Acwing算法基础课

一、Dynamic Programming

序号	题目描述	备注
01	01背包问题	背包要思考的就是拿还是不拿
02	完全背包问题	完全背包从小到大分析问题，因为要基于之前的改变
03	多重背包问题I	从拿不拿到拿几个
04	多重背包问题II	
05	多重背包问题III	多III也要从小到大分析问题，单调队列也是基于之前的改变完成的
- -	滑动窗口	这是为了做上一道题搞得
06	混合背包问题	就是之前的混合
07	二维费用的背包问题	就是费用变成二维
08	分组背包问题	
09	有依赖的背包问题	递归-难
10	背包问题求方案数	方案数本身就能动态规划，我靠

1. 01背包问题

- 填表的时候，表格元素肯定是价值。行索引指背包容量，列索引指有前i件物品。
- 通过获得子问题最优解就能得到全局最优解。所以每次只考虑子问题最优解。
- 为什么记录下了所有可能的情况，就是所有容量都考虑到了，且初始化指出了初始不拿的情况。
- 待补充

```
#include <iostream>
#include<algorithm>
using namespace std;
const int maxN = 10001;
int volArr[maxN];
int valArr[maxN];
int matrix[maxN][maxN];
int main()
```

```
{
    int num, total;
    cin >> num >> total;
    for (int i = 1;i <= num;i++) {
        cin >> volArr[i] >> valArr[i];
    }
    // i代表前几个物品 j代表背包空间
    for (int i = 1; i <= num; i++)
    {
        for (int j = 1; j <= total; j++) {
            if (j < volArr[i]) {
                matrix[i][j] = matrix[i - 1][j];
            }
            else {
                matrix[i][j] = max(matrix[i - 1][j], matrix[i - 1][j - volArr[i]]
+ valArr[i]);
            }
        }
    }
    cout << matrix[num][total];
}
```

2. 完全背包问题

- 两个维度，考虑背包每次扩容后，是加入新的物品还是沿用旧的解决方案。也就是***拿还是不拿***。

```
void versionOne(int num, int total) {
    for (int i = 1; i <= num; i++)
    {
        for (int j = 1; j <= total; j++) {
            if (j < volArr[i]) {
                matrix1[i][j] = matrix1[i - 1][j];
            }
            else {
                matrix1[i][j] = max(matrix1[i - 1][j], matrix1[i][j - volArr[i]]
+ valArr[i]);
            }
        }
    }
    cout << matrix1[num][total] << endl;
}
int main() {
    versionOne(num, total);
}
```

3. 多重背包问题I

- 比较暴力 - -!

```
案例   答案是200
10 100
10 17 3
9 15 3
9 9 1
14 28 5
18 20 4
6 12 5
21 32 4
19 25 3
24 28 2
24 34 2
```

```
void versionOne(int num, int total) {
    for (int i = 1; i <= num; i++)
    {
        for (int j = total; j >= 1; j--) {
            for (int k = 0; k <= numArr[i] && k * volArr[i] <= j; k++) {
                matrix1[i][j] = max(matrix1[i - 1][j], matrix1[i - 1][j - k *
volArr[i]] + k * valArr[i]);
            }
        }
    }
    cout << matrix1[num][total] << endl;
}

void versionTwo(int num, int total) {
    for (int i = 1; i <= num; i++)
    {
        for (int j = total; j >= volArr[i]; j--) {
            for (int k = 0; k <= numArr[i] && k * volArr[i] <= j; k++) {
                matrix2[j] = max(matrix2[j], matrix2[j - k*volArr[i]] +
k*valArr[i]);
            }
        }
    }
    cout << matrix2[total] << endl;
}
```

4. 多重背包问题II

- 从拿不拿进阶到***拿多少***问题。
- 多重背包I里 一种解法是拆成s个单物品,另一种解法是暴力寻找拿多少个。
- 多重背包II里 拆成2进制形式的组合物品，然后回归01背包问题。

```
const int maxN = 2001;
int matrix2[maxN];
int main()
{
    int num, total;
    cin >> num >> total;
    vector<int> volVec;
    vector<int> valVec;
```

```
    int volTemp, valTemp, numTemp;
    for (int i = 1;i <= num;i++) {
        cin >> volTemp >> valTemp >> numTemp;
        for (int j = 1; j <= numTemp; j *= 2) {
            numTemp -= j;
            volVec.push_back(volTemp * j);
            valVec.push_back(valTemp * j);
        }
        if (numTemp > 0) {
            volVec.push_back(numTemp * volTemp);
            valVec.push_back(numTemp  * valTemp);
        }

    }
    for (int i = 0; i < volVec.size(); i++) {
        for (int j = total; j >= volVec[i]; j--)
        {
            matrix2[j] = max(matrix2[j], matrix2[j - volVec[i]] + valVec[i]);
        }
    }
    cout << matrix2[total];
}
```

5. 多重背包问题III

- 单调队列动态更新局部最大值 简化了最内层循环。
- 思路就是从当前容量出发，遍历所有可能的拿情况，从拿0到尽量拿。
- 完全背包是除了不拿和尽量拿之外的解，都和-v项重合了，所以在-v项的基础上搞定，从低到高循环。
- 多重背包I，纯暴力。
- 多重背包II, 二进制减少遍历。
- 多重背包III, 动态遍历，也在-v项的基础上搞定，从低到高循环。

```
#include<cstring>
const int maxN = 20001;
int matrix[maxN];
int copyy[maxN];
int q[maxN];
int main()
{
    int num, total;
    cin >> num >> total;
    int volTemp, valTemp, numTemp;
    for (int i = 0;i < num;i++) {
        memcpy(copyy, matrix, sizeof(matrix));
        cin >> volTemp >> valTemp >> numTemp;
        for (int j = 0; j < volTemp; j++) {
            int head = 0;
            int tail = -1;
            for (int k = j; k <= total; k += volTemp) {
                if (head <= tail && k - q[head] > numTemp*volTemp) {
                    head++;
```

```
                }
                while (head <= tail && copyy[q[tail]] - (q[tail]-j) / volTemp *
valTemp <= copyy[k] - (k - j) / volTemp * valTemp) {
                    tail--;
                }
                if (head <= tail) {
                    matrix[k] = max(matrix[k], copyy[q[head]] + (k -
q[head])/volTemp * valTemp);
                }
                q[++tail] = k;

            }
        }
    }
    cout << matrix[total];
}
```

6. 混合背包问题

```
for (int i = 1; i <= n; i++)
    {
        cin >> vol >> val >> num;
        if (num == -1) {
            for (int j = total; j >= vol; j--) {
                matrix[j] = max(matrix[j], matrix[j - vol] + val);
            }
        }
        else if (num == 0) {
            for (int j = vol; j <= total; j++) {
                matrix[j] = max(matrix[j], matrix[j - vol] + val);
            }
        }
        else {
            for (int j = 1; j <= num; j *= 2) {
                num -= j;
                goods.push_back({j*vol,j*val,vol});
            }
            if (num > 0) {
                goods.push_back({ num * vol,num * val, vol});
            }
        }
    }
    for (auto good : goods) {
        for (int j = total; j >= good.vol_o; j--) {
            if (good.vol_l <= j) {
                matrix[j] = max(matrix[j], matrix[j - good.vol_l] + good.val_l);
            }
        }
    }
```

7. 二维费用的背包问题

```
for (int i = 1; i <= n; i++)
    {
        cin >> v >> m >> w;
        for (int j = total_v; j >= v; j--) {
            for (int k = total_w; k >= m; k--) {
                matrix[j][k] = max(matrix[j][k], matrix[j - v][k - m] + w);
            }
        }
    }
```

8. 分组背包问题

```
matrix[j] = max(matrix[j], matrix[j - vols[k]] + vals[k]);
```

9. 有依赖的背包问题

- 递归之后写东西，就是自底向上影响
- 递归之前写东西，就是自顶向下影响
- 这里把问题从，拿了就得拿父节点，变成，从父节点开始拿，无论拿哪一个都可以保证父节点拿过了，根节点就是max结果。
- 问题变换成了分组背包问题，一层一组
- 实际上没咋看懂--!

```
#include<iostream>
#include<algorithm>
#include<vector>
// 实际上没懂
using namespace std;

const int maxN = 101;       // 这里是指背包的容量
int matrix[maxN][maxN];
int v[maxN];
int w[maxN];
vector<int> g[maxN];
int n,total, root;

void dfs(int x) {
    for (int i = v[x]; i <= total; i++) {
        matrix[x][i] = w[x];
    }
    // i是该组物品的索引
    for (int i = 0; i < g[x].size(); i++) {
        // g[x][i]是该组物品的顺序输入索引；x是当前节点的顺序索引
        int son = g[x][i];
        dfs(son);
        // x是顺序输入的索引
        for (int j = total; j >= v[x]; j--) {
            // 暴力搜索
            for (int k = 0; k <= j-v[x]; k++) {
```

```
                matrix[x][j] = max(matrix[x][j], matrix[x][j - k] + matrix[son]
[k]);
            }
        }
    }
}

int main() {
    cin >> n >> total;
    int p;
    for (int i = 1; i <= n; i++) {
        cin >> v[i] >> w[i] >> p;        //体积 价值 父节点（组号）
        if (p == -1) {
            root = i;
        }
        else {
            g[p].push_back(i);
        }
    }
    dfs(root);
    cout << matrix[root][total];
}
```

10. 背包问题求方案数

- 我是sb

```
int main() {
    int n, total;
    cin >> n >> total;
    for (int i = 0; i <= total; i++) {
        cnt[i] = 1;
    }
    int vol, val;
    for (int i = 1; i <= n; i++) {
        cin >> vol >> val;
        for (int j = total; j >= vol; j--) {
            int value = matrix[j - vol] + val;
            if (value > matrix[j]) {
                cnt[j] = cnt[j - vol];
                matrix[j] = matrix[j - vol]+val;
            }
            else if (value == matrix[j]) {
                cnt[j] = (cnt[j] + cnt[j - vol]) % modd;
            }
        }
    }
    cout << cnt[total];
}
```

11.最长上升子序列

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

```
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    for (int i = 1; i <= n; i++) {
        f[i] = 1;
        for (int j = 1; j < i; j++) {
            // here
            if (a[i] > a[j])        f[i] = max(f[i], f[j] + 1);
        }
    }
    int res = 0;
    for (int i = 1; i <= n; i++) {
        res = max(res, f[i]);
    }
    cout << res;
}
```

12.没有上司的舞会

```
// 这里getHead可以在输入的时候做
int getHead() {
    bool flag[N];
    for (int i = 1; i <= n; i++) {
        flag[i] = false;
    }
    for (int i = 1; i <= n; i++) {
        for (int ii = h[i]; ii != -1; ii = ne[ii]) {
            flag[e[ii]] = true;
        }
    }
    for (int i = 1; i <= n; i++) {
        if (flag[i] == false) {
            return i;
        }
    }
}
// 树形DP
void dfs(int u) {
    f[u][1] = a[u];
    for (int ii = h[u]; ii != -1; ii = ne[ii]) {
        int node = e[ii];
        dfs(node);
        f[u][0] += max(f[node][0], f[node][1]);
        f[u][1] += f[node][0];
    }
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    memset(h, -1, sizeof(h));
    int x, y;
    for (int i = 0; i < n-1; i++) {
```

```
        cin >> x >> y;
        insert(y, x);
    }
    int head = getHead();
    dfs(head);
    int res = max(f[head][0], f[head][1]);
    cout << res;
}
```

13.

