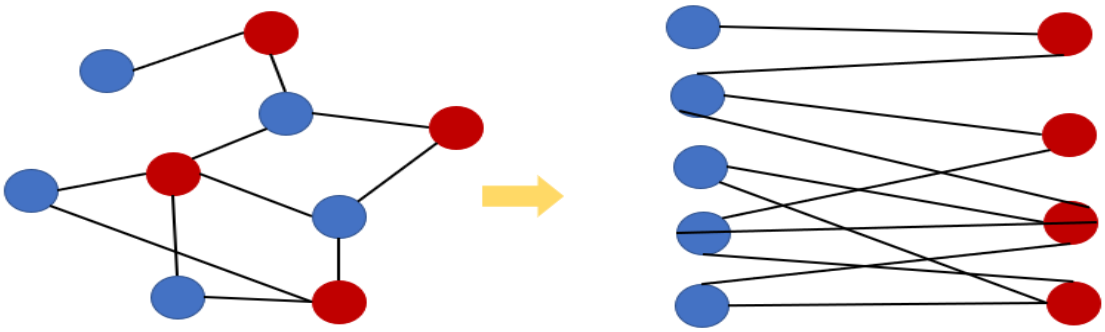
이분 그래프(Bipartite Graph)

- 같은 레벨의 정점끼리는 **같은색**
- 인접한 정점 끼리는 **다른 색**



- **인접한 정점끼리 서로 다른 색으로 칠해서 모든 정점을 두 가지 색으로 만 칠할 수 있는 그래프**
- **그래프의 모든 정점이 두 그룹으로 나눠지고 서로 다른 그룹의 정점이 간선으로 연결되어져 있는(같은 그룹에 속한 정점끼리는 서로 인접하지 않도록 하는) 그래프를 이분 그래프**

이분 그래프의 특징



- 이분 그래프인지 확인하는 방법은 BFS, DFS 탐색을 이용하면 된다.
- 이분 그래프는 BFS를 할 때 같은 레벨의 정점끼리는 모조건 같은 색으로 칠해진다.
- 모든 정점을 방문하며 간선을 검사하기 때문에 시간 복잡도는 O(V+E)

이분 그래프(Bipartite Graph)

이분 그래프인지 확인하는 방법

이분 그래프인지 확인하는 방법은 너비 우선 탐색(BFS), 깊이 우선 탐색(DFS)을 이용

서로 인접한 정점이 같은 색이면 이분 그래프가 아니다.

1. BFS, DFS로 탐색하면서 정점을 방문할 때마다 두 가지 색 중 하나를 칠한다.
2. 다음 정점을 방문하면서 자신과 인접한 정점은 자신과 다른 색으로 칠한다.
3. 탐색을 진행할 때 자신과 인접한 정점의 색이 자신과 동일하면 이분 그래프가 아니다.
   - BFS의 경우 정점을 방문하다가 만약 같은 레벨에서 정점을 다른 색으로 칠해야 한다면 무조건 이분 그래프가 아니다.
4. 모든 정점을 다 방문했는데 위와 같은 경우가 없다면 이분 그래프이다.

주의! 연결 그래프와 비연결 그래프(모든 정점을 돌면서 확인) 모두 고려!!

```
//판단할 체크 배열 선언
check= new int[V+1];
//result 값 초기화
result = true;
//모든 정점에서 시작하여 판단 필요(연결되지 않은 그래프인 경우도 있을 수 있음)
for(int i = 1; i < V + 1; i++) {
//하나의 정점이라도 이분 그래드팍 아니면 전체 그래프는 이분 그래프가 아니다
  if(!result) {
    break;
  }
//이미 판단된 정점은 배제하고 아니면 판단 함수 호출한다.
  if(check[i] == 0) {
    //dfs(i,TYPE1);
    bfs(i, TYPE1);
  }
}
```

이분 그래프(Bipartite Graph)

BFS

```
static void bfs(int idx, int type) {
  Queue<Integer> q = new LinkedList<Integer>();
  //첫번째 정점의 임의틔 타입을 넣어준다.
  check[idx] = type;
  q.offer(idx);
  int cur;
  while(!q.isEmpty()) {
  //정점을 뽑고 정점과 연결된 나머지 정점을 판단한다.
    cur = q.poll();
    for(int v : list[cur]) {
    //방문한 적이 없는 정점이면 현재 정점의 반대 Type을
    넣어주고,큐에 삽입 후 다음 정점으로 넘어간다.
      if(check[v] == 0) {
        check[v] = -check[cur];
        q.offer(v);
        continue;
      }
      //인접한 정점이 같은 값이면 이분 그래프가 아님으로
      //종료한다
      if(check[v] + check[cur] != 0) {
        result = false;
        return;
      }
    }
  }
}
```

이분 그래프(Bipartite Graph)

DFS

```
static void dfs(int idx, int type) {
  //현재 위치를 기존 타입으로 칠한다.
  check[idx] = type;
  //현재 정점과 연결된 나머지 정점을 판단한다
  for(int v : list[idx]) {
  //연결된 정점이 기존 타입과 같은 것으로 결정되어
  // 있으면 이분 그래프가 아님으로 멈춤
    if(check[v] == type) {
      result = false;
      return;
    }
  //방문하지 않았던 정점이면 다른 타입으로
  //재귀호출한다
    if(check[v] == 0) {
      dfs(v, -type);
    }
  }
}
```