Chuyên đề luyện chiều 25.2
QUY HOẠCH ĐỘNG TRÊN LƯỚI (GRID DP)

Trong bồi dưỡng học sinh giỏi Tin học, **Quy hoạch động (Dynamic Programming – DP)** luôn được xem là một trong những mảng kiến thức có vai trò then chốt và mang tính "bản lề". Thực tiễn giảng dạy và ôn luyện cho thấy, nếu học sinh chỉ dừng lại ở mức nắm được các thuật toán duyệt cơ bản như DFS, BFS hay các kỹ thuật xử lý mảng – chuỗi thông thường, các em có thể giải quyết tốt các bài toán ở mức **nhận biết và thông hiểu**, thậm chí đạt điểm khá trong các đề kiểm tra phổ thông. Tuy nhiên, để vượt qua các bài toán **phân loại** trong đề thi học sinh giỏi – nơi yêu cầu tư duy trừu tượng, khả năng mô hình hóa và tối ưu hóa thuật toán – thì **DP gần như là con đường bắt buộc**.

Trong hệ thống các dạng bài Quy hoạch động, **Quy hoạch động trên lưới (Grid DP)** là một chuyên đề đặc biệt quan trọng. Đây là dạng bài vừa **trực quan**, vừa **giàu khả năng mở rộng**, rất phù hợp để dẫn dắt học sinh từ những bài toán DP cơ bản đến các bài toán nâng cao. Tính trực quan thể hiện ở chỗ bài toán thường được mô tả dưới dạng một bản đồ hình chữ nhật gồm các ô vuông, trong đó học sinh có thể "nhìn thấy" trạng thái, bước di chuyển, chướng ngại vật và đích đến. Điều này giúp giảm đáng kể rào cản ban đầu khi tiếp cận DP – một mảng kiến thức vốn được xem là khó và trừu tượng.

Tuy nhiên, **độ khó của Grid DP không nằm ở hình thức mô tả**, mà nằm ở khả năng biến đổi và mở rộng điều kiện bài toán. Chỉ cần thay đổi một yếu tố nhỏ như:

- đếm số đường đi thay vì kiểm tra tồn tại đường đi,
- tối ưu chi phí thay vì chỉ đi hợp lệ,
- giới hạn số lần đổi hướng,
- cho phép nhiều kiểu di chuyển khác nhau,
- hoặc bổ sung thêm trạng thái phụ (thời gian, năng lượng, số bước đặc biệt…),

thì bài toán lập tức chuyển từ mức cơ bản sang mức **khó – rất khó**, đủ sức phân loại học sinh giỏi ở cấp huyện, tỉnh, thậm chí quốc gia. Chính vì vậy, Grid DP thường xuyên xuất hiện trong các đề thi chọn đội tuyển, đề thi HSG các cấp, cũng như trong các hệ thống bài tập uy tín như AtCoder, USACO hay LeetCode nâng cao.

Xuất phát từ vai trò và tầm quan trọng đó, chuyên đề **"Quy hoạch động trên lưới (Grid DP)"** được xây dựng với mục tiêu không chỉ giúp học sinh **biết cách làm bài**, mà quan trọng hơn là **hiểu bản chất tư duy DP**, biết cách thiết kế trạng thái, xây

dựng công thức chuyển, và đánh giá độ phức tạp thuật toán. Chuyên đề được thiết kế theo **cấu trúc chuẩn của chuyên đề Hùng Vương**, đảm bảo tính hệ thống và khả năng sử dụng trực tiếp trong giảng dạy đội tuyển.

Cụ thể, chuyên đề bao gồm:

- **Phần cơ sở lý thuyết**: trình bày tư duy chung của Grid DP, cách xác định trạng thái, hướng chuyển, điều kiện biên và các kỹ thuật tối ưu thường gặp.
- **Hệ thống bài tập vận dụng**: sắp xếp theo mức độ tăng dần từ cơ bản đến nâng cao, giúp học sinh từng bước làm quen, rèn luyện và mở rộng tư duy.
- **Hướng dẫn thuật toán và code C++ minh họa** cho từng dạng bài, đảm bảo học sinh không chỉ hiểu ý tưởng mà còn biết cách cài đặt chuẩn xác.
- **Phân tích lỗi thường gặp**: chỉ ra những sai lầm phổ biến của học sinh khi làm Grid DP, từ lỗi tư duy đến lỗi cài đặt.

Với cấu trúc như vậy, chuyên đề không chỉ là một tập hợp bài tập, mà còn là **một lộ trình học tập có định hướng**, giúp học sinh từng bước hình thành tư duy giải quyết các bài toán Quy hoạch động trên lưới – một kỹ năng quan trọng để chinh phục các bài toán Tin học ở mức độ cao.

PHẦN I. CƠ SỞ LÝ THUYẾT

1. Khái niệm Quy hoạch động trên lưới (Grid DP)

Quy hoạch động trên lưới (Grid DP) là nhóm bài toán Quy hoạch động trong đó không gian trạng thái được tổ chức dưới dạng một lưới hai chiều, thường là hình chữ nhật gồm các ô có tọa độ (i, j). Mỗi ô biểu diễn một trạng thái, và việc di chuyển giữa các ô tuân theo những quy tắc nhất định (sang phải, xuống dưới, chéo hoặc các dạng mở rộng khác).

