#07-2. 그래프 이론

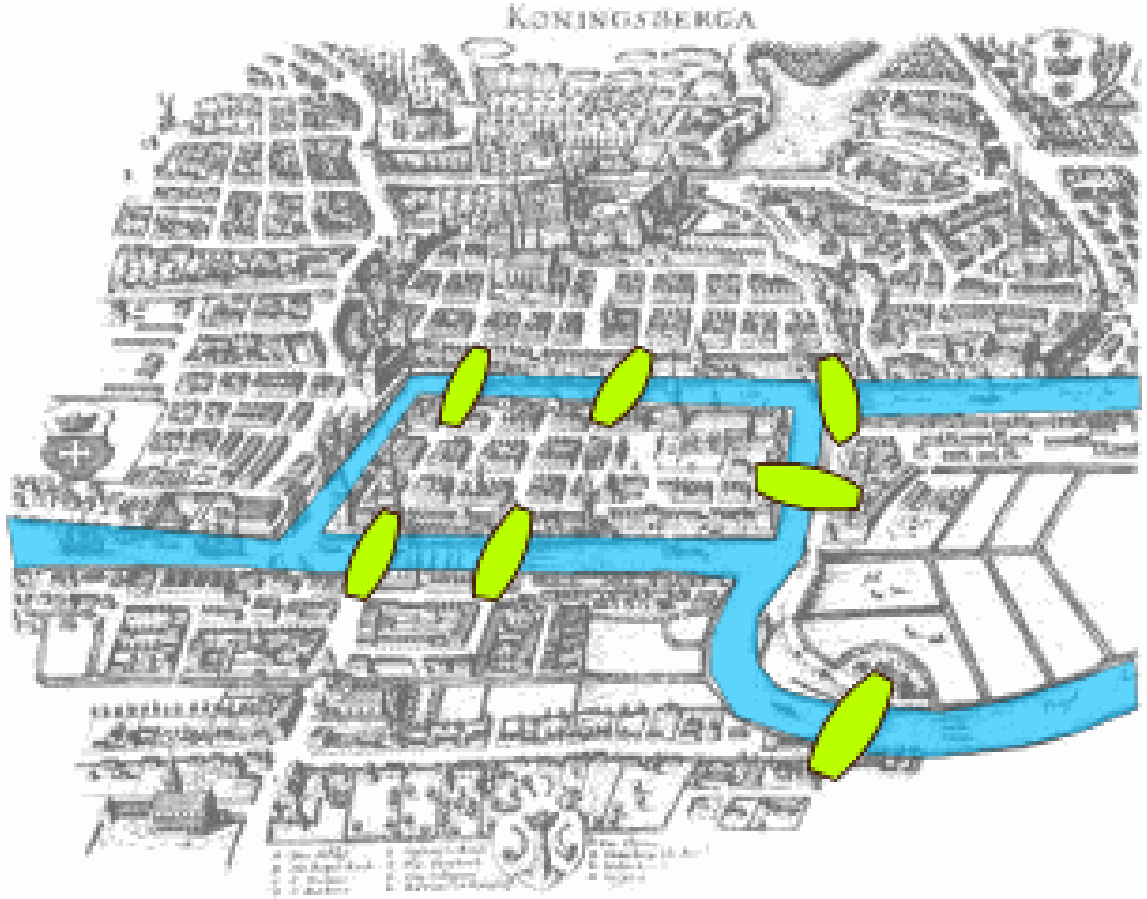
나정휘

https://justiceHui.github.io/

그래프

쾨니히스베르크의 다리

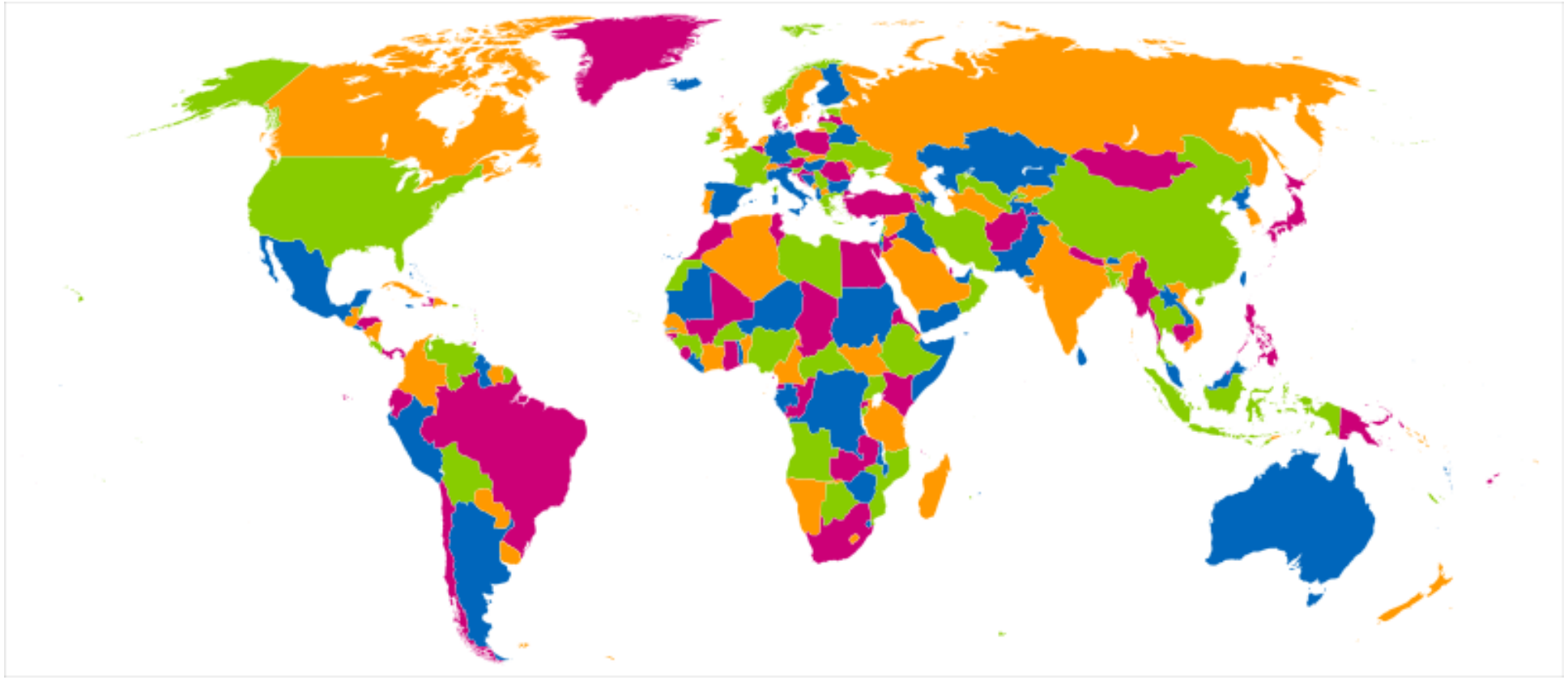
- 임의의 지점에서 출발해서 일곱 개의 다리를 정확히 한 번씩만 건너고 원래 위치로 돌아올 수 있을까?
- https://en.wikipedia.org/wiki/File:Konigsberg_bridges.png



