백준 2186번 - 문자판
문제
입력
출력
예제 입력1
예제 출력1
출처
알고리즘 분류
접근 방법
소스코드

백준 2186번 - 문자판

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2초	128MB	6427	1397	924	21.439%

문제

알파벳 대문자가 한 칸에 한 개씩 적혀있는 N×M 크기의 문자판이 있다. 편의상 모든 문자는 대문자라 생각하자. 예를 들어 아래와 같은 문자판을 보자.

K	**A**	**K**	**T**
X	E	A	S
Y	R	W	U
Z	B	Q	P

이 문자판의 한 칸(아무 칸이나 상관없음)에서 시작하여 움직이면서, 그 칸에 적혀 있는 문자들을 차례대로 모으면 하나의 단어를 만들 수 있다. 움직일 때는 상하좌우로 K개의 칸까지만 이동할 수 있다. 예를 들어 K=2일 때 아래의 그림의 가운데에서는 'X' 표시된 곳으로 이동할 수 있다.

		X		
		X		
X	X		X	X
		X		
		X		

반드시 한 칸 이상 이동을 해야 하고, 같은 자리에 머물러 있을 수 없다. 또, 같은 칸을 여러 번 방문할 수 있다.

이와 같은 문자판과 K, 그리고 하나의 영단어가 주어졌을 때, 이와 같은 영단어를 만들 수 있는 경로가 총 몇 개 존재하는지 알아내는 프로그램을 작성하시오.

위의 예에서 영단어가 BREAK인 경우에는 다음과 같이 3개의 경로가 존재한다. 앞의 수는 행 번호, 뒤의 수는 열 번호를 나타낸다.

- (4, 2) (3, 2) (2, 2) (1, 2) (1, 1)
- (4, 2) (3, 2) (2, 2) (1, 2) (1, 3)
- (4, 2) (3, 2) (2, 2) (2, 3) (1, 3)

입력

첫째 줄에 N(1 ≤ N ≤ 100), M(1 ≤ M ≤ 100), K(1 ≤ K ≤ 5)가 주어진다. 다음 N개의 줄에는 M개의 알파벳 대문자가 주어지는데, 이는 N×M 크기의 문자판을 나타낸다. 다음 줄에는 1자 이상 80자 이하의 영단어가 주어진다. 모든 문자들은 알파벳 대문자이며, 공백 없이 주어진다.

출력

첫째 줄에 경로의 개수를 출력한다. 이 값은 $2^{31} - 1$보다 작거나 같다.

예제 입력1

```
4 4 1
KAKT
XEAS
YRWU
ZBQP
BREAK
```

예제 출력1

```
3
```

출처

출처

알고리즘 분류

- 다이나믹 프로그래밍
- 그래프 이론
- 그래프 탐색
- 깊이 우선 탐색

접근 방법

임의의 $[y][x]$에서부터 시작하여, 주어진 문자열 s 에 매칭되는 경우를 생각해보자.

1번째 문자가 매칭되는 경우는 $board[y][x] = s[0]$ 이다.

2번째 문자가 매칭되는 경우는 $board[y'][x'] = s[1]$ 이다.

이것을 일반화하여, $[a][b]$에서 $N-1$번째 문자까지 매칭되었다면, $[a'][b']$에서 매칭되는 경우는 $board[a'][b'] = s[N]$일 때이다.

규칙성에 따라서, 특정 위치 $[i][j]$에서 매핑되는 모든 경우의 수는 $[a][b]$에서 (a, b는 i, j 이전의 위치) $N-1$번째 문자까지 매칭이 완료된 모든 경우의 수의 합이다.

아래와 같이 메모이제이션 테이블을 정의해보자.

`dp[i][j][k] := 문자열s의 k번째 인덱스까지 [i][j] 칸에 도달하는 모든 경우의 수`

점화식의 base-case는 $board[i][j][0] = s[0]$일 때이고, 해당 조건을 만족할 때 적절하게 top-bottom dp방식으로 구현하면 된다.

소스코드

```cpp
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
////////////////////////////////////////////////////////////////////
#include <bits/stdc++.h>
using namespace std;
int N, M, K, dp[105][105][85];
char board[105][105];
string s;
const int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
int dfs(int y, int x, int k){
    if(k == s.size() - 1) {
        return dp[y][x][k] = 1;
    }
    int &ret = dp[y][x][k];
    if(ret != -1)
        return ret;
    ret = 0;
    // 최대 K칸 만큼 이동 가능
    vector<pair<int,int>> next;
    for(int d=0; d<4; d++){
        for(int l=1; l<=K; l++){
            int ny = y + l * dir[d][0], nx = x + l * dir[d][1];
            if(ny < 0 || nx < 0 || ny >= N || nx >= M) continue;
            next.emplace_back(ny, nx);
        }
    }
```

```
    for(const auto &[ny, nx] : next){
        if(k+1 < s.size() && board[ny][nx] == s[k+1])
            ret += dfs(ny, nx, k+1);
    }
    return ret;
}
int main(void){
    FASTIO;
////////////////////////////////////////////////////////////////
    cin >> N >> M >> K;
    for(int i=0; i<N; i++){
        for(int j=0; j<M; j++){
            cin >> board[i][j];
        }
    }
    cin >> s;
    // dp[i][j][k] : [i][j] 까지 문자열의 k번째까지 매칭 되는 모든 경우의 수
    memset(dp, -1, sizeof(dp));
    int ans = 0;
    for(int i=0; i<N; i++){
        for(int j=0; j<M; j++){
            if(board[i][j] == s[0])
                ans += dfs(i, j, 0);
        }
    }

    cout << ans ;
    return 0;
}
```