Trong Grid DP, ta thường định nghĩa dp[i][j] là một đại lượng gắn với ô (i, j), chẳng hạn:

- Số cách để đi từ ô xuất phát đến ô (i, j);
- Chi phí nhỏ nhất hoặc lớn nhất để đến được ô (i, j);
- Giá trị tốt nhất đạt được khi dừng tại ô (i, j).

Bản chất của Grid DP không nằm ở việc "đi trên lưới", mà nằm ở cách mô hình hóa bài toán thành các trạng thái có quan hệ phụ thuộc lẫn nhau. Khi tiếp cận một bài Grid DP, học sinh cần đặc biệt chú ý ba yếu tố cốt lõi: Xác định đúng ý nghĩa của trạng thái dp[i][j]; Xác định chính xác các trạng thái có thể chuyển tới dp[i][j] và lựa chọn thứ tự duyệt phù hợp để đảm bảo các trạng thái cần thiết đã được tính trước.

2. Hai dạng bài Grid DP kinh điển

a) Bài toán đếm số cách

Ở dạng bài này, dp[i][j] thường biểu diễn số cách khác nhau để đi đến ô (i, j). Công thức chuyển trạng thái thường là tổng số cách từ các ô có thể đi tới ô hiện tại. Đặc trưng của dạng này là phép toán cộng, dễ phát sinh số lớn và thường phải lấy modulo.

b) Bài toán tối ưu (min/max)

Ở dạng bài này, dp[i][j] biểu diễn giá trị tối ưu (nhỏ nhất hoặc lớn nhất) khi đi đến ô (i, j). Công thức chuyển trạng thái thường sử dụng phép min hoặc max kết hợp với chi phí tại ô hiện tại. Hai dạng bài trên có cùng một mô hình tư duy DP, chỉ khác nhau ở ý nghĩa trạng thái và phép toán chuyển.

3. Kỹ năng xử lý biên và kiểm soát độ phức tạp

Grid DP rất dễ sai ở xử lý các trường hợp biên như hàng đầu tiên, cột đầu tiên, ô xuất phát, ô đích và các ô bị chặn. Do đó, học sinh cần hình thành thói quen kiểm tra biên và khởi tạo trạng thái ban đầu một cách cẩn thận.

Về độ phức tạp, phần lớn các bài Grid DP cơ bản có độ phức tạp $O(n \times m)$. Khi bài toán được mở rộng bằng cách thêm trạng thái phụ, độ phức tạp thường tăng lên $O(n \times m \times S)$, với S là số trạng thái phụ. Việc đánh giá đúng độ phức tạp giúp học sinh lựa chọn được phương pháp giải phù hợp.

PHẦN II. BÀI TOÁN VẬN DỤNG

BÀI TOÁN 1. ĐƯỜNG ĐI RIÊNG BIỆT (DUONGDI.*)

Nguồn: https://leetcode.com/problems/unique-paths/

Cho hai số nguyên dương m và n biểu diễn kích thước của một lưới hình chữ nhật gồm m hàng và n cột.

Một robot bắt đầu tại ô góc trên bên trái (1,1) và cần di chuyển tới ô góc dưới bên phải (m, n).

Tại mỗi bước, robot chỉ được phép:

- Di chuyển sang phải
- Di chuyển xuống dưới

Hãy xác định số đường đi khác nhau mà robot có thể thực hiện.

Dữ liệu vào

Một dòng chứa hai số nguyên dương m, n ($1 \le m, n \le 100$).

Dữ liệu ra

Một số nguyên duy nhất là số đường đi hợp lệ từ (1,1) đến (m,n).

Ví dụ

Input	Output
3 7	28

***Hướng dẫn thuật toán:**

Ta đặt dp[i][j] là số đường đi để tới ô (i,j) (Ta bắt đầu từ chỉ số 1). Vì chỉ đi phải hoặc xuống nên để tới (i,j) chỉ có hai nguồn: (i-1,j) và (i,j-1). Do đó:

dp[i][j] = dp[i-1][j] + dp[i][j-1]

Biên: dp[1][1] = 1; hàng 1 chỉ đi từ trái sang, cột 1 chỉ đi từ trên xuống.

Độ phức tạp: $O(m \times n)$

*** Code tham khảo:**

```
#include <bits/stdc++.h>
using namespace std;
```

```
int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);
   int m, n;
   cin >> m >> n;
   vector<vector<long long>> dp(m + 1, vector<long long>(n + 1, 0));
   dp[1][1] = 1;
   for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= n; j++) {
         if (i == 1 && j == 1) continue;
         dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
      }
   }
   cout << dp[m][n] << "\n";
   return 0;
}
```

*** Cảm nhận:**

Đây là bài "vỡ lòng" của Grid DP. Tôi thường dùng để kiểm tra 3 năng lực nền: (1) học sinh hiểu trạng thái dp là gì, (2) có biết xử lý biên không, (3) có viết được vòng lặp duyệt đúng thứ tự không. Những em làm sai bài này thường sẽ gặp khó khăn với các bài Grid DP có chướng ngại hoặc tối ưu sau đó.

