第零章　编程中常见的 C 和 C++知识以及 IDE 配置

1. C 与 C++的输入输出

- C++设置精度
  保留两位有效数字
  得加入#include <iomanip>
  std::cout << setprecision(2) << std::endl;
  精确到小数点 2 位
  std::cout << setiosflags(ios::fixed) << setprecision(2) << std::endl;
- C++不足填充
  std::cout << setfill('0') << setw(4) << d << std::endl;
- C++关同步
  为了解决使用 cin cout 产生的超时可以在 main()函数中加入
  std::ios::sync_with_stdio(false);
  注意：加入这句 printf、scanf 不能和 cin、cout 混用
- cin、printf 输入是以空格识别结束，如果想要获取整行在 C++中可以使用 getline(cin,***);
  如果想要获取一个字符可以使用 getchar();
- printf 输出%需要 printf("%d %%",ans)

2. 字符串处理常用的一些函数
   - 判断是否是字母 isalpha()
   - 判断是否是数字 isdigit()
   - 判断是否是数字或者字母 isalnum()
   - 判断是否是大写字母 isupper()
   - 判断是否是小写字母 islower()
   - 大小写转换 toupper() , tolower()
   - 字符串逆置 reverse()
   - 字符串转数字 atoi(), atof(), stoi(),
   - 数字转字符串 to_string() itoa()

3. Visual stdio 相关知识（适用 2013+）

- 使用查找替换加快编码速度
  查找快捷键：Crtl + F
  还可以用正则表达式查找替换 $&表示全部匹配项
- 解决 printf、scanf 不能用（三种方法）
  1) 右击项目 - 属性 - 配置属性 - C/C++ - 命令行
     命令行增加:/D_CRT_SECURE_NO_WARNINGS
  2) 右击项目 - 属性 - 配置属性 - C/C++ 预处理
     加入_CRT_SECURE_NO_WARNINGS
  3) 直接在文件中加入
     #define _CRT_SECURE_NO_WARNINGS

第一章 简单题

一、 数理逻辑

1. 权力的游戏

A B C D E F

1 2 1 3 2 1

将上述数字序列按照权值 A < B < C < D < E < F 找出最小值

【核心代码】

```
#include<iostream>
int main(){
	int a[] = {1,2,1,3,2,1};
	int min = sizeof(a)/sizeof(int) - 1;
	for(int i = sizeof(a)/sizeof(int) - 1 ; i >= 0;i --){
		if(a[min] > a[i]) min = i;
	}
	std::cout << min << " " << a[min];
	return 0;
}
```

【PAT】A1012

【思考】如何按权值进行排序？

2. 判断素数(质数)

给定一个数判断是否是素数？

【核心代码】

```
bool isPrime(int a){
	if(a == 1 ){
		return false;
	}else{
		int temp = sqrt(a);
		for(int i = 2; i <= temp; i ++){
			if(a % i == 0) return false;
		}
	}
	return true;
}
```

【PAT】A1015

3. 求最大公约数

【核心代码】

```
class Algorithm {
public:
	int gcd_1(int a, int b) {
		return (b == 0 ? a : gcd_1(b, a%b));
	}
	int gcd_2(int a, int b) {
		while (b != 0) {
```

```
                int r = b;
                b = a % b;
                a = r;
            }
            return a;
        }
        int gcd_3(int a, int b) {
            if(a == 0) return b;
            if (b == 0) return a;
            if (a % 2 == 0 && b % 2 == 0) return 2 * gcd_3(a >> 1, b >> 1);
            else if (a % 2 == 0)    return gcd_3(a >> 1, b);
            else if (b % 2 == 0) return gcd_3(a, b >> 1);
            else return gcd_3(abs(a - b), min(a, b));
        }
};
```

4. 求一个非负序列中第 K 大的数（比如中位数）

【核心代码】

```
#include<algorithm>
class Algorithm {
public:
        /*Find k-th numbers algorithm*/
        int table[100000] = { 0 }, block[316] = { 0 };
        void kth_insert(int n) {
            this->table[n]++;
            this->block[n / 316]++;
        }
        void kth_delete(int n) {
            this->table[n]--;
            this->block[n]--;
        }
        int getKth(int k) {
            int sum = 0, i = 0;
            while (sum + block[i] < k) sum += block[i++];
            i = i * 316;
            while (sum + table[i] < k) sum += table[i++];
            return i;
        }
};
```

5. 进制转换

10 进制转任意进制 B，任意进制 B 转 10 进制

【核心代码】

```
/*10 进制转任意进制 base*/
vector<int> v;
do{
        v.insert (v.begin(),n%base);
```

```
        n /= base;
}while(n!=0);
/*任意进制 base 转 10 进制*/
int sum = 0;
for(int it : v){
        sum = sum * base + it;
}
```

6. 反转素数

【PAT】Reversible Primes

A reversible prime in any number system is a prime whose "reverse" in that number system is also a prime. For example in the decimal system 73 is a reversible prime because its reverse 37 is also a prime. Now given any two positive integers N ($<10^5$) and D ($1<D\leqslant 10$), you are supposed to tell if N is a reversible prime with radix D.

Input Specification:

The input file consists of several test cases. Each case occupies a line which contains two integers N and D. The input is finished by a negative N.

Output Specification:

For each test case, print in one line Yes if N is a reversible prime with radix D, or No if not.

Sample Input:

73 10

23 2

23 10

-2

Sample Output:

Yes

Yes

No

【核心代码】

```
#include<iostream>
#include<vector>
#include<cmath>
using namespace std;
bool isPrime(int x) {
        if (x == 1) return false;
        for (int i = 2; i <= sqrt(x); i++) {
                if (x % i == 0) return false;
        }
        return true;
}
int getReverseNum(int x, int d) {
        vector<int> v;
        do {
                v.push_back(x % d);
                x /= d;
        } while (x != 0);
```

```
    int res = 0;
    for (auto it : v) {
        res = res * d + it;
    }
    return res;
}
int main() {
    int x, d;
    while (scanf("%d %d", &x, &d) == 2) {
        if (isPrime(x) && isPrime(getReverseNum(x, d))) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

7. 分数运算

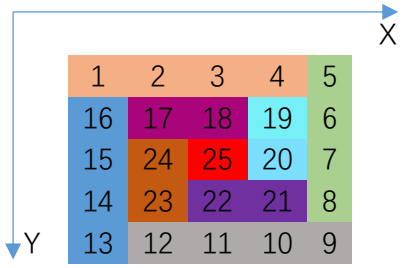
给定两个分数 a1/b1 a2/b2，对它们分别进行四种运算。

【核心代码】

```
struct frac {
    long long down, up;
};
long long gcd(long long a, long long b) {
    return b == 0 ? a : gcd(b, a%b);
}
frac simply(frac a) {
    long long c = gcd(abs(a.up), abs(a.down));
    a.up /= c;
    a.down /= c;
    return a;
}
frac cal(int op, frac a, frac b) {
    frac ans;
    switch (op){
        case '-':
            b.up = -b.up;
        case '+'
            ans.down = a.down * b.down / gcd(a.down, b.down);
            ans.up = a.up * ans.down / a.down + b.up * ans.down / b.down;
            break;
        case '/':
            if (b.up < 0) {
                b.up = - b.up;
                b.down = - b.down;
            }
            swap(b.up, b.down);
        case '*':
            ans.down = a.down * b.down;
            ans.up = a.up * b.up;
```

```
                break;
        }
        ans = simply(ans);
        return ans;
}
```

8. 螺旋矩阵①



按上述方案填充序列并输出。

【核心代码】

➢ 递归版本

```
void SetMatrix(int **matrix, int x, int y, int start, int n) {
        int i, j;
        if (n <= 0) return;
        if (n == 1) {
                matrix[x][y] = start;
                return;
        }
        for (i = x; i < x + n-1; i++)                /* 上部 */
                matrix[y][i] = start++;
        for (j = y; j < y + n-1; j++)                /* 右边 */
                matrix[j][x+n-1] = start++;
        for (i = x+n-1; i > x; i--)                  /* 底部 */
                matrix[y+n-1][i] = start++;
        for (j = y+n-1; j > y; j--)                  /* 左边 */
                matrix[j][x] = start++;
        SetMatrix(matrix, x+1, y+1, start, n-2);     /* 递归 */
}
```

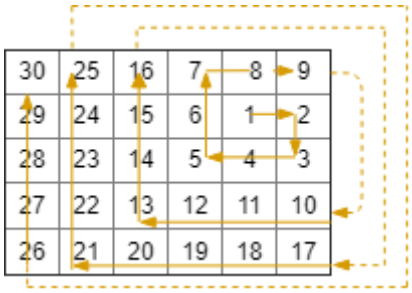
➢ 迭代版本

```
const int d[4][2] = { { 0, 1 },{ 1, 0 },{ 0, -1 },{ -1, 0 } };
class Solution {
public:
        vector<vector<int>> generateMatrix(int n) {
                vector<vector<int>> matrix(n, vector<int>(n, 0));
                int x = -1, y = -1, num = 1, dir = 0;
                for (int i = n - 1; i >= 0; i -= 2) {
                        x += 1; y += 1;
                        if (matrix[x][y] == 0) matrix[x][y] = num++;
                        for (int k = 0; k < 4; k++) {
                                for (int j = 0; j < i; j++) {
                                        x += d[k][0];
                                        y += d[k][1];
```

```
                                if (matrix[x][y] == 0) matrix[x][y] = num++;
                            }
                        }
                    }
                return matrix;
            }
};
```

【思考】如果给定不是方阵呢？

9. 螺旋矩阵②



【核心代码】

```
const int d[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
class Solution {
public:
    bool valid(int x, int y, int n, int m) {
        return x >= 0 && x < n && y >= 0 && y < m;
    }
    vector<vector<int>> spiralMatrixIII(int n, int m, int x, int y) {
        vector<vector<int>> ret = {{x, y}};
        int dir = 0;
        for (int k = 1; ; ++k) {
            for(int j = 0; j < 2; j++){
                for (int i = 0; i < k; ++i) {
                    x += d[dir][0];
                    y += d[dir][1];
                    if (valid(x, y, n, m)) ret.push_back({x, y});
                    if (ret.size() == n * m) return ret;
                }
                dir = (dir + 1) % 4;
            }
        }
    }
};
```

10. 缺失数

11. 丑数