그래프

지도 색칠

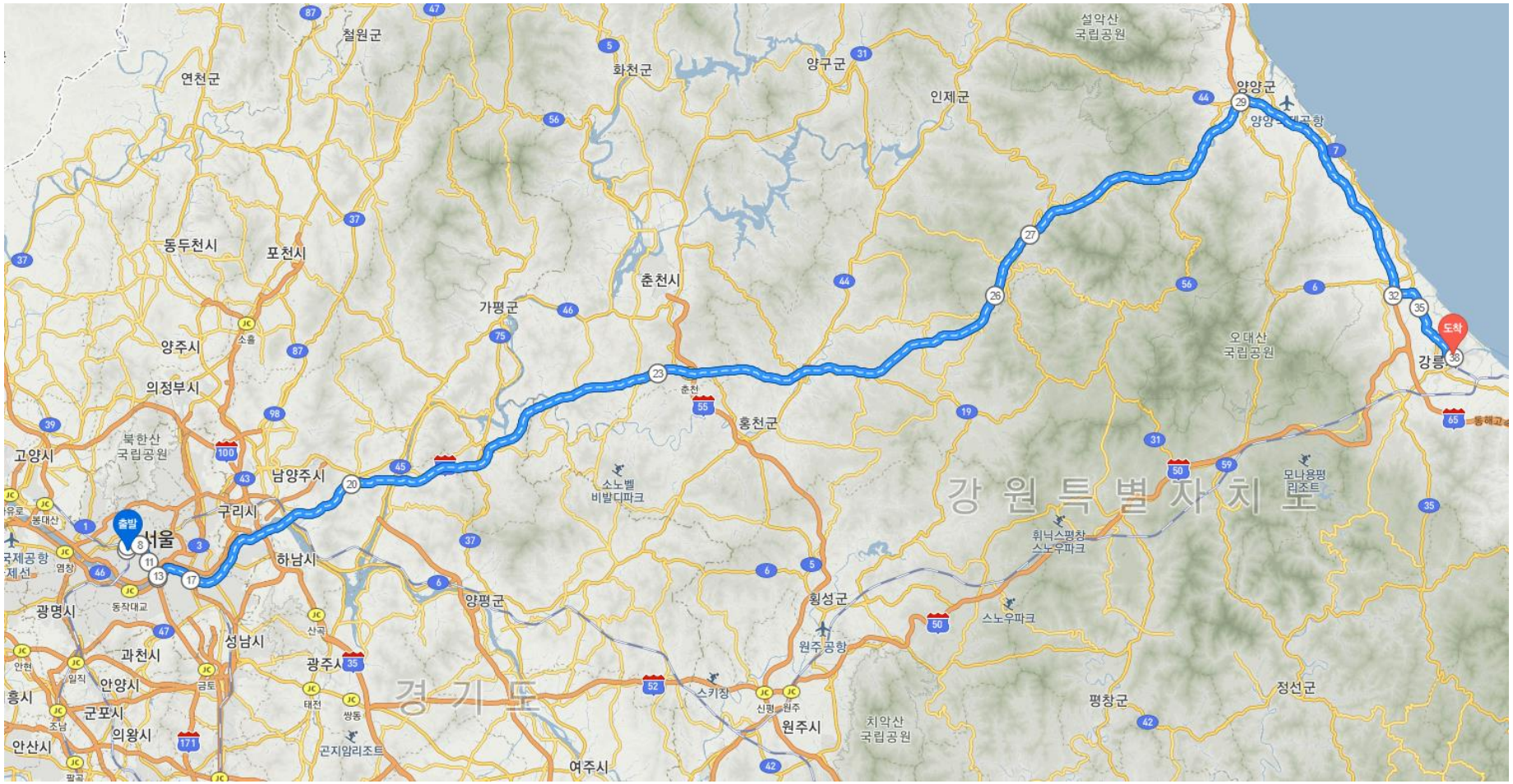
- 지도에 있는 모든 나라를 4개의 색으로 칠할 수 있을까?
- https://commons.wikimedia.org/wiki/File:World_map_with_four_colours.svg



그래프

길 찾기

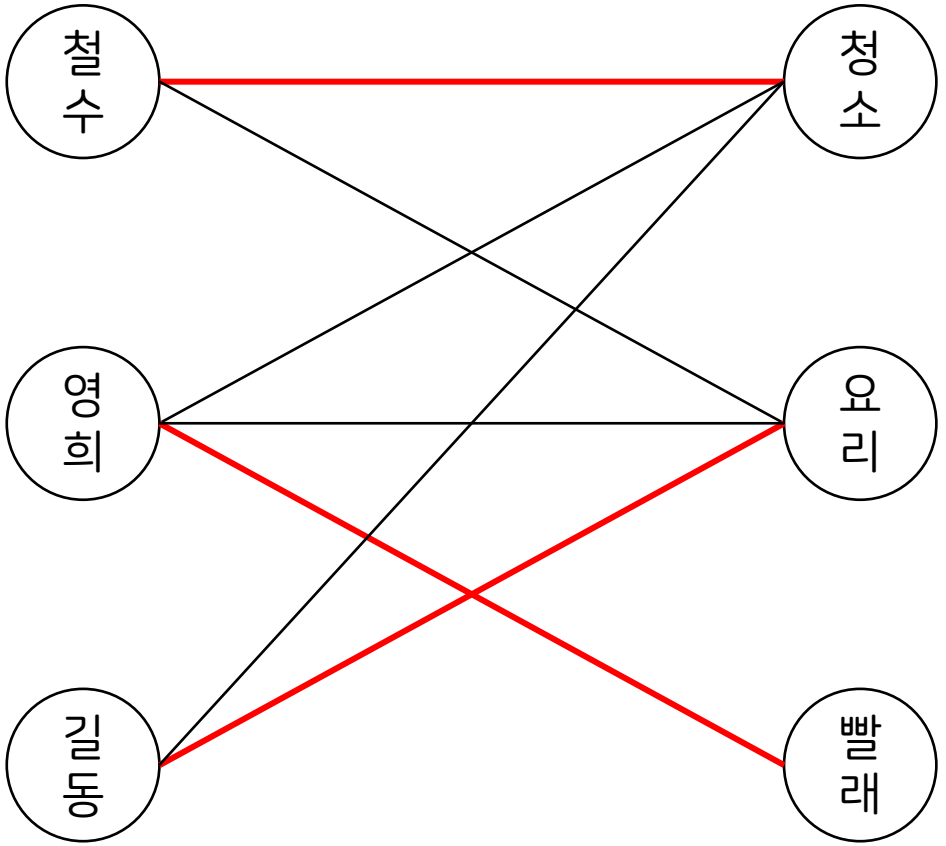
- 서울역에서 대전역으로 가는 가장 빠른 경로는?