***Một số lỗi học sinh hay mắc phải:**

- Nhầm dp[i][j] là số đường từ (i,j) đến đích thay vì từ đầu đến (i,j), dẫn tới duyệt sai thứ tự.
- Quên khởi tạo dp[1][1]=1 hoặc khởi tạo nhầm toàn bộ hàng/cột là 1 khi chưa phân tích biên.
- Dùng kiểu int gây tràn với m,n lớn (mặc dù bài gốc tránh tràn, nhưng thói quen nên dùng long long).
- Nếu nén 1D thì cập nhật sai thứ tự (đè mất dp[j] trước khi dùng).

BÀI TOÁN 2. Đếm đường đi có chướng ngại vật (DEMDUONG.*)

Nguồn: https://atcoder.jp/contests/dp/tasks/dp_h

Cho một lưới hình chữ nhật kích thước H × W. Mỗi ô trong lưới là:

- '.' : ô trống, có thể đi qua

- '#' : ô bị chặn, không thể đi qua

Bạn bắt đầu tại ô trên cùng bên trái (1,1) và cần đi đến ô dưới cùng bên phải (H,W).

Tại mỗi bước, bạn chỉ được phép:

- Đi xuống dưới
- Đi sang phải

Hãy tính số đường đi khác nhau từ (1,1) đến (H,W).

Dữ liệu vào

- Dòng đầu chứa hai số nguyên H, W (1 ≤ H, W ≤ 1000)
- H dòng tiếp theo, mỗi dòng là một xâu gồm W ký tự '.' hoặc '#'

Dữ liệu ra

In ra số đường đi hợp lệ từ (1,1) đến (H,W), lấy kết quả modulo $10^9 + 7$.

Ví dụ

Input	Output
3 4 .#.. ...#	3

*** Hướng dẫn thuật toán**

Ta vẫn đặt $dp[i][j]$ là số đường đi tới ô (i, j). Nếu ô (i, j) là vật cản thì $dp[i][j] = 0$.

Nếu ô trống thì: $dp[i][j] = (dp[i-1][j] + dp[i][j-1]) \; mod \; MOD$

Khởi tạo $dp[1][1] = 1$ (vì (1,1) đảm bảo trống). Duyệt i=1..H, j=1..W.

Độ phức tạp: $O(H \times W)$.

*** Code tham khảo:**

```
#include <bits/stdc++.h>
using namespace std;
static const long long MOD = 1000000007LL;
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int H, W;
  cin >> H >> W;
  vector<string> g(H);
```

```
    for (int i = 0; i < H; i++) cin >> g[i];
    vector<vector<long long>> dp(H, vector<long long>(W, 0));
    dp[0][0] = 1;
    for (int i = 0; i < H; i++) {
        for (int j = 0; j < W; j++) {
            if (g[i][j] == '#') { dp[i][j] = 0; continue; }
            if (i == 0 && j == 0) continue;
            long long ways = 0;
            if (i > 0) ways += dp[i - 1][j];
            if (j > 0) ways += dp[i][j - 1];
            dp[i][j] = ways % MOD;
        }
    }
    cout << dp[H - 1][W - 1] << "\n";
    return 0;
}
```

*** Cảm nhận:**

Bài này là "cửa ải" đầu tiên trong Grid DP vì nó buộc học sinh vừa giữ đúng tư duy DP vừa xử lý điều kiện hợp lệ và modulo. Trong luyện đội tuyển, tôi dùng bài này để rèn thói quen: Gặp bài đếm số cách với giới hạn lớn thì phải nghĩ tới modulo và phải kiểm soát kỹ các ô bị chặn.

***Một số lỗi học sinh hay mắc phải:**

- Quên đặt dp=0 tại ô '#' hoặc vẫn cộng dp từ ô '#'.
- Quên modulo sau phép cộng dẫn đến tràn số.
- Nhầm ký hiệu '.' và '#' hoặc nhầm chỉ số 0/1-index khi cài đặt.
- Khởi tạo dp[0][0]=1 nhưng quên xử lý trường hợp ô đầu/ô cuối luôn trống theo đề.

BÀI TOÁN 3. Đường đi có tổng giá trị lớn nhất (TONGMAX.*)

Nguồn: https://usaco.guide/gold/paths-grids

Cho một lưới hình chữ nhật kích thước n × m. Mỗi ô (i, j) trong lưới có một giá trị nguyên a[i][j].

Bạn xuất phát từ ô (1,1) (góc trên bên trái) và cần di chuyển tới ô (n,m) (góc dưới bên phải). Tại mỗi bước, bạn chỉ được phép:

- Đi sang phải
- Đi xuống dưới

Hãy tìm giá trị lớn nhất của tổng các giá trị trên một đường đi từ (1,1) tới (n,m).

Input:

- Dòng đầu chứa hai số nguyên n, m.
- n dòng tiếp theo, mỗi dòng gồm m số nguyên a[i][j].

Output:

- In ra một số nguyên là tổng giá trị lớn nhất có thể đạt được.