二、Sort

1. 快速排序

```
int arr[100000];
int partition(int begin, int end) {
    int head = arr[begin];
    int i = begin;
    int j = end + 1;
    while (true) {
        while (arr[++i] < head) {
            if (i == end)
                break;
        }
        while (arr[--j] > head) {
            if (j == begin)
                break;
        }
        if (i >= j)
            break;
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    arr[begin] = arr[j];
    arr[j] = head;
    return j;
}
void quickSort(int begin, int end) {
    if (begin >= end)       return;
    int part = partition(begin, end);
    quickSort(begin, part - 1);
    quickSort(part + 1, end);
}
int main()
{
    int num;
    cin >> num;
    for (int i = 0; i < num; i++)
    {
```

```
        cin >> arr[i];
    }
    // 为了防止最坏情况
    random_shuffle(arr, arr + num);
    quickSort(0, num - 1);
    for (int i = 0; i < num; i++)
    {
        cout << arr[i] << " ";
    }
}
```

2. 归并排序

```
int arr[100000];
int aux[100000];
void merge(int begin, int mid, int end) {
    int i = begin;
    int j = mid + 1;
    for (int k = begin; k <= end; k++) {
        aux[k] = arr[k];
    }
    for (int k = begin; k <= end; k++) {
        if (i > mid)        arr[k] = aux[j++];
        else if (j > end)       arr[k] = aux[i++];
        else if (aux[i] < aux[j])       arr[k] = aux[i++];
        else    arr[k] = aux[j++];
    }
}
void MergeSort(int begin, int end) {
    if (begin >= end)       return;
    int mid = (begin + end) / 2;
    MergeSort(begin, mid);
    MergeSort(mid + 1, end);
    merge(begin, mid, end);
}
int main() {
    int num;
    cin >> num;
    for (int i = 0; i < num; i++)
    {
        cin >> arr[i];
    }
    MergeSort(0, num - 1);
    for (int i = 0; i < num; i++)
    {
        cout << arr[i] << " ";
    }
}
```

3. 二分查找

```
int arr[100000];
int q[10000];
// 标准二分查找
```

```
int bs(int num, int begin, int end) {
    while (begin <= end) {
        int mid = (begin + end) / 2;
        if (num < arr[mid]) {
            end = mid -1;
        }
        else if (num > arr[mid]) {
            begin = mid + 1;
        }
        else
            return mid;
    }
    return -1;
}
int bsmin(int num, int begin, int end, int minN) {
    if (begin > end)    return max(minN, -1);
    int mid = (begin + end) / 2;
    if (arr[mid] == num) {
        minN = mid;
        return bsmin(num, begin, mid - 1, minN);
    }
    else if (arr[mid] > num)    return bsmin(num, begin, mid - 1, minN);
    else return bsmin(num, mid + 1, end, minN);
}
int bsmax(int num, int begin, int end, int maxN) {
    if (begin > end)    return max(maxN, -1);
    int mid = (begin + end) / 2;
    if (arr[mid] == num) {
        maxN = mid;
        return bsmax(num, mid + 1, end, maxN);
    }
    else if (arr[mid] > num)    return bsmax(num, begin, mid - 1, maxN);
    else return bsmax(num, mid + 1, end, maxN);
}
int bs_1(){
    int s = 0, e = alls.size() - 1;
    while (s < e) {
        int mid = (s + e) / 2;
        // e是大于等于x的第一个元素
        if (alls[mid] >= x)  e =  mid;
        else {
            s = mid + 1;
        }
    }
    return e;
}
```

4.高精度加法

```
const int N = 100010;
int A[N], B[N], C[N];
int add(int len) {
    int flag = 0;
    for (int i = 0; i < len; i++) {
        C[i] += A[i] + B[i] + flag;
```

```
        flag = C[i] / 10;
        C[i] %= 10;
    }
    if (flag) {
        C[len] = 1;
        return len + 1;
    }
    return len;
}
int main() {
    // 整行字符串的输入
    string as, bs;
    cin >> as >> bs;
    for (int i = as.size() - 1, j = 0; i >= 0; i--, j++) {
        A[j] = as[i] - '0';
    }
    for (int i = bs.size() - 1, j = 0; i >= 0; i--, j++) {
        B[j] = bs[i] - '0';
    }
    int maxlen = max(as.size(), bs.size());
    int tot = add(maxlen);
    for (int i = tot - 1; i >= 0; i--) {
        cout << C[i];
    }
}
```

5.高精度乘法

```
const int N = 100010;
int A[N];
int B;
vector<int> C;
void multiple(int size) {
    int t = 0;
    for (int i = 0; i < size; i++) {
        t += A[i] * B;
        // 每次都只存最低位
        C.push_back(t % 10);
        t /= 10;
    }
    while (t) {
        C.push_back(t % 10);
        t /= 10;
    }
    // 防止B是0
    while (C.size() > 1 && C.back() == 0) C.pop_back();
}
```

6.高精度减法

```
bool kai_cmp(int aSize, int bSize) {
    if (aSize != bSize) return aSize > bSize;
    for (int i = aSize; i >= 0; i--)
        if (A[i] != B[i])
```

```
            return A[i] > B[i];
    return true;
}
int kai_minus(bool flag, int len) {
    if (!flag) {
        int t = 0;
        for (int i = 0; i < len; i++) {
            C[i] = B[i] - A[i] + t;
            t = C[i] >= 0 ? 0 : -1;
            C[i] = (10 + C[i]) % 10;
        }
    }
    else {
        int t = 0;
        for (int i = 0; i < len; i++) {
            C[i] = A[i] - B[i] + t;
            t = C[i] >= 0 ? 0 : -1;
            C[i] = (10 + C[i]) % 10;
        }
    }
    int i = len - 1;
    while (i > 0 && C[i] == 0) {
        i--;
    }
    return i;
}
```

7.高精度除法

```
void division(int aSize) {
    int bSize = 0;
    int bb = B;
    while (bb) {
        bb /= 10;
        bSize++;
    }
    for (int i = 0; i < bSize;i++) {
        yy = yy * 10 + A[i];
    }
    for (int i = bSize-1; i < aSize; i++) {
        xx[i] = yy / B;
        yy = yy % B;
        if(i < aSize-1) yy = yy * 10 + A[i + 1];
    }
    // 输出
    bool flag = false;
    for (int i = bSize - 1; i < aSize; i++) {
        if (xx[i] != 0 && !flag) {
            flag = true;
        }
        if (flag) {
            cout << xx[i];
        }
    }
    // 防止A是0
```

```
    if (!flag)  cout << 0;
    cout << endl;
    cout << yy;
}
```

8.前缀和（子矩阵的和）

```
// 高端的解：s[i][j] = s[i][j - 1] + s[i - 1][j] - s[i - 1][j - 1] + a[i][j];
// s[x2][y2] - s[x2][y1 - 1] - s[x1 - 1][y2] + s[x1 - 1][y1 - 1]
int main() {
    cin >> n >> m >> q;
    int x;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> x;
            mt[i][j] = x + mt[i][j - 1];
        }
    }
    int x1, y1, x2, y2;
    while (q--) {
        cin >> x1 >> y1 >> x2 >> y2;
        int sum = 0;
        for (int i = x1; i <= x2; i++) {
            sum += mt[i][y2] - mt[i][y1 - 1];
        }
        cout << sum << endl;
    }
}
```

9.差分

```
// 差分是前缀和的逆运算
int a[N];
int b[N];
int main() {
    cin >> n >> m;
    int x;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        b[i] = a[i] - a[i - 1];
    }
    int l, r, c;
    while (m--) {
        cin >> l >> r >> c;
        b[l] += c;
        b[r+1] -= c;
    }
    int t = 0;
    for (int i = 1; i <= n; i++) {
        t += b[i];
        cout << t << " ";
    }
}
```

10.差分矩阵

```
int a[N][N];
int b[N][N];
void insert(int x1, int y1, int x2, int y2, int c) {
    b[x1][y1] += c;
    b[x2 + 1][y1] -= c;
    b[x1][y2 + 1] -= c;
    b[x2 + 1][y2 + 1] += c;
}
int main() {
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            insert(i, j, i, j, a[i][j]);
        }
    }
    int x1, y1, x2, y2, c;
    while (q--) {
        cin >> x1 >> y1 >> x2 >> y2 >> c;
        insert(x1, y1, x2, y2, c);
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j - 1];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cout << b[i][j] << " ";
        }
        cout << endl;
    }
}
```

11.双指针（最长连续不重复子序列）

```
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    // 后指针是i 前指针是j
    for (int i = 0, j = 0; i < n; i++) {
        s[a[i]]++;
        while (s[a[i]] > 1) {
            s[a[j]]--;
            j++;
        }
```

```
        res = max(res, i - j + 1);
    }
    cout << res;
}
```

12.位运算

```
void count(int x) {
    int cnt = 0;
    while (x) {
        cnt += x & 1;
        x = x >> 1;
    }
    cout << cnt << " ";
}
// lowbit   树状数组的基本操作
//在C++里 -x == ~x+1      负x等于x取反+1。
//那么 x & (-x) == x & (~x + 1)，会返回x的最右一位1以及右边的零。
int lowbit(int x){
    return x & (-x);
}
int main(){
    int n;
    cin >> n;
    int x;
    while(n--){
        int res = 0;
        cin >> x;
        while(x){
            x -= lowbit(x);
            res++;
        }
        cout << res << " ";
    }
}
```

13.区间和

```
typedef pair<int, int> PII;
const int N = 300010;
int n, m;
vector<int> alls;
vector<PII> add, query;
// 映射之后的数组;
int a[N];
// 前缀和数组;
int s[N];
// 把数轴上的点映射到1,2,3,4,5...上
int find(int x) {
    int s = 0, e = alls.size() - 1;
    while (s < e) {
        int mid = (s + e) / 2;
        // 大于等于x的第一个元素
        if (alls[mid] >= x)  e =  mid;
```

```
        else {
            s = mid + 1;
        }
    }
    return e + 1;
}
int main() {
    cin >> n >> m;
    int x, c;
    for (int i = 0; i < n; i++) {
        cin >> x >> c;
        add.push_back({ x, c });
        alls.push_back(x);
    }
    int l, r;
    for (int i = 0; i < m; i++) {
        cin >> l >> r;
        query.push_back({ l, r });
        alls.push_back(l);
        alls.push_back(r);
    }
    sort(alls.begin(), alls.end());
    alls.erase(unique(alls.begin(), alls.end()), alls.end());
    for (int i = 0; i < n; i++) {
        PII t = add[i];
        a[find(t.first)] += t.second;
    }
    // 前缀和算法
    for (int i = 1; i <= alls.size(); i++) {
        s[i] = s[i - 1] + a[i];
    }
    for (auto item : query) {
        cout << s[find(item.second)] - s[find(item.first) - 1] << endl;
    }
}
```

14.区间合并

```
int main() {
    cin >> n;
    int l, r;
    for (int i = 0; i < n; i++) {
        cin >> l >> r;
        ia[i] = { l,r };
    }
    sort(ia, ia + n);
    int begin = ia[0].l;
    int end = ia[0].r;
    for (int i = 1; i < n; i++) {
        if (ia[i].l <= end) {
            end = max(ia[i].r, end);    // 注意这里要取最大值
        }
        else {
            res++;
            end = ia[i].r;
```

```
        }
    }
    cout << res+1;
}
```

三、数据结构

1.数组模拟单链表

```
const int N = 100010;
int idx;
// head是头结点的索引，e存当前节点的值，ne存当前节点下一节点的索引
int head, e[N], ne[N];

void add(int x) {
    e[idx] = x;
    ne[idx] = head;
    head = idx++;
}

void insert(int x, int y) {
    e[idx] = y;
    ne[idx] = ne[x];
    ne[x] = idx++;
}

void del(int x) {
    ne[x] = ne[ne[x]];
}

int main()
{
    head = -1;
    int m, x, y;
    char c;
    cin >> m;
    for (int i = 0; i < m; i++) {
        cin >> c;
        if (c == 'H') {
            cin >> x;
            add(x);
        }
        else if (c == 'I') {
            cin >> x >> y;
            insert(x-1, y);
        }
        else {
            cin >> x;
            if (x == 0) {
                head = ne[head];
            }
            else {
                del(x - 1);
```

```
            }
        }
    }
    int pp = head;
    while (pp != -1) {
        cout << e[pp] << " ";
        pp = ne[pp];
    }
}
```