```
只含因子 2，3，5 的数称为丑数，求第 n 个丑数。
2，3，4，5，6，8，9，10，12…
```

【分析】p2 p3 p5 满足*p2 * 2 , *p3 * 3 *p5 * 5 >当前最大的才能有新数添加

12. 三维体的投影面积

【LeetCode】Projection Area of 3D Shapes

On a N * N grid, we place some 1 * 1 * 1 cubes that are axis-aligned with the x, y, and z axes.

Each value v = grid[i][j] represents a tower of v cubes placed on top of grid cell (i, j).

Now we view the projection of these cubes onto the xy, yz, and zx planes.

A projection is like a shadow, that maps our 3 dimensional figure to a 2 dimensional plane.

Here, we are viewing the "shadow" when looking at the cubes from the top, the front, and the side.

Return the total area of all three projections.

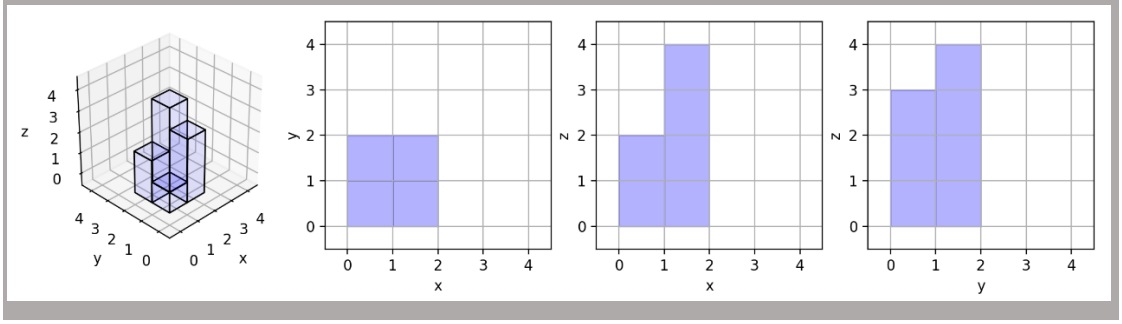
Example :

Input: [[1,2],[3,4]]

Output: 17

Explanation:

Here are the three projections ("shadows") of the shape made with each axis-aligned plane.



【核心代码】

```
class Solution {
public:
    int projectionArea(vector<vector<int>>& grid) {
        int res = 0;
        map<int,int> maxC;
        for(auto it : grid){
            int maxR = 0, cnt = 0;
            for(auto that : it){
                if(that > 0) res++;
                if(that > maxR) maxR = that;
                if(maxC.find(cnt) == maxC.end() || (maxC.find(cnt) != maxC.end() && maxC[cnt] < that))
                    maxC[cnt] = that;
                cnt++;
            }
            res += maxR;
        }
        for(auto it : maxC){
            res += it.second;
        }
        return res;
    }
};
```

13. 三维体的表面积

【LeetCode】Surface Area of 3D Shapes

On a N * N grid, we place some 1 * 1 * 1 cubes.

Each value v = grid[i][j] represents a tower of v cubes placed on top of grid cell (i, j).

Return the total surface area of the resulting shapes.

Example :

Input: [[2,2,2],[2,1,2],[2,2,2]]

Output: 46

【核心代码】

```
class Solution {
public:
    int surfaceArea(vector<vector<int>>& grid) {
        int dir[][2] = {{0,1},{1,0},{0,-1},{-1,0}};
        int n = grid.size();
        int res = 0;
        for(int i = 0 ; i < n; i++){
            for(int j = 0; j < n; j++){
                if(grid[i][j] > 0){
                    res += 2;
                    for(int k = 0; k < 4; k++){
                        int x = i + dir[k][0], y = j + dir[k][1], v = 0;
                        if(x >= 0 && x < n && y >= 0 && y < n){
                            v = grid[x][y];
                        }
                        res += max(grid[i][j] - v,0);
                    }
                }
            }
        }
        return res;
    }
};
```

14. 最小好进制

【LeetCode】Smallest Good Base

For an integer n, we call k>=2 a good base of n, if all digits of n base k are 1.

Now given a string representing n, you should return the smallest good base of n in string format.

Example :

Input: "4681"

Output: "8"

Explanation: 4681 base 8 is 11111.

Note:

The range of n is [3, 10^18].

The string representing n is always valid and will not have leading zeros.

【核心代码】

```
using ull = unsigned long long;
class Solution {
public:
    string smallestGoodBase(string str) {
        ull n = stoll(str);
        for (int m = log2(n); m >= 2; m--) {
            int b = pow (n, 1.0/m);
            ull sum = 1;
            ull prod = 1;
            for (int i = 0; i < m; i++) {
                sum += (prod *= b);
            }
            if ( sum == n ) return to_string(b);
        }
        return to_string(n-1);
    }
};
```

二、 字符串处理

15. 加法模拟

【核心代码】

```
string add(const string &a, const string &b) {
    int i = a.length() - 1, j = b.length() - 1, carry = 0;
    string res;
    while (i >= 0 || j >= 0 || carry == 1) {
        char temp = ((i >= 0 ? a[i] : '0') + (j >= 0 ? b[j] : '0') - '0' + carry);
        carry = temp / ('9'+1);
        temp = temp > '9' ? temp – 10 : temp;
        res = temp + res;
        i--;
        j--;
    }
    return res;
}
```

【思考】减法？乘法？除法？开根号？

16. 乘法模拟

【核心代码】

```
//本题是经典的高精度乘法，可以直接模拟竖式乘法计算。
//乘积的最大长度为两个乘数的长度之和。
class Solution {
public:
    string multiply(string num1, string num2) {
        int m = num1.size();
        int n = num2.size();
        vector<int> a(m, 0), b(n, 0), c(m+n, 0);
        for(int i = 0 ; i < m ; i++){
            a[m-1-i] = num1[i] - '0';
        }
        for (int i = 0 ; i < n; i++){
            b[n-1-i] = num2[i] - '0';
        }
        int i, j;
        for (i = 0; i < m; i ++){
            for (j = 0; j < n ;j++){
                c[i+j] += a[i] * b[j];
                c[i+j+1] += c[i+j] / 10;
                c[i+j] = c[i+j] %10;
            }
        }
        string result = "";
        int l = m + n - 1;
        while (l > 0 && c[l] == 0)
            l--;
```

```
        for (int i = l; i >= 0 ; i-- ){
            result += c[i] + '0';
        }
        return result;
    }
};
```

17. 中缀表达式计算求值

给定一个包含整数 () + - * \ 的中缀表达式，求其计算结果。

Operation	#	(	* /	+ -	)
In Stack	0	1	5	3	6
Out	0	6	4	2	1

【核心代码】

18. 科学记数法

给定一个数以及精度，输出其科学记数法表达形式。或者给定科学记数法，输出原始数。

【核心代码】

```
string deal(string s,int &e,int n) {
    int k = 0;
    while (s[0] == '0') s.erase(s.begin());
    if (s[0] == '.') {
        s.erase(s.begin());
        while (s[0] == '0') {
            s.erase(s.begin());
            e--;
        }
    }
    else {
        while (k < s.length() && s[k] != '.') {
            k++;
            e++;
        }
        if (k < s.length()) s.erase(s.begin() + k);
    }
    if (s.length() == 0) e = 0;
    k = 0;
    string res = "";
    while (k < n) {
        if (k < s.length()) res += s[k];
        else res += '0';
        k++;
    }
    return res;
}
```

19. 数字分割

将 123456789 按照每三位用逗号分隔即 123,456,789

【分析】利用正则表达式进行搜索，正则表达式为 (\d)(?=(\d{3})+(?!\d)) 。

【提示】在 c++中正则表达式运用需要用到库函数 regex ，相关函数有 regex_search()、regex_replace()等，本题用到的函数是 regex_replace()。注意首先要将数字转成字符串。

【核心代码】

```
string s = to_string(num);
regex pattern("(\\d)(?=(\\d{3})+(?!\\d))");
cout << regex_replace(s,pattern,"$1,");
```

【思考】数字转字符串方法？字符串转数字的方法?

【PAT】A1001

20. 公共后缀（前缀）

在给定的 n 个字符串中寻找 n 个字符串的公共后缀（前缀）。

Example:

Input:

I am singing~

Waiting~

With stone falling~

Output:

ing~

【核心代码】

```
string findCommonSuffix(vector<string> str) {
        string ans;
        for (int i = 0; i < str.size(); i++) {
                string s = str[i];
                reverse(s.begin(), s.end());
                if (i == 0) ans = s;
                for (int j = 0; j < ans.length(); j++) {
                        if (ans[j] != s[j]) {
                                ans = s.substr(0, j);
                        }
                }
        }
        reverse(ans.begin(), ans.end());
        return ans;
}
```

21. 是否是合法数字？

输入 n 个数字，判断是否合法？

【核心代码】

➢ 方法一：常规字符串处理

```
for(int i = 0; i < n; i++) {
        scanf("%s", a);
        sscanf(a, "%lf", &temp);
        sprintf(b, "%lf",temp);
        int flag = 0;
        for(int j = 0; j < strlen(a); j++) {
                if(a[j] != b[j]) {
                        flag = 1;
```

```
            }
        }
        if(flag) {
            cout << flag << endl;
        }
}
```

- 方法二：正则表达式

```
bool isValid(string temp,double &ans) {
        regex re("(-?\\d*)(\\.\\d*)?");
        if (regex_match(temp, re)) {
            ans = stod(temp);
            return true;
        }
        return false;
}
```

22. 是否是子串

【LeetCode】Is Subsequence

Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t. t is potentially a very long (length ~= 500,000) string, and s is a short string (<=100).

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:

s = "abc", t = "ahbgdc"

Return true.

Example 2:

s = "axc", t = "ahbgdc"

Return false.

【核心代码】

```
class Solution {
public:
    bool isSubsequence(string s, string t) {
        int pos = 0;
        for(auto it : s){
            bool flag = false;
            for(int i = pos; i < t.length(); i++){
                if(it == t[i]){
                    pos = i + 1;
                    flag = true;
                    break;
                }
            }
            if(flag == false) return false;
        }
```