그래프

작업 할당

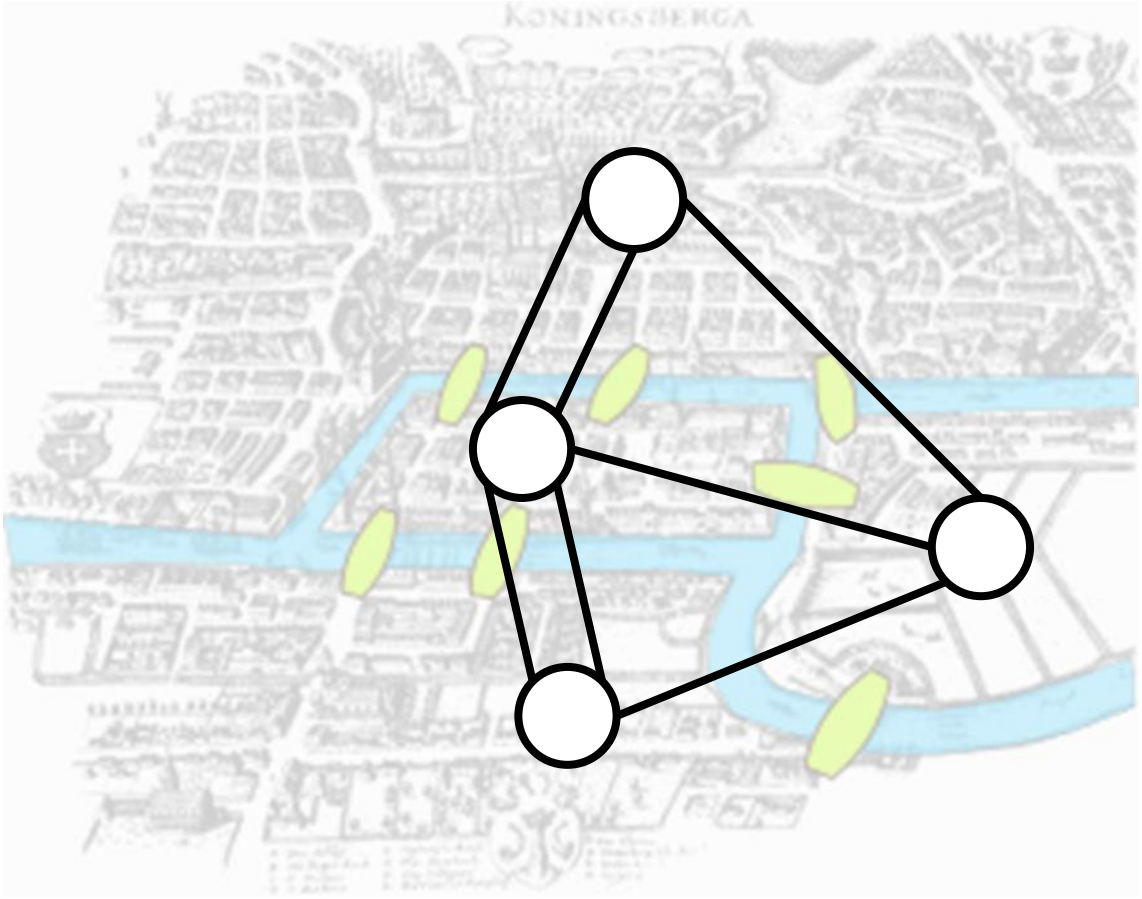
- N개의 사람과 N개의 작업이 있을 때, 각 사람에게 작업을 하나씩 할당할 수 있을까?



그래프

오일러 회로 (Euler Tour)

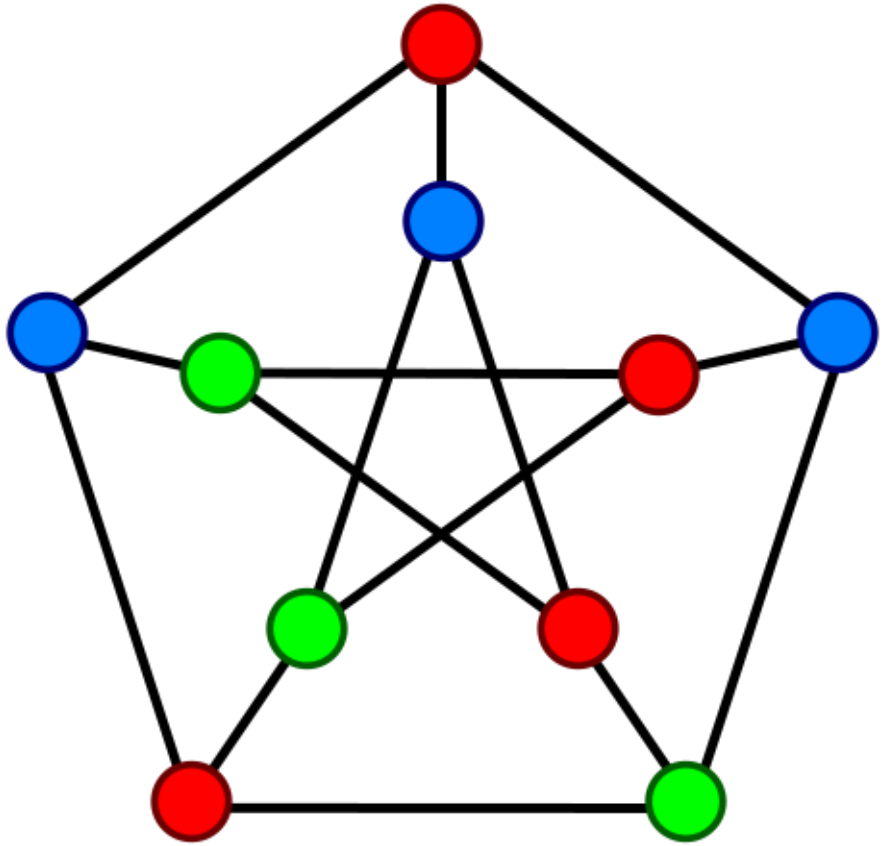
- 한 정점에서 출발해 모든 간선을 한 번씩 지나서 시작점으로 돌아오는 회로를 찾는 문제
- https://en.wikipedia.org/wiki/File:Konigsberg_bridges.png

