2021. 05. 08 BFS 이론정리

너비 우선 탐색(BFS, Breadth-First Search)

너비 우선 탐색이란

루트 노드(혹은 다른 임의의 노드)에서 시작해서 인접한 노드를 먼저 탐색하는 방법

- 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법이다.
- 즉, 깊게(deep) 탐색하기 전에 넓게(wide) 탐색하는 것이다.
- 사용하는 경우: **두 노드 사이의 최단 경로** 혹은 **임의의 경로를 찾고 싶을 때** 이 방법을 선택한다.
  - Ex) 지구상에 존재하는 모든 친구 관계를 그래프로 표현한 후 Ash와 Vanessa 사이에 존재하는 경로를 찾는 경우
  - 깊이 우선 탐색의 경우 - 모든 친구 관계를 다 살펴봐야 할지도 모른다.
  - 너비 우선 탐색의 경우 - Ash와 가까운 관계부터 탐색
- 너비 우선 탐색(BFS)이 깊이 우선 탐색(DFS)보다 좀 더 복잡하다.

너비 우선 탐색(BFS)의 특징

- 직관적이지 않은 면이 있다.
  - BFS는 시작 노드에서 시작해서 거리에 따라 단계별로 탐색한다고 볼 수 있다.
- BFS는 **재귀적으로 동작하지 않는다.**
- 이 알고리즘을 구현할 때 가장 큰 차이점은, 그래프 탐색의 경우 **어떤 노드를 방문했었는지 여부를 반드시 검사** 해야 한다는 것이다.
  - 이를 검사하지 않을 경우 무한루프에 빠질 위험이 있다.
- BFS는 방문한 노드들을 차례로 저장한 후 꺼낼 수 있는 자료 구조인 **큐(Queue)를 사용한다**.
  - 즉, **선입선출(FIFO)** 원칙으로 탐색
  - 일반적으로 큐를 이용해서 반복적 형태로 구현하는 것이 가장 잘 동작한다.
- 'Prim', 'Dijkstra' 알고리즘과 유사하다.

BFS의 장점

1. 노드의 수가 적고 깊이가 얕은 경우 빠르게 동작할 수 있다.
2. 단순 검색 속도가 깊이 우선 탐색(DFS)보다 빠름
3. 너비를 우선 탐색하기에 답이 되는 경로가 여러개인 경우에도 최단경로임을 보장한다.
4. 최단경로가 존재한다면 어느 한 경로가 무한히 깊어진다해도 최단경로를 반드시 찾을 수 있다.

BFS의 단점

1. 재귀호출의 DFS와는 달리 큐에 다음에 탐색할 정점들을 저장해야 하므로 저장공간이 많이 필요하다.
2. 노드의 수가 늘어나면 탐색해야하는 노드 또한 많아지기에 비현실적이다.

** 마인드

1. 탐색 시작 노드를 큐에 삽입한다.
2. 큐에서 노드를 꺼낸 뒤 해당 노드의 인접 노드 중 방문하지 않은 노드를 모두 큐에 삽입 후 방문 처리
3. 2번을 반복

간단한 bfs (queue) 구현

```
#include <cstdio>
#include <vector>
#include <queue>

using namespace std;

const int MAX = 500;
int n,m;
vector<int> v[MAX];
bool check_bfs[MAX];

void BFS(){
  queue<int> q; //Queue 생성
  q.push(0); //초기 시작점 저장
  check_bfs[0] = true; //초기 방문 체크

  while(!q.empty()){
    int current = q.front();
    q.pop();
    printf("%d ",current);

    for(int i=0;i<v[current].size();i++){
      int next = v[current][i];

      if(!check_bfs[next]){
        check_bfs[next] = true;
        q.push(next);
      }
    }
  }
}
```

```
int main() {

  scanf("%d %d",&n,&m);

  for(int i=0;i<m;i++){
    int a,b;
    scanf("%d %d",&a,&b);

    v[a].push_back(b);
    v[b].push_back(a);
  }

  BFS();
  return 0;
}
```