概率 DP

引入

概率 DP 用于解决概率问题与期望问题，建议先对 概率 & 期望 的内容有一定了解。一般情况下，解决概率问题需要顺序循环，而解决期望问题使用逆序循环，如果定义的状态转移方程存在后效性问题，还需要用到 高斯消元 来优化。概率 DP 也会结合其他知识进行考察，例如 状态压缩，树上进行 DP 转移等。

DP 求概率

这类题目采用顺推，也就是从初始状态推向结果。同一般的 DP 类似的，难点依然是对状态转移方程的刻画，只是这类题目经过了概率论知识的包装。

例题 Codeforces 148 D Bag of mice

题目大意：袋子里有 w 只白鼠和 b 只黑鼠，公主和龙轮流从袋子里抓老鼠。谁先抓到白色老鼠谁就赢，如果袋子里没有老鼠了并且没有谁抓到白色老鼠，那么算龙赢。公主每次抓一只老鼠，龙每次抓完一只老鼠之后会有一只老鼠跑出来。每次抓的老鼠和跑出来的老鼠都是随机的。公主先抓。问公主赢的概率。

过程

设 $f_{i,j}$ 为轮到公主时袋子里有 i 只白鼠，j 只黑鼠，公主赢的概率。初始化边界，$f_{0,j}=0$ 因为没有白鼠了算龙赢，$f_{i,0}=1$ 因为抓一只就是白鼠，公主赢。 考虑 $f_{i,j}$ 的转移：

- 公主抓到一只白鼠，公主赢了。概率为 $\frac{i}{i+j}$；
- 公主抓到一只黑鼠，龙抓到一只白鼠，龙赢了。概率为 $\frac{j}{i+j}\cdot\frac{i}{i+j-1}$；
- 公主抓到一只黑鼠，龙抓到一只黑鼠，跑出来一只黑鼠，转移到 $f_{i,j-3}$。概率为 $\frac{j}{i+j}\cdot\frac{j-1}{i+j-1}\cdot\frac{j-2}{i+j-2}$；
- 公主抓到一只黑鼠，龙抓到一只黑鼠，跑出来一只白鼠，转移到 $f_{i-1,j-2}$。概率为 $\frac{j}{i+j}\cdot\frac{j-1}{i+j-1}\cdot\frac{i}{i+j-2}$；

考虑公主赢的概率，第二种情况不参与计算。并且要保证后两种情况合法，所以还要判断 i,j 的大小，满足第三种情况至少要有 3 只黑鼠，满足第四种情况要有 1 只白鼠和 2 只黑鼠。

实现

参考实现

```cpp
#include <cstring>
#include <iomanip>
#include <iostream>
using namespace std;

using ll = long long;
int w, b;
double dp[1010][1010];

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> w >> b;
  memset(dp, 0, sizeof(dp));
  for (int i = 1; i <= w; i++) dp[i][0] = 1;  // 初始化
  for (int i = 1; i <= b; i++) dp[0][i] = 0;
  for (int i = 1; i <= w; i++) {
    for (int j = 1; j <= b; j++) {  // 以下为题面概率转移
      dp[i][j] += (double)i / (i + j);
      if (j >= 3) {
        dp[i][j] += (double)j / (i + j) * (j - 1) / (i + j - 1) *
(j - 2) /
                    (i + j - 2) * dp[i][j - 3];
      }
      if (i >= 1 && j >= 2) {
        dp[i][j] += (double)j / (i + j) * (j - 1) / (i + j - 1) *
i /
                    (i + j - 2) * dp[i - 1][j - 2];
      }
    }
  }
  cout << fixed << setprecision(9) << dp[w][b] << '\n';
  return 0;
}
```

习题

- CodeForces 148 D Bag of mice
- POJ3071 Football
- CodeForces 768 D Jon and Orbs

DP 求期望

例一

例题 POJ2096 Collecting Bugs

题目大意：一个软件有 s 个子系统，会产生 n 种 bug。某人一天发现一个 bug，这个 bug 属于某种 bug 分类，也属于某个子系统。每个 bug 属于某个子系统的概率是 $\frac{1}{s}$，属于某种 bug 分类的概率是 $\frac{1}{n}$。求发现 n 种 bug，且 s 个子系统都找到 bug 的期望天数。