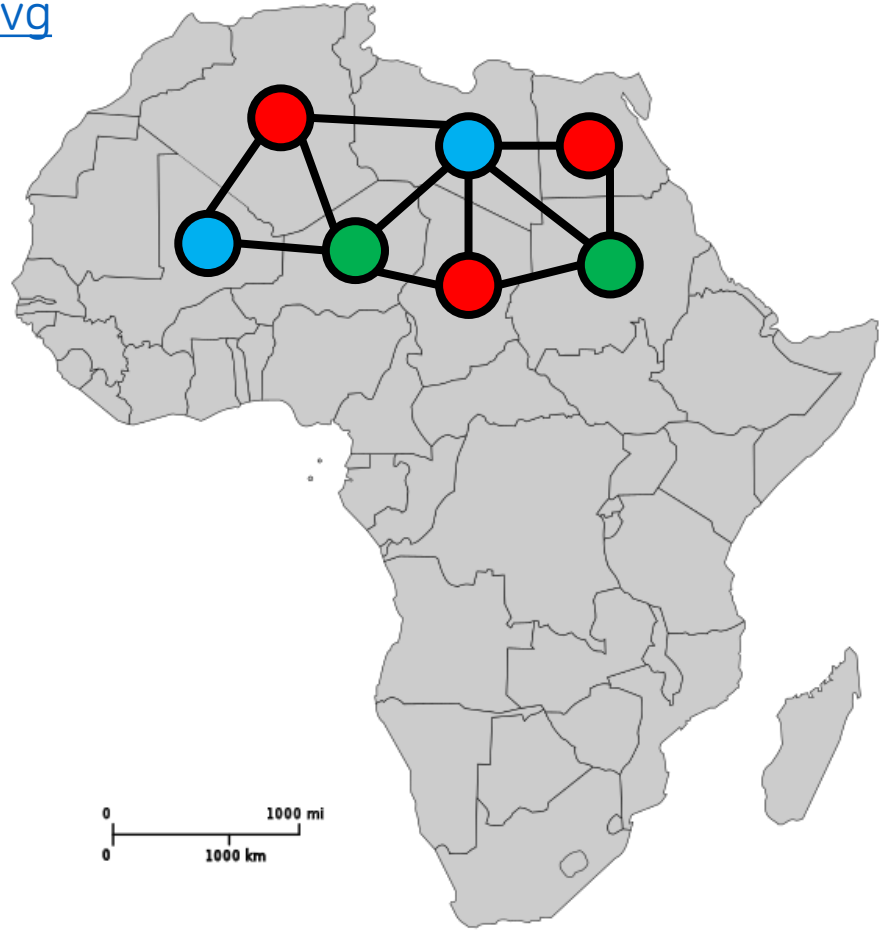


그래프

그래프 채색 (Graph Coloring)

- 간선으로 연결된(서로 인접한) 정점을 서로 다른 색으로 칠하는 문제
- https://commons.wikimedia.org/wiki/File:Petersen_graph_3-coloring.svg
- https://en.wikipedia.org/wiki/File:Blank_Map-Africa.svg