2.数组模拟双链表

```
int e[N], l[N], r[N];
void init() {
    // 0和1都是哑点，就是实际上不存内容。
    r[0] = 1;
    l[1] = 0;
    idx = 2;
}
void insert(int k, int x) {
    e[idx] = x;
    l[r[k]] = idx;
    r[idx] = r[k];
    l[idx] = k;
    r[k] = idx++;
}
void del(int k) {
    r[l[k]] = r[k];
    l[r[k]] = l[k];
}
```

3.KMP

```
int n, m;
int ne[M]; //next[]数组，避免和头文件next冲突
char s[N], p[M];  //s为模式串， p为匹配串

int main()
{
    cin >> n >> s+1 >> m >> p+1;  //下标从1开始

    //求next[]数组 从2开始是以为ne[1]为0 i代表主串 j代表匹配串
    for(int i = 2, j = 0; i <= m; i++)
    {
        // j等于0或者匹配的时候 才能算下一个next值
        while(j && p[i] != p[j+1]) j = ne[j];
        // 当前是i和j+1做匹配
        if(p[i] == p[j+1]) j++;
        ne[i] = j;
    }
    //匹配操作
    for(int i = 1, j = 0; i <= n; i++)
    {
        while(j && s[i] != p[j+1]) j = ne[j];
```

```
        if(s[i] == p[j+1]) j++;
        if(j == m)  //满足匹配条件，打印开头下标，从0开始
        {
            //匹配完成后的具体操作
            //如：输出以0开始的匹配子串的首字母下标
            //printf("%d ", i - m); (若从1开始，加1)
            j = ne[j];              //再次继续匹配
        }
    }
}
```

4.trie树

```
// trie树 高效存储和查找字符串集合的数据结构
const int N = 100010;
// idx=0 是根节点&&空节点
// cnt尾节点单词个数
int son[N][26], cnt[N];
// 每个节点对应一个idx；idx存在son里
int idx;
void insert(string &str) {
    int p = 0;
    for (int i = 0; str[i]; i++) {
        int u = str[i] - 'a';
        if (!son[p][u]) {
            son[p][u] = ++idx;
        }
        p = son[p][u];
    }
    cnt[p]++;
}
int query(string& str) {
    int p = 0;
    for (int i = 0; str[i]; i++) {
        int u = str[i] - 'a';
        if (!son[p][u]) {
            return 0;
        }
        p = son[p][u];
    }
    return cnt[p];
}
```

```
class Trie {
public:
    struct Node {
        bool is_end;
        Node* son[26];
        Node() {
            is_end = false;
            for (int i = 0; i < 26; i++) son[i] = NULL;
        }
    };
    Node* root;
```

```
    Trie() {
        root = new Node();
    }

    void insert(string word) {
        Node* p = root;
        for (int i = 0; i < word.size(); i++) {
            int u = word[i] - 'a';
            if (!p->son[u]) p->son[u] = new Node();
            p = p->son[u];
        }
        p->is_end = true;
    }

    bool search(string word) {
        Node* p = root;
        for (int i = 0; i < word.size(); i++) {
            int u = word[i] - 'a';
            if (!p->son[u]) return false;
            p = p->son[u];
        }
        return p->is_end;
    }

    bool startsWith(string prefix) {
        Node* p = root;
        for (int i = 0; i < prefix.size(); i++) {
            int u = prefix[i] - 'a';
            if (!p->son[u]) return false;
            p = p->son[u];
        }
        return true;
    }
};
```

5.最大异或对

```
const int N = 100010;
const int M = 31 * 1e5;
int son[M][2];
int a[N];
int idx, n;
void insert(int x) {
    int p = 0;
    for (int i = 30; i >= 0; i--) {
        int u = x >> i & 1;     //取x的第i位二进制数
        if (!son[p][u]) son[p][u] = ++idx;
        p = son[p][u];
    }
}
int query(int x) {
    int p = 0;
```

```
    int res = 0;
    for (int i = 30; i >= 0; i--) {
        int u = x >> i & 1;
        if (son[p][1 - u]) {
            res = res * 2 + 1;
            p = son[p][1 - u];
        }
        else {
            res = res * 2 + 0;
            p = son[p][u];
        }
    }
    return res;
}
int main() {
    cin >> n;
    for (int i = 0; i < n;i++) {
        cin >> a[i];
        insert(a[i]);
    }
    int res = 0;
    for (int i = 0; i < n; i++) {
        res = max(res, query(a[i]));
    }
    cout << res;
}
```

6.合并集合

> 并查集

```
// 返回x的祖宗节点 + 路径压缩
int find(int x) {
    if (x != p[x]) {
        // 把节点的父节点指向根节点
        p[x] = find(p[x]);
    }
    return p[x];
}
void merge(int x, int y) {
    int px = find(x);
    int py = find(y);
    if (px != py) {
        p[px] = py;
    }
}
void query(int x, int y) {
    int px = find(x);
    int py = find(y);
    if (px == py) {
        cout << "Yes" << endl;
    }
    else {
        cout << "No" << endl;
    }
```

```
}
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        p[i] = i;
    }
    while (m--) {
        char op;
        int x, y;
        cin >> op >> x >> y;
        if (op == 'M') {
            merge(x, y);
        }
        else {
            query(x, y);
        }
    }
}
```

7.数组模拟栈

```
int stack[N];
int tt = -1;
int main() {
    for (int i = 0; i < m; i++) {
        if (str == "push") {
            stack[++tt] = x;
        }
        else if (str == "pop") {
            tt--;
        }
        else if (str == "empty") {
            if (tt == -1)   cout << "YES" << endl;
            else    cout << "NO" << endl;
        }
        else if (str == "top"){
            cout << stack[tt] << endl;
        }
    }
}
```

8.数组模拟队列

```
int queue[N];
int tt = -1;
int hh = 0;
int main() {
    for (int i = 0; i < m; i++) {
        if (str == "push") {
            queue[++tt] = x;
        }
        else if (str == "pop") {
            hh++;
        }
```

```
        else if (str == "empty") {
            if (tt < hh) cout << "YES" << endl;
            else    cout << "NO" << endl;
        }
        else if (str == "front"){
            cout << queue[hh] << endl;
        }
    }
}
```

9.模拟堆

```
int idx = 0;
// h堆，p指针，hp存的是编号， ph存的是堆索引，hp表示由堆区指向指针区，ph表示由指针区指向堆区
int h[N], ph[N], hp[N];
// 这里a b和idx一样是堆索引
void heap_swap(int a, int b) {
    swap(ph[hp[a]], ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(h[a], h[b]);
}
void sink(int k, int len) {
    while (k * 2 <= len) {
        int j = k * 2;
        if (j < len && h[j + 1] < h[j]) {
            j++;
        }
        if (h[k] < h[j]) {
            break;
        }
        heap_swap(k, j);
        k = j;
    }
}
void swim(int k) {
    while (k / 2 > 0 && h[k / 2] > h[k]) {
        int j = k / 2;
        heap_swap(k, j);
        k = j;
    }
}
void del(int k) {
    int u = ph[k];
    heap_swap(u, idx);
    idx--;
    sink(u, idx);
    swim(u);
}
void change(int k, int x) {
    h[ph[k]] = x;
    sink(ph[k], idx);
    swim(ph[k]);
}
int main() {
    cin >> n;
```

```
    int m = 0;
    for (int i = 1; i <= n; i++) {
        string str;
        cin >> str;
        if (str == "I") {
            cin >> h[++idx];
            ph[++m] = idx;
            hp[idx] = m;
            swim(idx);
        }
        // print min
        else if (str == "PM") {
            cout << h[1] << endl;
        }
        // delte min
        else if (str == "DM") {
            heap_swap(1, idx);
            idx--;
            sink(1, idx);
        }
        // delelte
        else if (str == "D") {
            int k;
            cin >> k;
            del(k);
        }
        // change
        else {
            int k, x;
            cin >> k >> x;
            change(k, x);
        }
    }
}
```

10.哈希（模拟散列表）

```
// 大于100000的第一个质数，散列表的长度
const int N = 100003;
int h[N], e[N], ne[N], idx;
void insert(int x) {
    int hash = (x % N + N) % N;
    e[idx] = x;
    ne[idx] = h[hash];
    h[hash] = idx++;
}
void query(int x) {
    int hash = (x % N + N) % N;
    int p = h[hash];
    while (p != -1) {
        if (e[p] == x) {
            cout << "Yes" << endl;
            return;
        }
        p = ne[p];
```

```
    }
    cout << "No" << endl;
}
int main() {
    int m;
    cin >> m;
    char op;
    int x;
    memset(h, -1, sizeof(h));
    while (m--) {
        cin >> op >> x;
        if (op == 'I') {
            insert(x);
        }
        else {
            query(x);
        }
    }
}
```

11.滑动窗口

> 模板

```
int hh = 0, tt = -1;
for (int i = 0; i < n; i ++ )
{
    while (hh <= tt && check_out(q[hh])) hh ++ ;  // 判断队头是否滑出窗口
    while (hh <= tt && check(q[tt], i)) tt -- ;
    q[ ++ tt] = i;
}
```

> 滑动窗口的最大小值

```
const int N = 1e6 + 10;
int a[N];
int q[N];
int hh = 0, tt = -1;
int n, k;
int main() {
    cin >> n >> k;
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }
    for(int i = 0; i < n; i++){
        if(i - q[hh] >= k)  hh++;
        while(hh <= tt && a[q[tt]] >= a[i]){
            tt--;
        }
        q[++tt] = i;
        if(i >= k-1) cout << a[q[hh]] << " ";
    }
    cout << endl;
    int hh = 0, tt = -1;
```

```
    // 最大值 单调减队列
    for(int i = 0; i < n; i++){
        if(i - q[hh] >= k)  hh++;
        while(hh <= tt && a[q[tt]] <= a[i]){
            tt--;
        }
        q[++tt] = i;
        if(i >= k-1) cout << a[q[hh]] << " ";
    }
}
```

四、搜索和图论

1.图的深度优先搜索

```
// 这里u是结点编号
void dfs(int u) {
    flag[u] = true;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!flag[j]) {
            dfs(j);
        }
    }
}
```

2.树的重心

```
void insert(int x, int y) {
    e[idx] = y;
    ne[idx] = h[x];
    h[x] = idx++;
}
int dfs(int u) {
    int cnt = 1;
    flag[u] = true;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!flag[j]) {
            int child_cnt = dfs(j);
            res[u] = max(res[u], child_cnt);
            cnt += child_cnt;
        }
    }
    res[u] = max(res[u], n-cnt);
    return cnt;
}
int main() {
    memset(h, -1, sizeof(h));
    cin >> n;
    int x, y;
    for (int i = 0; i < n - 1; i++) {
```

```
        cin >> x >> y;
        insert(x, y);
        insert(y, x);
    }
    dfs(1);
    int minR = res[1];
    for (int i = 2; i <= n;i++) {
        minR = min(minR, res[i]);
    }
    cout << minR;
}
```

3.图的广度优先搜索

```
int hh, t;
void bfs() {
    dis[1] = 0;
    q[++tt] = 1;
    while (hh <= tt) {
        int i = q[hh++];
        int dt = dis[i];
        for (int ii = h[i]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            if (dis[j] == -1) {
                q[++tt] = j;
                dis[j] = dt + 1;
            }
        }
    }
}
```