过程

令 $f_{i,j}$ 为已经找到 i 种 bug 分类，j 个子系统的 bug，达到目标状态的期望天数。这里的目标状态是找到 n 种 bug 分类，s 个子系统的 bug。那么就有 $f_{n,s}=0$，因为已经达到了目标状态，不需要用更多的天数去发现 bug 了，于是就以目标状态为起点开始递推，答案是 $f_{0,0}$。

考虑 $f_{i,j}$ 的状态转移：

- $f_{i,j}$，发现一个 bug 属于已经发现的 i 种 bug 分类，j 个子系统，概率为 $p_1=\frac{i}{n}\cdot\frac{j}{s}$
- $f_{i,j+1}$，发现一个 bug 属于已经发现的 i 种 bug 分类，不属于已经发现的子系统，概率为 $p_2=\frac{i}{n}\cdot(1-\frac{j}{s})$
- $f_{i+1,j}$，发现一个 bug 不属于已经发现 bug 分类，属于 j 个子系统，概率为 $p_3=(1-\frac{i}{n})\cdot\frac{j}{s}$
- $f_{i+1,j+1}$，发现一个 bug 不属于已经发现 bug 分类，不属于已经发现的子系统，概率为 $p_4=(1-\frac{i}{n})\cdot(1-\frac{j}{s})$

再根据期望的线性性质，就可以得到状态转移方程：

$$\begin{aligned}f_{i,j}&=p_1\cdot f_{i,j}+p_2\cdot f_{i,j+1}+p_3\cdot f_{i+1,j}+p_4\cdot f_{i+1,j+1}+1\\&=\frac{p_2\cdot f_{i,j+1}+p_3\cdot f_{i+1,j}+p_4\cdot f_{i+1,j+1}+1}{1-p_1}\end{aligned}$$

实现

参考实现

```cpp
#include <iomanip>
#include <iostream>
using namespace std;
int n, s;
double dp[1010][1010];

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n >> s;
  dp[n][s] = 0;
  for (int i = n; i >= 0; i--) {
    for (int j = s; j >= 0; j--) {
      if (i == n && s == j) continue;
      dp[i][j] = (dp[i][j + 1] * i * (s - j) + dp[i + 1][j] * (n
- i) * j +
                  dp[i + 1][j + 1] * (n - i) * (s - j) + n * s) /
                 (n * s - i * j);  // 概率转移
    }
  }
  cout << fixed << setprecision(4) << dp[0][0] << '\n';
  return 0;
}
```

例二

例题 「NOIP2016」换教室

题目大意：牛牛要上 n 个时间段的课，第 i 个时间段在 c_i 号教室，可以申请换到 d_i 号教室，申请成功的概率为 p_i，至多可以申请 m 节课进行交换。第 i 个时间段的课上完后要走到第 $i+1$ 个时间段的教室，给出一张图 v 个教室 e 条路，移动会消耗体力，申请哪几门课程可以使他因在教室间移动耗费的体力值的总和的期望值最小，也就是求出最小的期望路程和。

过程

对于这个无向连通图，先用 Floyd 求出最短路，为后续的状态转移带来便利。以移动一步为一个阶段（从第 i 个时间段到达第 $i+1$ 个时间段就是移动了一步），那么每一步就有 p_i 的概率到 d_i，不过在所有的 d_i 中只能选 m 个，有 $1-p_i$ 的概率到 c_i，求出在 n 个阶段走完后的最小期望路程和。 定义 $f_{i,j,0/1}$ 为在第 i 个时间段，连同这一个时间段已经用了 j 次换教室的机会，在这个时间段换（1）或者不换（0）教室的最小期望路程和，那么答案就是 $\min\{f_{n,i,0}, f_{n,i,1}\}, i \in [0, m]$。注意边界 $f_{1,0,0} = f_{1,1,1} = 0$。

