状态压缩dp-初期2022 10---11

chenjiuri_zhuangyadpbasic

$first\_provlem$

[Traveling by Stagecoach]

20mins

```
和旅行商人问题之间有什么差别？
1. 不一定要遍历完所有的城市，有固定的城市终点。
2. 引入关于车票的相关属性，要关注车票的选择。

城市数量
m达到30之多，无法用一个数组来表达当前去了哪一些城市。
但是事实上只需要使用到达n个城市即可.

定义f[s][j];这里表示，表示当前在j个城市，票的使用情况为s，去到终点城市的情况的最小时间。
初始化问题： f[0...2^n][end]=0;
终点--》相邻的点。
怎么进行状态转移？
```

solve

```
f[s][j]，此时到达j城市，此时车票是剩下集合为s的最短时间。
这里和我的状态设计相反。

s从大到小枚举，可以保证，当前枚举的状态的迁移问题已经解决。
这里可以之间的状态关系可以类比一个有向图。问题很简单的转化成在dag上的dp问题。
这样不断地迁移问题，最终计算出问题的解。
```

code

```cpp

#include <iostream>
#include <algorithm>
#include <string.h>
#include <iomanip>
using namespace std;

bool MAIN();
int main()
{
    // ios::sync_with_stdio(false);
    // cin.tie(nullptr), cout.tie(nullptr);
```

```
    while (MAIN())
        ;
    return 0;
}
typedef long long ll;
const int maxn = 11;
const double inf = 10010;
//------code-----٩(ˊωˋ*)و ------靓仔代码-----٩(ˊωˋ*)و ------------

int d[40][40]; //邻接矩阵来储存图。
double f[1 << maxn][40];
double ans;
int c[maxn]; //表示每一车票具体地情况。

bool MAIN()
{
    int n, m, p, a, b;
    cin >> n >> m >> p >> a >> b;
    if (n == 0 && m == 0 && p == 0 && a == 0 && b == 0)
        return false;
    memset(d, -1, sizeof(d));
    ans = inf;
    for (int i = 0; i < 1 << n; i++)
        // fill(f[i], f[i] + m, inf);
        for (int j = 0; j <= m; j++)
            f[i][j] = inf;
    for (int i = 0; i < n; i++)
        cin >> c[i];
    for (int i = 1; i <= p; i++)
    {
        int x, y, t;
        cin >> x >> y >> t;
        d[x - 1][y - 1] = d[y - 1][x - 1] = t;
    }
    //进行初始化;
    f[(1 << n) - 1][a - 1] = 0;                //为了更加好操作，表述上稍微转化。
    for (int s = (1 << n) - 1; s >= 0; s--) //当前枚举一个车票地使用情况。
    {
        ans = min(ans, f[s][b - 1]);
        for (int u = 0; u < m; u++)      //枚举当前所在点。
            for (int t = 0; t < n; t++) //枚举相关地转移点。
                if (s >> t & 1) //当前票如果没有被使用
                    for (int v = 0; v < m; v++)
                        if (d[u][v] >= 0) //当前两条路之间有路径，。//向下转移
方程
                            f[s & ~(1 << t)][v] = min(f[s & ~(1 << t)]
[v], f[s][u](double)d[u][v] / c[t]);
    }
    // for (int i = (1 << n) - 1; i >= 0; i--)
```

```
    //      ans = min(ans, f[i][b - 1]);
    if (ans == inf)
        cout << "Impossible" << '\n';
    else
        cout << fixed << setprecision(3) << ans << '\n';
    return true;
}
```

生长思考：

- 发散
  - 记忆化搜索应该怎么么实现?
- $debug$的惨痛教训。
  - 在这里为了方便，一般将集合的标记，从0开始标记。
  - 由于没有统一贯穿这一个原则。所以导致了图的构建的时候出了问题。

$the-second-problem$

吃奶酪

20mins

- 定义$f_{s,now}$表示当前老鼠已经遍历了状态s，当前在now点。他经历的最短路程。
- 向下寻找向下更新状态即可。这里可以类比一个有向完全图的dp.

```
更新顺序为：
如果是dfs会一直向下搜索。
不适合这样的递推思路。
```

主要问题：

- 不会设计状态转移方程。
  - 前面两个问题的状态转移方程设计。还是要从之前的经验中回想得到一些启发。
- 为什么不利用相关的图论几个关于路程的经典算法?