4.图中点的层次

```
void insert(int x, int y) {
    e[idx] = y;
    ne[idx] = h[x];
    h[x] = idx++;
}
void bfs() {
    dis[1] = 0;
    q[++tt] = 1;
    while (hh <= tt) {
        int i = q[hh++];
        int dt = dis[i];
        for (int ii = h[i]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            if (dis[j] == -1) {
                q[++tt] = j;
                dis[j] = dt + 1;
            }
        }
    }
}
```

5.有向图的拓扑排序

```
// 把入度==0 作为判定无前置节点的条件
void bfs() {
    queue<int> qi;
    for (int i = 1; i <= n; i++) {
        if (indg[i] == 0) {
            qi.push(i);
            flag[i] = true;
        }
    }
    while (qi.size()) {
        int x = qi.front();
        qi.pop();
        res.push(x);
        for (int ii = h[x]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            if (!flag[j]) {
                indg[j]--;
                if (indg[j] == 0) {
                    qi.push(j);
                    flag[j] = true;
                }
            }
        }
    }
}
```

5.Dijkstra

> $O(n^2)$，遍历的是结点，适合稠密图

```
int adj[N][N];
int dist[N];
bool flag[N];
int dijkstra() {
    memset(dist, 0x3f, sizeof(dist));
    dist[1] = 0;
    for (int i = 0; i < n; i++) {
        int t = -1;
        for (int j = 1; j <= n;j++) {
            if (!flag[j] && (t == -1 || dist[j] < dist[t])) {
                t = j;
            }
        }
        flag[t] = true;
        for (int j = 1; j <= n; j++) {
            dist[j] = min(dist[j], dist[t] + adj[t][j]);
        }
    }
    if (dist[n] == 0x3f3f3f3f)  return-1;
    return dist[n];
}
```

6.Dijkstra_heap

> O(mlog$_n$) 广搜每一条边，然后让节点进入优先队列，适合稀疏图

```
#include<queue>
typedef pair<int, int> PII;
void dijkstra() {
    memset(dis, 0x3f, sizeof(dis));
    dis[1] = 0;
    // 优先队列的使用 要指定使用什么数据结构存，还要指定排序方式，小顶堆用升序greater
    priority_queue<PII, vector<PII>, greater<PII>> pq;
    pq.push({ 0, 1 });
    while (pq.size()) {
        PII x = pq.top();
        pq.pop();
        int first = x.first;    // 到源点距离
        int second = x.second;  // 结点编号
        if (flag[second])   continue;       // 这里要注意
        flag[second] = true;
        for (int i = h[second]; i != -1; i = ne[i]) {
            int ii = e[i];
            if (dis[ii] > dis[second] + w[i]) {     // 这里要注意
                dis[ii] = dis[second] + w[i];
                pq.push({ dis[ii], ii });
            }
        }
    }
}
```

7.Bellmen-ford/有边数限制的最短路

```
struct edge {
    int from;
    int to;
    int cost;
};
vector<edge> ve;

void bellmanFord() {
    memset(dis, 0x3f, sizeof(dis));
    dis[1] = 0;
    for (int i = 0; i < k; i++) {
        memcpy(backup, dis, sizeof(dis));
        for (int j = 0; j < m; j++) {
            edge e = ve[j];
            // n-1次 松弛所有边
            dis[e.to] = min(dis[e.to], backup[e.from] + e.cost);
        }
    }
}
int main() {
    cin >> n >> m >> k;
    int x, y, z;
    for(int i = 0; i < m; i++){
```

```
        cin >> x >> y >> z;
        ve.push_back({ x, y, z });
    }
    bellmanFord();
    if (dis[n] > 0x3f3f3f3f / 2)    cout << "impossible";
    else cout << dis[n];
}
```

8.SPFA

```
// 这里flag是为了防止 已经在队列中的结点，再次被加入队列，可能会超时
int flag[N];
void spfa() {
    memset(dis, 0x3f, sizeof(dis));
    queue<int> qi;
    qi.push(1);
    flag[1] = true;
    dis[1] = 0;
    while (qi.size()) {
        int x = qi.front();
        qi.pop();
        flag[x] = false;
        for (int ii = h[x]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            int wei = w[ii];
            int dt = dis[x] + wei;
            if (dt < dis[j]) {
                dis[j] = dt;
                if (!flag[j]) {
                    qi.push(j);
                    flag[j] = true;
                }
            }
        }
    }
}
int main() {
    cin >> n >> m;
    int x, y, z;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        cin >> x >> y >> z;
        insert(x, y, z);
    }
    spfa();
    if (dis[n] == 0x3f3f3f3f)       cout << "impossible";
    else cout << dis[n];
}
```

9.SPFA判定负环

- 从虚拟源点到某结点的边数超过n-1，则视为有负环。

```
bool spfa() {
```

```
    queue<int> qi;
    // 添加一个虚拟源点，所有点到源点距离为0， 从一次bfs后开始写
    for (int i = 1; i <= n; i++) {
        qi.push(i);
        flag[i] = true;
    }
    while (qi.size()) {
        int x = qi.front();
        qi.pop();
        flag[x] = false;
        for (int ii = h[x]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            int wei = w[ii];
            if (dis[j] > dis[x] + wei) {
                dis[j] = dis[x] + wei;
                // cnt代表步数
                // 不能直接++ 是因为有重边 也能让j更新n次。
                cnt[j] = cnt[x] + 1;
                if (cnt[j] >= n) {
                    return true;
                }
                if (!flag[j]) {
                    flag[j] = true;
                    qi.push(j);
                }
            }
        }
    }
    return false;
}
```

10.Floyd

```
void floyd() {
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
            }
        }
    }
}
int main() {
    // 初始化工作也很重要
    memset(dis, 0x3f, sizeof(dis));
    cin >> n >> m >> k;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) {
                dis[i][j] = 0;
            }
        }
    }
    for (int i = 0; i < m; i++) {
        int x, y, z;
```

```
        cin >> x >> y >> z;
        dis[x][y] = min(z, dis[x][y]);
    }
    floyd();
    for (int i = 0; i < k; i++) {
        int x, y;
        cin >> x >> y;
        if (dis[x][y] > 0x3f3f3f3f / 2) {
            cout << "impossible" << endl;
        }
        else {
            cout << dis[x][y] << endl;
        }
    }
}
```

11.Prim算法求最小生成树

> 跟dijkstra特别特别像 --除了更新步骤

```
void Prim() {
    memset(dis, 0x3f, sizeof(dis));
    dis[1] = 0;
    for (int i = 1; i <= n; i++) {
        // 找离集合最近的点
        int t = -1;
        for (int j = 1; j <= n; j++) {
            if (!flag[j] && ( t==-1 || dis[j] < dis[t] )) {
                t = j;
            }
        }
        flag[t] = true;
        // 更新其他点到集合的距离
        for (int j = 1; j <= n; j++) {
            if (!flag[j]) {
                dis[j] = min(dis[j], adj[t][j]);
            }
        }
    }
}

int main() {
    cin >> n >> m;
    // 初始化
    memset(adj, 0x3f, sizeof(adj));
    int x, y, z;
    for (int i = 0; i < m; i++) {
        cin >> x >> y >> z;
        adj[x][y] = min(adj[x][y], z);
        adj[y][x] = min(adj[y][x], z);
    }
    Prim();
    int sum = 0;
    bool fg = false;
    for (int i = 1; i <= n; i++) {
```

```
        if (dis[i] == 0x3f3f3f3f) {
            cout << "impossible";
            fg = true;
            break;
        }
        else {
            sum += dis[i];
        }
    }
    if (!fg) {
        cout << sum;
    }
}
```

12.Kruskal算法求最小生成树

```
struct Edge {
    int from;
    int to;
    int cost;
    bool operator< (const Edge& e)const {
        return cost < e.cost;
    }
};
Edge edges[M];
int find(int x) {
    if (x != p[x]) {
        p[x] = find(p[x]);
    }
    return p[x];
}
void merge(int a, int b) {
    int pa = find(a);
    int pb = find(b);
    if (pa != pb) {
        p[pa] = pb;
    }
}
bool kruskal() {
    int cnt = 0;
    for (int i = 0;i < m;i++)
    {
        int a = edges[i].from, b = edges[i].to, w = edges[i].cost;
        if (find(a) != find(b)){
            merge(a, b);
            cnt++;
            sum += w;
        }
    }
    if (cnt == n - 1) {
        return true;
    }
    return false;
}
int main() {
```

```
    cin >> n >> m;
    int x, y, z;
    for (int i = 1; i <= n; i++) {
        p[i] = i;
    }
    for (int i = 0; i < m; i++) {
        cin >> x >> y >> z;
        edges[i] = { x, y, z };
    }
    sort(edges, edges + m);
    if (kruskal()) {
        cout << sum;
    }
    else {
        cout << "impossible";
    }
}
```

13.染色法判定二分图

```
bool dfs(int x, int c) {
    color[x] = c;
    for (int ii = h[x]; ii != -1; ii = ne[ii]) {
        int j = e[ii];
        if (color[j] == 0) {
            if (!dfs(j, -c)) {
                return false;
            }
        }
        else if (color[j] == color[x]) {
            return false;
        }
        else;
    }
    return true;
}
int main() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    int x, y;
    for (int i = 0; i < m; i++) {
        cin >> x >> y;
        add(x, y);
        add(y, x);
    }
    bool fg = true;
    for (int i = 1; i <= n; i++) {
        if (!color[i]) {
            if (!dfs(i, 1)) {
                fg = false;
                break;
            }
        }
    }
    if (fg) cout << "Yes";
```

```
    else    cout << "No";
}
```

14.匈牙利算法

> 二分图的最大匹配

```
// 二分图的匹配：给定一个二分图 G，在 G 的一个子图 M 中，
//M 的边集 {E} 中的任意两条边都不依附于同一个顶点，
//则称 M 是一个匹配。

// 匹配
int match[N];
// 预订
bool flag[N];
bool hungary(int x) {
    for (int ii = h[x]; ii != -1; ii = ne[ii]) {
        int j = e[ii];
        if (!flag[j]) {
            flag[j] = true;
            if (match[j] == 0 || hungary(match[j])) {
                match[j] = x;
                return true;
            }
        }
    }
    return false;
}
int main() {
    memset(h, -1, sizeof(h));
    int res = 0;
    cin >> n1 >> n2 >> m;
    int x, y;
    for (int i = 0; i < m; i++) {
        cin >> x >> y;
        add(x, y);
    }
    for (int i = 1; i <= n1; i++) {
        memset(flag, false, sizeof(flag));
        if (hungary(i)) {
            res++;
        }
    }
    cout << res;
}
```

15.八数码

```
int bfs(string start) {
    string end = "12345678x";
    queue<string> qs;
    unordered_map<string, int> dis;
    qs.push(start);
    dis[start] = 0;
```

```
    int dx[4]{ -1, 0, 1, 0 }, dy[4]{ 0, 1, 0, -1 };
    while (qs.size()) {
        string xx = qs.front();
        qs.pop();
        int distance = dis[xx];
        if (end==xx) return distance;
        int k = xx.find('x');
        int x = k / 3, y = k % 3;
        for (int i = 0; i < 4; i++) {
            int a = x + dx[i];
            int b = y + dy[i];
            if (a >= 0 && a < 3 && b >= 0 && b < 3) {
                swap(xx[k], xx[a * 3 + b]);
                if (!dis.count(xx)) {
                    dis[xx] = distance + 1;
                    qs.push(xx);
                }
                swap(xx[k], xx[a * 3 + b]);
            }
        }
    }
    return -1;
}
```

五、数学知识

1.试除法判断质数

```
if (x < 2) {
    cout << "No" << endl;
    continue;   // 0和1都不是质数
}
bool flag = true;
for (int i = 2; i <= x/i; i++) {
    if (x % i == 0) {
        cout << "No" << endl;
        flag = false;
        break;
    }
}
if (flag) {
    cout << "Yes" << endl;
}
```