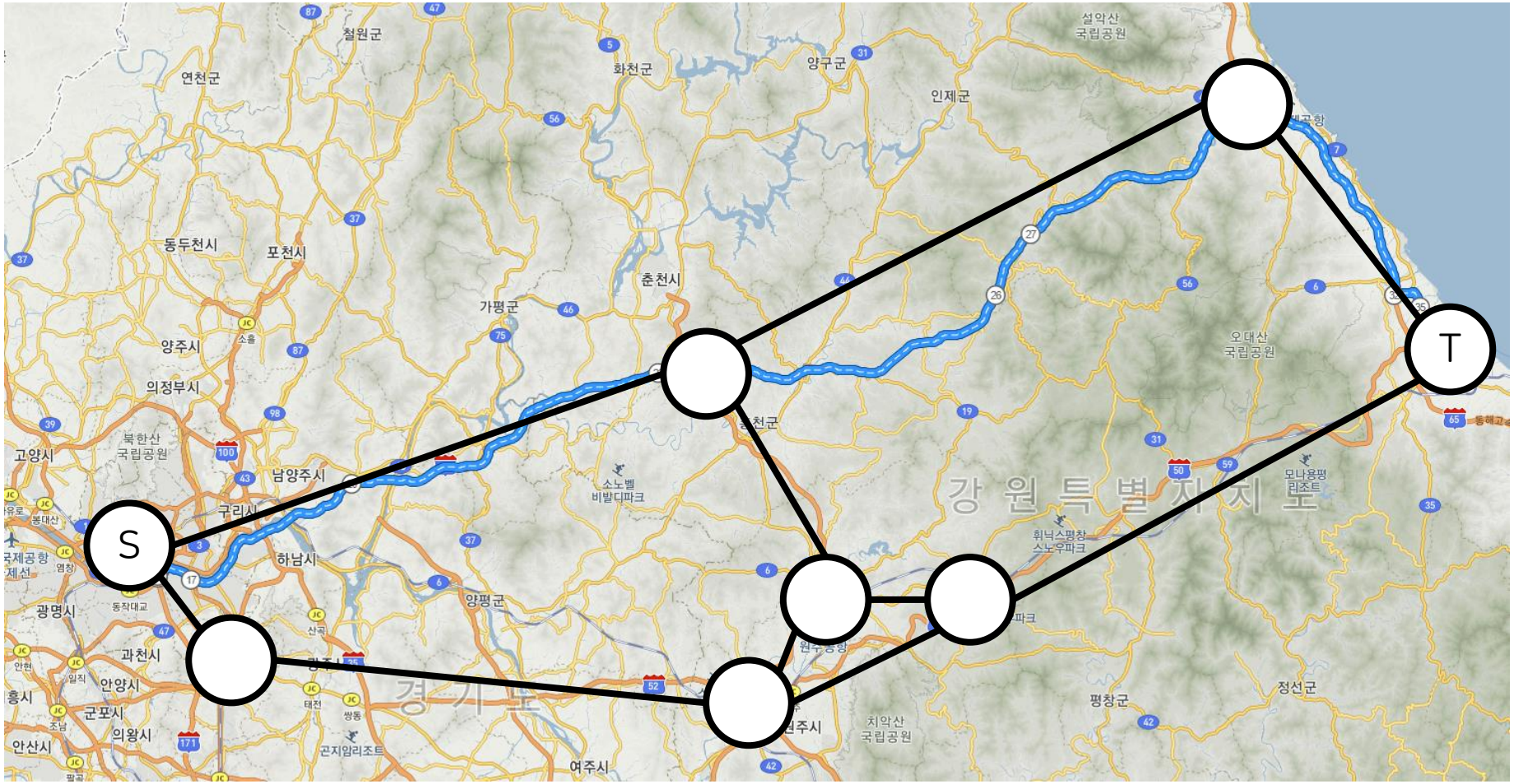




그래프

최단 경로 (Shortest Path)

