# 目录

# 7.20 练习题

## 1. Labyrinth CF1063B

**【问题描述】**

你正在玩一款电脑游戏。在其中一关，你位于一个 n 行 m 列的迷宫。每个格子要么是可以通过的空地，要么是障碍。迷宫的起点位于第 r 行第 c 列。你每一步可以向上、下、左、右中的一个方向移动一格，前提是那一格不是障碍。你无法越出迷宫的边界。 不幸的是，你的键盘快坏了，所以你只能向左移动不超过 x 格，并且向右移动不超过 y 格。因为上下键情况良好，所以对向上和向下的移动次数没有限制。 现在你想知道在满足上述条件的情况下，从起点出发，有多少格子可以到达（包括起点）？

**【输入格式】**

第一行包含两个数 n,m (1 <= n, m, <= 2000),表示迷宫的行数和列数。 第二行包含两个数 r,c (1 <= r <= n, 1 <= c <= m)表示起点位于第 r 行第 c 列。 第三行包含两个数 x,y (1 <= x,y <= 10^9),表示最多向左或向右移动的次数。 接下来 n 行描述这个迷宫，每行为一个长为 m 的由 '.' 和 '' 组成的字符串。 '.' 表示空地， '' 表示障碍。 输入保证起点不是障碍。

**【输出格式】**

输出一个整数，表示从起点出发可以到达的格子数，包括起点本身。

**【样例输入】**

```
4 5
3 2
1 2
.....
.***.
...**
*....
```

**【样例输出】**

```
10
```

**题目链接：https://www.luogu.com.cn/problem/CF1063B**

**【题目分析】**

带左右方向限制次数的 bfs，最容易想到的方案是对 bfs 进行优化，在 bfs 的过程中记录左右移动次数，不超过限制，相比朴素的宽度搜索，在遍历到同一个点的时候，两条不同的路径，向左和向右移动的次数也不同。这时候就会产生大量的位置相同但左右移动次数不同的状态，要想办法剪枝，可以利用 vis 数组对不可能的最优解进行优化，如果到某个点后已经使用的左右移动次数比此前访问过的次数多，就不会是最优解。

另一种方案是 01bfs，当边权只有 0/1 时，可以借助双端队列进行 bfs，在队列中记录坐标以及当前还剩下的左右移动次数，把不用花代价的方案放到队首，把花代价的放到队尾，每次从队首取即可，这样任意时刻队列中的点 dist 差值不会超过 1，相比普通的 bfs,能避免出现“一个点可能有多种通达的方法，但是，不同的通达方法也有不同的左右移动次数，第一次到达这个点的时候，可能左右移动次数并不是最小的，导致下一个点无法访问”这样的问题。

**【参考代码】**

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <deque>
using namespace std;

struct node{
    int a, b, lc, rc;
};

const int N = 2022;
int n, m, r, c, x, y, cnt;
```

```
char g[N][N];
deque<node> q;
bool v[N][N];

int main(){
    scanf("%d%d%d%d%d%d", &n, &m, &r, &c, &x, &y);
    for (int i = 1; i <= n; i++)    scanf("%s", g[i] + 1);

    q.push_back({r, c, x, y});

    while(q.size()){
        auto t = q.front();
        q.pop_front();
        int a = t.a, b = t.b;
        if(v[a][b])    continue;

        v[a][b] = true, cnt++;

        if (a - 1 >= 1 && g[a - 1][b] == '.')
            q.push_front({a - 1, b, t.lc, t.rc});
        if (a + 1 <= n && g[a + 1][b] == '.')
            q.push_front({a + 1, b, t.lc, t.rc});
        if (b - 1 >= 1 && g[a][b - 1] == '.' && t.lc)
            q.push_back({a, b - 1, t.lc - 1, t.rc});
        if (b + 1 <= m && g[a][b + 1] == '.' && t.rc)
            q.push_back({a, b + 1, t.lc, t.rc - 1});
    }
    printf("%d\n", cnt);
    return 0;
}
```

## 2. Proud Merchants HDU-3466

**【问题描述】**

有 n(1<=N<=500)件物品，M(1<=M<=5000)元钱。每件物品最多买一次，且每件物品除了价格 P 和价值 V 外，还有限制 Q，代表你当前至少有钱数 Q 时，商家才愿意把东西卖给你。求使用不多于 M 的钱最多获得的价值。

**【输入格式】**

多组输入，每组输入以两个整数 N，M 为开始，分别表示物品数量以及钱数。接下来有 N 行，每一行分别表示第 i 个物品的价格 Pi，价值 Vi，以及限制 Qi。输入以 EOF 结尾。

**【输出格式】**

对于每组输入，输出一个整数代表购买物品的最大的价值

**【样例输入】**