2.分解质因数

```
void getD(int x){
    for(int i = 2; i <= x/i; i++){
        if(x % i == 0){
            int cnt = 0;
            while(x % i == 0){
                x /= i;
                cnt++;
```

```
            }
            cout << i << " " << cnt << endl;
        }
    }
    if(x > 1)   cout << x << " " << 1 << endl;
    cout << endl;
}
int main(){
    int n;
    cin >> n;
    while(n--){
        int x;
        cin >> x;
        getD(x);
    }
}
```

3.试除法判断约数

```
void get_divisors(int x) {
    vector<int> vi;
    for (int i = 1; i <= x / i; i++) {
        if (x % i == 0) {
            vi.push_back(i);
            if (i != x / i) {
                // 原理：一个数的俩约数，一小一大，判定完小的，大的就是自然而然的
                vi.push_back(x / i);
            }
        }
    }
    sort(vi.begin(), vi.end());
    for (auto t : vi) {
        cout << t << " ";
    }
    cout << endl;
}
```

4.欧拉函数

```
// 2e9 有110个约数
void getCommonDivisor(int x) {
    int res = x;
    for (int i = 2; i <= x/i; i++) {
        if (x % i == 0) {
            res = res / i * (i - 1);
            while (x % i == 0) {
                x /= i;
            }
        }
    }
```

```
    if(x > 1)   res = res / x * (x - 1);
    cout << res << endl;
}
```

5.快速幂

```
typedef unsigned long long ull;
// 把b拆解成二进制 比线性更快
void getQuickPower(ull a, ull b, ull p) {
    ull res = 1;
    ull x = a;
    while (b) {
        if (b & 1) {
            res = (ull)res * x % p;
        }
        b >>= 1;
        x = (ull)x*x % p;
    }
    cout << res << endl;
}
```

6.扩展欧几里得算法

```
// 欧几里得算法 辗转相除法
int gcd(int x, int y) {
    while (y) {
        int t = x % y;
        x = y;
        y = t;
    }
    return x;
}
// 扩展欧几里得算法
int exgcd(int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int x1, y1;
    int d = exgcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - a / b * y1;
    return d;
}
int main() {
    int n;
    cin >> n;
    int a, b, x, y;
    while (n--) {
        cin >> a >> b;
        exgcd(a, b, x, y);
        cout << x << " " << y << endl;
    }
```

```
}
```

1. 扩展欧几里得

用于求解方程 $ax+by=gcd(a,b)$ 的解

当 $b=0$ 时　$ax+by=a$ 故而 $x=1,y=0$

当 $b\neq 0$ 时

因为

$$gcd(a,b)=gcd(b,a\%b)$$

而

$$bx'+(a\%b)y'=gcd(b,a\%b)$$

$$bx'+(a-\lfloor a/b\rfloor *b)y'=gcd(b,a\%b)$$

$$ay'+b(x'-\lfloor a/b\rfloor *y')=gcd(b,a\%b)=gcd(a,b)$$

故而

$$x=y',\quad y=x'-\lfloor a/b\rfloor *y'$$

因此可以采取递归算法 先求出下一层的x'和y' 再利用上述公式回代即可

7.表达整数的奇怪方法

```
// 中国剩余定理
```

六、贪心

1.区间选点

```
vector<PII> vp;
int main()
{
    cin >> n;
    int x, y;
    for (int i = 0; i < n; i++) {
        cin >> x >> y;
        vp.push_back({ x, y });
    }
    sort(vp.begin(), vp.end());
    int res = 0;
    int end = -0x3f3f3f3f;
    for (int i = 0; i < n; i++) {
        if (end < vp[i].first) {
            end = vp[i].second;
            res++;
        }
        else {
            // 按右端点排的话，不用这一步。
            end = min(end, vp[i].second);
        }
    }
```

```
    cout << res;
}
```

2.最大不相交区间数量

```
// 同上
```

3.区间分组

```
int main() {
    cin >> n;
    int x, y;
    for (int i = 0; i < n; i++) {
        cin >> x >> y;
        vp.push_back({ x, y });
    }
    sort(vp.begin(), vp.end());
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int i = 0; i < n; i++) {
        if (pq.empty() || pq.top() >= vp[i].first)  pq.push(vp[i].second);
        else {
            pq.pop();
            pq.push(vp[i].second);
        }
    }
    cout << pq.size();
}
```

4.区间覆盖

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 100010;
int n;
struct interval {
    int l;
    int r;
    bool operator<(const interval& M)const {
        return l < M.l;
    }
}intervals[N];

int main() {
    int x, y;
    cin >> x >> y;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> intervals[i].l >> intervals[i].r;
    }
    sort(intervals, intervals + n);
```

```
    bool flag = false;
    int res = 0;
    for (int i = 0; i < n; i++) {
        int j = i;
        int r = -2e9;
        while (j < n && intervals[j].l <= x) {
            r = max(r, intervals[j].r);
            j++;
        }
        if (r < intervals[j].l) {
            res = -1;
            break;
        }
        res++;
        if (r >= y) {
            flag = true;
            break;
        }
        x = r;
        i = j - 1;
    }
    if (flag)  cout << res << endl;
    else cout << -1 << endl;
}
```

5.合并果子

```
```

七、树

1.基本操作

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int value) {
        val = value;
        left = NULL, right = NULL;
    }
};
// 插入一些结点 瞎写的
void insert(TreeNode*& root, int  value) {
    root->left = new TreeNode(value);
    root->right = new TreeNode(value);
}
// 二叉树高
int height(TreeNode *root) {
    if (!root)  return 0;
    return max(height(root->left) , height(root->right)) + 1;
}
// 二叉树结点个数
```

```
int node_num(TreeNode* root) {
    if (!root)  return 0;
    return node_num(root->left) + node_num(root->right) + 1;
}
// 二叉树遍历 先序
void preorder(TreeNode *root) {
    if (!root)  return;
    cout << root->val;
    preorder(root->left);
    preorder(root->right);
}
// 中序
void inorder(TreeNode* root) {
    if (!root)  return;
    inorder(root->left);
    cout << root->val;
    inorder(root->right);
}
// 后序
void postorder(TreeNode* root) {
    if (!root)  return;
    postorder(root->left);
    postorder(root->right);
    cout << root->val;
}
// 判断是否为平衡树
bool isBalance(TreeNode* root) {
    if (!root) return true;
    return isBalance(root->left) && isBalance(root->right) && abs(height(root-
>left) - height(root->right)) <= 1;
}
```

2.二叉搜索树（BST）

```
```

Acwing算法提高课

一、搜索

1.Flood Fill(BFS)

> 池塘计数

```
int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> a[i][j];
        }
    }
    int res = 0;
    queue<pair<int, int>> que;
```

```
    int dx[8]{ -1, -1, -1, 0, 0, 1, 1, 1 };
    int dy[8]{ -1, 0, 1, -1, 1, -1, 0, 1 };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (a[i][j] == '.' && !flag[i][j]) {
                res++;
                que.push({ i, j });
                flag[i][j] = true;
                while (que.size()) {
                    auto ele = que.front();
                    que.pop();
                    int ii = ele.first;
                    int jj = ele.second;
                    for (int k = 0; k < 8; k++) {
                        int x = ii + dx[k];
                        int y = jj + dy[k];
                        if (x >= 0 && x < n && y >= 0 && y < m && a[x][y] == '.'
&& !flag[x][y]) {
                            que.push({ x, y });
                            flag[x][y] = true;
                        }
                    }
                }
            }
        }
    }
    cout << res;
}
```

城堡问题

```
int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> g[i][j];
        }
    }
    queue<pair<int, int>> que;
    int is_wall[4]{ 0, 0, 0, 0 }; // 西 北 东 南
    int dx[4]{ 0 , -1, 0, 1 };  // 西 北 东 南
    int dy[4]{ -1, 0 , 1,  0 };
    int area = 0;
    int maxS = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (!flag[i][j]) {
                int s = 1;
                que.push({ i, j });
                flag[i][j] = true;
                area++;
                while (que.size()) {
                    auto ele = que.front();
                    que.pop();
                    int ii = ele.first;
```

```
                    int jj = ele.second;
                    for (int k = 0; k < 4; k++) {
                        is_wall[k] = g[ii][jj] >> k & 1;
                        int x = ii + dx[k];
                        int y = jj + dy[k];
                        if (x >= 0 && x < n && y >= 0 && y < m && !is_wall[k] &&
!flag[x][y]) {
                            que.push({ x, y });
                            S++;
                            flag[x][y] = true;
                        }
                    }
                }
                maxS = max(S, maxS);
            }
        }
    }
    cout << area << endl;
    cout << maxS;
}
```

```
void bfs(int sx, int sy, bool& has_higher, bool& has_lower) {
    queue<pair<int, int>> que;
    que.push({ sx, sy });
    flag[sx][sy] = true;
    while (que.size()) {
        auto ele = que.front();
        que.pop();
        for (int i = ele.first - 1; i <= ele.first + 1; i++) {
            for (int j = ele.second - 1; j <= ele.second + 1; j++) {
                if (i < 0 || i >= n || j < 0 || j >= n)     continue;
                if (g[i][j] != g[ele.first][ele.second]) {
                    if (g[i][j] > g[ele.first][ele.second])     has_higher =
true;
                    else        has_lower = true;
                }
                else if(!flag[i][j]){
                    que.push({ i , j });
                    flag[i][j] = true;
                }
            }
        }
    }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> g[i][j];
        }
    }
    int ln = 0, hn = 0;
    queue<pair<int, int>> que;
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (flag[i][j] == 0) {
                bool has_lower = false;
                bool has_higher = false;
                bfs(i, j, has_higher, has_lower);
                if (!has_lower) ln++;
                if (!has_higher) hn++;

            }
        }
    }
    cout << hn << " " << ln;
}
```

2.BFS

> 迷宫问题

```
//这里应该从终点开始走回到起点，就能输出正序路径了。
void bfs(int sx, int sy) {
    que[++tt] = { sx, sy };
    flag[sx][sy] = true;
    pre[sx][sy] = { -1, -1 };
    int dx[4]{ -1, 0, 1, 0 };
    int dy[4]{ 0, 1, 0, -1 };
    while (hh <= tt) {
        auto t = que[hh++];
        int x = t.first, y = t.second;
        if (x == n - 1 && y == n - 1) {
            return;
        }
        for (int i = 0; i < 4; i++) {
            int xx = x + dx[i], yy = y + dy[i];
            if (xx >= 0 && xx < n && yy >= 0 && yy < n && !g[xx][yy] && !flag[xx]
[yy]) {
                que[++tt] = { xx, yy };
                pre[xx][yy] = { x, y };
                flag[xx][yy] = true;
            }
        }
    }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> g[i][j];
        }
    }
    bfs(0, 0);
    int px = n - 1;
    int py = n - 1;
    while (px != -1) {
        cout << "(" << px << ", " << py << ")" << endl;
```

```
        auto idx = pre[px][py];
        px = idx.first;
        py = idx.second;
    }
}
```

> 武士风度的牛

```
int bfs(int cx, int cy) {
    memset(dis, -1, sizeof(dis));
    int dx[]{ -2, -2, 2, 2, -1, -1, 1, 1 };
    int dy[]{ -1, 1, 1, -1, -2, 2, 2, -2 };
    que[++tt] = { cx, cy };
    dis[cx][cy] = 0;
    while (hh <= tt) {
        auto t = que[hh++];
        if (g[t.first][t.second] == 'H')    return dis[t.first][t.second];
        for (int i = 0; i < 8; i++) {
            int x = t.first + dx[i], y = t.second + dy[i];
            if (x >= 0 && x < n && y >= 0 && y < m && dis[x][y] == -1 && g[x][y]
!= '*') {
                dis[x][y] = dis[t.first][t.second] + 1;
                que[++tt] = { x, y };
            }
        }
    }
}
int main() {
    cin >> m >> n;
    int cx, cy;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> g[i][j];
            if (g[i][j] == 'K') {
                cx = i, cy = j;
            }
        }
    }
    int res = bfs(cx, cy);
    cout << res;
}
```