考虑 $f_{i,j,0/1}$ 的状态转移：

- 如果这一阶段不换，即 $f_{i,j,0}$。可能是由上一次不换的状态转移来的，那么就是 $f_{i-1,j,0}+w_{c_{i-1},c_i}$，也有可能是由上一次交换的状态转移来的，这里结合条件概率和全概率的知识分析可以得到 $f_{i-1,j,1}+w_{d_{i-1},c_i}\cdot p_{i-1}+w_{c_{i-1},c_i}\cdot(1-p_{i-1})$，状态转移方程就有

$$f_{i,j,0}=min(f_{i-1,j,0}+w_{c_{i-1},c_i},f_{i-1,j,1}+w_{d_{i-1},c_i}\cdot p_{i-1}+w_{c_{i-1},c_i}\cdot(1-p_{i-1}))$$

- 如果这一阶段交换，即 $f_{i,j,1}$。类似地，可能由上一次不换的状态转移来，也可能由上一次交换的状态转移来。那么遇到不换的就乘上 $(1-p_i)$，遇到交换的就乘上 p_i，将所有会出现的情况都枚举一遍出进行计算就好了。这里不再赘述各种转移情况，相信通过上一种阶段例子，这里的状态转移应该能够很容易写出来。

实现

参考实现

```cpp
#include <algorithm>
#include <iomanip>
#include <iostream>

using namespace std;

constexpr int MAXN = 2010;
int n, m, v, e;
int f[MAXN][MAXN], c[MAXN], d[MAXN];
double dp[MAXN][MAXN][2], p[MAXN];

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n >> m >> v >> e;
  for (int i = 1; i <= n; i++) cin >> c[i];
  for (int i = 1; i <= n; i++) cin >> d[i];
  for (int i = 1; i <= n; i++) cin >> p[i];
  for (int i = 1; i <= v; i++)
    for (int j = 1; j < i; j++) f[i][j] = f[j][i] = 1e9;

  int u, V, w;
  for (int i = 1; i <= e; i++) {
    cin >> u >> V >> w;
    f[u][V] = f[V][u] = min(w, f[u][V]);
  }

  for (int k = 1; k <= v; k++)
    for (int i = 1; i <= v; i++)  // 前面的，按照前面的题解进行一个状态转移
      for (int j = 1; j < i; j++)
        if (f[i][k] + f[k][j] < f[i][j]) f[i][j] = f[j][i] = f[i][k] + f[k][j];

  for (int i = 1; i <= n; i++)
    for (int j = 0; j <= m; j++) dp[i][j][0] = dp[i][j][1] = 1e9;

  dp[1][0][0] = dp[1][1][1] = 0;
  for (int i = 2; i <= n; i++)  // 有后效性方程
    for (int j = 0; j <= min(i, m); j++) {
      dp[i][j][0] = min(dp[i - 1][j][0] + f[c[i - 1]][c[i]],
                        dp[i - 1][j][1] + f[c[i - 1]][c[i]] * (1 - p[i - 1]) +
                            f[d[i - 1]][c[i]] * p[i - 1]);
      if (j != 0) {
        dp[i][j][1] = min(dp[i - 1][j - 1][0] + f[c[i - 1]][d[i]] * p[i] +
                              f[c[i - 1]][c[i]] * (1 - p[i]),
                          dp[i - 1][j - 1][1] +
                              f[c[i - 1]][c[i]] * (1 - p[i - 1])
```

```
* (1 - p[i]) +
                                f[c[i - 1]][d[i]] * (1 - p[i - 1])
* p[i] +
                                f[d[i - 1]][c[i]] * (1 - p[i]) *
p[i - 1] +
                                f[d[i - 1]][d[i]] * p[i - 1] *
p[i]);
      }
    }

  double ans = 1e9;
  for (int i = 0; i <= m; i++) ans = min(dp[n][i][0], min(dp[n]
[i][1], ans));
  cout << fixed << setprecision(2) << ans;

  return 0;
}
```

比较这两个问题可以发现，DP 求期望题目在对具体是求一个值或是最优化问题上会对方程得到转移方式有一些影响，但无论是 DP 求概率还是 DP 求期望，总是离不开概率知识和列出、化简计算公式的步骤，在写状态转移方程时需要思考的细节也类似。

