#09-3. 그래프 탐색

나정휘

https://justiceHui.github.io/

그래프의 탐색

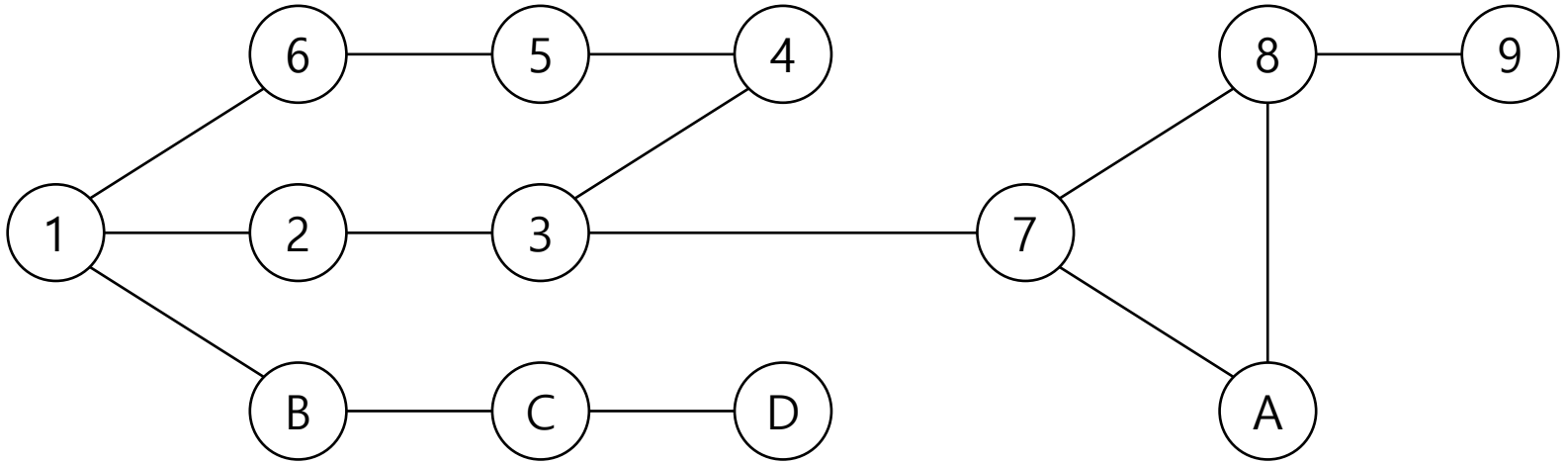
그래프의 탐색

- 그래프의 정점을 적당한 순서로 한 번씩 모두 보는 방법이라고 생각하면 편함
- 깊이 우선 탐색
- 너비 우선 탐색

그래프의 탐색

깊이 우선 탐색 (Depth-First Search, DFS)

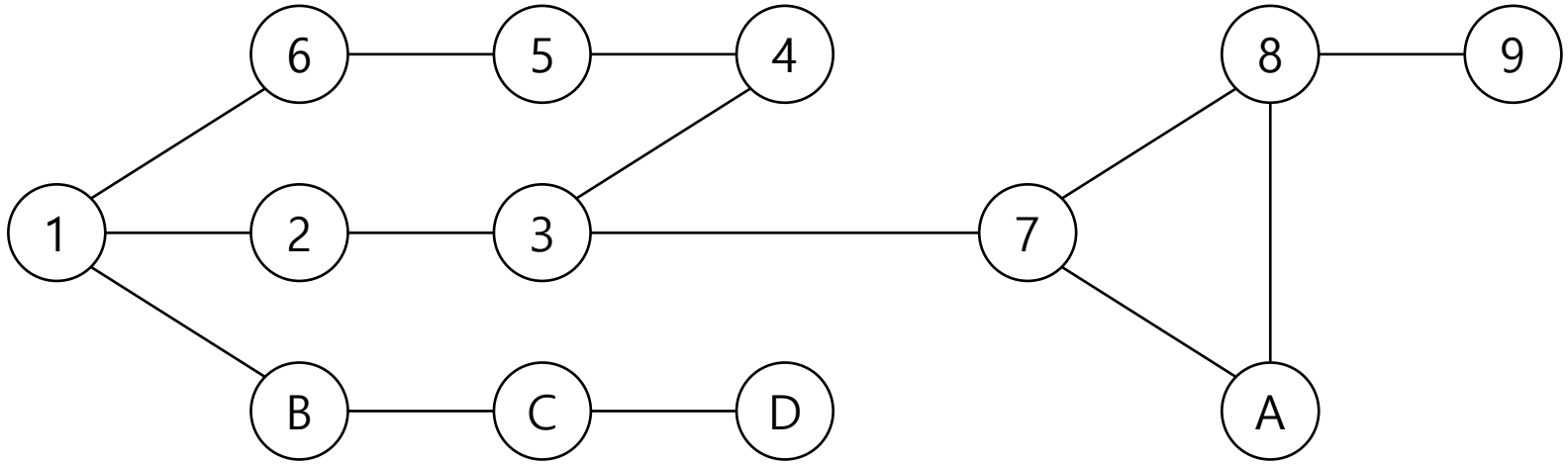
- 한 정점에서 시작
- 현재 정점과 인접한 정점 중 아직 방문하지 않은 정점을 하나 선택해 방문
- 만약 인접한 정점을 모두 방문한 상태라면 이전 정점으로 복귀