```
            return true;
        }
};
```

23. 重复的字符

给定一个字符串，要求找出其中不合法的字符，其中不合法字符的规定是该字符总是以连续 K 个出现，同时将 K 个以一个替代，输出最后的结果。字符总是以[a-z0-9_]出现。

【核心算法】

```
    string s;
    int n;
    cin >> n >> s;
    string ans = s;
    smatch sm;
    regex p("(\\w)\\1{" + to_string(n - 1) + "}");// n 个重复 character 的正则
    map<string, bool> rep;
    while (regex_search(s, sm, p)) {
        string stemp = ans;
        string temp = sm.str().substr(0, 1);//得到重复项
        string par = "(" + temp + ")\\1{" + to_string(n - 1) + "}"; // n 个 temp 的正则
        stemp = regex_replace(stemp, regex(par), "");//把所有重复 n 个的去除
        //若去除后依然搜索不到 temp 存在，表明是
        if (!regex_search(stemp, regex(temp)) && rep.find(temp) == rep.end()) {
            rep[temp] = true;//防止已输出的再次输出  Example:3 eeeeeeeee
            cout << temp;
            ans = regex_replace(ans,regex(par), temp);//n 个替换成 1 个
        }
        s = sm.suffix().str();
    }
    cout << endl << ans;
```

24. 重复的子串

【LeetCode】Repeated Substring Pattern

Given a non-empty string check if it can be constructed by taking a substring of it and appending multiple copies of the substring together. You may assume the given string consists of lowercase English letters only and its length will not exceed 10000.

【核心算法】

```
class Solution {
public:
    bool repeatedSubstringPattern(string s) {
        int k = s.length() / 2;
        for(int i = 1; i <= k; i++){
            bool tag = true;
            for(int j = 0; j + i < s.length(); j += i){
                if(s.substr(j,i) != s.substr(j + i,i)) {tag = false; break;}
            }
            if(tag) return true;
```

```
            }
            return false;
        }
};
```

25. ZigZag Conversion

【LeetCode 改编】ZigZag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N

A P L S I I G

Y   I   R
```

Now, given a string and a number, you are supposed to output the zigzag pattern.

【核心代码】

```
vector<string> convert(string s, int numRows) {
        vector<string> v(numRows);
        if (numRows == 1) { v[0] = s; return v; }
        int k = numRows * 2 - 2;
        int n = s.length();
        int m = n % k == 0 ? n / k : (n / k + 1);
        for (int i = 0; i < m; i++) {
                int cnt = 2;
                for (int j = 0; j < k && (i * k + j) < n; j++) {
                        if (j < numRows) {
                                v[j] += s[i * k + j];
                        }
                        else {
                                v[j - cnt] += s[i * k + j];
                                cnt += 2;
                        }

                }

        }
        return v;
}
void myprint(vector<string> v, int numRows) {
        int k = numRows - 2 < 0 ? 0 : numRows - 2;
        for (int i = 0; i < numRows; i++) {
                if (i == 0 || i == numRows - 1) {
                        for (auto it : v[i]) {
```

```
                        cout << " " << it << " ";
                        for (int j = 0; j < k; j++) {
                                cout << "      ";
                        }
                }
        }
        else {
                for (int j = 0; j < v[i].length(); j++) {
                        if (j % 2 != 0) {//每行斜线部分
                                for (int l = 0; l < k - i; l++) {
                                        cout << "      ";
                                }
                                cout << " " << v[i][j] << " ";
                                for (int r = 0; r < i - 1; r++) {
                                        cout << "      ";
                                }
                        }
                        else {//每行垂直部分
                                cout << " " << v[i][j] << " ";
                        }
                }
        }
        cout << endl;
    }
}
```

26. 罗马数字与整数的互换

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	**Value**
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

Example 1:

Input: "III"

Output: 3

Input: 58

Output: "LVIII"

Explanation: C = 100, L = 50, XXX = 30 and III = 3.

【核心代码】

```
string intToRoman(int num) {
        string res = "";
        map<int, string> m;
        m[0] = "", m[1] = "I", m[5] = "V", m[10] = "X", m[50] = "L", m[100] = "C", m[500] = "D", m[1000] = "M";
        m[4] = "IV", m[9] = "IX", m[40] = "XL", m[90] = "XC", m[400] = "CD", m[900] = "CM";
        for (int i = 1; i < 10000; i *= 10) {
                string t = "";
                int temp = (num % 10) * i;
                num /= 10;
                while (temp != 0) {
                        if (m.find(temp) != m.end()) { t = m[temp] + t;    temp = 0; }
                        else { t += m[i];    temp -= i; }

                }
                res = t + res;
        }
        return res;
}
int romanToInt(string s) {
        int res = 0;
        map<string, int> m;
        m["I"] = 1, m["V"] = 5, m["X"] = 10, m["L"] = 50, m["C"] = 100, m["D"] = 500, m["M"] = 1000;
        for (int i = 0; i < s.length() - 1; i++) {
                if (m[s.substr(i, 1)] >= m[s.substr(i + 1, 1)]) res += m[s.substr(i, 1)];
                else res -= m[s.substr(i, 1)];
        }
        res += m[s.substr(s.length() - 1, 1)];
        return res;
}
```

27. Decoded String at Index

【LeetCode】Decoded String at Index

An encoded string S is given. To find and write the decoded string to a tape, the encoded string is read one character at a time and the following steps are taken:

If the character read is a letter, that letter is written onto the tape.

If the character read is a digit (say d), the entire current tape is repeatedly written d-1 more times in total.

Now for some encoded string S, and an index K, find and return the K-th letter (1 indexed) in the decoded string.

Example 1:

Input: S = "leet2code3", K = 10

Output: "o"

Explanation:

The decoded string is "leetleetcodeleetleetcodeleetleetcode".

The 10th letter in the string is "o".

【分析】很自然的想法求出整个串，这样肯定不行，内存和时间都有可能超限。观察到有很多重复项和无关项，所以去除重复项和无关项，可以减小复杂度。怎样去除重复项和无关项？很显然，我们得从尾部入手。为了能够正确去除重复项和无关项，我们必须求出整个串的长度。当前长度大于给定 index 时，继续去除重复项和无关项。当前长度小于给定 index 时，说明所求的是重复的后面部分。**例如 S = "leet2code3", K = 7，去除重复项后，S = "leet2code", K = 7；去除无关项后，S = "leet2", K = 7；去除重复项 S = "leet", K = 3，去除无关项后，S = "lee", K = 3。**

【核心代码】

```
class Solution {
public:
    string decodeAtIndex(string S, int K) {
        long size = 0;
        int N = S.size();
        // Find size = length of decoded string
        for (int i = 0; i < N; ++i) {
            if (isdigit(S[i])) size *= S[i] - '0';
            else size++;
        }
        for (int i = N-1; i >=0; --i) {
            K %= size;
            if (K == 0 && isalpha(S[i]))
                return (string) "" + S[i];
            if (isdigit(S[i])) size /= S[i] - '0';
            else size--;
        }
    }
};
```

28. Orderly Queue

【LeetCode】Orderly Queue

A string S of lowercase letters is given. Then, we may make any number of moves.

In each move, we choose one of the first K letters (starting from the left), remove it, and place it at the end of the string.

Return the lexicographically smallest string we could have after any number of moves.

Example 1:

Input: S = "cba", K = 1

Output: "acb"

Explanation:

In the first move, we move the 1st character ("c") to the end, obtaining the string "bac".

In the second move, we move the 1st character ("b") to the end, obtaining the final result "acb".

Example 2:

Input: S = "baaca", K = 3

Output: "aaabc"

Explanation:

In the first move, we move the 1st character ("b") to the end, obtaining the string "aacab".

In the second move, we move the 3rd character ("c") to the end, obtaining the final result "aaabc".

【分析】经过观察发现，当 K > 1 时，结果应当为字符串的排序序列；当 K = 1 时，是受限的有序序列。

【核心代码】

```
class Solution {
public:
    string orderlyQueue(string S, int K) {
        if( K > 1){
            sort(S.begin(),S.end());
            return S;
        }else{
            int len = S.length();
            string ans = S;
            S += S;
            for(int i = 0 ; i < len; i++){
                string temp = S.substr(i,len);
                ans = min(ans,temp);
            }
            return ans;
        }
    }
};
```

29.

第二章 排序题和模拟题

一、 排序题

- 简单题

一般都是静态排序，在输入结束后进行一次排序即可得到输出结果，或者是排序后经过筛选得到输出结果。

- 中等题

一般都是动态排序，每次输入都会改变某些输入的相关状态,从而导致顺序的变化以及某些状态的无效。

【PAT 原题】Recommendation System

Recommendation system predicts the preference that a user would give to an item. Now you are asked to program a very simple recommendation system that rates the user's preference by the number of times that an item has been accessed by this user.

Input Specification:

Each input file contains one test case. For each test case, the first line contains two positive integers: N (≤ 50,000), the total number of queries, and K (≤ 10), the maximum number of recommendations the system must show to the user. Then given in the second line are the indices of items that the user is accessing -- for the sake of simplicity, all the items are indexed from 1 to N. All the numbers in a line are separated by a space.

Output Specification:

For each case, process the queries one by one. Output the recommendations for each query in a line in the format:

query: rec[1] rec[2] ... rec[K]

where query is the item that the user is accessing, and rec[i] (i=1, ... K) is the i-th item that the system recommends to the user. The first K items that have been accessed most frequently are supposed to be recommended in non-increasing order of their frequencies. If there is a tie, the items will be ordered by their indices in increasing order. Note: there is no output for the first item since it is impossible to give any recommendation at the time. It is guaranteed to have the output for at least one query.

Sample Input:

12 3

3 5 7 5 5 3 2 1 8 3 8 12

Sample Output:

5: 3

7: 3 5

5: 3 5 7

5: 5 3 7

3: 5 3 7

2: 5 3 7

1: 5 3 2

8: 5 3 1

3: 5 3 1

8: 3 5 1

12: 3 5 8

二、 模拟题

1. 排队问题

【PAT 改编】Having meals in resturant

There is a famous resturant in the street, so many people would like to have meals. However, the resturant doesn't have enough tables to provide for all customers. If customers want to have meals at the resturtans, they will wait in a queue. It is assumed that every customers can have at most 2 hours. Addtionally, the resturant provide the VIP service. There are several tables for VIP customers. When the VIP tables is available, the VIP customers have the priviledge to take it. If there is no VIP customers, everyone can take it. But when the VIP tables is not available, they will be considered as any ordinary customers.