> 抓住那头牛

```
int main() {
    cin >> n >> k;
    que[++tt] = n;
    tm[n] = 0;
    int res = 0;
    memset(tm, -1, sizeof(tm));
    while (hh <= tt) {
        int ft = que[hh];
        if (ft == k) {
            res = tm[hh];
```

```
            break;
        }
        if (tm[ft + 1] == -1) {
            que[++tt] = ft + 1;
            tm[ft + 1] = tm[ft] + 1;
        }
        if (tm[ft - 1] == -1) {
            que[++tt] = ft - 1;
            tm[ft - 1] = tm[ft] + 1;
        }
        if (tm[ft * 2] == -1) {
            que[++tt] = ft * 2;
            tm[ft * 2] = tm[ft] + 1;
        }
        hh++;
    }
    cout << res;
}
```

3.多源BFS

> 矩阵距离

```
//求每个节点到任意一个值为1节点的最小曼哈顿距离
void bfs() {
    int dx[]{ -1, 0, 1, 0 };
    int dy[]{ 0 , -1, 0, 1 };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (g[i][j] == 1) {
                dis[i][j] = 0;
                que[++tt] = { i, j };
            }
        }
    }
    while (hh <= tt) {
        auto t = que[hh++];
        for (int k = 0; k < 4; k++) {
            int x = t.first + dx[k], y = t.second + dy[k];
            if (x >= 0 && x < n && y >= 0 && y < m && dis[x][y] == -1) {
                dis[x][y] = dis[t.first][t.second] + 1;
                que[++tt] = { x, y };
            }
        }
    }
}
int main() {
    cin >> n >> m;
    char ele;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> ele;
            g[i][j] = ele - '0';
        }
    }
```

```
    memset(dis, -1, sizeof(dis));
    bfs();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << dis[i][j] << " ";
        }
        cout << endl;
    }
}
```

4.最小步数模型

> 魔板

```
int hh = 0, tt = -1;
vector<string> que;
unordered_map<string, bool> flag;
unordered_map<string, pair<char, string>> pre;

string opA(string s) {
    reverse(s.begin(), s.end());
    return s;
}
string opB(string s) {
    s.insert(s.begin(), s[3]);
    s.erase(s.begin() + 4);
    s.insert(s.end(), s[4]);
    s.erase(s.begin() + 4);
    return s;
}
string opC(string s) {
    s.insert(s.begin() + 1, s[6]);
    s.erase(s.begin() + 7);
    s.insert(s.begin() + 6, s[3]);
    s.erase(s.begin() + 3);
    return s;
}
void bfs(string& target) {
    string source = "12345678";
    tt++;
    que.push_back(source);
    flag[source] = true;
    pre[source] = { 'D', "12345678" };
    while (hh <= tt) {
        string ft = que[hh++];
        if (ft == target)   return;
        string sa = opA(ft);
        string sb = opB(ft);
        string sc = opC(ft);
        if (!flag.count(sa)) {
            tt++;
            que.push_back(sa);
            flag[sa] = true;
            pre[sa] = { 'A', ft };
        }
```

```
        if (!flag.count(sb)) {
            tt++;
            que.push_back(sb);
            flag[sb] = true;
            pre[sb] = { 'B', ft };
        }
        if (!flag.count(sc)) {
            tt++;
            que.push_back(sc);
            flag[sc] = true;
            pre[sc] = { 'C', ft };
        }
    }
}
int main() {
    string target;
    char ele;
    for (int i = 0; i < 8; i++) {
        cin >> ele;
        target += ele;
    }
    bfs(target);
    int cnt = 0;
    string ress;
    string nw = target;
    while (nw != "12345678") {
        cnt++;
        ress += pre[nw].first;
        nw = pre[nw].second;
    }
    reverse(ress.begin(), ress.end());
    cout << cnt << endl;
    cout << ress;
}
```

5.双端队列bfs

> 电路维修

```
typedef pair<int, int> PII;
int n, m;
const int N = 510;
bool flag[N][N];
char g[N][N];
int dis[N][N];
int bfs() {
    memset(flag, false, sizeof(flag));
    memset(dis, 0x3f, sizeof(dis));
    deque<PII> deq;
    char cs[5] = "\\/\\/";
    int dx[4]{ -1, -1, 1, 1 }, dy[4]{ -1, 1, 1, -1 };    //能到达点的偏移量
    int ix[4]{ -1, -1, 0, 0 }, iy[4]{ -1, 0, 0, -1 };        //应该的斜线索引偏移量，
中心点到斜线
    deq.push_back({ 0, 0 });
    dis[0][0] = 0;
```

```
    while (deq.size()) {
        auto t = deq.front();
        deq.pop_front();
        int x = t.first, y = t.second;
        if (x == n && y == m)       return dis[x][y];
        if (flag[x][y])     continue;
        flag[x][y] = true;
        for (int i = 0; i < 4; i++) {
            int a = x + dx[i], b = y + dy[i];
            if (a >= 0 && a <= n && b >= 0 && b <= m) {
                int w = (g[x + ix[i]][y + iy[i]] != cs[i]);
                int d = dis[x][y] + w;
                if (d < dis[a][b]) {
                    dis[a][b] = d;
                    if (w)      deq.push_back({ a, b });
                    deq.push_front({ a, b });
                }
            }
        }
    }
}
int main() {
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m;
        for (int i = 0; i < n; i++) {
            cin >> g[i];
        }
        if ((n + m) & 1) {
            cout << "NO SOLUTION";
            continue;
        }
        cout << bfs();
    }
}
```

6.DFS-连通性模型

> 迷宫

```
const int N = 105;
int n;
int ha, la, hb, lb;
char g[N][N];
bool flag[N][N];
int dx[4]{ -1, 0, 1, 0 };
int dy[4]{ 0, 1, 0, -1 };

bool dfs(int sx, int sy) {
    if (sx == hb && sy == lb)   return true;
    flag[sx][sy] = true;
    for (int i = 0; i < 4; i++) {
        int x = sx + dx[i];
        int y = sy + dy[i];
```

```
        if (x >= 0 && x < n && y >= 0 && y < n && !flag[x][y] && g[x][y] != '#')
{
            if (dfs(x, y))  return true;
        }
    }
    return false;
}

int main() {
    int k;
    cin >> k;
    while (k--) {
        cin >> n;
        for (int i = 0; i < n; i++) {
            cin >> g[i];
        }
        cin >> ha >> la >> hb >> lb;
        if (g[ha][la] == '#' || g[hb][lb] == '#') {
            cout << "NO" << endl;
            continue;
        }
        memset(flag, false, sizeof(flag));
        if (dfs(ha, la))    cout << "YES";
        else cout << "NO";
    }
}
```

> 红与黑

```
const int N = 25;
int n, m;
char g[N][N];
bool flag[N][N];
int res = 0;
int dx[4]{ -1, 0, 1, 0 };
int dy[4]{ 0, 1, 0, -1 };
int dfs(int sx, int sy) {
    flag[sx][sy] = true;
    res++;
    for (int i = 0; i < 4; i++) {
        int x = sx + dx[i], y = sy + dy[i];
        if (x >= 0 && x < n && y >= 0 && y < m && !flag[x][y] && g[x][y] != '#')
{
            dfs(x, y);
        }
    }
    return res;
}
int main() {
    while (true) {
        cin >> m >> n;
        if (m == 0 && n == 0)       break;
        res = 0;
        memset(flag, false, sizeof(flag));
        int sx, sy;
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                cin >> g[i][j];
                if (g[i][j] == '@') {
                    sx = i, sy = j;
                }
            }
        }
        cout << dfs(sx, sy) << endl;
    }
}
```

7.DFS-搜索顺序

> 马走日

```
const int N = 15;
const int M = N * N;
int n, m, x, y;
bool flag[N][N];
int res = 0;
int cnt = 0;
int dx[]{ -2, -2, 2, 2, -1, -1, 1, 1 };
int dy[]{ -1, 1, 1, -1, -2, 2, 2, -2 };
void dfs(int sx, int sy) {
    flag[sx][sy] = true;
    cnt++;
    if (cnt == n * m)   res++;
    for (int i = 0; i < 8; i++) {
        int x = sx + dx[i], y = sy + dy[i];
        if (x >= 0 && x < n && y >= 0 && y < m && !flag[x][y]) {
            dfs(x, y);
            flag[x][y] = false;
            cnt--;
        }
    }
}
int main() {
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m >> x >> y;
        memset(flag, false, sizeof(flag));
        cnt = 0;
        res = 0;
        dfs(x, y);
        cout << res << endl;
    }
}
```

> 单词接龙

```
const int N = 21;
int n;
```

```
int maxv = 0;
string strs[N];
int g[N][N];
int used[N];
void dfs(int edi, int nwl) {
    maxv = max(maxv, nwl);
    for (int i = 0; i < n; i++) {
        if (g[edi][i] && used[i] < 2) {
            used[i]++;
            dfs(i, nwl + strs[i].size() - g[edi][i]);
            used[i]--;
        }
    }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> strs[i];
    }
    char bg;
    cin >> bg;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            string a = strs[i], b = strs[j];
            for (int k = 1; k < min(a.size(), b.size()); k++) {
                if (a.substr(a.size() - k, k) == b.substr(0, k)) {
                    g[i][j] = k;
                    break;
                }
            }
        }
    }
    for (int i = 0; i < n; i++) {
        if (strs[i][0] == bg) {
            used[i]++;
            dfs(i, strs[i].size());
            used[i]--;
        }
    }
    cout << maxv;
}
```

> 分成互质组

```
// 没学懂 待二刷
int n;
const int N = 10;
int arr[N];
bool flag[N];
int group[N][N];
int ans = n;
//互质是公约数只有1的两个整数，叫做互质整数。
// 欧几里得算法，结果是最大公约数，如果是1，说明互质。
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
```

```
}
// 判定新元素和当前组是否all互质
bool check(int group[], int gc, int i) {
    for (int j = 0; j < gc; j++) {
        if (gcd(arr[group[j]], arr[i]) > 1) {
            return false;
        }
    }
    return true;
}
// 当前组数、当前组的size， 当前有归宿的元素索引+1，当前组从start下标开始搜
void dfs(int g, int gc, int tc, int start) {
    if (g >= ans)   return; //如果当前解法已经比最小结果大了，就没必要继续下去了。
    if (tc == n)    ans = g;
    bool st = true;
    for (int i = start; i < n; i++) {
        if (!flag[i] && check(group[g], gc, i)) {
            flag[i] = true;
            group[g][gc] = i;
            dfs(g, gc + 1, tc + 1, i + 1);
            flag[i] = false;

            st = false;
        }
    }
    if (st) dfs(g + 1, 0, tc, 0);
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    dfs(1, 0, 0, 0);
    cout << ans;
}
```

8.DFS-剪枝与优化

> 优化

- 优化搜索顺序
  - 先搜索分支比较少的节点。（排序）
- 排除等效冗余
  - 组合搜索而不是排列搜索
  - 相同值的搜索
- 可行性剪枝
- 最优性剪枝
- 记忆化搜索（类似DP）