그래프의 탐색

깊이 우선 탐색 (Depth-First Search, DFS)

- 만약 인접한 정점을 모두 방문한 상태라면 이전 정점으로 돌아감
  - 스택이나 재귀를 이용해서 구현할 수 있음
  - PS할 때는 보통 재귀를 사용해서 구현함
  - 스택을 사용하는 구현은 함수 호출 스택을 직접 구현하면 됨




```cpp
#include <bits/stdc++.h>
using namespace std;

int N, M, S, G[1010][1010], C[1010];

void DFS(int v){
    cout << v << " ";
    C[v] = 1;
    for(int i=1; i<=N; i++){
        if(G[v][i] && !C[i]) DFS(i);
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> S;
    for(int i=0; i<M; i++){
        int u, v; cin >> u >> v;
        G[u][v] = G[v][u] = 1;
    }
    DFS(S);
}
```

그래프의 탐색

깊이 우선 탐색 (Depth-First Search, DFS)

- 인접 리스트 / 간선 리스트를 사용한 구현

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, M, S, C[1010];
vector<int> G[1010];

void DFS(int v){
    cout << v << " ";
    C[v] = 1;
    for(auto i : G[v]) if(!C[i]) DFS(i);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> S;
    for(int i=0; i<M; i++){
        int u, v; cin >> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    DFS(S);
}
```

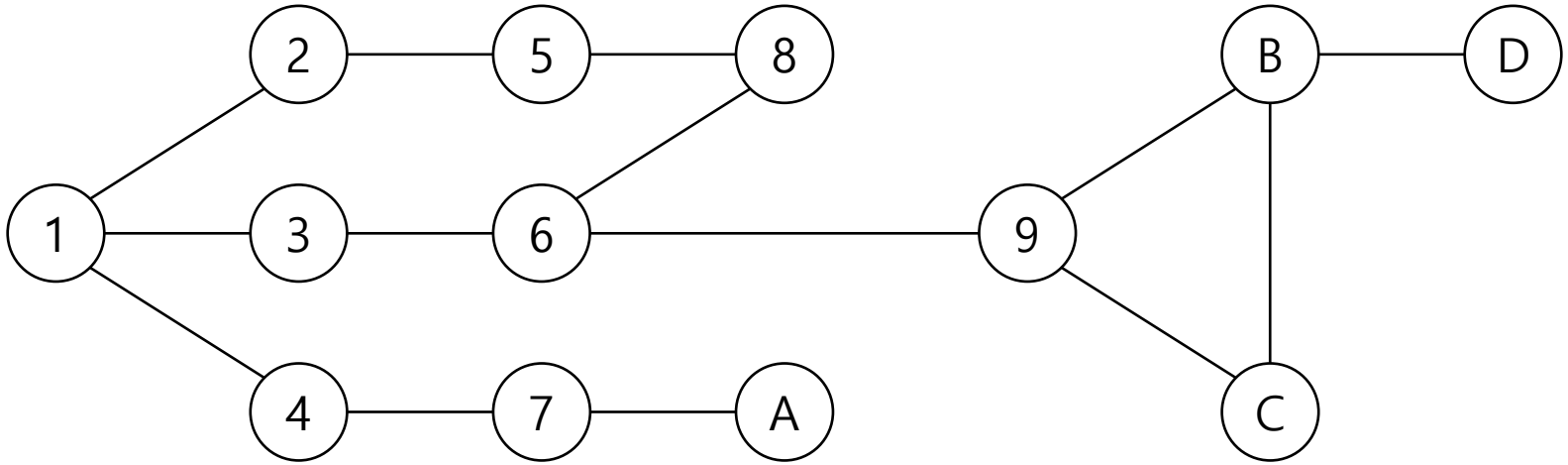
```cpp
#include <bits/stdc++.h>
using namespace std;