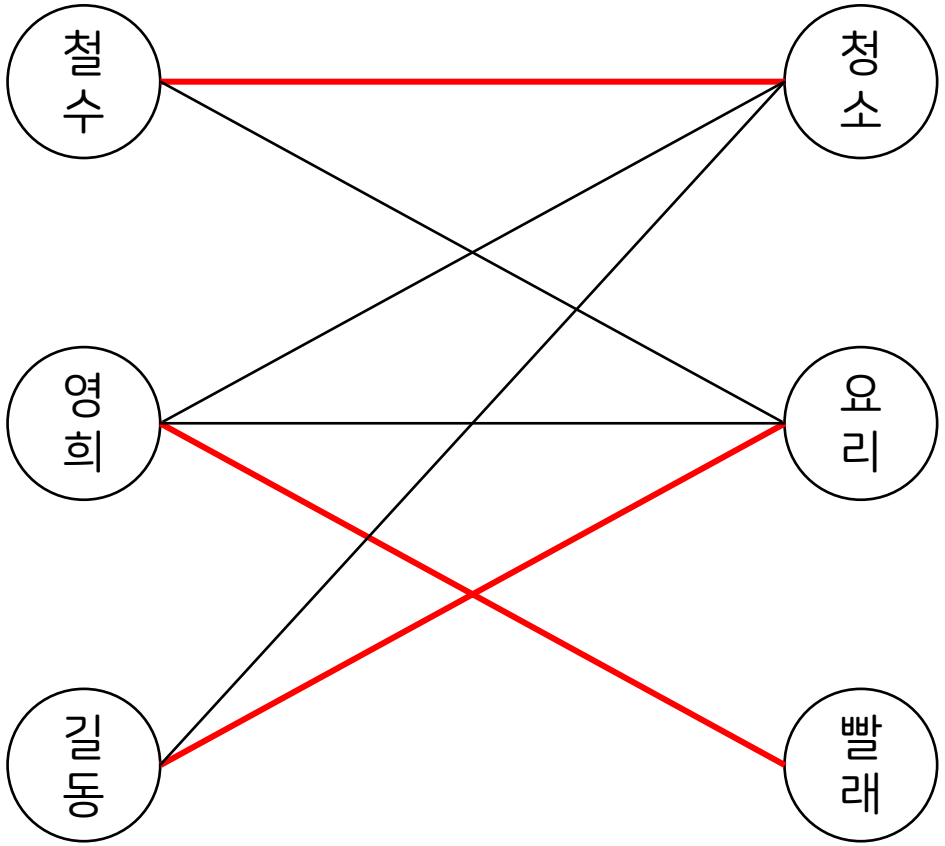
- 정점 S에서 정점 T로 가는 가중치가 가장 짧은 경로를 찾는 문제

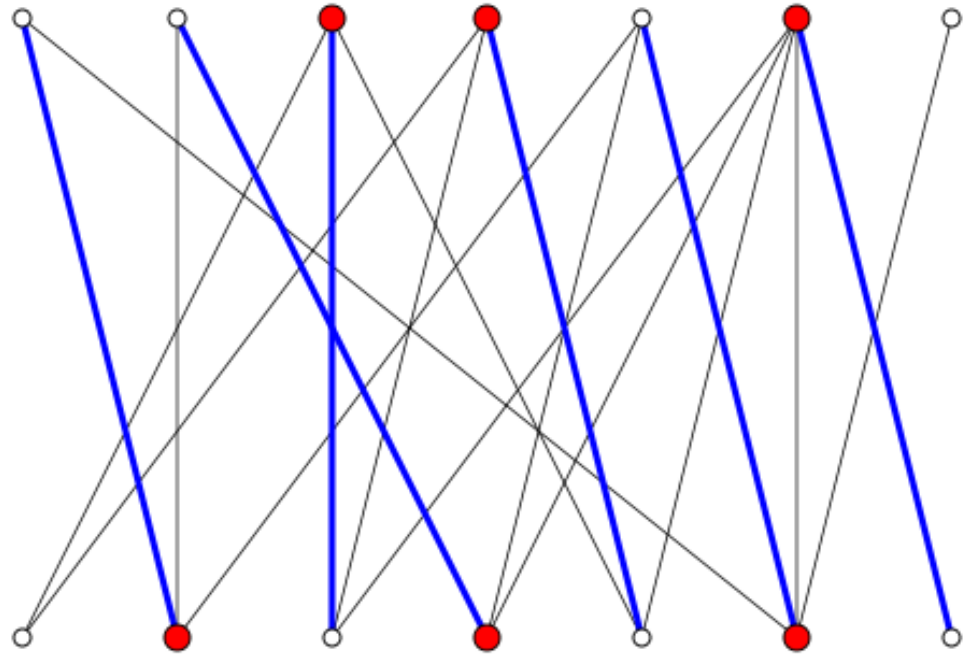


그래프

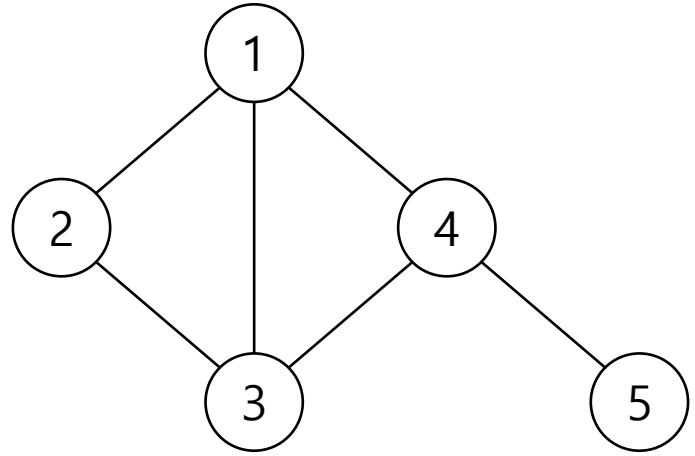
그래프 매칭 (Graph Matching)

- 서로 인접하지 않도록(같은 정점을 공유하지 않도록) 최대한 많은 간선을 선택하는 문제
- https://commons.wikimedia.org/wiki/File:Koenigs-theorem-graph.svg





그래프

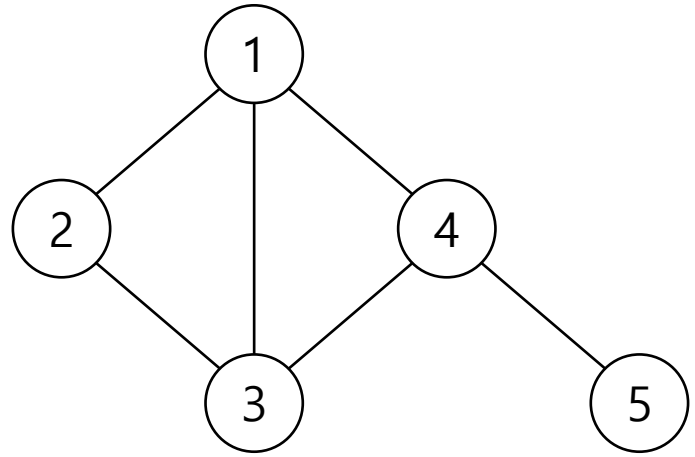


그래프

- 정점 집합과 두 정점을 연결하는 간선 집합으로 구성된 구조
  - G = (V, E)
  - V: 정점들의 집합
  - E ⊆ { {x,y} | x,y ∈ V }: 간선들의 집합
  - ex. V = {1, 2, 3, 4, 5}, E = { {1,2}, {2,3}, {3,4}, {4,1}, {4,5}, {1,3} }

- 간선에 방향이 없으면 무향 그래프 (Undirected Graph)
- 간선에 방향이 있으면 유향 그래프 (Directed Graph)
- 간선에 가중치가 있으면 가중치 그래프 (Weighted Graph)
- 모든 정점이 간선을 통해 직/간접적으로 연결되어 있으면 연결 그래프 (Connected Graph)

그래프



그래프 관련 용어

- 차수(degree): 한 정점에 연결되어 있는 간선의 수
  - 입력 차수(in-degree): 한 정점으로 들어오는 간선의 수
  - 출력 차수(out-degree): 한 정점에서 나가는 간선의 수
- 보행(walk): 정점과 변이 교대로 등장하는 나열, 각 간선의 앞/뒤는 간선이 연결하는 두 정점
  - ex. 1 – {1,2} – 2 – {2,3} – 3 – {3,1} – 1 – {1,2} - 2
- 트레일(trail): 간선이 중복되지 않는 보행
  - ex. 1 – {1,2} – 2 – {2,3} – 3 – {3,1} – 1 – {1,4} - 4
- 경로(path): 정점이 중복되지 않는 트레일
  - ex. 1 – {1,2} – 2 – {2,3} - 3
- 사이클(cycle): 시작과 끝점이 동일하고, 나머지는 정점의 중복이 없는 트레일
  - ex. 1 – {1,2} – 2 – {2,3} – 3 – {3,1} – 1
- 연결 요소(Connected Component): 경로를 통해 서로 연결된 정점들의 집합

그래프

관심있는 것

- 경로, 사이클
  - euler tour/path: 모든 간선을 정확히 한 번만 방문하는 사이클/경로가 존재하는지
  - hamiltonian cycle/path: 모든 정점을 정확히 한 번만 방문하는 사이클/경로가 존재하는지
  - shortest path: 어떤 정점에서 다른 정점으로 가는 경로의 가중치 최솟값은 얼마나 되는지
- 연결성, 최소 신장 트리
  - connectivity: 모든 정점이 간선으로 연결되어 있는지
  - k-edge connectivity: 간선 k개를 임의로 제거해도 여전히 연결되어 있는지
  - minimum spanning tree: 모든 정점이 연결되도록 만들기 위해 얼마나 많은 가중치가 필요한지
- 채색, 매칭, 정점 덮개, 독립 집합
  - coloring: 인접한 색을 서로 다른 색으로 잘 칠할 수 있는지
  - matching: 정점들을 얼마나 많이 짝지을 수 있는지
  - vertex cover: 모든 간선을 감시하기 위해 얼마나 많은 정점을 점유해야 하는지
  - independent set: 사람들이 서로 이웃하지 않도록 정점에 배치할 때, 얼마나 많은 사람을 배치할 수 있는지
- 그래프의 형태
  - tree/forest: 그래프에 사이클이 존재하는지
  - bipartite graph: 홀수 길이 사이클이 존재하는지
  - planar graph: 이차원 평면 위에 간선이 서로 교차하지 않도록 그릴 수 있는지

그래프

특별한 형태의 그래프

- 사이클이 없는 방향 그래프(Directed Acyclic Graph)
- 완전 그래프(Complete Graph): 서로 다른 두 정점이 한 개의 간선으로 모두 연결된 그래프
- 이분 그래프(Bipartite Graph): 정점을 서로 연결되지 않은 집합 2개로 나눌 수 있는 그래프
  - 완전 이분 그래프(Complete Bipartite Graph)
- 숲(Forest): 사이클이 없는 무향 그래프
- 트리(Tree): 연결 요소가 1개인 숲
- 선인장(Cactus): 모든 간선이 최대 1개의 사이클에만 속한 그래프

그래프

특별한 형태의 그래프

- 2 Connected Graph: 정점을 임의로 1개 제거해도 연결되어 있는 그래프
  - k Connected Graph
- 2-edge Connected Graph: 간선을 임의로 1개 제거해도 연결되어 있는 그래프
  - k-edge Connected Graph
- 현 그래프(Chordal Graph): 길이 4 이상의 모든 사이클이 chord를 갖는 그래프
  - chord: 사이클을 구성하는 간선은 아니지만, 사이클에 포함된 두 정점을 연결하는 간선
- 평면 그래프(Planar Graph): 2차원 평면에 간선이 교차하지 않도록 그릴 수 있는 그래프
  - Outer Planar Graph: 모든 정점이 가장 바깥 평면과 인접하도록 그릴 수 있는 평면 그래프

질문?