```
2 10
10 15 10
5 10 5
3 10
5 10 5
3 5 6
2 7 3
```

**【样例输出】**

```
5
11
```

**题目链接：**https://vjudge.csgrandeur.cn/problem/HDU-3466

**【题目分析】**

考察对 dp 无后效性的理解以及对背包物品顺序的理解，题目要求隐含了物品选取有后效性的问题，物品购买的顺序会影响后续物品的购买，需要贪心地进行购买，假设 A,B 两个物品,要使得先后顺序让所需空间最小，才能让后面的价值可能越大

假如 A 先放，则需要空间 q1+p2;

B 先放，则所需空间为 q2+p1;

假如要让 A 排在前面,则排序方式为 q1+p2<q2+p1

转化一下就是 q1-p1 > q2-p2，也就是 qi-pi 大的先买。

这里同时考察了对背包物品顺序的掌握程度，在 dp 过程中，购买第 i 件物品时，是把第 i 件物品当作先买来考虑的，所以说最终的 dp 顺序要把 qi-pi 从小到大排序，这样表示越往后的购买顺序越优先。

**【参考代码】**

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <cstdlib>
using namespace std;
#define N 510
#define met(a, b) memset(a, b, sizeof(a))

int dp[N*10], n, m;

struct node
{
    int v, q, p, qp;
}a[N];
int cmp(node p, node q)
{
    return p.qp<q.qp;
}
int main()
{
    while(scanf("%d %d", &n, &m)!=EOF)
    {
        met(a, 0);
        met(dp, 0);
        for(int i=1; i<=n; i++)
        {
            scanf("%d%d%d", &a[i].p, &a[i].q, &a[i].v);
            a[i].qp = a[i].q - a[i].p;
        }
        sort(a+1, a+n+1, cmp);
        for(int i=1; i<=n; i++)
        {
            for(int j=m; j>=a[i].q; j--)///dp[j]表示剩余 j 元钱所能购买物品的最大价值;
                dp[j] = max(dp[j], dp[j-a[i].p]+a[i].v);
        }
        printf("%d\n", dp[m]);
```

```
    }
    return 0;
}
```

## 3. Jamie and Interesting Graph CF916C

**【问题描述】**

给定 n 个点 m 条边，满足 2<=n<=1e5，n-1<=m<=min(n(n-2)/2,1e5)。要求构造一张无向图，满足以下条件：

包含 n 个点，m 条边；

边权范围在[1,1e9]内；

1 到 n 的最短路径长度是质数；

最小生成树的边权和为质数，且不超过 1e14；

没有重边和自环。

**【输入格式】**

两个整数 n 和 m。

**【输出格式】**

第一行输出最短路径的长度和最小生成树的大小。

后 m 行输出图。

**【样例输入】**

```
4 4
```

**【样例输出】**

```
7 7
1 2 3
2 3 2
3 4 2
2 4 4
```

**题目链接：https://www.luogu.com.cn/problem/CF916C**

**【题目分析】**

构造。考察构造的思路，方案有很多，比较好想的一种方案是将最短路和最小生成树看作一条链，这条链的权值和为质数，考虑如何让这条路最小，可以把所有选上的边都标上 1 ，为了保证和是一个质数，随便选一个大于 10^5 的质数，例如 1e5+3，让 1 到 2 的一条边等于 10^5+3-n+2，这样其他想要选的边全标 1，这样最小生成树和最短路就构造好了，其余的边补齐即可。

**【参考代码】**

```
#include <bits/stdc++.h>
using namespace std;
const int p = 1e5 + 3;
int n, m;
int main() {
    cin >> n >> m;
    cout << p << ' ' << p << "\n" << "1 2 " << p - n + 2 << "\n";
    for(int i = 2; i < n; i++)
        cout << i << ' ' << i + 1 << " 1\n";
    m -= n - 1;
    for(int i = 0, j = n; m--; ) {
        if(++j > n)
            j = ++i + 2;
        cout << i << ' ' << j << " 1000000000\n";
    }
    return 0;
```

```
}
```

## 4. Toss a Coin to Your Graph... CF1679D

**【问题描述】**

珂朵莉给了你一个有向图，边数最大为2*10^5，每个点有一个点权，任选起点，走k步，问经过的点的最大权值最小能是多少？k≤10^18，无解输出-1，没有重边和自环，但是会有环。