int N, M, S, C[1010];
int Head[1010], Next[20202], To[20202], E;

void AddEdge(int s, int e){
    int id = ++E;
    To[id] = e;
    Next[id] = Head[s];
    Head[s] = id;
}

void DFS(int v){
    cout << v << " ";
    C[v] = 1;
    for(int i=Head[v]; i; i=Next[i]) if(!C[To[i]]) DFS(To[i]);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> S;
    vector<pair<int,int>> V;
    for(int i=0; i<M; i++){
        int u, v; cin >> u >> v;
        AddEdge(u, v);
    }
    DFS(S);
}
```

질문?

그래프의 탐색

너비 우선 탐색 (Breadth-First Search, BFS)

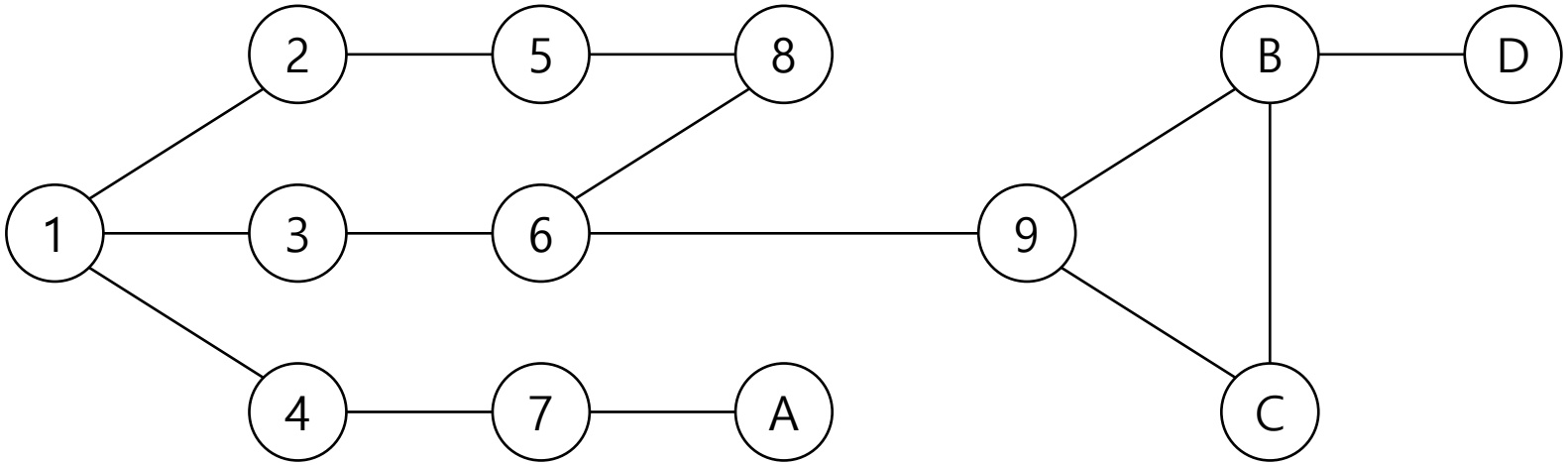
- 한 정점에서 시작
- 현재 정점과 인접한 정점 중 아직 방문하지 않은 정점을 모두 방문



그래프의 탐색

너비 우선 탐색 (Breadth-First Search, BFS)

- 현재 정점과 인접한 정점 중 아직 방문하지 않은 정점을 모두 방문
  - 정점들의 방문 "대기열"을 만들어야 하므로 큐를 사용해서 구현
  - 시작 정점과 거리가 가까운 정점부터 방문하게 됨
  - 큐에 넣는 시점에 방문 체크를 해야 함
    - 큐에서 빼는 시점에 체크하면 안 됨