Given the numbers of tables N(index 1 to N), the numbers of customers M.The next line follows the index of VIP tables. Then M lines follow, the customers' arriving time, the customers' eating time, the customers' cost, the VIP tag of customers(0 is not VIP, 1 is VIP). It is guaranteed that the arriving time is between 08:00:00 and 23:00:00 while the resturant is open. It is assumed that no two customers arrives at the same time.

Now, first you should ouput customers' serving time, waiting time, cost, table index. Then you should output the total income of every table. Notice that the output must be listed in chronological order of the serving time. If the serving time is the same, the output must be listed in non-decreasing order of the cost . If the cost is equal, the output must be listed in non-decreasing order of the waiting time.If all the same,the ouput should listed in non-decreasing order of the table index. The waiting time must be rounded up to an integer minute(s). If one cannot get a table before the closing time, their information must NOT be printed.

Example:

Input: 4 9

1

20:52:00 10 100 0

08:00:00 20 80 0

08:02:00 30 80 0

20:51:00 10 80 0

08:10:00 5 70 0

08:12:00 10 100 1

20:50:00 10 80 0

08:01:00 15 80 1

20:53:00 10 20 1

Output:

2. 栈模拟

【PAT 改编】Pop sequence

Given a sequence 1 2 … n and a stack, using push and pop , we will get a new sequence. However, some sequences cannot get from the stack by using push and pop.Your task is to verify that a new sequence is valid or not. We assume that the capacity of stack is enough.

Example:

Input: 7

1 2 3 4 5 6 7

2 1 3 4 5 6 7

7 4 3 2 5 1 6

Output: True

True

False

【核心代码】

```
vector<boo> isValidPopSequence(int n, vector< vector<int> > nums){
    vector<bool> res;
    for(auto &v : nums){
        stack<int> s;
        int current = 0;
        for(int i = 1; i <= n; i++){
            s.push(i);
            while(!s.empty() && s.top() == v[current]) {
                s.pop();
                current++;
            }
        }
        if(current == n) res.push_back(true);
        else res.push_back(false);
    }
    return res;
}
```

3. 行走的机器人

【LeetCode 原题】Walking robot simulation

The robot on the infinite grid starts at point (0, 0) and faces north. The robot can receive the following three types of commands:

- `-2`: turn left 90 degrees
- `-1`: turn 90 degrees to the right
- `1 <= x <= 9`: move forward by `x` unit length

There are some grid squares that are considered obstacles.

The `i` first obstacle is at the grid point `(obstacles[i][0], obstacles[i][1])`

If the robot tries to get over the obstacle, it will stay on the previous grid square of the obstacle, but still continue the rest of the route.

Returns the **square** of the largest Euclidean distance from the origin to the robot .

Example 1:

```
Input: commands = [4,-1,3], obstacles = []
Output: 25
Explanation: The robot will arrive (3, 4)
```

Example 2:

```
Input: commands = [4,-1,4,-2,4], obstacles = [[2,4]]
Output: 65
Explanation : The robot will be trapped before going left (1, 8) (1 , 4)
```

Note:

1. `0 <= commands.length <= 10000`
2. `0 <= obstacles.length <= 10000`
3. `-30000 <= obstacle[i][0] <= 30000`
4. `-30000 <= obstacle[i][1] <= 30000`
5. The answer is guaranteed to be less than `2 ^ 31`

【核心代码】

```
class Solution {
public:
    int robotSim(vector<int>& commands, vector<vector<int>>& obstacles) {
        int x = 0, y    = 0, dir[4][2] = {{0,1},{1,0},{0,-1},{-1,0}}, direction = 0, res = 0;
        map<int,map<int,int>> m;
        for(int i = 0; i < obstacles.size(); i ++){
            m[obstacles[i][0]][obstacles[i][1]] = 1;
        }
        for(int i = 0 ; i < commands.size(); i++){
            if(commands[i] == -1){
                direction = (direction + 1) % 4;
            }else if(commands[i] == -2){
                direction = ( direction + 3 ) % 4;
            }else{
                for(int j = 1; j <= commands[i]; j++ ){
                    int nx = x + dir[direction][0];
                    int ny = y + dir[direction][1];
                    if(m[nx][ny] == 1) break;
                    x = nx;
                    y = ny;
                }
            }
            res = max(res, x*x+y*y);
        }
        return res;
    }
};
```

4.

第三章 常用的算法思想

一、 二分查找

【LeetCode】Binary Search

Given a **sorted** (in ascending order) integer array nums of n elements and a target value, write a function to search targetin nums. If target exists, then return its index, otherwise return -1.

【LeetCode】 Find Smallest Letter Greater Than Target

Given a list of sorted characters letters containing only lowercase letters, and given a target letter target, find the smallest element in the list that is larger than the given target.

Letters also wrap around. For example, if the target is target = 'z' and letters = ['a', 'b'], the answer is 'a'.

【LeetCode】Search a 2D Matrix

Write an efficient algorithm that searches for a value in an *m* x *n* matrix. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

【LeetCode】Sqrt(x)

Implement int sqrt(int x).

Compute and return the square root of *x*, where *x* is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

【LeetCode 原题】Koko Eating Bananas

Koko loves to eat bananas. There are N piles of bananas, the i-th pile has piles[i] bananas. The guards have gone and will come back in H hours.

Koko can decide her bananas-per-hour eating speed of K. Each hour, she chooses some pile of bananas, and eats K bananas from that pile. If the pile has less than K bananas, she eats all of them instead, and won't eat any more bananas during this hour.

Koko likes to eat slowly, but still wants to finish eating all the bananas before the guards come back.

Return the minimum integer K such that she can eat all the bananas within H hours.

Example 1:

```
Input: piles = [3,6,7,11], H = 8
Output: 4
```

Example 2:

```
Input: piles = [30,11,23,4,20], H = 5
Output: 30
```

Example 3:

```
Input: piles = [30,11,23,4,20], H = 6
Output: 23
```

Note:

- 1 <= piles.length <= 10^4
- piles.length <= H <= 10^9
- 1 <= piles[i] <= 10^9

【核心代码】

```
class Solution {
public:
    int minEatingSpeed(vector<int>& piles, int H) {
        int num = piles.size();
        int low = 1, high = *max_element(piles.begin(),piles.end());
        while(low < high){
            int mid = (low + high) / 2;
            int sum = 0;
            for(auto &it : piles){
                sum += ((it - 1) / mid + 1);
            }
            if(sum > H) low = mid + 1;
            else high = mid;
        }
        return high;
    }
};
```

【PAT 改编】Target sum

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. You are asked to find the contiguous sublist which has target sum S and return the sublists. If there is no solution,you should output the sublist sum is more than S but minimized.

Example1:

Input: [3, 2, 1, 5, 4, 6, 8, 7, 16, 10, 15, 11, 9, 12, 14, 13], 15

Output: 1-5
4-6
7-8
11-11

Explanation: with values 3+2+1+5+4=15, return index 1 to 5;etc;

Example2:

Input: [2 4 5 7 9], 13

Output: 2-4
4-5

Explanation: There is no solution for 13, and 16 is more than 13 but minimized. with values 4 + 5 + 7 = 16, return index 2 to 4;etc;

【LeetCode 原题】Minimum Size Subarray Sum

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum ≥ **s**. If there isn't one, return 0 instead.

Example:

Input: s = 7, nums = [2,3,1,2,4,3]

Output: 2

Explanation: the subarray [4,3] has the minimal length under the problem constraint.

Follow up:

If you have figured out the *O*(*n*) solution, try coding another solution of which the time complexity is *O*(*n* log *n*)

【核心代码】

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
        int min = INT_MAX;
        int num = nums.size();
        nums.insert(nums.begin(),0);
        for(int i = 1; i<= num ; i++){
            nums[i] += nums[i-1];
        }
        for(int i = 1; i<= num ; i++){
            int j = lower_bound(nums.begin() + i, nums.end(),nums[i-1] + s)-nums.begin();
            if( j <= num && j - i + 1 < min) min = j - i + 1;
        }
        return (min == INT_MAX ? 0 : min);
    }
};
```

【LeetCode 原题】Nth Magical Number

A positive integer is *magical* if it is divisible by either A or B.

Return the N-th magical number. Since the answer may be very large, **return it modulo** 10^9 + 7.

Example 1:

Input: N = 1, A = 2, B = 3

Output: 2

Example 2:

Input: N = 4, A = 2, B = 3

Output: 6

Example 3:

Input: N = 5, A = 2, B = 4

Output: 10

Example 4:

Input: N = 3, A = 6, B = 4

Output: 8

Note:

1. 1 <= N <= 10^9
2. 2 <= A <= 40000
3. 2 <= B <= 40000

【分析】

➢ 方法一：

任意给定一个数返回比它小最接近的神奇数的位置？ K = num / A + num / B - num / gcd(A,B)

然后通过二分查找找到 the N-th magical number。

➢ 方法二：

注意到这个序列以 K 个一组，观察其规律。

2 3 4 6

8 9 10 12

14 15 16 18

【PAT】Radix

Given a pair of positive integers, for example, 6 and 110, can this equation 6 = 110 be true? The answer is yes, if 6 is a decimal number and 110 is a binary number.

Now for any pair of positive integers *N*1 and *N*2, your task is to find the radix of one number while that of the other is given.

Input Specification:

Each input file contains one test case. Each case occupies a line which contains 4 positive integers:

N1 N2 tag radix

Here N1 and N2 each has no more than 10 digits. A digit is less than its radix and is chosen from the set { 0-9, a-z } where 0-9 represent the decimal numbers 0-9, and a-z represent the decimal numbers 10-35. The last number radix is the radix of N1 if tag is 1, or of N2 if tag is 2.

Output Specification:

For each test case, print in one line the radix of the other number so that the equation N1 = N2 is true. If the equation is impossible, print Impossible. If the solution is not unique, output the smallest possible radix.

Sample Input 1:

6 110 1 10

Sample Output 1:

2

Sample Input 2:

1 ab 1 2

Sample Output 2:

Impossible

【分析】要想找到一个数的进制与另一数相等，必定要进行搜索。首先将进制已知的数转换成十进制，然后确定另一数的进制取值范围，进行二分查找。最小值的确定，可以通过遍历每一位数值得到最大的就是所求；最大值的确定，当另一数为“10”时，此时进制应最大。

【核心代码】