**【输入格式】**

第一行三个整数，n，m，k分别表述点、边、步长，第二行有n个数分别表示n个的点权ai，后续有m行分别表示有向图的路径。

**【输出格式】**

输出最大权值的最小值。

**【样例输入】**

```
6 7 4
1 10 2 3 4 5
1 2
1 3
3 4
4 5
5 6
6 2
2 5
```

**【样例输出】**

```
4
```

**题目链接：https://www.luogu.com.cn/problem/CF1679D**

**【题目分析】**

最大权值最小，考虑二分做法，寻找题目中的单调性：如果把所有点权超过某个阈值的点从图上删除，那么图上剩余点的数量与这个阈值是有单调关系的，阈值越小，被删掉的点就越多，同时剩余点能组成的最长链的长度可能就越短，这种单调关系可以帮助我们二分结果。

在二分操作中，对点权ai<=mid的点进行建图，将问题转化为是否存在长度len>=k的路径，那么这个图会存在链和环两种情况，如果存在环，那么一定有长度大于等于k点路径，直接进行拓扑排序或者dfs找环即可，对于链的情况，用拓扑排序求最长路即可。

**【参考代码】**

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int maxn=1e6+5;
const int inf=0x3f3f3f3f;
typedef long long ll;
typedef pair<int,int> PII;
int n,m,k;
struct Edge
{
    int from,to;
};

vector<int>G[maxn];
vector<Edge>edges;
```

```
void add(int from,int to)
{
    edges.push_back({from,to});
    int m=edges.size();
    G[from].push_back(m-1);
}


bool vis[maxn];
int a[maxn],indegree[maxn],dep[maxn];

bool check(int x)
{
    for(int i=1;i<=n;i++)
     {
         vis[i]=0;
         dep[i]=-inf;
         indegree[i]=0;
     }
    for(int i=1;i<=n;i++)
    {
        if(a[i]<=x)
            vis[i]=1;
    }
    for(int i=0;i<edges.size();i++)
    {
        Edge e=edges[i];
        int u=e.from,v=e.to;
        if(a[u]<=x&&a[v]<=x)
            indegree[v]++;
    }
    queue<int>q;
    for(int i=1;i<=n;i++)
    {
        if(vis[i]&&!indegree[i])
            {
                q.push(i);
                dep[i]=1;
            }
    }
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        for(int i=0;i<G[u].size();i++)
        {
            Edge e=edges[G[u][i]];
            int v=e.to;
            if(!vis[v])
                continue;
            dep[v]=max(dep[u]+1,dep[v]);
```

```
            if(dep[v]>=k)//找到最长路径了
                return 1;
            indegree[v]--;
            if(indegree[v]==0)
                q.push(v);
        }
    }

    for(int i=1;i<=n;i++)
    {
        if(vis[i]&&indegree[i])//说明有环
            return 1;
    }
    return 0;

}
int main()
{
    cin>>n>>m>>k;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
    }
    for(int i=0;i<m;i++)
    {
        int u,v;
        cin>>u>>v;
        add(u,v);
    }
    if(k==1)
    {
        cout<<*max_element(a+1,a+1+n)<<endl;
        return 0;
    }
    int ans=inf;
    int l=0,r=1e9;
    while(l<=r)
    {
        int mid=(l+r)>>1;
        if(check(mid))
        {
            r=mid-1;
            ans=min(ans,mid);
        }
        else
            l=mid+1;
    }
    if(ans==inf)
        cout<<-1;
    else
        cout<<ans;
```

```
}
```

## 5. Robot on the Board 2 CF1607F

**【问题描述】**

有一个机器人在 n×m 的棋盘上移动，每个格子中写着 L、U、R、D 其中一个字母，依次代表机器人到这个格子后会向左、下、右、上方向走。 机器人不能重复经过一个格子，也不能走出棋盘，问机器人在哪个格子开始走可以走到的格子最多，以及最多能走到多少个格子。

一共 T 组数据。

**【输入格式】**

第一行输入一个数，T;以下 T 组测试数据,每组数据第一行输入 n,m,然后输入 n 行，每行一个长度为 m 的只含 L、R、U、D 的字符串。

**【输出格式】**

对于每组数据，输出一行三个数，前两个数表示机器人起始位置，第三个数表示机器人最多走多少格。若有多个位置符合要求，任意输出一个即可。

**【样例输入】**