> 按组合数和排列数枚举

- 待做

> 小猫爬山

```
int n, w;
const int N = 20;
int arr[N];
int ans = N;
int sum[N]; //记录第i个车已载多重
//搜索到第cat_n只猫， 搜索到第car_n个车
void dfs(int cat_n, int car_n) {
    if (car_n >= ans)   return; //最优性剪枝
    if (cat_n == n) {
        ans = car_n;
    }
    // 之前已经有猫的车
    for (int i = 0; i < car_n; i++) {
        if (sum[i] + arr[cat_n] <= w) {     //可行性剪枝
            sum[i] += arr[cat_n];
            dfs(cat_n + 1, car_n);
            sum[i] -= arr[cat_n];
        }
    }
    // 当前的空车
    sum[car_n] = arr[cat_n];
    dfs(cat_n + 1, car_n + 1);
    sum[car_n] = 0;
}
int main() {
    cin >> n >> w;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    sort(arr, arr + n);
    reverse(arr, arr + n);      //优化搜索顺序
    //组合搜索 每次进去都是新车
    dfs(0, 0);
    cout << ans;
}
```

> 数独

```
//有亿点点难
const int N = 9, M = 1 << N;
char str[100];
int row[N];
int col[N];
int cell[3][3];
int ones[M], mp[M];
void init() {
    for (int i = 0; i < N; i++) {
        //  row表示第i行 的 1-9的二进制有无 11111111表示都没有,能放
        row[i] = col[i] = (1 << N) - 1;
    }
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cell[i][j] = (1 << N) - 1;
```

```
        }
    }
}
// 在当前x，y位置填或者删一个数，
void draw(int x, int y, int t, bool is_set) {
    if (is_set) {
        str[x * N + y] = '1' + t;
    }
    else {
        str[x * N + y] = '.';
    }
    int v = 1 << t;
    if (!is_set)    v = -v;
    // 0表示有了 不能放，所以如果填数，就减掉当前的1；
    row[x] -= v;
    col[y] -= v;
    cell[x / 3][y / 3] -= v;
}
int lowbit(int x) {
    return x & -x;
}
int get(int x, int y) {
    return row[x] & col[y] & cell[x / 3][y / 3];
}
bool dfs(int cnt) {
    if (!cnt)   return true;
    int minv = 10;
    int x, y;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (str[i * N + j] == '.') {
                int state = get(i, j);
                if (ones[state] < minv) {
                    minv = ones[state];
                    x = i, y = j;
                }
            }
        }
    }
    int state = get(x, y);
    for (int i = state; i; i -= lowbit(i)) {
        int  t = mp[lowbit(i)];
        draw(x, y, t, true);
        if (dfs(cnt - 1))   return true;
        draw(x, y, t, false);
    }
    return false;
}
int main() {
    for (int i = 0; i < N; i++) mp[1 << i] = i;     //mp[i] = log_2_i;
    // 从00000000到11111111，每个数的二进制表示里有多少个1；
    for (int i = 0; i < 1 << N; i++) {
        for (int j = 0; j < N; j++) {
            ones[i] += i >> j & 1;
        }
    }
```

```
    while (true) {
        cin >> str;
        if (strcmp(str, "end") == 0)    break;
        init();
        int cnt = 0;
        for (int i = 0, k = 0; i < N; i++) {
            for (int j = 0; j < N; j++, k++) {
                if (str[k] != '.') {
                    int t = str[k] - '1';
                    draw(i, j, t, true);
                }
                else cnt++;
            }
        }
        dfs(cnt);
        cout << str << endl;
    }
}
```

> 木棒

```
//小猫上车的枚举顺序是，猫递增，枚举上哪个车
//木棒拼接的枚举顺序是，木棒递增，枚举用哪些木棍    不同的原因是，木棒的长度最终是固定的，比较方便
// 也是没懂。。。
const int N = 65;
int arr[N];
int n;
int length = 0;
int sumv = 0;
bool flag[N];
// 当前组数、当前下标
bool dfs(int g, int gl, int start) {
    if (g * length == sumv) return true;
    if (gl == length)   return dfs(g + 1, 0, 0);

    for (int i = start; i < n; i++) {
        if (!flag[i] && gl + arr[i] <= length) {
            flag[i] = true;
            if (dfs(g, gl + arr[i], i + 1))     return true;
            flag[i] = false;
            // 如果方案开头用某长度的木棒失败了，该方案一定失败
            // 另一种搜索，中间用了该长度的木棒可以等价于开头用该长度木棒，两木棒互换，矛盾
            if (!gl)    return false;
            // 如果方案结尾用某长度的木棒失败了，该方案一定失败
            // 另一种搜索，中间用了该长度的木棒可以替换为失败结尾的木棒，矛盾
            if (gl + arr[i] == length)  return false;
            // 如果方案用某长度的木棒失败了，那相同长度的木棒也会造成失败，直接略过。
            int j = i;
            while (j < n && arr[j] == arr[i])   j++;
            i = j - 1;
        }
    }
    return false;
}
```

```
int main() {
    while (cin >> n) {
        if (n == 0)     break;
        memset(flag, false, sizeof(flag));
        length = 0;
        sumv = 0;
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
            sumv += arr[i];
            length = max(length, arr[i]);       //可行性优化
        }
        sort(arr, arr + n); //优化搜索顺序
        reverse(arr, arr + n);
        while (true) {
            if (sumv % length == 0 && dfs(0, 0, 0))
            {
                cout << length << endl;
                break;
            }
            length++;
        }
    }
}
```

> 生日蛋糕

```
// 很显然没懂
const int INF = 1e9;
const int N = 25;
int n, m;
int minv[N], mins[N];
int R[N], H[N];
int ans = INF;
//层数、体积、表面积
void dfs(int u, int v, int s) {
    // 可行性剪枝
    if (v + minv[u] > n)            return;
    // 最优化剪枝
    if (s + mins[u] >= ans)     return;
    // 表面积公式放松，体积公式代入 边界取等
    if (s + 2 * (n - v) / R[u + 1] >= ans)      return;
    if (!u) {
        if (v == n) ans = s;
        return;
    }
    for (int r = min(R[u + 1] - 1, (int)sqrt(n - v)); r >= u; r--) {
        for (int h = min(H[u + 1] - 1, (n - v) / r / r); h >= u; h--) {
            int t = 0;
            if (u == m) t = r * r;
            R[u] = r, H[u] = h;
            dfs(u - 1, v + r * r * h, s + 2 * r * h + t);
        }
    }
}
int main() {
```

```
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        minv[i] = minv[i - 1] + i * i * i;
        mins[i] = mins[i - 1] + i * i * 2;
    }
    R[m + 1] = H[m + 1] = INF;
    dfs(m, 0, 0);
    if (ans == INF)  ans = 0;
    cout << ans << endl;
}
```

9.迭代加深

> 加成序列

```

```

10.双向DFS

> 送礼物

```

```

11.IDA*

> 排书

```

```

> 回转游戏

```

```

12.A*

> 第K短路

```

```

> 八数码

```

```

13.双向广搜

> 字符变换

```

```

二、图

- 无负权边：
  - 朴素版Dijkstra，适合稠密图 O(n^2)
  - 堆优化Dijkstra，适合稀疏图 O(mlogm)
- 负权边：
  - Bellman-Ford O(mn)
  - SPFA，一般用这个 O(m)
- 难点：问题的转化和抽象，转换为图论经典模型。

> 为什么dijkstra算法不能处理负权边

- 因为贪心性质永远选择离源点最近的点先处理，集齐n个点就停了。但，如果有负边，那么先处理离源点远的点，再经过一个负边，可能反倒离源点近了。如果停的早，就出现错误解了。

1.单源最短路

> 热浪

- Dijkstra_heap和SPFA的对比：
  - 代码基本一样。但Dj_heap只能让相同节点但不同dis入队列，选小的用，用自带的dis；SPFA可以不让相同节点入队列，用全局新的dis就行。
  - Dj_heap只更新一次距离（贪心），SPFA可能更新多次。

```
typedef pair<int, int> PII;
const int N = 2510;
const int M = 6200 * 2 + 10;
int n, m, bg, ed;
int h[N], e[M], ne[M], w[M], idx;
int dis[N];
bool flag[N];
void add(int x, int y, int wei) {
    e[idx] = y;
    w[idx] = wei;
    ne[idx] = h[x];
    h[x] = idx++;
}
// 适合稀疏图
int heap_dijkstra() {
    memset(dis, 0x3f, sizeof(dis));
    priority_queue<PII, vector<PII>, greater<PII>> pq;
    pq.push({ 0, bg });
    dis[bg] = 0;
    while(pq.size()) {
        auto ft = pq.top();
        int weight = ft.first;
        int nodei = ft.second;
        pq.pop();
        // 存在nodei多次加入pq的情况，最短的情况肯定先更新了（贪心性质）,就不再更新。
        if (flag[nodei])        continue;
        flag[nodei] = true;
        for (int ii = h[nodei]; ii != -1; ii = ne[ii]) {
```

```
            int dist = weight + w[ii];
            if (dist < dis[e[ii]]) {
                dis[e[ii]] = dist;
                pq.push({ dist, e[ii]});
            }
        }
    }
    if (dis[ed] == 0x3f3f3f3f)      return -1;
    return dis[ed];
}

int main() {
    cin >> n >> m >> bg >> ed;
    memset(h, -1, sizeof(h));
    int x, y, wei;
    for (int i = 0; i < m; i++) {
        cin >> x >> y >> wei;
        add(x, y, wei);
    }
    cout << heap_dijkstra();
}
// SPFA
const int N = 2510;
const int M = 6200 * 2 + 10;
int n, m, bg, ed;
int h[N], e[M], ne[M], w[M], idx;
int dis[N];
bool flag[N];       //重复的点在队列中没有意义
void add(int x, int y, int wei) {
    e[idx] = y;
    w[idx] = wei;
    ne[idx] = h[x];
    h[x] = idx++;
}
int SPFA() {
    memset(dis, 0x3f, sizeof(dis));
    queue<int> que;
    dis[bg] = 0;
    que.push(bg);
    flag[bg] = true;
    while (que.size()) {
        auto ft = que.front();
        que.pop();
        flag[ft] = false;
        for (int ii = h[ft]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            int dist = dis[ft] + w[ii];
            if (dist < dis[j]) {
                dis[j] = dist;
                if (!flag[j]) {
                    que.push(j);
                    flag[j] = true;
                }
            }
        }
    }
```

```
    if (dis[ed] == 0x3f3f3f3f)      return -1;
    return dis[ed];
}
int main() {
    cin >> n >> m >> bg >> ed;
    memset(h, -1, sizeof(h));
    int x, y, wei;
    for (int i = 0; i < m; i++) {
        cin >> x >> y >> wei;
        add(x, y, wei);
    }
    cout << SPFA();
}
```

> 信使

```
//Floyd
const int N = 110;
int n, m;
int dis[N][N];
int floyd() {
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
            }
        }
    }
    int maxv = 0;
    for (int i = 1; i <= n; i++) {
        if (dis[1][i] == 0x3f3f3f3f) {      // dis[x][y] > 0x3f3f3f3f / 2; 如果有负权边，则判不可达需要这样
            return -1;
        }
        maxv = max(maxv, dis[1][i]);
    }
    return maxv;
}
int main() {
    cin >> n >> m;
    memset(dis, 0x3f, sizeof(dis));
    int x, y, wei;
    for (int i = 1; i <= n; i++) {
        dis[i][i] = 0;
    }
    for (int i = 0; i < m; i++) {
        cin >> x >> y >> wei;
        dis[y][x] = dis[x][y] = min(wei, dis[x][y]);
    }
    cout << floyd();
}
```