Ràng buộc

- $1 \leq n, m \leq 2000$
- $|a[i][j]| \leq 10^9$

Input	Output
2 3 1 2 3 4 5 6	16

***Hướng dẫn thuật toán:**

Đây là bài tối ưu hóa nên trạng thái dp[i][j] là "tổng lớn nhất" để tới (i,j). Quan hệ chuyển trạng thái:

dp[i][j] = max(dp[i-1][j], dp[i][j-1]) + a[i][j]

Biên: dp[1][1]=a[1][1], hàng 1 chỉ từ trái, cột 1 chỉ từ trên.

Độ phức tạp $O(n \times m)$.

***Code mẫu:**

```
#include <bits/stdc++.h>
using namespace std;
int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);
   int n, m;
   cin >> n >> m;
   vector<vector<long long>> a(n + 1, vector<long long>(m + 1, 0));
   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= m; j++)
         cin >> a[i][j];
```

```
    const long long NEG_INF = -(1LL<<62);
    vector<vector<long long>> dp(n + 1, vector<long long>(m + 1, NEG_INF));
    dp[1][1] = a[1][1];
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (i == 1 && j == 1) continue;
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + a[i][j];
        }
    }
    cout << dp[n][m] << "\n";
    return 0;
}
```

***Cảm nhận:**

Bài này là bước chuyển tư duy quan trọng: từ "đếm số cách" sang "tối ưu giá trị". Trong đội tuyển, tôi hay giao bài này sau khi học sinh đã thuần thục bài đếm đường đi. Em nào còn lẫn lộn giữa dp cộng và dp lấy max/min thì sẽ bộc lộ rất rõ ở bài này.

*** Một số lỗi học sinh hay mắc:**

- Dùng 0 làm giá trị khởi tạo cho dp khiến sai khi a[i][j] âm (phải dùng -INF).
- Xử lý biên không đúng: ví dụ dp[1][j] phải đi từ dp[1][j-1] chứ không được lấy max với -INF mà chưa khởi tạo.
- Nhầm mục tiêu: tìm đường đi dài nhất theo số bước thay vì tổng giá trị.
- Tràn số khi tổng có thể vượt int; cần long long.

BÀI TOÁN 4. Đường đi có tổng chi phí nhỏ nhất (CHIPHI_MIN.*)

Nguồn: https://leetcode.com/problems/minimum-path-sum/

Cho một bảng số hình chữ nhật gồm m hàng và n cột. Mỗi ô chứa một số nguyên không âm. Bạn bắt đầu từ ô ở góc trên bên trái (1,1) và chỉ được phép di chuyển sang phải hoặc xuống dưới để đến ô ở góc dưới bên phải (m,n).

Yêu cầu: Tìm tổng nhỏ nhất của các số trên một đường đi hợp lệ từ ô (1,1) đến ô (m,n).

Dữ liệu vào:

- Dòng 1: Hai số nguyên m, n ($1 \le m, n \le 200$)
- m dòng tiếp theo, mỗi dòng gồm n số nguyên không âm a[i][j] ($0 \le a[i][j] \le 200$)

Dữ liệu ra:

- Một số nguyên duy nhất là tổng nhỏ nhất tìm được.

Ví dụ

Input	Output
3 3 1 3 1 1 5 1 4 2 1	7

Giải thích

Đường đi có tổng nhỏ nhất là: 1 → 3 → 1 → 1 → 1 (tổng = 7).

***Hướng dẫn thuật toán:**

Đây là bài tối ưu min. Đặt dp[i][j] là tổng nhỏ nhất để tới (i,j).

Công thức: dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]

Biên: dp[1][1]=grid[1][1]. Hàng 1 và cột 1 chỉ có một hướng đi tới.

Độ phức tạp $O(m \times n)$.

***Code mẫu:**

```
#include <bits/stdc++.h>
using namespace std;
int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);
   int m, n;
   cin >> m >> n;
   vector<vector<long long>> g(m + 1, vector<long long>(n + 1, 0));
   for (int i = 1; i <= m; i++)
      for (int j = 1; j <= n; j++)
         cin >> g[i][j];
   const long long INF = (1LL<<62);
   vector<vector<long long>> dp(m + 1, vector<long long>(n + 1, INF));
   dp[1][1] = g[1][1];
   for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= n; j++) {
         if (i == 1 && j == 1) continue;
```

```
            dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + g[i][j];
        }
    }
    cout << dp[m][n] << "\n";
    return 0;
}
```

***Cảm nhận:**

Bài min-path là dạng kinh điển, thường xuất hiện dưới nhiều bối cảnh (chi phí, năng lượng, thời gian). Với học sinh HSG, điều quan trọng không phải thuộc công thức, mà là nhìn ra bài toán "tối ưu đường đi" và đặt đúng trạng thái. Tôi thường yêu cầu học sinh làm thêm phiên bản nén 1D để rèn tư duy tối ưu bộ nhớ.