```
#include<iostream>
#include<cctype>
#include<algorithm>
#include<string>
using namespace std;
long long TransferToNum(string d, int radix) {
	long long ans = 0;
```

```
        for (int i = 0; i < d.length(); i++) {
                ans = ans * radix + ( isdigit(d[i]) ? d[i] - '0' : ( d[i] - 'a' + 10 ) );
        }
        return ans;
}
int getMinRadix(string d) {
        char ans = *max_element(d.begin(), d.end());
        return ( isdigit(ans) ? ans - '0' : ans - 'a' + 10 ) + 1;
}
int main() {
        string d[2];
        int tag, radix;
        cin >> d[0] >> d[1] >> tag >> radix;
        long long low, high, mid, d1;
        d1 = TransferToNum(d[tag == 1 ? 0 : 1], radix);
        low = getMinRadix(d[tag != 1 ? 0 : 1]);
        high = max(d1, low) + 1;
        bool flag = false;
        while (low < high) {
                mid = (low + high) >> 1;
                long long ans = TransferToNum(d[tag != 1 ? 0 : 1], mid);
                if (ans > d1 || ans < 0) {
                        high = mid;
                }
                else if (ans == d1) {
                        flag = true;
                        break;
                }
                else {
                        low = mid + 1;
                }
        }
        if (flag) printf("%lld", mid);
        else printf("Impossible");
        return 0;
}
```

二、 贪心

【LeetCode】Boats to Save People

The i-th person has weight people[i], and each boat can carry a maximum weight of limit.

Each boat carries at most 2 people at the same time, provided the sum of the weight of those people is at most limit.

Return the minimum number of boats to carry every given person. (It is guaranteed each person can be carried by a boat.)

Example 1:

Input: people = [1,2], limit = 3

Output: 1

Explanation: 1 boat (1, 2)

Example 2:

Input: people = [3,2,2,1], limit = 3

Output: 3

Explanation: 3 boats (1, 2), (2) and (3)

Example 3:

Input: people = [3,5,3,4], limit = 5

Output: 4

Explanation: 4 boats (3), (3), (4), (5)

Note:

1 <= people.length <= 50000

1 <= people[i] <= limit <= 30000

【核心算法】

```
class Solution {
public:
    int numRescueBoats(vector<int>& people, int limit) {
        int n = people.size(), cnt = 0;
        sort(people.begin(),people.end());
        int j =  0;
        for(int i = n -1; i >= 0 && i >= j; i--){
            if(people[i] + people[j] <= limit){
                j++;
            }
            cnt++;
        }
        return cnt;
    }
};
```

【LeetCode】Jump Game

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

Example 1:

Input: [2,3,1,1,4]

Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: [3,2,1,0,4]

Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximumjump length is 0, which makes it impossible to reach the last index.

【核心代码】

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int i = 0;
        while(i < nums.size()){
            bool isGreat = false;//是否能够在当前跳跃范围内跳得更远
            for(int j = 1; j <= nums[i]; j++ ){
                if(i + j < nums.size() && nums[i+j] > nums[i] - j){
                    i += j;
                    isGreat = true;
                    break;
                }
            }
            if(isGreat == false){
                i += nums[i];
                //如果当前到达的地方无法跳跃并且还未到终点，则返回 fasle
                if(i < nums.size() - 1 && nums[i] == 0) return false;
            }
            if( i >= nums.size() - 1) return true;//如果跳过终点或者跳到终点，则返回 true
        }
    }
};
```

【LeetCode】Monotone Increasing Digits

Given a non-negative integer N, find the largest number that is less than or equal to N with monotone increasing digits.(Recall that an integer has monotone increasing digits if and only if each pair of adjacent digits x and y satisfy x <= y.)

Example 1:

Input: N = 10

Output: 9

Example 2:

Input: N = 1234

Output: 1234

Example 3:

Input: N = 332

Output: 299

Note: N is an integer in the range [0, 10^9].

【核心代码】

```
class Solution {
public:
    int monotoneIncreasingDigits(int N) {
        vector<int> v;
        do{
            v.push_back(N % 10);
            N /= 10;
        }while(N != 0);
        for(int i = 1; i < v.size(); i++){
            if(v[i] > v[i-1]){
                for(int j = i - 1; j >= 0; j--) v[j] = 9;
                v[i]--;
            }
        }
        int res = 0, base = 1;
        for(auto &it : v){
            res += base*it;
            base *= 10;
        }
        return res;
    }
};
```

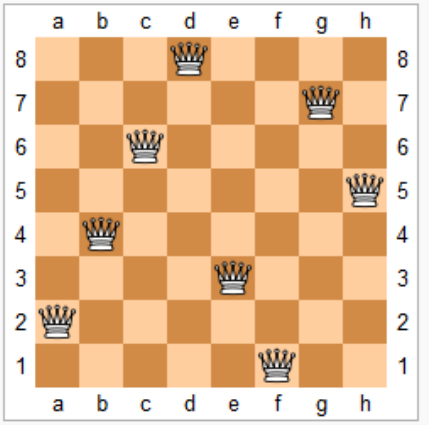
三、 分治

四、 双指针

五、 回溯剪枝

【LeetCode 原题】N-Queens

The *n*-queens puzzle is the problem of placing *n* queens on an *n*×*n* chessboard such that no two queens attack each other.Given an integer *n*, return all distinct solutions to the *n*-queens puzzle.Each solution contains a distinct board configuration of the *n*-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space respectively.



One solution to the eight queens puzzle

【核心代码】

```
class Solution {
public:
    vector<int> gNQueen;
    vector<vector<int>> res;
    vector<vector<string>> solveNQueens(int n) {
        gNQueen.resize(n);
        Nqueen(0,n);
        vector<vector<string>> ans;
        for(auto it : res){
            vector<string> temp;
            for(int i = 0; i < n; i++){
                string s;
                for(int j = 0; j < n; j++){
                    if( j != it[i]) s += '.';
                    else s += 'Q';
                }
                temp.push_back(s);
            }
            ans.push_back(temp);
        }
        return ans;
    }
    bool check_pos_valid(int maxRow,int data){
        for(int i = 0; i < maxRow; i++){
```

```
            if(gNQueen[i] == data || abs (data - gNQueen[i]) == abs(i - maxRow)) return false;
        }
        return true;
    }
    void Nqueen(int index, int n){
        for (int loop = 0; loop < n; loop++){ // row : index // column : loop
            if (check_pos_valid(index, loop)) {
                gNQueen[index] = loop;
                if (n - 1 == index) {
                    res.push_back(gNQueen);
                    gNQueen[index] = 0;
                    return;
                }
                Nqueen(index + 1,n);
                gNQueen[index] = 0;
            }
        }
    }
};
```

【Uva】困难的串

Sequences containing an occurrence of two adjoining identical subsequences will be called``easy''. Other sequences will be called ``hard''.Given n and L, output the nth hard sequence (composed of letters drawn from the first L letters in the alphabet). For example, with L = 3, the first 7 hard sequences are:

A

AB

ABA

ABAC

ABACA

ABACAB

ABACABA

【核心代码】

```
#include<stdio.h>
int n, L, cnt;
int S[100];
int dfs(int cur) {                                          // 返回 0 表示已经得到解，无须继续搜索
  if(cnt++ == n) {
    for(int i = 0; i < cur; i++) {
      if(i % 64 == 0 && i > 0) printf("\n");
      else if(i % 4 == 0 && i > 0) printf(" ");
      printf("%c", 'A'+S[i]); // 输出方案
    }
    printf("\n%d\n", cur);
    return 0;
  }
```

```
    for(int i = 0; i < L; i++) {
        S[cur] = i;
        int ok = 1;
        for(int j = 1; j*2 <= cur+1; j++) {                    // 尝试长度为 j*2 的后缀
            int equal = 1;
            for(int k = 0; k < j; k++)                         // 检查后一半是否等于前一半
                if(S[cur-k] != S[cur-k-j]) { equal = 0; break; }
            if(equal) { ok = 0; break; }                       // 后一半等于前一半，方案不合法
        }
        if(ok) if(!dfs(cur+1)) return 0;                       // 递归搜索。如果已经找到解，则直接退出
    }
    return 1;
}
int main() {
    while(scanf("%d%d", &n, &L) == 2 && n > 0) {
        cnt = 0;
        dfs(0);
    }
    return 0;
}
```

第四章 数据结构

一、 Linked List

- 反转链表

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

Example:

Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

【核心代码】

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *         int val;
 *         ListNode *next;
 *         ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        if(k == 1) return head;
        int cnt = 0;
        ListNode* temp = head, * h = head;
        for(;temp;cnt++,temp = temp->next);
        head = new ListNode(0);
        ListNode* lastTail = head;
        for(int i = 0; i < cnt / k; i++){
            ListNode *next = NULL , *old = h;
            temp = old->next;
            for(int j = 0; j < k -1; j++){
                next = temp->next;
                temp->next = old;
                old = temp;
                temp = next;
            }
            lastTail->next = old;
            lastTail = h;
            h = temp;
        }
        lastTail->next = h;
        return head->next;
    }
};
```

- 逆置链表
- 合并 k 个有序链表
- 旋转链表
- 倒数第 k 个元素
- 两两交换
- 划分链表
- 去除重复元素
- 复制带随机指针的链表

【LeetCode】Copy List with Random Pointer

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

Definition for singly-linked list with a random pointer.