그래프의 표현 방식

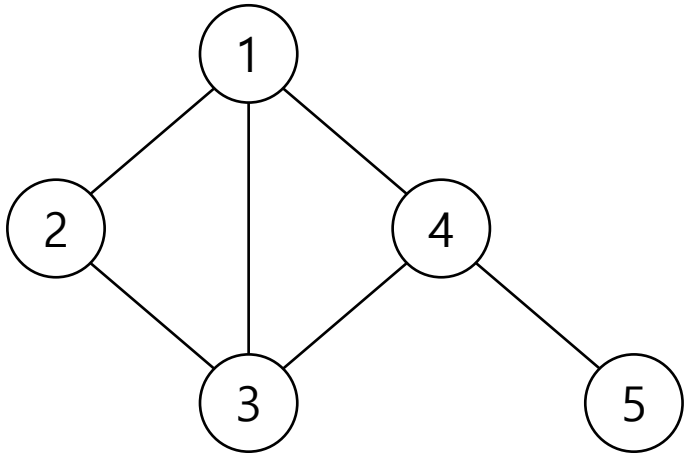
그래프의 표현 방식

- 그래프를 이용한 작업을 수행하기 편하도록 그래프를 효율적으로 저장하는 방식
- 인접 행렬
- 인접 리스트
- 간선 리스트

그래프의 표현 방식

인접 행렬

- |V| * |V| 크기의 2차원 배열 G
- u에서 v로 가는 간선이 있으면 G[u][v] = 1
  - 무향 그래프면 G[u][v] = G[v][u]
  - 가중치 그래프면 G[u][v] = {가중치}
- 공간 복잡도: $O(|V|^2)$

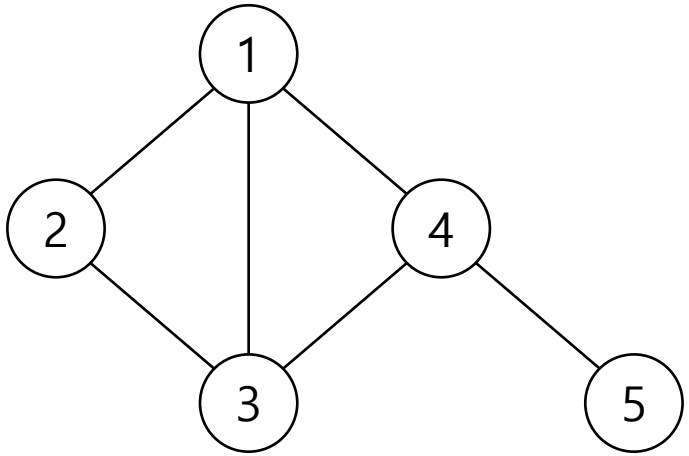


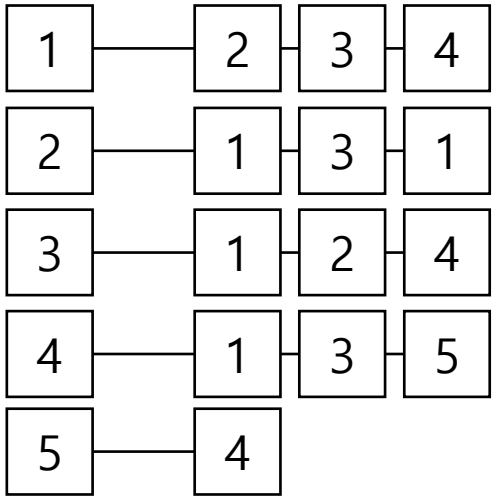
0	1	1	1	0
1	0	1	0	0
1	1	0	1	0
1	0	1	0	1
0	0	0	1	0

그래프의 표현 방식

인접 리스트

- 연결 리스트 |V|개 관리
- u번째 리스트는 u에서 출발하는 간선을 관리함
- 공간 복잡도: O(|V| + |E|)
  - O(|V| + sum deg(v))인데 sum deg(v) = 2E임
  - Handshaking Lemma





그래프의 표현 방식

인접 행렬 vs 인접 리스트

- 공간 복잡도
  - 인접 행렬: $O(|V|^2)$
  - 인접 리스트: $O(|V| + |E|)$
- 임의의 두 정점 u, v를 연결하는 간선이 있는지 확인
  - 인접 행렬: O(1)
  - 인접 리스트: O(deg(u))
- 모든 간선을 순회
  - 인접 행렬: $O(|V|^2)$
  - 인접 리스트: $O(|V| + |E|)$
- 구현
  - 인접 행렬: 2차원 배열
  - 인접 리스트: 연결 리스트 또는 동적 배열 사용 (STL 사용)

그래프의 표현 방식

간선 리스트

- 인접 리스트와 비슷하지만 인접 리스트의 단점을 상쇄
  - 연결 리스트 또는 동적 배열(std::vector) 기반이라 느림
  - 간선을 모두 순회하기 불편함
  - STL 없으면 구현하기 귀찮음

- 뒤에서 DFS/BFS 공부할 때 코드와 함께 알아보자!

질문?

