Week 10

2024-09-05

1194 달이 차오른다 가자

BFS + 비트마스킹

시작점과 도착점이 있고

소문자 위치의 키를 획득해야만 대문자 위치를 갈 수 있음

1194 달이 차오른다 가자

N <= 50, M <= 50, 키는 6 개니깐

dist[50][50][1 << 6] 으로 거리 배열을 만들어줌

50 * 50 * (1 << 6) == 160,000 라서 모든 경우를 탐색한다 해도 시간은 충분함

배열크기 1 << 6 은 0 ~ 2^6 -1 이므로

0b000000 부터 0b111111 까지의 모든 값을 가질 수 있음

ex) r == 10, c == 8 이고 a, c 키를 먹은 상태라면

(10, 8, 0b000101) 로 나타낼 수 있다

1194 달이 차오른다 가자

BFS 에서 일반적으로 (x, y) 를 큐에 넣는것과 다르게

(x, y, state) 를 넣고 탐색을 진행한다

입력 받기

```
int n, m;
cin >> n >> m;
vector<string> a(n);

queue<tuple<int,int,int>> q;
memset(dist, -1, sizeof(dist));

for(int i = 0; i < n; i++) {
    cin >> a[i];
    for(int j = 0; j < m; j++) {
        if(a[i][j] == '0') {
            q.emplace(i, j, 0);
            dist[i][j][0] = 0;
        }
    }
}

while(!q.empty()) {
    auto [x, y, state] = q.front();
    q.pop();
    if(a[x][y] == '1') {
        return !(cout << dist[x][y][state] << '\n');
    }
```

1194 달이 차오른다 가자

탐색할때 경우의 수

1. 키를 먹는 경우

   n_state = state | 1 << key

   로 바꿔주고 다음칸으로 이동

2. 문에 도착한 경우

   if(state && 1 << key) 인 경우에만 이동

3. 그냥 길인 경우

   현재 state 그대로 이동

키를 먹는 경우

```
for(int k = 0; k < 4; k++) {
    int nx = x + dx[k];
    int ny = y + dy[k];
    if(nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
    if(a[nx][ny] == '#') continue;
    if('a' <= a[nx][ny] && a[nx][ny] <= 'f') {
        int now = a[nx][ny] - 'a';
        int n_state = state | 1 << now;
        if(dist[nx][ny][n_state] == -1) {
            dist[nx][ny][n_state] = dist[x][y][state] + 1;
            q.emplace(nx, ny, n_state);
        }
```

키를 먹는 경우

키를 먹을 때 1 번 먹고 2 번을 먹든
2 번 먹고 1 번을 먹든 순서는 중요하지 않음

BFS 탐색을 하고있기 때문에 최적의 경우만 구해짐 .

2, 1 을 먹고 x,y 에 도착했고
나중에 1, 2 를 먹고 x y 에 도착했다면
두번째에는 dist 가 갱신이 안 됨
dist[x][y][0b000011] 이 -1 이 아니기 때문이다
여러개인 경우에도 마찬가지

문에 도착한 경우

```
} else if('A' <= a[nx][ny] && a[nx][ny] <= 'F') {
    int now = a[nx][ny] - 'A';
    if(state & 1 << now && dist[nx][ny][state] == -1) {
        dist[nx][ny][state] = dist[x][y][state] + 1;
        q.emplace(nx, ny, state);
    }
```

그냥 길인 경우

```
} else {
    if(dist[nx][ny][state] != -1) continue;
    dist[nx][ny][state] = dist[x][y][state] + 1;
    q.emplace(nx, ny, state);
}
```

도착 못 하는 경우

```
            }
        }

        cout << -1 << '\n';

        return 0;
    }
```