```
6 7 4
7

1 1
R

1 3
RRL

2 2
DL
RU

2 2
UD
RU

3 2
DL
UL
RU

4 4
RRRD
RUUD
URUD
ULLR

4 4
DDLU
RDDU
UUUU
RDLD
```

【样例输出】

```
1 1 1
1 1 3
1 1 4
2 1 3
3 1 5
4 3 12
1 1 4
```

题目链接：https://www.luogu.com.cn/problem/CF1607F

【题目分析】

爆搜练习题，从每个点出发能走的路径是唯一确定的。每一次选一个没确定答案的点开始搜索，开 path 数组记录这次搜索经过的点。搜索过程中，如果遇到已经确定答案的点或者到达边界，就回溯更新 path 上的所有点的答案；若遇到这次搜索曾走过的点，说明这次搜索遇到了一个环，就回溯找环的起点，并把环上所有点的距离更新为环的长度，注意边界处理。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const signed N = 2010;
char G[N][N];
int cnt[N][N], pathx[N * N], pathy[N * N];
int n, m;
void sou(int x, int y)
{
    int len = 0, done = 0;
    while(x >= 1 && x <= n && y >= 1 && y <= m)
    {
        if(cnt[x][y] == -1) //这条路上有环
        {
            int cirbegin = len;
            while(pathx[cirbegin] != x || pathy[cirbegin] != y)
                cirbegin--; //往回找环的起点
            for(int i = cirbegin; i <= len; i++) //更新环上所有点的答案
                cnt[pathx[i]][pathy[i]] = len - cirbegin + 1;
            done = len - cirbegin + 1;
            len = cirbegin - 1;
            break;
        }
        if(cnt[x][y]) //到达已经有答案的点
        {
            done = cnt[x][y];
            break;
        }

        pathx[++len] = x, pathy[len] = y;
        cnt[x][y] = -1; //标记这次搜索中走过该点

        switch(G[x][y])
        {
            case 'L': y--; break;
            case 'R': y++; break;
            case 'D': x++; break;
```

```
            case 'U': x--; break;
        }
    }

    for(int i = 1; i <= len; i++) //回溯更新路径上所有点的答案
        cnt[pathx[len+1-i]][pathy[len+1-i]] = i + done;
}

signed main()
{
    int T; cin >> T; while(T--)
    {
        cin >> n >> m;
        for(int i = 1; i <= n; i++) cin >> G[i] + 1;

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

        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= m; j++)
                if(!cnt[i][j]) sou(i, j);

        int xx = 0, yy = 0;
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= m; j++)
                if(cnt[xx][yy] < cnt[i][j])
                    xx = i, yy = j;
        cout << xx << ' ' << yy << ' ' << cnt[xx][yy] << '\n';
    }

    return 0;
}
```

## 6. A and B and Lecture Rooms CF519E

**【问题描述】**

A 和 B 在准备参加编程比赛。

A 和 B 学习的大学的房间由走廊连接。大学一共有 n 个房间，由 n−1 条走廊连接，房间的编号是从 1 到 n 的数字编号。

A 和 B 在大学的某些房间里进行比赛。在每场比赛之后，他们会一起在一个房间里讨论问题。A 和 B 希望这个讨论问题的房间到分别他们两个人比赛房间的距离相等。两个房间之间的距离指他们之间最短路的边数。

因为A 和B 每天都在不一样的房间里比赛，所以他们请求你告诉他们在接下来比赛的m 天里可以用来讨论问题的房间有多少个？

**【输入格式】**

第一行包括整数 n(1<=n<=1e5)，表示房间数量。接下来的 n-1 行描述所有的走廊，这 n-1 行中的第 i 行包括两个整数 ai 和 bi，表示第 i 条走廊连接了房间 ai 和 bi。接下来一行输入比赛的天数 m(1<=m<=1e5)，再接下来的 m 行，第 j 行包含两个整数 xj 和 yj，表示第 j 天 A 将在 xj 房间比赛，B 将在 yj 房间比赛。

**【输出格式】**

在第 i 天输出当天分别到 A、B 比赛的房间距离相等的房间数量。

**【样例输入】**

```
4
1 2
1 3
2 4
1
2 3
```

**【样例输出】**

```
1
```

**题目链接：https://www.luogu.com.cn/problem/CF519E**

**【题目分析】**

由于树上任意两点的路径是唯一的，可以先考虑路径 A→B 上有哪些点能对答案产生贡献。产生贡献的点为路径的中点，且该点为树的一个节点。也就是说，只有路径 A→B 上有奇数个点时，才会有节点对答案产生贡献。

