알고리즘 정리

2022년 3월 14일 월요일 오후 10:41

[BFS와 DFS]

-BFS : 너비 우선 탐색

-DFS: 깊이 우선 탐색

https://velog.io/@lucky-korma/DFS-BFS%EC%9D%98-%EC%84%A4%EB%AA%85-%EC%B0%A8%EC%9D%B4%EC%A0%90

1). DFS : tree에서 깊이부터 본다.

: 예를들면 현재 위치부터 동서남북 4way움직임 배열과, , 도착하는 모든 경우의 수를 다 해보는 것. ..

ㅁ 보통 재귀로 구현한다. 재귀는 void형으로 하는게 여러모로 편한듯하다

ㅁ DFS도 visit 표시가 필요하다. 각 재귀함수의 시작에서 visit 체크, return 직전에 해제 하면된다. return 포인트가 여러개인 경우 절대 uncheck를 잊지마라!

ㅁ ++n 등을 사용할땐 특히나 재귀에서는 주의해라. 재귀하고 돌아왔을 때 n 값을 또 사용할때 그 자체가 증가해서 답에 오류 발생한다.

2). BFS : level별로 본다.

: Queue를 사용하여, (FIFO 자료구조)

ㅁ BFS 의 근본은 level 순서대로 들어가야 한다는 것이다.!!

같은 level의 정보 쭉 들어가고 -> 그 다음 level의 정보 쭉 들어감

이렇게 안하면 Queue를 끝까지 다 pop해야 최단경로 (level)의 정답을 찾는 비효율이 발생한다

ㅁVisit을 표시해주는 시점은 Q에서 해당위치 자료를 꺼낸 다음에 하라. 넣기전에 하면 조건체크 헷갈린다.

● 기타꿀팁

- 다음의 3가지 문제유형에서

1). 그래프의 모든 node 방문하는 것이 주요 문제 : BFS DFS 둘다 상관 x

2). 경로의 특징을 저장해야 하는 문제 : DFS (BFS는 경로의 특징을 가지지 못함)

3). 최단거리 구해야 하는 문제 : BFS가 유리. 미로찾기 등

이외에도 그래프가 정말 크다면 DFS고려. 규모가 크지 않다면 BFS

<FloodFill 알고리즘> : 나눠진 영역의 개수를 세는 알고리즘

- BFS탐색과, Queue를 사용하는 것이 대표적

[Union find 알고리즘]

: 서로 다른 집합들을 합쳐가는 알고리즘

: 트리 자료구조형으로 구현하는 것이 가장 효율적이다. depth path가 탐색시간이기 때문에 O(N)보다는 빠름

1). 구현방법 : 재귀 + arr의 사용

arr[자식노드] = 부모노드; 즉 각 노드에서 부모노드를 가리키도록 구현. (이거 hash map으로도 구현 가능할듯)

2). API

- find 함수 : 찾고자하는 x, y 등에서 시작해서 재귀로 루트노드를 찾음. 루트노드는 자료명[x]== x 즉 자기 자신을 부모노드로 가지는 곳
- Union (합치기) : x가 속한 집합과, y가 속한 집합 각각의 루트노드 a, b를 찾은 뒤. 자료명[a] =b 이렇게 서로 루트노드를 이어준다. 이렇게 루트노드끼리 붙여야 탐색속도 이득

[Binary Search 이진탐색] -parametric search 에도 응용가능

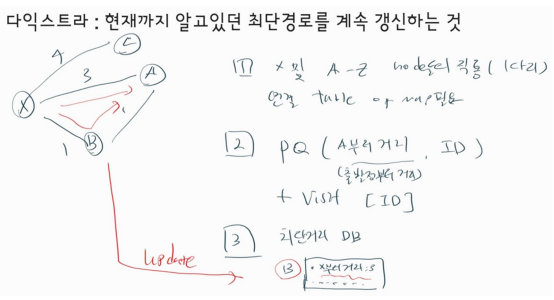
```
int BinarySearch (int MAX, 찾는대상목록 등){
    int left = 1 // (min 값)
    int right = MAX;

    while(right>=left){
        int mid = (right+left)/2;

        /// 탐색 과정 거침 ... ///

        //1. 정답이라면 => 값 반환

        // ㅁ 오답인 경우
        //2-1. 증가시켜야 하는경우
        left = mid+1;
        //2-2. 감소시켜야 하는경우
        right = mid-1;
    }
     return -1;
}
```

[DP Dynamic Programming]

하나의 큰 문제를 여러 동일한 작은 문제들로 나눌 수 있을때. (그리고 딴거로 안풀릴때.. 의심)

-문제를 분석할때 단계의 맨앞이나 맨 뒤를 기준으로 한단계 진행하거나, 빼서 DP를 어떻게 설계하면 좋을지 생각

[Djik]



ㅁ 그리디(Greedy) 알고리즘 : 단계별의 모든 경우의 수를 탐색하는 것이 아닌, 매 단계의 최선의 선택만을 하는 것 -사용하려면 알고리즘이 항상 최적해를 찾는다는 정당성이 필요하다 -eg. 강의실 최대로 많이 예약하는 시간표 구하기 : greedy 알고리즘이 최적해가 아니게 되는 경우는 없다	