```
struct RandomListNode {
    int label;
    RandomListNode *next, *random;
    RandomListNode(int x) : label(x), next(NULL), random(NULL) {}
};
```

【核心代码】

```
class Solution {
public:
    map<int,RandomListNode*> m;
    RandomListNode *copyRandomList(RandomListNode *head) {
        if(head == NULL) return NULL;
        RandomListNode* h = new RandomListNode(head->label);
        m[head->label] = h;
        if( head->next && m.find(head->next->label) == m.end() || head->next == NULL){
            h->next = copyRandomList(head->next);
        }else{
            h->next = m[head->next->label];
        }
        if(head->random && m.find(head->random->label) == m.end() || head->random == NULL){
            h->random = copyRandomList(head->random);
        }else{
            h->random = m[head->random->label];
        }
        return h;
    }
};
```

二、 Hash

【LeetCode】Two Sum

Given an array of integers, find two numbers such that they add up to a specific target number. The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

【核心代码】

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> hash;
        for(int i = 0; i < nums.size(); i++){
            if(hash.find(target-nums[i]) != hash.end()){
                return {hash[target-nums[i]],i};
            }
            hash[nums[i]] = i;
        }
    }
};
```

【PAT】Hashing - Average Search Time

The task of this problem is simple: insert a sequence of distinct positive integers into a hash table first. Then try to find another sequence of integer keys from the table and output the average search time (the number of comparisons made to find whether or not the key is in the table). The hash function is defined to be *H*(*key*)=*key*%*TSize* where *TSize* is the maximum size of the hash table. Quadratic probing (with positive increments only) is used to solve the collisions.

Note that the table size is better to be prime. If the maximum size given by the user is not prime, you must re-define the table size to be the smallest prime number which is larger than the size given by the user.

Input Specification:

Each input file contains one test case. For each case, the first line contains 3 positive numbers: MSize, N, and M, which are the user-defined table size, the number of input numbers, and the number of keys to be found, respectively. All the three numbers are no more than 104. Then N distinct positive integers are given in the next line, followed by M positive integer keys in the next line. All the numbers in a line are separated by a space and are no more than 105.

Output Specification:

For each test case, in case it is impossible to insert some number, print in a line X cannot be inserted. where X is the input number. Finally print in a line the average search time for all the M keys, accurate up to 1 decimal place.

Sample Input:

4 5 4

10 6 4 15 11

11 4 15 2

Sample Output:

15 cannot be inserted.

2.8

【核心代码】

```
#include<iostream>
#include<vector>
#include<cmath>
#include<iomanip>
using namespace std;
bool isPrime(int n) {
	if (n == 1) return false;
	int k = sqrt(n);
	for (int i = 2; i <= k; i++) {
		if (n % i == 0) return false;
	}
	return true;
}
int main() {
	int tableSize, n, m;
	cin >> tableSize >> n >> m;
	while (!isPrime(tableSize)) tableSize++;
	vector<int> v(tableSize);
	for (int i = 0; i < n; i++) {
		int temp;
		cin >> temp;
		bool flag = false;
		for (int j = 0; j < tableSize; j++) {
			int pos = (temp + j * j) % tableSize;
			if (v[pos] == 0) {
				v[pos] = temp;
				flag = true;
				break;
			}
		}
		if (flag == false) cout << temp << " cannot be inserted.\n";
	}
	int ans = 0;
	for (int i = 0; i < m; i++) {
		int temp;
		cin >> temp;
		int cnt = 1;
		for (int j = 0; j < tableSize; j++) {
			int pos = (temp + j * j) % tableSize;
			if (v[pos] == temp || v[pos] == 0) break;
			cnt++;
		}
		ans += cnt;
	}
	cout << setiosflags(ios::fixed) << setprecision(1) << ans * 1.0 / m;
	return 0;
}
```

【LeetCode】Find All Anagrams in a String

Given a string **s** and a **non-empty** string **p**, find all the start indices of **p**'s anagrams in **s**.

Strings consists of lowercase English letters only and the length of both strings **s** and **p** will not be larger than 20,100.

The order of output does not matter.

Example 1:

Input:

s: "cbaebabacd" p: "abc"

Output:

[0, 6]

Explanation:

The substring with start index = 0 is "cba", which is an anagram of "abc".

The substring with start index = 6 is "bac", which is an anagram of "abc".

【分析】对每 p.length()个字符进行哈希统计，若与 p 相同，则返回最前面的 index。注意：统计时需要方法，否则会超时。由于前 p.length()个字符与下一个 p.length()个字符只有一个字符不同，此时只需对那个字符再原有 hash 上修改。

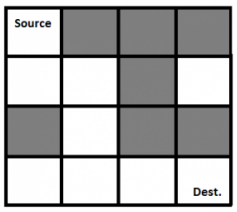
【核心代码】

```
class Solution {
public:
    vector<int> findAnagrams(string s, string p) {
        if (s.empty()) return {};
        vector<int> res, ms(256, 0), mp(256, 0);
        for (int i = 0; i < p.size(); ++i) {
            ++ms[s[i]]; ++mp[p[i]];
        }
        if (ms == mp) res.push_back(0);
        for (int i = p.size(); i < s.size(); ++i) {
            ++ms[s[i]];
            --ms[s[i - p.size()]];
            if (ms == mp) res.push_back(i - p.size() + 1);
        }
        return res;
    }
};
```

三、 递归和 DFS

迷宫问题（Maze Problem）

A Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. A rat starts from source and has to reach the destination.The rat can move only in four directions: forward, backward, left and right.



【核心代码】

```
class Solution {
public:
    vector< vector<int> > direction = { {0,1},{1,0},{0,-1},{-1,0} };
    vector<pair<int, int> > temp;
    vector< vector<pair<int, int> > > path;
    vector< vector<pair<int, int> > > solutionMaze(vector< vector<bool> > maze) {
        int n = maze.size(), m = maze.front().size();
        vector< vector<bool> > visit(n, vector<bool>(m, false));
        temp.push_back(make_pair(0, 0));
        dfs(maze, visit, 0, 0, n - 1, m - 1);
        return path;
    }
    bool checkValid(int x, int y, int xend, int yend) {
        if (x < 0 || y < 0 || x > xend || y > yend) return false;
        else return true;
    }
    void dfs(vector<vector<bool> > &maze, vector<vector<bool> > &visit, int x ,int y ,int xend, int yend) {
        if (x == xend && y == yend) {
            path.push_back(temp);
            return;
        }
        if (!checkValid(x, y, xend, yend)) return;
        for (int i = 0; i < 4; i++) {
            int nx = x + direction[i][0];
            int ny = y + direction[i][1];
            if (checkValid(nx, ny, xend, yend) && maze[nx][ny] && !visit[nx][ny]) {
                visit[nx][ny] = true;
                temp.push_back(make_pair(nx,ny));
                dfs(maze, visit, nx, ny, xend, yend);
                visit[nx][ny] = false;
                temp.pop_back();
            }
        }
    }
};
```

背包问题

0-1 Knapsack Problem

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

【分析】问题的大意就是在 n 个物品中，选出若干个，在不超过背包的容量前提下使得背包内物品价值最大。对于每件物品来说都有选与不选，这样就形成了分支。这个问题有两个终止条件：一个是当前重量超过了 W；另一个就是所有物品都选完了。

【核心代码】

```
void dfs(int index, int sum,int weight){
    if(index == n) return;
    dfs(index+1,sum,weight);
    if(weight+w[index] <= W){
        if(sum+v[index] > maxValue)
            maxValue = sum + v[index];
        dfs(index+1, sum+v[index], weight + w[index]);
    }
}
```

组个数 Forming A Number

Given n numbers {a1, a2, a3, … an}, we need to tell the total number of ways we can form a number 'N' using thesum of the given numbers. (allowing repetitions).

Example: Input: [1, 3, 5], 6

Output:[[1,1,1,1,1,1],

[1,1,1,3],

[1,5],

[3,3]]

【核心代码】

➢ 版本一

```
class Solution {
public:
    vector<vector<int> > res;
    vector<int> tempans;
    vector<vector<int> > solution(vector<int> nums, int n) {
        tempans.resize(n,0);
        dfs(nums, n, nums.size() - 1, 0, 0);
        for (auto &it : res) {
            sort(it.begin(), it.end());
            while (it.front() == 0) it.erase(it.begin());
        }
        sort(res.begin(), res.end());
        return res;
    }
    void dfs(vector<int> nums, int n, int index, int tempsum, int tempk) {
        if (tempsum == n) {
            res.push_back(tempans);
            return;
```

```
            }
            // 必须逆序
            while (index >= 0) {
                if (tempsum + nums[index] <= n) {
                    tempans[tempk] = nums[index];
                    dfs(nums, n, index, tempsum + nums[index], tempk + 1);
                }
                if (index == 0) return;
                index--;
            }
        }
};
```

- 版本二

```
class Solution {
public:
    vector<vector<int> > res;
    vector<int> tempans;
    vector<vector<int> > solution(vector<int> nums, int n) {
        dfs(nums, n, 0, 0, 0);
        return res;
    }
    void dfs(vector<int> nums, int n, int index, int tempsum, int tempk) {
        if (tempsum == n) {
            res.push_back(tempans);
            return;
        }
        if (index == nums.size()) return;
        if (tempsum + nums[index] <= n) {
            tempans.push_back(nums[index]);
            dfs(nums, n, index, tempsum + nums[index], tempk + 1);
            tempans.pop_back();
        }
        dfs(nums, n, index + 1, tempsum, tempk);
    }
};
```

子集问题

【LeetCode】Subsets

Given a set of distinct integers, nums, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

- 方法一：迭代求解。设 S 是当前工作集，初始为空；ans 是结果集，初始为一个空集。当向工作集 S 添加一个元素时，结果集 ans 应当对各个集加入一个元素后在加入到结果集。当所有的元素添加完毕后，结果集 ans 就是所求的所有子集。例如原始 S={}，ans={{}}，添加 1 进入 S 得 S={1}，ans={{},{1}}，添加 2 进入 S 得 S={1,2}，ans={{},{1},{2},{1,2}}。

【核心代码】

```
vector<vector<int>> subsetsWithDup(vector<int> &S) {
    if (S.empty()) return{};
    vector<vector<int>> res(1);
```

```
    sort(S.begin(), S.end());
    for (int i = 0; i < S.size(); ++i) {
        int size = res.size();
        for (int j = 0; j < size; ++j) {
            res.push_back(res[j]);
            res.back().push_back(S[i]);
        }
    }
    return res;
}
```

➢ 方法二：递归求解。

【核心算法】

```
vector<int> S;
vector<int> out;
vector<vector<int>> res;
void getSubsets(int cur){
    res.push_back(out);
    for (int i = cur; i < S.size(); ++i) {//当前处理的元素不会在后面处理
        out.push_back(S[i]);
        getSubsets(i + 1);
        out.pop_back();
    }
}
```