而当路径 A→B 上有偶数个点时，没有一个点能满足到 A 的距离和到 B 的距离相等。

接下来考虑不在路径上的点，当前考虑 A→B 中点存在时的情况。显然，只要一个点到 A 点和 B 点的路径均包含路径 A→B 的中点，该点就对答案产生贡献。也就是说，当一个点到 A 点和 B 点的路径中遇到的第一个路径 A→B 上的点为 A→B 中点时，这个点就对答案产生贡献。

因为该点到 A→B 中点的路径是唯一的，而 A→B 中点到 A 点和 B 点距离相等，所以该点到 A 点和 B 点距离是相等的。同理，当 A→B 中点不存在时，树上任意一个点都不对答案产生贡献，即答案为 0。

接下来考虑计算答案，我们需要维护 d（x）表示 x 的深度，size（x）表示 x 的子树大小，mid（x，y）表示 xy 的中点，lca（x，y）表示 x 和 y 的最近公共祖先，son（x）表示 x 的儿子集合，fa（x）表示 x 的父节点，这些信息都可以帮助我们快速计算答案。

当 A 和 B 深度不一样时，此时 A→B 中点一定在 A，B 当中深度较大的那个点 lca（A，B）的路径上，设点 C 在 A→B 的路径上并且为 lca（mid（A，B））的子节点，mid（A，B）以上的节点均不会对答案产生贡献，则答案为 size（mid(A,B)）-size（C）。

当 A 和 B 深度相等时，此时 A→B 中点一定为 lca（A，B），设点 C，D 在 A→B 的路径上并且为 lca（mid（A，B））的子节点，则答案为 n-size（C）-size（D）。

**【参考代码】**

```
#include<bits/stdc++.h>
using namespace std;
char G[N][N];
#define N 200010
#define LL long long
#define lson l,m,rt<<1
#define rson m+1,r,rt<<1|1
using namespace std;
struct edge
{
    int v,next;
    edge() {}
    edge(int v,int next):v(v),next(next) {}
} e[N];
int head[N],tot;
int num[N],deep[N];
int fa[N][20];
void init()
{
    memset(head,-1,sizeof(head));
    tot=0;
}
```

```
void addedge(int u,int v)
{
    e[tot]=edge(v,head[u]);
    head[u]=tot++;
}
void dfs(int x,int f)
{
    for(int i=1; i<20; i++)
    {
        if(deep[x]<(1<<i))break;
        fa[x][i]=fa[fa[x][i-1]][i-1];
    }
    for(int i=head[x]; ~i; i=e[i].next)
    {
        int v=e[i].v;
        if(v==f)continue;
        fa[v][0]=x;
        deep[v]=deep[x]+1;
        dfs(v,x);
        num[x]+=num[v];
    }
}
int lca(int x,int y)
{
    if(deep[x]<deep[y])swap(x,y);
    int t=deep[x]-deep[y];
    for(int i=0; i<=19; i++)
        if((1<<i)&t)x=fa[x][i];
    for(int i=19; i>=0; i--)
    {
        if(fa[x][i]!=fa[y][i])
        {
            x=fa[x][i];
            y=fa[y][i];
        }
    }
    if(x==y)return x;
    return fa[x][0];
}
int main()
{
    int n,m;
    while(scanf("%d",&n)>0)
    {
        init();
        for(int i=1; i<n; i++)
        {
            int u,v;
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
```

```
        }
        for(int i=1;i<=n;i++)num[i]=1;
        deep[1]=1;
        dfs(1,0);
        scanf("%d",&m);
        while(m--)
        {
            int x,y;
            scanf("%d%d", &x,&y);
            if(deep[x]<deep[y])swap(x,y);
            if((deep[x]-deep[y])&1) printf("0\n");
            else
            {
                int xy=lca(x, y),tx=x;
                int t=(deep[x]+deep[y]-2*deep[xy])/2-1;
                for(int i=0;i<20;i++)if(t&(1<<i))x=fa[x][i];
                if(deep[tx]==deep[y])
                {
                    t=deep[y]-deep[xy]-1;
                    for(int i=0;i<20;i++)if(t&(1<<i))y=fa[y][i];
                    printf("%d\n",n-num[x]-num[y]);
                }
                else printf("%d\n", num[fa[x][0]]-num[x]);
            }
        }
    }
}
```