---

solve

```
直接类比上面个的旅行商问题：
f[s][now].当前在now点，经历的集合为s的最小距离。
不引入原点为一个点。
最小子规模的问题为只经历一个奶酪，当前点在唯一个点上的最小距离。
从小到大枚举，可以确保更小的集合已经被枚举完全。
和旅行商问题，相差不大；
```

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

```
void MAIN();
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    MAIN();
}
typedef long long ll;
const int maxn = 17;
//------code-----٩(ˊωˋ*)و ------靓仔代码-----٩(ˊωˋ*)و ----talk is cheap ,
show me the code--------
double x[maxn], y[maxn];
double f[1 << maxn][maxn];
double inf = 1e9;
double d[maxn][maxn];
double get(int i, int j)
{
    return sqrt((x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j]) * (y[i] -
y[j]));
}

void MAIN()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> x[i] >> y[i];
    // for (int i = 1 << (n + 1) - 1; i >= 0; i--)
    //     fill(f[i], f[i] + n + 1, inf); //
    memset(f, 127, sizeof(f));   //这样可以给浮点数赋值无穷大
    for (int i = 0; i <= n; i++) //距离初始化
        for (int j = 0; j <= n; j++)
            d[i][j] = get(i, j); //地址进行初始化。
    int flag = (1 << n) - 1;
    for (int i = 0; i < n; i++)      //对压缩方面的技巧不够，定义不娴熟。如果是
定义1为没去。
        f[1 << i][i] = d[i + 1][0]; //初始化就是它们与原点的距离。
    for (int s = 1; s <= flag; s++) //如果当前位置上是1说明没有选，如果是0说明
已经选择了。
        for (int u = 0; u < n; u++) //枚举当前的所在点。
            if (s >> u & 1)          //如果对前枚举的点并没有经过，说明讨论迁移
没有意义。此时当前位置上为1
                for (int v = 0; v < n; v++)
                    if (u != v && (s >> v & 1))
               //当前位置上必须是0;
                        f[s][u] = min(f[s][u], f[s & ~(1 << u)][v] + d[u
+ 1][v + 1]); //顺序上出错了。
```

```
                //思考怎么把当前位置变成0。
                                                    //下面处理的问题应该是-+-+7
    double ans = inf;
    for (int i = 0; i < n; i++)
        ans = min(ans, f[(1 << n) - 1][i]);
    cout << fixed << setprecision(2) << ans << '\n';
}
```

---

the_third_problem

```
给定一个网格，网格上面有一些黑砖头。
现在要用一些1*2的砖头铺在上面。
注意黑色的砖头不可以被遮盖。
现在问，铺满有多少种方式。
答案对mod取%
```

思路：

- 第一个问题，万物皆可以暴力。优化基于暴力。
  - 这里的暴力思路，有顺序的进行一个选择枚举。从右上角开始选择策略。每一种策略产生一些影响。我们用一个专门的数组，来记录一个这个图上，哪一个砖头被铺了。然后带着一整个数组向下递归就行了。
  - 最终的复杂度是：$O(2^{nm} \times n \times m)$。
- 在上述进行优化
  - 状态压缩思想。
    - 若干种状态是就是一张图，大的离谱。不可能用一个二进制数字表达。
    - 发现对于向下迁移的过程中，上述行都是被贴满的，下面的行和原来不变，而且和状态迁移相关的（影响迁移状态，策略影响的只是下一行）。
    - 从上面这一个发现，可以产生什么启发?
      - 关于滚动数组。滚动数组的思想是
        - 关注迁移顺序，划分阶段，关注没有变化的。
    - $f[j][s]$现在第搞j层，上面所有层已经填满，现在层上的情况是s.

```
#include <bits/stdc++.h>
using namespace std;
const int MAX_N = 30;
int dp[2][1 << MAX_N];
int n, m;
int M;
bool color[MAX_N][MAX_N];

void solve()
```

```
{
    int *crt = dp[0], *next = dp[1];
    crt[0] = 1;
    for (int i = n - 1; i >= 0; i--)
    {
        for (int j = m - 1; j >= 0; j--)
        {
            for (int used = 0; used < 1 << m; used++)
            {
                if ((used >> j & 1) || color[i][j])
                {
                    next[used] = crt[used & ~(1 << j)];
                }
                else
                {
                    int res = 0;
                    if (j + 1 < m && !(used >> (j + 1) & 1) && !color[i]
[j + 1])
                    { //横着放。
                        res += crt[used | 1 << j];
                    }
                    if (i + 1 < n && !color[i + 1][j])
                    {
                        res += crt[used | 1 << j];
                    }
                    next[used] = res % M;
                }
            }
            swap(next, crt);
        }
    }
    printf("%d\n", crt[0]);
}
```