```
void BFS(int v){
    queue<int> Q;
    Q.push(v); C[v] = 1;
    while(!Q.empty()){
        v = Q.front(); Q.pop();
        cout << v << " ";
        for(int i=1; i<=N; i++){
            if(G[v][i] && !C[i]) Q.push(i), C[i] = 1;
        }
    }
}
```

그래프의 탐색

너비 우선 탐색 (Breadth-First Search, BFS)

- 인접 리스트 / 간선 리스트를 사용한 구현

```cpp
int N, M, S, C[1010];
vector<int> G[1010];

void BFS(int v){
    queue<int> Q;
    Q.push(v); C[v] = 1;
    while(!Q.empty()){
        v = Q.front(); Q.pop();
        cout << v << " ";
        for(auto i : G[v]) if(!C[i]) Q.push(i), C[i] = 1;
    }
}
```

```cpp
int Head[1010], Next[20202], To[20202], E;

void BFS(int v){
    queue<int> Q;
    Q.push(v); C[v] = 1;
    while(!Q.empty()){
        v = Q.front(); Q.pop();
        cout << v << " ";
        for(int i=Head[v]; i; i=Next[i]) if(!C[To[i]])
            Q.push(To[i]), C[To[i]] = 1;
    }
}
```

그래프의 탐색

깊이 우선 탐색 vs 너비 우선 탐색

- 구현 방법
  - DFS: 재귀 or 스택
  - BFS: 큐
- 방문 순서
  - BFS: 거리가 가까운 정점부터 방문
  - DFS: 좋은 성질을 갖고 있지만 어려워서 지금은 생략
- 시간 복잡도
  - 인접 행렬 사용하면 $O(|V|^2)$
  - 인접 리스트 / 간선 리스트 사용하면 $O(|V| + |E|)$

- 두 알고리즘 모두 연결되어 있는 정점들만 방문함
  - 연결 그래프가 아니면 컴포넌트마다 따로 탐색해야 함
  - BOJ 11724. 연결 요소의 개수

질문?

그래프의 탐색

BOJ 2178 미로 탐색

- N * M 크기의 격자에 장애물이 있음
- (1, 1)에서 (N, M)으로 이동할 때 지나야 하는 칸의 최소 개수를 구하는 문제

- 이동할 수 있는 칸을 정점으로 생각하고 인접한 칸들을 간선으로 연결하면
- 한 점에서 다른 점으로 가는 최단 거리 + 1을 구하는 문제
- BFS는 최단 거리를 구하는 알고리즘이므로 BFS를 사용하면 문제를 해결할 수 있음

- 인접한 4방향 탐색
  - (i+1, j) (i-1, j) (i, j+1) (i, j-1)
  - int di[4] = {1, -1, 0, 0}, dj[4] = {0, 0, 1, -1}; 선언하고
  - 반복문 돌면서 i + di[k], j + dj[k] 를 보는 방식으로 구현

그래프의 탐색

```cpp
#include <bits/stdc++.h>
using namespace std;
constexpr int di[] = {1, -1, 0, 0};
constexpr int dj[] = {0, 0, 1, -1};

int N, M, A[111][111], D[111][111];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++){
            char c; cin >> c;
            A[i][j] = c - '0';
        }
    }
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) D[i][j] = -1;

    queue<pair<int,int>> Q;
    Q.push({1, 1}); D[1][1] = 1; // Q.emplace(1, 1);
    while(!Q.empty()){
        // auto [i,j] = Q.front(); Q.pop();
        int i = Q.front().first;
        int j = Q.front().second;
        Q.pop();
        for(int k=0; k<4; k++){
            int r = i + di[k], c = j + dj[k];
            if(r < 1 || c < 1 || r > N || c > M) continue;
            if(A[r][c] == 1 && D[r][c] == -1) D[r][c] = D[i][j] + 1, Q.push({r, c});
        }
    }
    cout << D[N][M];
}
```

질문?