인접 행렬의 거듭 제곱

BOJ 14289 본대 산책 3

- N개의 정점과 M개의 간선으로 구성된 무향 그래프
- 1번 정점에서 시작해서 D개의 간선을 지난 뒤 다시 1번 정점으로 돌아오는 보행의 개수를 구하는 문제
- $N \leq 50$, $M \leq 1000$, $D \leq 10^9$

- 인접 행렬 G를 생각해 보자.
- G[u][v]는 u에서 출발해서 1개의 간선을 지난 뒤 v에 도달하는 보행의 개수

인접 행렬의 거듭 제곱

BOJ 14289 본대 산책 3

- N개의 정점과 M개의 간선으로 구성된 무향 그래프
- 1번 정점에서 시작해서 D개의 간선을 지난 뒤 다시 1번 정점으로 돌아오는 보행의 개수를 구하는 문제
- $N \le 50$, $M \le 1000$, $D \le 10^9$

- 인접 행렬 G를 생각해 보자.
- G[u][v]는 u에서 출발해서 1개의 간선을 지난 뒤 v에 도달하는 보행의 개수

- u에서 출발해서 2개의 간선을 지난 뒤 v에 도달하는 보행의 개수는?
- sum(G[u][x] * G[x][v])
- G^2[u][v]는 u에서 출발해서 2개의 간선을 지난 뒤 v에 도달하는 보행의 개수

인접 행렬의 거듭 제곱

BOJ 14289 본대 산책 3

- N개의 정점과 M개의 간선으로 구성된 무향 그래프
- 1번 정점에서 시작해서 D개의 간선을 지난 뒤 다시 1번 정점으로 돌아오는 보행의 개수를 구하는 문제
- $N \le 50$, $M \le 1000$, $D \le 10^9$

- 인접 행렬 G를 생각해 보자.
- G[u][v]는 u에서 출발해서 1개의 간선을 지난 뒤 v에 도달하는 보행의 개수

- u에서 출발해서 2개의 간선을 지난 뒤 v에 도달하는 보행의 개수는?
- sum(G[u][x] * G[x][v])
- G^2[u][v]는 u에서 출발해서 2개의 간선을 지난 뒤 v에 도달하는 보행의 개수

- u에서 출발해서 k개의 간선을 지난 뒤 v에 도달하는 보행의 개수는?
- (u에서 x까지 k-1개의 간선으로 가는 경우의 수) * (x에서 v까지 1개의 간선으로 가는 경우의 수)
- sum(G^{k-1}[u][x] * G[x][v])
- G^k[u][v]는 u에서 출발해서 k개의 간선을 지난 뒤 v에 도달하는 보행의 개수

인접 행렬의 거듭 제곱

BOJ 14289 본대 산책 3

- N개의 정점과 M개의 간선으로 구성된 무향 그래프
- 1번 정점에서 시작해서 D개의 간선을 지난 뒤 다시 1번 정점으로 돌아오는 보행의 개수를 구하는 문제
- $N \le 50$, $M \le 1000$, $D \le 10^9$

- $G^k[u][v]$는 u에서 출발해서 k개의 간선을 지난 뒤 v에 도달하는 보행의 개수

- N * N 행렬을 곱하는 것은 $O(N^3)$에 가능
- 어떤 값을 D번 곱하는 것은 O(log D)번의 곱셈으로 가능

- 따라서 $O(N^3 \log D)$ 시간에 문제를 풀 수 있음

인접 행렬의 거듭 제곱

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using matrix = vector<vector<ll>>;
constexpr ll MOD = 1e9+7;

ll N, M, D;

matrix Multiply(const matrix &a, const matrix &b){
    int n = a.size();
    matrix c(n, vector<ll>(n));
    for(int i=0; i<n; i++)
        for(int k=0; k<n; k++)
            for(int j=0; j<n; j++)
                c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % MOD;
    return c;
}

matrix Pow(matrix a, ll b){
    int n = a.size();
    matrix res(n, vector<ll>(n));
    for(int i=0; i<n; i++) res[i][i] = 1;
    for(; b; b >>= 1, a = Multiply(a, a)) if(b & 1) res = Multiply(res, a);
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    matrix G(N, vector<ll>(N));
    for(int i=0,u,v; i<M; i++) cin >> u >> v, G[u-1][v-1] = G[v-1][u-1] = 1;
    cin >> D;
    cout << Pow(G, D)[0][0];
}
```

질문?

인접 행렬의 거듭 제곱

그래프의 연결성 판별

- 두 정점을 잇는 최단 경로의 길이는 최대 N – 1
  - N개의 간선을 사용하면 N+1개의 정점을 방문하게 되므로 중복 방문을 하기 때문
- 따라서 $G_S = G^0 + G^1 + G^2 + \ldots + G^{N-1}$ 는 그래프의 모든 최단 경로를 포함함
- G_S의 원소 중 0이 있으면 연결 그래프가 아니고, 모든 원소가 0이 아니면 연결 그래프임
- 이 방법이 최선일까?

인접 행렬의 거듭 제곱

그래프의 연결성 판별

- 두 정점을 잇는 최단 경로의 길이는 최대 N – 1
  - N개의 간선을 사용하면 N+1개의 정점을 방문하게 되므로 중복 방문을 하기 때문
- 따라서 $G_S = G^0 + G^1 + G^2 + \dots + G^{N-1}$ 는 그래프의 모든 최단 경로를 포함함
- G_S의 원소 중 0이 있으면 연결 그래프가 아니고, 모든 원소가 0이 아니면 연결 그래프임
- 이 방법이 최선일까?

- GN-1만 보면 안 되는 이유
  - 길이가 k < N-1인 보행으로 도달 가능하지만 길이가 N-1인 보행으로는 불가능할 수 있기 때문
  - 길이가 k < N-1인 보행으로 도달한 이후에 못 움직이도록 막는다면?

인접 행렬의 거듭 제곱

그래프의 연결성 판별

- 두 정점을 잇는 최단 경로의 길이는 최대 N – 1
  - N개의 간선을 사용하면 N+1개의 정점을 방문하게 되므로 중복 방문을 하기 때문
- 따라서 $G_S = G^0 + G^1 + G^2 + \dots + G^{N-1}$ 는 그래프의 모든 최단 경로를 포함함
- G_S의 원소 중 0이 있으면 연결 그래프가 아니고, 모든 원소가 0이 아니면 연결 그래프임
- 이 방법이 최선일까?

- GN-1만 보면 안 되는 이유
  - 길이가 k < N-1인 보행으로 도달 가능하지만 길이가 N-1인 보행으로는 불가능할 수 있기 때문
  - 길이가 k < N-1인 보행으로 도달한 이후에 못 움직이도록 막는다면?

- $G_I = G + I_N$ 으로 정의하자.
- G_I^{N-1}에 0이 있으면 연결 그래프가 아니고, 모든 원소가 0이 아니면 연결 그래프임
  - 길이가 k < N-1인 보행을 self-loop에 묶어 놓을 수 있음

질문?