SP703 SERVICE - Mobile Service 移动服务

这是题目链接

Stage one: 人畜无害的DFS

- 设$cost[p][q]$为 p 移动到位置 q 需要花费的价钱
- i为正在接受dfs操作的第i个请求
- a b c 分别为 3 个流动员工的位置

  以 未来式 开始 dfs，得出第i个诉求的未来最优解

转移方程:

$$ f(i,a,b,c)=min\left{ \begin{aligned} cost[a][i]+f(i+1,i,b,c) \ cost[b][i]+f(i+1,a,i,c)\ cost[c][i]+f(i+1,a,b,i) \end{aligned} \right. $$ 实现到代码中为：

```
int dfs(int i, int a, int b, int c)
{
 int costa =  dfs(i+1, x[i], b, c) + cost[a][x[i]];
 int costb =  dfs(i+1, a, x[i], c) + cost[b][x[i]];
 int costc =  dfs(i+1, a, b, x[i]) + cost[c][x[i]];
 int ans = min(costa, min(costb, costc));
 return ans;
}
```

边界考虑：

1. 当考虑的第i个请求时i大于请求位置的总数n，则不予讨论
2. 当第i个请求时请求位置刚好有一位员工，则需要的花费为0,直接开始处理下一个请求

故获得以下边界代码：

```
if (i > n) return 0;
  if (x[i] == a or x[i] == b or x[i]==c)
    return dfsf(i+1, a, b, c);
```

综合下来，dfs 函数为：

```
int dfs(int i, int a, int b, int c)
{
  if (i > n) return 0;
  if (x[i] == a or x[i] == b or x[i]==c)
    return dfs(i+1, a, b, c);
```

```
    int costa =  dfs(i+1, x[i], b, c) + cost[a][x[i]];
    int costb =  dfs(i+1, a, x[i], c) + cost[b][x[i]];
    int costc =  dfs(i+1, a, b, x[i]) + cost[c][x[i]];
    int ans = min(costa, min(costb, costc));
    return ans;
}
```

我们就快乐地完成了dfs函数的编写！接着我们便可以在主函数中使用它来完成stage one力！

```
#include<iostream>
#include<algorithm>
using namespace std;
int L; //位置总和
int n; //共有n个请求
int cost[300][300]; //花费
int x[2000]; //请求

int dfs(int i, int a, int b, int c)
{
    if (i > n) return 0;
    if (x[i] == a or x[i] == b or x[i]==c)
      return dfs(i+1, a, b, c);
    int costa =  dfs(i+1, x[i], b, c) + cost[a][x[i]];
    int costb =  dfs(i+1, a, x[i], c) + cost[b][x[i]];
    int costc =  dfs(i+1, a, b, x[i]) + cost[c][x[i]];
    int ans = min(costa, min(costb, costc));
    return ans;
}

int main()
{
    int N;
    cin >> N;
    while (N-->0)
    {
      cin>>L>>n;
      for (int i = 1; i <= L; ++i)
        for (int j = 1; j <= L; ++j)
          cin >> cost[i][j];
      for (int i = 1; i <= n; ++i)
        cin >> x[i];
      cout << dfs(1, 1, 2, 3)<<endl;
    }
}
```

就这样，stage one 就快乐的完成力！

你以为这就结束了？那你可就真是...

> "Too $young$ too $simple$, $sometimes$ $naive$"

这个时候提交答案应该只有4个点能够AC, 其余的都是会TLE的

这个时候我们就进入了stage two...

Stage two: 优化

顾名思义 我们需要将目前$O(NL^3)$复杂度的dfs算法进行优化，使其能够避免TLE

> 优化非常类似于矿工配餐，点这里看上次的笔记

观察现有的dfs函数，可以发现变量c在大多数情况下显的垄余，所以我们不妨...

- 将第$x[i-1]$视作c，从而在函数中只需要3个量（i,a，b）

且不难看出dfs函数为不会影响到全局变量的纯函数$pure$ $function$, 所以就可以...

- 祭出大杀器 *记忆化搜索* 来处理复杂度

优化后（理论上）复杂度降为$O(NL^2)$

```
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;
int L; //位置总和
int n; //共有n个请求
int cost[300][300]; //花费
int x[2000]; //请求
int set [1001][201][201];//记忆化数组

int dfs(int i, int a, int b)
{
  if (i > n) return 0;
  if (a > b) return dfs(i, b, a);
  if (set[i][a][b]!=0) return set[i][a][b];
  if (x[i]==x[i-1])
    return dfs(i+1, a, b);
  if (x[i]==a)
    return dfs(i+1,  x[i-1],b );
  if (x[i]==b)
    return dfs(i+1, a, x[i-1]);
  int costa =  dfs(i+1, b, x[i-1]) + cost[a][x[i]];
  int costb =  dfs(i+1, a, x[i-1]) + cost[b][x[i]];
  int costi =  dfs(i+1, a, b) + cost[x[i-1]][x[i]];
  set[i][a][b] = min(costa, min(costb, costi));
  return set[i][a][b];
}

int main()
{
  int N;
  cin >> N;
```

```
  while (N-->0)
  {
    cin>>L>>n;
    for (int i = 1; i <= L; ++i)
      for (int j = 1; j <= L; ++j)
        cin >> cost[i][j];
    for (int i = 1; i <= n; ++i)
      cin >> x[i];
    x[0]=3;//初始化x[i-1]
    memset(set,0,sizeof(set));
    cout << dfs(1, 1, 2)<<endl;
  }
}
```

我们还可以通过滚动数组实现在祝函数中迭代达成dfs:使dfs函数中第$x[i]$个请求等价于$x[i% 2]$个请求,第$x[i+1]$个请求等价于$x[(i+1)% 2]$个请求

```
#include<iostream>
#include<algorithm>
using namespace std;
int L, n;
int cost[500][500];
int x[3001];
int f[2][501][501];
int main()
{
  cin >> L >> n;
  for (int i = 1; i <= L; ++i)
    for (int j = 1; j <= L; ++j)
      cin >> cost[i][j];
  for (int i = 1; i <= n; ++i)
   cin >> x[i];

  x[0] = 3;

  for (int i = n; i > 0; --i) {
      int c = x[i-1];
      for (int a = 1; a <= L; ++a)
        for (int b = 1; b <= L; ++b) {
          if (a == b or b == c or c == a) {
            f[i%2][a][b] = 1 << 29;
          }
          else {
            int r1 =  f[(i+1)%2][b][c] + cost[a][x[i]];
            int r2 =  f[(i+1)%2][a][c] + cost[b][x[i]];
            int r3 =  f[(i+1)%2][a][b] + cost[c][x[i]];
            f[i%2][a][b] = min(r1, min(r2, r3));
          }
        }
  }
```

```
    cout << f[1][1][2] << "\n";     //每组数据处理完后要换行！
  }
```

这题便做完了

~~dfs 十分钟，优化2小时~~

若是喜欢整理的笔记的话在github上点个star再走啊，谢谢啦～