내리막 길

https://www.acmicpc.net/problem/1520

- dp[i][j] = (i, j)에서 시작해서 (N,M)으로 가는 내리막 길의 개수
- 이동하는 방향이 4방향이므로 탐색하면서 내리막길로 내려간다. 수가 감소하는 방향으로만 이동할 수 있기 때문에 사이클은 생기지 않는다.
- dp[i][j] += dp[y][x]
- Top – Down 방식의 탐색이 효과적이다.

```
int solve(int y, int x) {
    if (y == N && x == M){
        return 1;
    }

    if (dp[y][x] != -1) {
        return dp[y][x];
    }

    dp[y][x] = 0;
    for (int k = 0; k < 4; ++k){
        int ny = y + dy[k];
        int nx = x + dx[k];
        if (A[ny][nx] > 0 && A[y][x] > A[ny][nx]) {
            dp[y][x] += solve(ny, nx);
        }
    }

    return dp[y][x];
}
```

내리막 길

https://www.acmicpc.net/problem/1520

- dp[i][j] = (i, j)에서 시작해서 (N,M)으로 가는 내리막 길의 개수
- 이동하는 방향이 4방향이므로 탐색하면서 내리막길로 내려간다. 수가 감소하는 방향으로만 이동할 수 있기 때문에 사이클은 생기지 않는다.
- dp[i][j] += dp[y][x]
- 좌표를 높이 기준으로 내림차순으로 정렬한다.
- 높은 순으로 차례대로 방문하면서 dp 테이블을 채워 나아간다.

```cpp
struct Cell {
    int row, col, val;

    /*bool operator < (const Cell _cell) {
        return val < _cell.val;
    }*/
};

bool cmp(const Cell &u, const Cell &v) {
    return u.val > v.val;
}
```

```cpp
sort(v.begin(), v.end(), cmp);

dp[1][1] = 1;
for (int i = 0; i < v.size(); i++) {

    int y = v[i].row;
    int x = v[i].col;

    for (int k = 0; k < 4; k++) {

        int ny = y + dy[k];
        int nx = x + dx[k];

        if (A[ny][nx] >= 0 && A[ny][nx] < A[y][x]) {
            dp[ny][nx] += dp[y][x];
        }
    }
}
```