*** Một số lỗi học sinh hay mắc:**

- Quên xử lý hàng 1/cột 1 dẫn tới dùng INF sai cách.
- Dùng int thay vì long long (dù dữ liệu nhỏ, nhưng thói quen tốt).
- Nén 1D nhưng cập nhật sai thứ tự (cần j tăng dần để dp[j] là từ trên, dp[j-1] là từ trái).
- Nhầm bài này với Dijkstra (thực ra chỉ có 2 hướng nên DP là đủ).

BÀI TOÁN 5. Đường đi của bò Bessie (BESSIE.*)

Nguồn: https://usaco.org/index.php?cpid=1157&page=viewproblem2

Bò Bessie đang đi từ đồng cỏ yêu thích của mình về chuồng. Khu đồng cỏ được biểu diễn bằng một lưới vuông kích thước N × N (2 ≤ N ≤ 50). Ô đồng cỏ nằm ở góc trên bên trái (1,1) và chuồng nằm ở góc dưới bên phải (N,N). Để về nhà nhanh nhất, Bessie chỉ được phép di chuyển:

- Sang phải (R)
- Xuống dưới (D)

Trong một số ô của lưới có các kiện rơm (ký hiệu là 'H') mà Bessie không thể đi qua, do đó cô phải đi vòng tránh các ô này.

Do đang cảm thấy mệt, Bessie chỉ muốn đổi hướng di chuyển không quá K lần (1 ≤ K ≤ 3).

Hãy xác định số đường đi khác nhau mà Bessie có thể đi từ đồng cỏ đến chuồng. Hai đường đi được coi là khác nhau nếu tồn tại ít nhất một ô mà Bessie đi qua ở đường này nhưng không đi qua ở đường kia.

Dữ liệu vào

- Dòng 1: Số nguyên T (1 ≤ T ≤ 50) – số bộ test
- Với mỗi bộ test:
  + Dòng 1: Hai số nguyên N và K (2 ≤ N ≤ 50, 1 ≤ K ≤ 3)
  + N dòng tiếp theo, mỗi dòng là một chuỗi gồm N ký tự:
    * '.' : ô trống
    * 'H' : ô có kiện rơm (không được đi qua)
  Lưu ý: Ô (1,1) và ô (N,N) luôn là ô trống.

Dữ liệu ra

- Với mỗi bộ test, in ra một dòng là số đường đi hợp lệ.

Ví dụ

Input	Output
7 3 1 ... ... 3 2 ... ... 3 3 ... ... 3 3 ... .H. ... 3 2 .HH HHH HH. 3 3	2 4 6 2 0 0 6

.H. H.. ... 4 3 ...H .H.. .... H...	

***Hướng dẫn thuật toán:**

Đây là bài "đẹp" để chọn đội tuyển vì buộc học sinh xây dựng thêm trạng thái: không chỉ quan tâm đang ở ô nào, mà còn cần biết hướng vừa đi và đã đổi hướng bao nhiêu lần.

Ý tưởng:

Đặt dp[i][j][dir][t] = số cách đi tới ô (i,j) sao cho bước đi cuối cùng có hướng dir, và đã đổi hướng t lần.

- dir = 0 nếu bước cuối là sang phải (R)
- dir = 1 nếu bước cuối là đi xuống (D)

Quy tắc đổi hướng: nếu bước trước đó có hướng khác bước hiện tại thì t tăng 1.

Đặc biệt: bước đầu tiên (từ ô (1,1) đi R hoặc D) KHÔNG tính là đổi hướng.

Để xử lý bước đầu tiên gọn gàng, ta dùng thêm trạng thái dir=2 (NONE) tại ô xuất phát:

dp[0][0][2][0] = 1 (0-index)

Khi chuyển từ dir=NONE sang R/D thì số lần đổi hướng không tăng.

Chuyển trạng thái (0-index, ô trống):

1) Đi sang phải: từ (i, j-1) -> (i, j)

newdir = R

newt = t + (prevdir == D ? 1 : 0) (prevdir==NONE thì +0)

2) Đi xuống: từ (i-1, j) -> (i, j)

newdir = D

newt = t + (prevdir == R ? 1 : 0) (prevdir==NONE thì +0)

Chỉ giữ những trạng thái newt ≤ K.

Đáp án: sum_{dir in {R,D}} sum_{t=0..K} dp[n-1][n-1][dir][t].

Độ phức tạp: $O(T * N^2)$ $vì$ $K \leq 3, N \leq 50.$

***Code mẫu:**

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int T;
    cin >> T;
    while (T--) {
        int N, K;
        cin >> N >> K;
        vector<string> g(N);
        for (int i = 0; i < N; i++) cin >> g[i];
        // dir: 0=R, 1=D, 2=NONE (only used at start)
        long long dp[50][50][3][4] = {}; // N<=50, K<=3
        dp[0][0][2][0] = 1;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (g[i][j] == 'H') continue;
                for (int dir = 0; dir < 3; dir++) {
                    for (int t = 0; t <= K; t++) {
                        long long ways = dp[i][j][dir][t];
                        if (!ways) continue;
                        // Move Right
                        if (j + 1 < N && g[i][j + 1] != 'H') {
                            int nt = t;
                            if (dir == 1) nt++;         // D -> R : turn
                            // dir==2 (NONE) or dir==0 (R) : no new turn
                            if (nt <= K) dp[i][j + 1][0][nt] += ways;
                        }
                        // Move Down
                        if (i + 1 < N && g[i + 1][j] != 'H') {
                            int nt = t;
                            if (dir == 0) nt++;         // R -> D : turn
                            // dir==2 (NONE) or dir==1 (D) : no new turn
```