> 香甜的黄油

```cpp
const int INF = 0x3f3f3f3f;
const int N = 810, M = 3000;    // 这里无向图要注意
int n, m, t;
int cow_num[N];
int h[N], e[M], ne[M], w[M], idx;
int dis[N];
bool flag[N];

void add(int a, int b, int c) {
    e[idx] = b;
    w[idx] = c;
    ne[idx] = h[a];
    h[a] = idx++;
}

int spfa(int bg) {
    memset(dis, 0x3f, sizeof(dis));
    memset(flag, false, sizeof(flag));
    queue<int> que;
    que.push(bg);
    dis[bg] = 0;
    flag[bg] = true;
    while (que.size()) {
        int ft = que.front();
        que.pop();
        flag[ft] = false;
        for (int ii = h[ft]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            int dist = dis[ft] + w[ii];
            if (dist < dis[j]) {
                dis[j] = dist;
                if (!flag[j]) {
                    que.push(j);
                    flag[j] = true;
                }
            }
        }
    }
    int res = 0;
    for (int i = 1; i <= n; i++) {
        if (dis[i] == INF && cow_num[i])    return INF;     // 这里退出的条件一定要细琢磨
        res += dis[i] * cow_num[i];
    }
    return res;
}

int main() {
    memset(h, -1, sizeof(h));
    cin >> t >> n >> m;
    int pos;
    for (int i = 0; i < t; i++) {
        cin >> pos;
        cow_num[pos]++;
    }
    int a, b, c;
```

```
    for (int i = 0; i < m; i++) {
        cin >> a >> b >> c;
        add(a, b, c);
        add(b, a, c);
    }
    int res = INF;
    for (int i = 1; i <= n; i++) {
        res = min(res, spfa(i));
    }
    cout << res;
}
```

> 最小花费

```
void dijkstra() {
    memset(dis, 0, sizeof(dis));
    dis[bg] = 1.0;
    for (int i = 1; i <= n; i++) {
        int t = -1;
        for (int j = 1; j <= n; j++) {
            if (!flag[j] && (t == -1 || dis[j] > dis[t])) {
                t = j;
            }
        }
        flag[t] = true;
        for (int j = 1; j <= n; j++) {
            dis[j] = max(dis[j], dis[t] * g[t][j]);
        }
    }
}
int main() {
    cin >> n >> m;
    int a, b, c;
    memset(g, 0x3f, sizeof(g));
    for (int i = 0; i < n; i++) {
        cin >> a >> b >> c;
        double z = (100.0 - c) / 100;
        g[a][b] = g[b][a] = max(z, g[a][b]);
    }
    cin >> bg >> ed;
    dijkstra();
    printf("%.8lf", 100 / dis[ed]);
}
```

> 最优乘车

```
// 这道题的关键是 重新建立图
// 通过 连接 大巴路径上 的先修后继结点 + 最短路 思想 来完成； 未区分哪条路径。
const int N = 510;
int n, m;
bool g[N][N];
int dis[N];
int stop[N];
```

```
void bfs() {
    memset(dis, -1, sizeof(dis));
    queue<int>  que;
    que.push(1);
    dis[1] = 0;
    while (que.size()) {
        int ft = que.front();
        que.pop();
        for (int i = 1; i <= n; i++) {
            if (g[ft][i] && dis[i] == -1) {
                dis[i] = dis[ft] + 1;
                que.push(i);
            }
        }
    }
}
int main() {
    cin >> m >> n;
    string line;
    getline(cin, line);     // 读换行符
    for (int i = 0; i < m; i++) {
        getline(cin, line);
        stringstream ssin(line);
        int cnt = 0, p;
        while (ssin >> p)   stop[cnt++] = p;
        for (int j = 0; j < cnt; j++) {
            for (int k = j + 1; k < cnt; k++) {
                g[stop[j]][stop[k]] = true;
            }
        }
    }
    bfs();
    if (dis[n] == -1)       cout << "NO";
    else
    {
        cout << max(0, dis[n] - 1);         // 特判起点和终点重合
    }
}
```

> 昂贵的聘礼

```
// 没懂
// 建立虚拟源点，从最远处开始寻找到节点1的最短路
const int INF = 0x3f3f3f3f;
const int N = 110;
int n, m;       // 物品数 等级差限制
int g[N][N];
int level[N], dis[N], flag[N];

int dijkstra(int down, int up) {
    memset(dis, 0x3f, sizeof(dis));
    memset(flag, false, sizeof(flag));
    dis[0] = 0;
    for (int i = 1; i <= n + 1; i++) {
        int t = -1;
```

```
        for (int j = 0; j <= n; j++) {
            if (!flag[j] && (t == -1 || dis[j] < dis[t])) {
                t = j;
            }
        }
        flag[t] = true;
        for (int j = 1; j <= n; j++) {
            if (level[j] >= down && level[j] <= up) {
                dis[j] = min(dis[j], dis[t] + g[t][j]);
            }
        }
    }
    return dis[1];
}
int main() {
    cin >> m >> n;
    memset(g, 0x3f, sizeof(g));
    for (int i = 0; i <= n; i++)        g[i][i] = 0;
    for (int i = 1; i <= n; i++) {
        int price, cnt;
        cin >> price >> level[i] >> cnt;
        g[0][i] = min(price, g[0][i]);
        while (cnt--) {
            int id, cost;
            cin >> id >> cost;
            g[id][i] = min(cost, g[id][i]);
        }
    }
    int res = INF;
    for (int i = level[1] - m; i <= level[1]; i++)      res = min(res,
dijkstra(i, i + m));
    cout << res;
}
```

2.Floyd

> 用途

- 多源最短路
- 传递闭包
- 找总和最小环
- 恰好经过k条边的最短路

> 牛的旅行

```
#define x first
#define y second
using namespace std;
typedef pair<int, int> PII;
const int N = 150;
const double INF = 1e20;
```

```cpp
int n;
PII q[N];               // 点坐标
char g[N][N];       // 邻接矩阵
double d[N][N], maxd[N];        // 两点间坐标距离， 连通图内距离某点最远的点
double get_dis(PII a, PII b) {
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}
void floyd() {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) cin >> q[i].x >> q[i].y;
    for (int i = 0; i < n; i++) cin >> g[i];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i != j) {
                if (g[i][j] == '1') d[i][j] = get_dis(q[i], q[j]);
                else
                {
                    d[i][j] = INF;
                }
            }
        }
    }
    floyd();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (d[i][j] < INF) {
                maxd[i] = max(maxd[i], d[i][j]);
            }
        }
    }
    double res1 = 0;
    for (int i = 0; i < n; i++) res1 = max(res1, maxd[i]);
    double res2 = INF;
    // 暴力枚举 所有可能的路径直径
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (d[i][j] >= INF) {
                res2 = min(res2, maxd[i] + maxd[j] + get_dis(q[i], q[j]));
            }
        }
    }
    printf("%lf\n", max(res1, res2));       // 默认6位
}
```

3.最小生成树

> 最短网络

```
typedef pair<int, int> PII;
const int N = 105;
int n;
int g[N][N];
bool st[N];
int dis[N];

void prim() {
    memset(dis, 0x3f, sizeof(dis));
    priority_queue<PII, vector<PII>, greater<PII>> pq;
    dis[1] = 0;
    pq.push({ 0, 1 });
    while (pq.size()) {
        auto ft = pq.top();
        pq.pop();
        int first = ft.first;
        int second = ft.second;
        if (st[second])     continue;
        st[second] = true;
        for (int i = 1; i <= n; i++) {
            // 对角线为0会使算法失效，算法运行过程中是有可能更新已在集合中的点的dis的，要么边算边加，要么禁止改。
            if (i != second && dis[i] > g[second][i] && !st[i]) {
                dis[i] = g[second][i];
                pq.push({ dis[i], i });
            }
        }
    }
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            cin >> g[i][j];
        }
    }
    prim();
    int res = 0;
    for (int i = 1; i <= n; i++) {
        res += dis[i];
    }
    printf("%d", res);
}
```

> 局域网

```
const int N = 105, M = N * 2;
int p[N];
bool st[M];
```

```cpp
int n, m;

struct Edge {
    int from;
    int to;
    int cost;
    bool operator<(const Edge& other) const {
        return cost < other.cost;
    }
} edges[M];


int find(int x) {
    if (x != p[x])  p[x] = find(p[x]);
    return p[x];
}
void merge(int x, int y) {
    int px = find(x);
    int py = find(y);
    if (px != py) {
        p[px] = py;
    }
}

int kruskal() {
    int ans = 0;
    for (int i = 0; i < m; i++) {
        if (find(edges[i].from) != find(edges[i].to)) {
            merge (edges[i].from, edges[i].to);
            ans += edges[i].cost;
        }
    }
    int all = 0;
    for (int i = 0; i < m; i++) {
        all += edges[i].cost;
    }
    return all - ans;
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        p[i] = i;
    }
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        edges[i] = { a, b, c };
    }
    sort(edges, edges + m);
    cout << kruskal();
}
```

4.最近公共祖先

> 朴素lca

```
int n, m, s;
const int N = 5e5 + 10, M = 5e5 + 10;
int d[N], f[N];     // 节点深度、节点的父节点
int h[N], e[N * 2], ne[N * 2], idx;

void add(int x, int y) {
    e[idx] = y;
    ne[idx] = h[x];
    h[x] = idx++;
}

void dfs(int u, int fa) {
    f[u] = fa;
    d[u] = d[fa] + 1;
    for (int ii = h[u]; ii != -1; ii = ne[ii]) {
        int j = e[ii];
        if(j != fa) dfs(ii, u);
    }
}

int lca(int u, int v) {
    if (d[u] > d[v]) {
        swap(u, v);
    }
    while (d[u] < d[v]) {
        v = f[v];
    }
    while (u != v) {
        u = f[u], v = f[v];
    }
    if (u == -1)    u = s;
    return u;
}

int main() {
    cin >> n >> m >> s;
    memset(h, -1, sizeof(h));
    for (int i = 1; i < n; i++) {
        int x, y;
        cin >> x >> y;
        add(x, y);
        add(y, x);
    }
    dfs(s, -1);
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        cout << lca(a, b) << endl;
```

```
    }
}
```

> 倍增lca

```
const int N = 4e4 + 10, M = N * 2;
int n, root;
int f[N][16], d[N];
int h[N], e[M], ne[M], idx;
int q[N];

void add(int x, int y) {
    e[idx] = y;
    ne[idx] = h[x];
    h[x] = idx++;
}
void bfs(int root) {
    memset(d, 0x3f, sizeof(d));
    d[0] = 0, d[root] = 1;
    int hh = 0, tt = -1;
    q[++tt] = root;
    while (hh <= tt) {
        int t = q[hh++];
        for (int ii = h[t]; ii != -1; ii = ne[ii]) {
            int j = e[ii];
            if (d[j] > d[t] + 1) {
                d[j] = d[t] + 1;
                q[++tt] = j;
                f[j][0] = t;
                for (int k = 1; k <= 15; k++) {
                    f[j][k] = f[f[j][k - 1]][k - 1];
                }
            }
        }
    }
}

int lca(int a, int b) {
    if (d[a] < d[b]) swap(a, b);
    for (int k = 15; k >= 0; k--) {
        if (d[f[a][k]] >= d[b]) {
            a = f[a][k];
        }
    }
    if (a == b) return a;
    for (int k = 15; k >= 0; k--) {
        if (f[a][k] != f[b][k]) {
            a = f[a][k];
            b = f[b][k];
        }
    }
    return f[a][0];
}

int main() {
```

```
    cin >> n;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < n; i++) {
        int x, y;
        cin >> x >> y;
        if (y == -1)    root = x;
        else {
            add(x, y);
            add(y, x);
        }
    }
    bfs(root);
    int m;
    cin >> m;
    while (m--) {
        int a, b;
        cin >> a >> b;
        int p = lca(a, b);
        if (p == a)         cout << 1 << endl;
        else if (p == b)        cout << 2 << endl;
        else
        {
            cout << 0 << endl;
        }
    }
}
```

> tarjan算法

```
```