➢ 方法三：二进制状态法。以 000，001，…，111 代表各个数的出现情况。

【核心算法】

```
vector<vector<int>> ans;
for (int i = 0; i < pow(2, n); i++) {
    int j = i, k = n;
    vector<int> a;
    while (k) {
        if (j & 1) a.push_back(k);
        j >>= 1;
        k--;
    }
    ans.push_back(a);
}
```

【LeetCode】Subsets II

Given a collection of integers that might contain duplicates, nums, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

➢ 方法一：同上，只要改变结果集添加元素的起始位置。例如原始 S={}，ans={{}}，添加 1 进入 S 得 S={1}，ans={{},{1}}；添加 2 进入 S 得 S={1,2}，ans={{},{1},{2},{1,2}}；添加 2 进入 S 得 S={1,2，2}，ans={{},{1},{2},{1,2}，{2,2},{1,2,2}}。

【核心算法】

```
vector<vector<int>> subsetsWithDup(vector<int> &S) {
    if (S.empty()) return{};
    vector<vector<int>> res(1);
    sort(S.begin(), S.end());
    int size = 1, last = S[0];
    for (int i = 0; i < S.size(); ++i) {
        if (last != S[i]) {
            last = S[i];
            size = res.size();
        }
        int newSize = res.size();
        for (int j = newSize - size; j < newSize; ++j) {
            res.push_back(res[j]);
            res.back().push_back(S[i]);
        }
    }
    return res;
}
```

- 方法二：要注意到什么时候有重复。当一个元素加了以后回溯后,判断下一个元素是否相同，如果相同就会造成重复。因此在上述回溯后加入 while(i < S.size()-1&&S[i]==S[i+1])i++;

【LeetCode】Sum of Subsequence Widths

Given an array of integers A, consider all non-empty subsequences of A.

For any sequence S, let the *width* of S be the difference between the maximum and minimum element of S.

Return the sum of the widths of all subsequences of A.

As the answer may be very large, **return the answer modulo 10^9 + 7**.

Example 1:

Input: [2,1,3]

Output: 6

Explanation:

Subsequences are [1], [2], [3], [2,1], [2,3], [1,3], [2,1,3].

The corresponding widths are 0, 0, 0, 1, 1, 2, 2.

The sum of these widths is 6.

Note:

- 1 <= A.length <= 20000
- 1 <= A[i] <= 20000

【分析】

Intuition

Let's try to count the number of subsequences with minimum A[i] and maximum A[j].

Algorithm

We can sort the array as it doesn't change the answer. After sorting the array, this allows us to know that the number of subsequences with minimum A[i] and maximum A[j] is 2^{j-i-1}. Hence, the desired answer is

$$\sum_{i>j} 2^{j-i-1}\left(A_j - A_i\right)$$

$$=\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 2^{j-i-1}A_j - \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 2^{j-i-1}A_i$$

$$=\left(\left(2^0A_1+2^1A_2+\cdots+2^{n-2}A_{n-1}\right)+\left(2^0A_2+2^1A_3+\cdots+2^{n-3}A_{n-1}\right)+\cdots+\left(2^0A_{n-1}\right)\right)$$

$$-\sum_{i=0}^{n-2}\left(2^0+2^1+\cdots+2^{n-i-2}\right)A_i$$

$$=\left(\left(2^0\right)A_1+\left(2^0+2^1\right)A_2+\cdots+\left(2^0+2^1+\cdots+2^{n-2}\right)A_{n-1}\right)-\sum_{i=0}^{n-2}\left(2^{n-i-1}-1\right)A_i$$

$$=\sum_{j=0}^{n-1}\left(2^j-1\right)A_j-\sum_{i=0}^{n-1}\left(2^{n-i-1}-1\right)A_i$$

【LeetCode】Bitwise ORs of Subarrays

We have an array A of non-negative integers.

For every (**contiguous**) subarray B = [A[i], A[i+1], ..., A[j]] (with i <= j), we take the bitwise OR of all the elements in B, obtaining a result A[i] | A[i+1] | ... | A[j].

Return the number of possible results. (Results that occur more than once are only counted once in the final answer.)

【核心算法】

```
class Solution {
public:
    int subarrayBitwiseORs(vector<int>& A) {
        unordered_set<int> s, ans;
        s.insert(0);
        for(auto it : A){
            unordered_set<int> temp;
            for(auto that : s)
                temp.insert(that | it);
            temp.insert(it);
            s = temp;
            ans.insert(s.begin(),s.end());
        }
        return ans.size();
    }
};
```

排列问题

【LeetCode】Permutations

Given a collection of numbers, return all possible permutations.

For example,

[1,2,3] have the following permutations:

[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].

- 方法一：递归+填充。

【核心算法】

```
vector<int> num;
vector<int> visited;
vector<int> out;
vector<vector<int> > res;
void permuteDFS(int level) {
    if (level == num.size()) res.push_back(out);
    else {
        for (int i = 0; i < num.size(); ++i) {
            if (visited[i] == 0) {
                visited[i] = 1;
                out.push_back(num[i]);
                permuteDFS(level + 1);
                out.pop_back();
                visited[i] = 0;
            }
        }
    }
}
```

- 方法二：递归+交换。

【核心算法】

```
vector<vector<int> > res;
void permuteDFS(vector<int> &num, int cur) {
    if (cur >= num.size()) res.push_back(num);
    for (int i = cur; i < num.size(); ++i) {
        swap(num[cur], num[i]);
        permuteDFS(num, cur + 1);
        swap(num[cur], num[i]);
    }
}
```

- 方法三：迭代。设 S 是当前工作集，初始为空；ans 是结果集即保存 S 的全排列，初始为一个空集。当向工作集 S 添加一个元素时，结果集 ans 应当对各个集的不同位置加入一个元素，然后更新结果集 ans。例如 S={}，ans={{}}，向 S 加入 1，则 S={1}，ans={{1}}；再向 S 加入 2，则 S={1,2}，ans={{1,2},{2,1}}；最后向 S 中加入 3，则 S={1,2,3}，ans={{1,2,3},{1,3,2},{3,1,2},{2,1,3},{2,3,1},{3,2,1}}。

【核心算法】

【LeetCode】Permutations II

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,

[1,1,2] have the following unique permutations:

[1,1,2], [1,2,1], and [2,1,1].

- 方法一：同上，只要在循环中最前面加入 if (i > 0 && num[i] == num[i - 1] && visited[i - 1] == 0) continue。
- 方法二：同上，只要在循环中最前面加入 if (i != start && nums[i] == nums[start]) continue。

【Uva】素数环 Prime Ring Problem

输入正整数 n，把整数 1, 2, 3,…, n 组成一个环，使得相邻两个整数之和均为素数。输出时从整数 1 开始逆时针排列。同一个环应恰好输出一次。n≤16。

【核心代码】

```
vector<int> temp(n, 1);
vector<vector<int>> res;
map<int,bool> visited;
void dfs(int cur) {
    if (cur == n && isPrime(temp[0] + temp[n - 1])) {
        res.push_back(temp);
        return;
    }
    for (int i = 2; i <= n; i++) {
        if (visited[i] == false && isPrime(i + temp[cur - 1])) {
            visited[i] = true;
            temp[cur] = i;
            dfs(cur + 1);
            visited[i] = false;
        }
    }
}
```

总结

正确的剪枝可以给 dfs 提高很大效率。

dfs 回溯的写法会造成效率有很大差别(PAT1131)。

版本一

```
if (node == end) {
    path = tempPath;
    return;
}
for (int i = 0; i < v[node].size(); i++) {
    if (visit[v[node][i]] == 0) {
        visit[v[node][i]] = 1;
        tempPath.push_back(v[node][i]);
        dfs(v[node][i]);
        visit[v[node][i]] = 0;
        tempPath.pop_back();
    }
}
```

版本二

```
visited[node] = 1;
tempPath.push_back(node);
if(node == end){
    path = tempPath;
    visited[node] = 0;
    tempPath.pop_back();
    return;
}
for(int i = 0; i < v[node].size();i++){
    int u = v[node][i];
    if(!visited[u]){
        dfs(u);
    }
}
visited[node] = 0;
tempPath.pop_back();
```

四、 迭代和 BFS

最少查找次数

给定一个目标值 T，以及一个初始值 K，查找规则如下：每次只能向前一个或者向后一个或者查找双倍。

Input:T = 17 K = 5；

Output:5->10->9->18->17

【核心算法】

```
struct node {
	int local, step,tag;
	// tag = -1  初始 tag = 0  向前 tag = 1  向后 tag = 2  加倍
};
int main() {
	int start, end;
	cin >> start >> end;
	unordered_map<int, node> mapp;
	queue<node> q;
	q.push({ start,0,-1 });
	mapp[start] = q.front();
	node temp;
	while (!q.empty()) {
		temp = q.front(); q.pop();
		if (temp.local == end)    break;
		if (temp.local < end && mapp.find(temp.local + 1) == mapp.end()) {
			q.push({ temp.local + 1,temp.step + 1,0 });
			mapp[temp.local + 1] = q.back();
		}
		if (temp.local > 0 && mapp.find(temp.local - 1) == mapp.end()) {
			q.push({ temp.local - 1,temp.step + 1,1 });
			mapp[temp.local - 1] = q.back();
		}
		if (temp.local < end && mapp.find(temp.local * 2) == mapp.end()) {
			q.push({ temp.local * 2,temp.step + 1,2 });
			mapp[temp.local * 2] = q.back();
		}
	}
	cout << temp.step << endl;
	vector<int> ans;
	ans.insert(ans.begin(), temp.local);
	while (temp.tag != -1) {
		switch (temp.tag){
			case 0:
				temp = mapp[temp.local - 1];
				break;
			case 1 :
				temp = mapp[temp.local + 1];
				break;
```

```
                case 2:
                    temp = mapp[temp.local / 2];
                    break;
                default:
                    break;
            }
            ans.insert(ans.begin(), temp.local);
        }
        return 0;
}
```

五、 树与二叉树

二叉树的深度

【LeetCode 原题】The maximum depth of binary tree

Given a binary tree, find its maximum depth.The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its depth = 3.