```
                    if (nt <= K) dp[i + 1][j][1][nt] += ways;
                }
            }
        }
    }
    long long ans = 0;
    for (int t = 0; t <= K; t++) {
        ans += dp[N - 1][N - 1][0][t];
        ans += dp[N - 1][N - 1][1][t];
    }
    cout << ans << "\n";
  }
  return 0;
}
```

***Cảm nhận:**

Đây là bài tôi rất thích dùng ở giai đoạn “chọn đội tuyển” vì K nhỏ (≤3) khiến nhiều em chủ quan, nhưng bài lại bắt buộc tư duy trạng thái đúng bản chất. Học sinh yếu thường đếm nhầm số lần đổi hướng hoặc xử lý sai bước đầu tiên. Học sinh khá sẽ làm được với dp 4 chiều, và học sinh rất khá sẽ tối ưu cài đặt gọn, không rối ở phần biên.

Về mặt sư phạm, bài này dạy một bài học quan trọng: khi thêm ràng buộc (giới hạn số lần đổi hướng), ta thêm đúng “thông tin cần nhớ” vào trạng thái DP. Không thêm thừa, không thiếu.

***Một số lỗi học sinh hay mắc:**

- Đếm sai số lần đổi hướng: nhiều em tính cả bước đầu tiên là 1 lần đổi hướng.
- Chỉ dùng dp[i][j][t] (quên lưu hướng) → không thể biết có đổi hướng hay không.
- Không xử lý riêng trạng thái xuất phát, dẫn tới phải ‘chữa cháy’ bằng nhiều if rối.
- Quên loại các ô 'H' trong cả trạng thái hiện tại và trạng thái chuyển tới.
- Tràn chỉ số khi dùng 1-index và quên dịch khi kiểm tra biên.

PHẦN III. BÀI TẬP TỰ LUYỆN

1) AtCoder ABC183 E – Quân hậu trên bàn cờ

https://atcoder.jp/contests/abc183/tasks/abc183_e

Cho một bàn cờ hình chữ nhật gồm H hàng và W cột. Mỗi ô trên bàn cờ là một trong hai loại:

- '.' : ô trống, có thể đi qua
- '#' : ô bị chặn, không thể đi qua

Bạn bắt đầu tại ô ở góc trên bên trái (1,1) và muốn di chuyển đến ô ở góc dưới bên phải (H,W).

Từ một ô bất kỳ, bạn có thể di chuyển:

- Sang phải bất kỳ số ô liên tiếp
- Xuống dưới bất kỳ số ô liên tiếp
- Chéo xuống phải bất kỳ số ô liên tiếp

với điều kiện các ô trên đường đi đều là ô trống ('.').

Hãy tính số cách khác nhau để đi từ (1,1) đến (H,W). Hai cách đi được coi là khác nhau nếu tập các ô đi qua là khác nhau.

Dữ liệu vào

- Dòng 1: Hai số nguyên H, W (1 ≤ H, W ≤ 2000)
- H dòng tiếp theo, mỗi dòng là một chuỗi độ dài W, chỉ gồm các ký tự '.' và '#'

Dữ liệu ra

- In ra số cách đi hợp lệ, lấy kết quả theo modulo 1 000 000 007.

Ví dụ

Input	Output
3 3 ... .#. ...	6

Ràng buộc

- Không được đi qua ô bị chặn '#'
- Chỉ được di chuyển theo 3 hướng đã nêu
- Kết quả có thể rất lớn, cần lấy modulo 1 000 000 007
- Thuật toán phải phù hợp với H, W ≤ 2000

***Code mẫu:**

```
#include <bits/stdc++.h>
using namespace std;
const long long MOD = 1000000007LL;
int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);
   int H, W;
   cin >> H >> W;
   vector<string> S(H);
   for (int i = 0; i < H; i++) cin >> S[i];
   vector<vector<long long>> dp(H, vector<long long>(W, 0));
   vector<vector<long long>> rowSum(H, vector<long long>(W, 0));
   vector<vector<long long>> colSum(H, vector<long long>(W, 0));
   vector<vector<long long>> diagSum(H, vector<long long>(W, 0));
   if (S[0][0] == '#') {
      cout << 0 << "\n";
      return 0;
   }
   dp[0][0] = 1;
   rowSum[0][0] = 1;
   colSum[0][0] = 1;
   diagSum[0][0] = 1;
   for (int i = 0; i < H; i++) {
      for (int j = 0; j < W; j++) {
         if (i == 0 && j == 0) continue;
         if (S[i][j] == '#') {
            dp[i][j] = 0;
            rowSum[i][j] = 0;
            colSum[i][j] = 0;
            diagSum[i][j] = 0;
            continue;
         }
         long long ways = 0;
         ways = (ways + (j > 0 ? rowSum[i][j - 1] : 0)) % MOD;
```