下一个重要问题：认识代码：

- 关于动态规划状态迁移的问题：
    - 认识$dp[2][s]$
        - 滚动数组。
        - next,curent.
            - crt[s]:下一个状态（就是下一个子问题系列）中选择情况为s的状态的解填满下续格子的方法的方法个数。
            - next[s],下一个问题中，状态为s的解的个数。
    - 理解枚举s的顺序。
        - 先枚举当前解决问题的状态，然后可以通过位运算快速的定位到相关的子问题。
        - 这里是改写最原始的记忆化搜索，因此这里采取了方程逆推的方式。

- 理解状态转移方程。
- 理解初始化。
  - 初始化方式$f_{0,0} = 1$
  - 理解该初始化的可行性。
- 理解最终的结果为$next$【0】

$the\_fourth\_problem$

Mondriaan's Dream

```
用1*2的砖头铺满格子。
h,w<=11;
```

20mins

模仿上面的思路:

- 定义状态$f[2][1 << MAX_N]$表示当前的状态情况。
- 滚动数组实现

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

void MAIN();
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    MAIN();
}
typedef long long ll;
const int maxn = 2e5 + 10;
//------code-----٩(ˊωˋ*)و ------靓仔代码-----٩(ˊωˋ*)و ----talk is cheap ,
show me the code--------

ll f[2][1 << 12];
ll *nxt, *crt;

void MAIN()
{
    int n, m;
    while (cin >> n >> m)
    {
        if (n == 0 && n == 0)
            break;
        nxt = f[1];
        crt = f[0];
```

```
        memset(f, 0, sizeof f);
        crt[0] = 1;
        for (int i = n - 1; i >= 0; i--)
            for (int j = m - 1; j >= 0; j--)
            {
                for (int s = 0; s < 1 << m; s++)
                {
                    if (s >> j & 1) //当前这个位置上已经放有了。那么怎么搞？
                        nxt[s] = crt[s & ~(1 << j)];
                    else
                    {
                        ll res = 0;                              //往两边鸡
                        if (j + 1 < m && !(s >> (j + 1) & 1)) //横着放
                            res += crt[s | 1 << (j + 1)];
                        if (i + 1 < n)
                            res += crt[s | 1 << j];
                        nxt[s] = res;
                    }
                }
                swap(nxt, crt);
            }
        cout << crt[0] << '\n';
    }
}
```

另外的一个yls的思路；

- 逐步的一列一列的放，优先考虑打横放。
- 子问题定义为，放置某一列时，前面几列已经放满，放当前列的情况

---

the_sixth_problem

E - Corn Fields

```
简介，在草坪上种东西，
要求两两之间不相邻，有一些格子上不能种东西。
求出符合条件的所有方案：
```

20mins

- 第一种思路
    - $f_{i,j}$:当前在第i行的放置，放置方式情况为j的情况之下的所有合法方案。
    - 那么直接迁移$f_{i-1,s}$的状态。首先明确迁移的前提：两两不相邻。某个方格之上不可以种东西。
    - 复杂度估算为$2^m * 2^m * n * m$;
- 向上面fourth的问题一样
    - 逆推的角度来看：

    - 子问题定义中另外一个关注的角度：在放置第(i, j)个格子的，当前（i，0..j-1）和（i,j.....m-1）的情况下之后合理方案的后面的格子中合理的放法。
    - 两种转移，放或者不放。另外也要判断两种方向的可行性
  - 正推的角度上看：
  - 复杂度应该为$O(n \times m \times 2^m)$快得多。
- 思考初始化方式
  - 初始化的方式，只要保证最小规模的子问题状态得到满足即可。
    - 第一个方向，定义之内的状态。就是该问题真正有意义的一个状态。
    - 第二个方向，定义之外的状态，通过该状态加入迁移，可以初始化其它状态。
- 困惑，没有进展的本质是什么？就是没有很好的把握问题的解空间。