习题

- POJ2096 Collecting Bugs
- HDU3853 LOOPS
- HDU4035 Maze
- 「NOIP2016」换教室
- 「SCOI2008」奖励关

有后效性 DP

CodeForces 24 D Broken robot

题目大意：给出一个 $n \times m$ 的矩阵区域，一个机器人初始在第 x 行第 y 列，每一步机器人会等概率地选择停在原地，左移一步，右移一步，下移一步，如果机器人在边界则不会往区域外移动，问机器人到达最后一行的期望步数。

过程

在 $m = 1$ 时每次有 $\frac{1}{2}$ 的概率不动，有 $\frac{1}{2}$ 的概率向下移动一格，答案为 $2 \cdot (n - x)$。 设 $f_{i,j}$ 为机器人机器人从第 i 行第 j 列出发到达第 n 行的期望步数，最终状态为 $f_{n,j} = 0$。 由于机器人会等

概率地选择停在原地，左移一步，右移一步，下移一步，考虑 $f_{i,j}$ 的状态转移：

- $f_{i,1} = \frac{1}{3} \cdot (f_{i+1,1} + f_{i,2} + f_{i,1}) + 1$
- $f_{i,j} = \frac{1}{4} \cdot (f_{i,j} + f_{i,j-1} + f_{i,j+1} + f_{i+1,j}) + 1$
- $f_{i,m} = \frac{1}{3} \cdot (f_{i,m} + f_{i,m-1} + f_{i+1,m}) + 1$

在行之间由于只能向下移动，是满足无后效性的。在列之间可以左右移动，在移动过程中可能产生环，不满足无后效性。 将方程变换后可以得到：

- $2f_{i,1} - f_{i,2} = 3 + f_{i+1,1}$
- $3f_{i,j} - f_{i,j-1} - f_{i,j+1} = 4 + f_{i+1,j}$
- $2f_{i,m} - f_{i,m-1} = 3 + f_{i+1,m}$

由于是逆序的递推，所以每一个 $f_{i+1,j}$ 是已知的。 由于有 m 列，所以右边相当于是一个 m 行的列向量，那么左边就是 m 行 m 列的矩阵。使用增广矩阵，就变成了 m 行 m+1 列的矩阵，然后进行 高斯消元 即可解出答案。

实现

参考实现

```cpp
#include <cstdio>
#include <cstring>
using namespace std;

constexpr int MAXN = 1e3 + 10;

double a[MAXN][MAXN], f[MAXN];
int n, m;

void solve(int x) {
  memset(a, 0, sizeof a);
  for (int i = 1; i <= m; i++) {
    if (i == 1) {
      a[i][i] = 2;
      a[i][i + 1] = -1;
      a[i][m + 1] = 3 + f[i];
      continue;
    } else if (i == m) {
      a[i][i] = 2;
      a[i][i - 1] = -1;
      a[i][m + 1] = 3 + f[i];
      continue;
    }
    a[i][i] = 3;
    a[i][i + 1] = -1;
    a[i][i - 1] = -1;
    a[i][m + 1] = 4 + f[i];
  }

  for (int i = 1; i < m; i++) {
    double p = a[i + 1][i] / a[i][i];
    a[i + 1][i] = 0;
    a[i + 1][i + 1] -= a[i][i + 1] * p;
    a[i + 1][m + 1] -= a[i][m + 1] * p;
  }

  f[m] = a[m][m + 1] / a[m][m];
  for (int i = m - 1; i >= 1; i--)
    f[i] = (a[i][m + 1] - f[i + 1] * a[i][i + 1]) / a[i][i];
}

int main() {
  scanf("%d %d", &n, &m);
  int st, ed;
  scanf("%d %d", &st, &ed);
  if (m == 1) {
    printf("%.10f\n", 2.0 * (n - st));
    return 0;
  }
```

```
50      for (int i = n - 1; i >= st; i--) {
51        solve(i);
52      }
53      printf("%.10f\n", f[ed]);
54      return 0;
55    }
```

习题

- CodeForce 24 D Broken robot
- HDU 4418 Time Travel
- 「HNOI2013」游走

参考文献

kuangbin 概率 DP 总结