```
            ways = (ways + (i > 0 ? colSum[i - 1][j] : 0)) % MOD;
            ways = (ways + (i > 0 && j > 0 ? diagSum[i - 1][j - 1] : 0)) % MOD;
            dp[i][j] = ways;
            rowSum[i][j] = dp[i][j];
            if (j > 0) rowSum[i][j] = (rowSum[i][j] + rowSum[i][j - 1]) % MOD;
            colSum[i][j] = dp[i][j];
            if (i > 0) colSum[i][j] = (colSum[i][j] + colSum[i - 1][j]) % MOD;
            diagSum[i][j] = dp[i][j];
            if (i > 0 && j > 0) diagSum[i][j] = (diagSum[i][j] + diagSum[i - 1][j - 1]) % MOD;
        }
    }
    cout << dp[H - 1][W - 1] % MOD << "\n";
    return 0;
}
```

2) AtCoder DP Contest Y – Grid 2

Nguồn: https://atcoder.jp/contests/dp/tasks/dp_y/

Có một lưới gồm H hàng và W cột. Kí hiệu (i, j) là ô ở hàng i (tính từ trên xuống) và cột j (tính từ trái qua).

Trong lưới có N ô là tường (không thể đi qua), lần lượt là: (r1, c1), (r2, c2), …, (rN, cN).

Các ô còn lại đều là ô trống. Đảm bảo rằng (1,1) và (H,W) là ô trống.

Taro bắt đầu từ ô (1,1) và muốn đến ô (H,W) bằng cách lặp lại việc di chuyển sang phải hoặc xuống dưới tới một ô trống kề cạnh.

Hãy tính số đường đi khác nhau từ (1,1) đến (H,W), lấy kết quả theo modulo 10^9 + 7.

Giới hạn:

- Tất cả giá trị đầu vào là số nguyên
- 2 ≤ H, W ≤ 100000
- 1 ≤ N ≤ 3000
- 1 ≤ r_i ≤ H
- 1 ≤ c_i ≤ W
- Các ô (r_i, c_i) đôi một khác nhau
- (1,1) và (H,W) là ô trống

Dữ liệu vào:

- Dòng 1: H W N
- N dòng tiếp theo: r_i c_i (i = 1..N)

Dữ liệu ra

In ra số đường đi từ (1,1) đến (H,W) theo modulo 10^9 + 7.

Ví dụ

Input	Output
3 4 2 2 2 1 4	3

***Code mẫu:**

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <utility>
using namespace std;
const long long MOD = 1000000007LL;
long long modpow(long long a, long long e) {
   long long r = 1 % MOD;
   a %= MOD;
   while (e > 0) {
      if (e & 1) r = (r * a) % MOD;
      a = (a * a) % MOD;
      e >>= 1;
   }
   return r;
}
int main() {
   ios::sync_with_stdio(false);
   cin.tie(NULL);
   int H, W, N;
   cin >> H >> W >> N;
   vector<pair<int,int> > pts;
   pts.reserve(N + 1);
```

```
for (int i = 0; i < N; i++) {
   int r, c;
   cin >> r >> c;
   pts.push_back(make_pair(r, c));
}
pts.push_back(make_pair(H, W));
sort(pts.begin(), pts.end());
int MAX = H + W + 5;
vector<long long> fact(MAX), invfact(MAX);
fact[0] = 1;
for (int i = 1; i < MAX; i++) fact[i] = (fact[i - 1] * i) % MOD;
invfact[MAX - 1] = modpow(fact[MAX - 1], MOD - 2);
for (int i = MAX - 2; i >= 0; i--) {
   invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
}
// nCr modulo MOD
// nCr = fact[n] * invfact[r] * invfact[n-r]
struct Comb {
   vector<long long> *fact, *invfact;
   long long nCr(int n, int r) const {
      if (r < 0 || r > n) return 0LL;
      long long res = (*fact)[n];
      res = (res * (*invfact)[r]) % MOD;
      res = (res * (*invfact)[n - r]) % MOD;
      return res;
   }
} comb;
comb.fact = &fact;
comb.invfact = &invfact;
auto ways = [&](int r1, int c1, int r2, int c2) -> long long {
   int dr = r2 - r1;
   int dc = c2 - c1;
   if (dr < 0 || dc < 0) return 0LL;
   return comb.nCr(dr + dc, dr);
```

```
    };
    int M = pts.size();
    vector<long long> dp(M, 0);
    for (int i = 0; i < M; i++) {
        int ri = pts[i].first;
        int ci = pts[i].second;
        long long cur = ways(1, 1, ri, ci);
        for (int j = 0; j < i; j++) {
            int rj = pts[j].first;
            int cj = pts[j].second;
            if (rj <= ri && cj <= ci) {
                long long sub = (dp[j] * ways(rj, cj, ri, ci)) % MOD;
                cur -= sub;
                if (cur < 0) cur += MOD;
            }
        }
        dp[i] = cur % MOD;
    }
    cout << dp[M - 1] % MOD << "\n";
    return 0;
}
```

3) LeetCode 63 – Unique Paths II

Nguồn: https://leetcode.com/problems/unique-paths-ii/

Một robot đang đứng tại ô góc trên bên trái của một lưới hình chữ nhật kích thước m × n. Robot chỉ có thể di chuyển theo hai hướng:

- Sang phải
- Xuống dưới

Trong lưới có một số ô bị chặn (vật cản), robot không thể đi qua các ô này.

Hãy xác định số đường đi khác nhau để robot có thể đi từ ô bắt đầu (1,1) đến ô đích (m,n).

Hai đường đi được coi là khác nhau nếu tồn tại ít nhất một ô mà robot đi qua trong một đường nhưng không đi qua trong đường còn lại.

Dữ liệu vào

- Số nguyên m, n $(1 \leq m, n \leq 100)$

- Một ma trận obstacleGrid kích thước $m \times n$:

  + 0: ô trống (có thể đi qua)

  + 1: ô có vật cản (không thể đi qua)

  Đảm bảo ô (1,1) và (m,n) có thể là ô trống hoặc có vật cản.

Dữ liệu ra

- In ra số đường đi hợp lệ từ (1,1) đến (m,n).

Ví dụ

Input	Output
3 3 0 0 0 0 1 0 0 0 0	2

Yêu cầu & ràng buộc

- Robot chỉ được di chuyển sang phải hoặc xuống dưới
- Không được đi vào ô có vật cản (giá trị 1)
- Kết quả nằm trong phạm vi số nguyên 32-bit
- Thuật toán phải phù hợp với m, n ≤ 100

*Code mẫu:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
   ios::sync_with_stdio(false);    cin.tie(nullptr);
   int m, n;
   cin >> m >> n;
   vector<vector<int>> a(m, vector<int>(n));
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         cin >> a[i][j];
      }
   }
   // dp[j] = số cách đến ô (i, j) ở hàng hiện tại
   vector<long long> dp(n, 0);
   // ô bắt đầu
```

```
    dp[0] = (a[0][0] == 0) ? 1 : 0;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (a[i][j] == 1) {
                dp[j] = 0;              // bị chặn thì không có cách nào đến
            } else if (j > 0) {
                dp[j] += dp[j - 1];     // từ trên (dp[j]) + từ trái (dp[j-1])
            }
        }
    }
    cout << dp[n - 1] << "\n";
}
```

PHẦN IV. KẾT LUẬN

Quy hoạch động trên lưới (Grid DP) là một chuyên đề nền tảng nhưng giàu chiều sâu, có vai trò đặc biệt quan trọng trong việc xây dựng lộ trình bồi dưỡng học sinh giỏi Tin học từ mức cơ bản đến mức chọn đội tuyển. Thông qua hệ thống các bài toán được sắp xếp theo mức độ tăng dần, học sinh không chỉ được củng cố kỹ năng lập trình, mà quan trọng hơn là từng bước hình thành tư duy thuật toán có hệ thống. Với năm bài vận dụng trong chuyên đề, học sinh được rèn luyện một cách toàn diện các năng lực cốt lõi của Quy hoạch động, bao gồm:

(1) Xác định và diễn đạt chính xác ý nghĩa của trạng thái DP;

(2) Xử lý đúng các trường hợp biên và điều kiện hợp lệ trên lưới;

(3) Nhận ra sự thống nhất về tư duy giữa các bài toán đếm số cách và các bài toán tối ưu;

(4) Mở rộng mô hình DP bằng cách bổ sung trạng thái khi bài toán xuất hiện những ràng buộc mới.

Qua quá trình huấn luyện đội tuyển, tác giả nhận thấy rằng chất lượng tư duy của học sinh không thể đánh giá chỉ qua việc chương trình chạy đúng, mà cần được thể hiện rõ trong cách các em trình bày lời giải. Vì vậy, giáo viên nên thường xuyên yêu cầu học sinh nêu rõ: ý nghĩa của từng trạng thái dp, các nguồn chuyển trạng thái, cũng như lý do lựa chọn thứ tự duyệt. Đây không chỉ là yêu cầu về mặt trình bày, mà còn là tiêu chí quan trọng để đánh giá mức độ hiểu bản chất bài toán của học sinh.

Khi được khai thác đúng cách, Grid DP không chỉ giúp học sinh giải quyết một nhóm bài toán cụ thể, mà còn đóng vai trò như cầu nối tư duy giữa các dạng DP khác nhau, tạo nền tảng vững chắc để các em tiếp cận những bài toán thuật toán phức tạp hơn trong các kỳ thi học sinh giỏi ở cấp độ cao.

Trong quá trình biên soạn chuyên đề, do điều kiện thời gian và năng lực cá nhân còn hạn chế, không tránh khỏi những thiếu sót nhất định về nội dung cũng như cách trình bày. Tác giả rất mong nhận được những ý kiến đóng góp, trao đổi và phản biện mang tính xây dựng từ các thầy cô giáo, đồng nghiệp và những người quan tâm, nhằm tiếp tục hoàn thiện chuyên đề, nâng cao chất lượng giảng dạy và bồi dưỡng học sinh giỏi Tin học trong thời gian tới.