【核心代码】

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root == NULL) return 0;
        else return max(maxDepth(root->left),maxDepth(root->right))+1;
    }
};
```

【思考】非递归方法？

【LeetCode 原题】Minimum Depth of Binary Tree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its minimum depth = 2.

【核心代码】

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int minDepth(TreeNode* root) {
        int ret = getMin(root,1);
        return (ret == INT_MAX ? 0 : ret);

    }
    int getMin(TreeNode* root, int depth){
        if(root == NULL) return INT_MAX;
        else if(root->left == NULL && root->right ==NULL) return depth;
        else return min(getMin(root->left,depth+1),getMin(root->right,depth+1));
    }
};
```

二叉树遍历

- 先序遍历
- 中序遍历
- 后序遍历
- 层次遍历

【LeetCode 原题】The level order traversal of binary tree

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example:

Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

【核心代码】

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
```

```
 *      int val;
 *      TreeNode *left;
 *      TreeNode *right;
 *      TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if(root == NULL) return res;
        queue<TreeNode*> q;
        q.push(root);
        TreeNode* last = root;
        vector<int> v;
        while(!q.empty()){
            TreeNode* temp = q.front();
            q.pop();
            v.push_back(temp->val);
            if(temp->left) q.push(temp->left);
            if(temp->right) q.push(temp->right);
            if(last == temp){
                last = q.back();
                res.push_back(v);
                v.clear();
            }
        }
        return res;
    }
};
```

【思考 1】如何实现锯齿形层次遍历即偶数从左到右、奇数从右到左？

【思考 2】如何输出每层的叶子节点（PAT1004）？

【思考 3】如何从先序、中序、后序过程中得到层次序列？

【原创】Convert String to Tree

In LeetCode, there is always asked to convert the string to the binary tree.Now you are asked to implement the function.It is assumed that all input is valid.

Example Input : [1, null, 2, 3 , 4 , null , 5]

Output: 1

```
    \
     2
    /  \
   3    4
       /
      5
```

Note: Space can appear in input.

【核心代码】

```
TreeNode* stringToTreeNode(string input) {
```

```
    trimLeftTrailingSpaces(input);
    trimRightTrailingSpaces(input);
    input = input.substr(1, input.length() - 2);
    if (!input.size()) {
        return NULL;
    }
    string item;
    stringstream ss;
    ss.str(input);
    getline(ss, item, ',');
    TreeNode* root = new TreeNode(stoi(item));
    queue<TreeNode*> nodeQueue;
    nodeQueue.push(root);
    while (true) {
        TreeNode* node = nodeQueue.front();
        nodeQueue.pop();
        if (!getline(ss, item, ',')) {
            break;
        }
        trimLeftTrailingSpaces(item);
        trimRightTrailingSpaces(item);
        if (item != "null") {
            int leftNumber = stoi(item);
            node->left = new TreeNode(leftNumber);
            nodeQueue.push(node->left);
        }
        if (!getline(ss, item, ',')) {
            break;
        }
        trimLeftTrailingSpaces(item);
        trimRightTrailingSpaces(item);
        if (item != "null") {
            int rightNumber = stoi(item);
            node->right = new TreeNode(rightNumber);
            nodeQueue.push(node->right);
        }
    }
    return root;
}
```

- 根据某两个序列构造二叉树

①先序+中序

可以唯一确定一棵二叉树，主要的思想：先根据先序遍历确定根结点，然后根据中序遍历将先序遍历划分为左子树的先序遍历和右子树的先序遍历，然后迭代求解，具体实现算法如下：

②中序+后序

可以唯一确定一棵二叉树，主要的思想：先根据后序遍历确定根结点，然后根据中序遍历将层序遍历划分为左子树的后序遍历和右子树的后序遍历，然后迭代求解，具体实现算法如下：

③层序+中序

可以唯一确定一棵二叉树，主要的思想：先根据层序遍历确定根结点，然后根据中序遍历将层序遍历划分为左子树的层序遍历和右子树的层序遍历，然后迭代求解，具体实现算法如下：

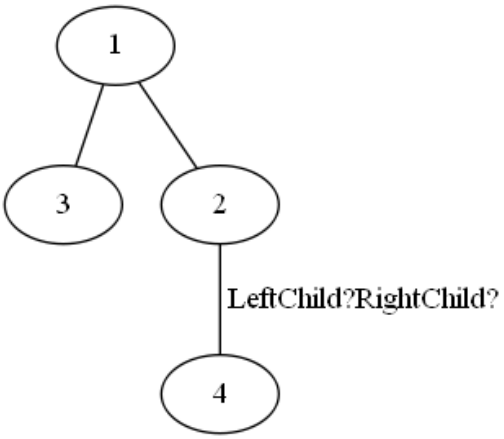
```
vector<int> pre, in, post, level;
void level_in(int inL, int inR, vector<int> level) {
	if (level.size() == 0) return;
	pre.push_back(level[0]);
	int k = inL;
	for (; in[k] != level[0] && k < inR; k++);
	vector<int> leftLevel, rightLevel;
	for (int i = 1; i < level.size(); i++) {
		bool isLeft = false;
		for (int j = inL; j < k; j++) {
			if (level[i] == in[j]) {
				isLeft = true;
				break;
			}
		}
		if (isLeft) leftLevel.push_back(level[i]);
		else rightLevel.push_back(level[i]);
	}
	level_in(inL, k - 1, leftLevel);
	level_in(k + 1, inR, rightLevel);
	post.push_back(level[0]);
}
```

④先序+后序

不可以唯一确定一棵二叉树，不能唯一确定是因为可能存在下面这种情况：pre:1 3 2 4 post:3 4 2 1，这种情况不能区分根节点的左右孩子。输出可能的一种中序遍历，具体算法如下：

```
vector<int> pre, post,in;
bool isUnique = true;
void getIn(int preL, int preR, int postL, int postR) {
	if (preL > preR || postL > postR) return;
	int k;
	for (k = preL + 1 ; k < preR && pre[k] != post[postR - 1]; k++);
	int num = k – preL - 1;
	if (k - preL != 1 || k > preR) { //记得区别子树只有一个结点,可能导致判断唯一性失效
		getIn(preL + 1, k - 1, postL, postL + num - 1);
	}
	else {
		isUnique = false;//不能唯一确定
	}
	in.push_back(pre[preL]);
	getIn(k, preR,postL + numL, postR - 1);
```

```
}
```



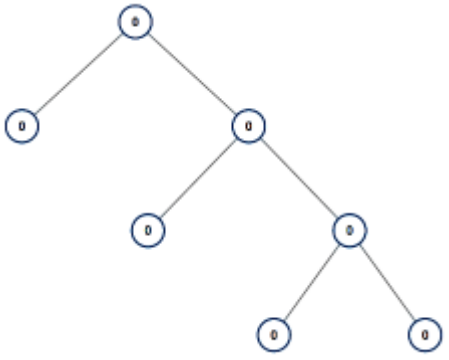
先序遍历：1 3 2 4

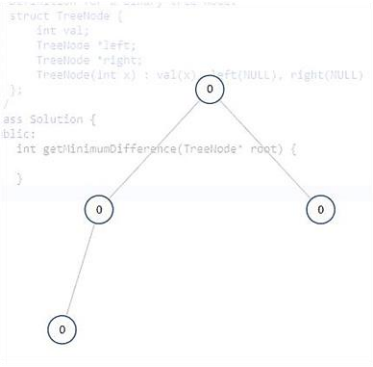
后序遍历：3 4 2 1

完全二叉树

完全二叉树的定义有两种：

1. 国外定义为每个结点要么有两个结点要么没有结点。
2. 国内定义为按满二叉树编号后，所有结点编号是连续的。





Is it a complete binary tree?

Given the in-order and post-order sequences of the binary tree,output the level-order sequence and to tell if it is a complete binary tree.

【核心代码】

➢ 方法一：利用层序遍历中不能有空结点，即遇到空结点不能在循环中

```
void getLevel(node* root){
    queue<node*> q;
    q.push(root);
    bool isNULL = false;
    while (!q.empty()) {
        node* temp = q.front(); q.pop();
        level.push_back(temp->key);
        if (temp->lchild) {
            if (isNULL) isComplete = false;
            q.push(temp->lchild);
```

```
            }
            else {
                isNULL = true;
            }
            if (temp->rchild) {
                if (isNULL) isComplete = false;
                q.push(temp->rchild);
            }
            else {
                isNULL = true;
            }
        }
}
```

➢ 方法二：利用父结点编号与子结点编号之间的关系 index = 2 * index + 1，检测编号是否连续。

【LeetCode】All Possible Full Binary Trees

A full binary tree is a binary tree where each node has exactly 0 or 2 children.

Return a list of all possible full binary trees with N nodes. Each element of the answer is the root node of one possible tree.

Each node of each tree in the answer must have node.val = 0.

You may return the final list of trees in any order.

【核心代码】

```
class Solution {
public:
    vector<TreeNode*> allPossibleFBT(int N) {
        if(N % 2 == 0) return {};
        if(N == 1) return {new TreeNode(0)};
        vector<TreeNode*> ans;
        for(int i = 1; i < N; i += 2){
            vector<TreeNode*> L = allPossibleFBT(i), R = allPossibleFBT(N - 1 - i);
            for(int j = 0; j < L.size(); j++)
                for(int k = 0; k < R.size(); k++){
                    TreeNode* root = new TreeNode(0);
                    root->left = L[j]; root->right = R[k];
                    ans.push_back(root);
                }
        }
        return ans;
    }
};
```

二叉搜索树

二叉搜索树（BST）是二叉树的一种特殊表示形式，它满足如下特性：

1. 每个节点中的值必须大于（或等于）存储在其左侧子树中的任何值。
2. 每个节点中的值必须小于（或等于）存储在其右子树中的任何值。

【LeetCode 原题】Is It a Binary Search Tree

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

```
Input:
    2
   / \
  1   3
Output: true
```

Example 2:

```
    5
   / \
  1   4
     / \
    3   6
Output: false
Explanation: The input is: [5,1,4,null,null,3,6]. The root node's value
             is 5 but its right child's value is 4.
```

【核心代码】

➢ 方法一

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:

    bool isValidBST(TreeNode *root) {
        return isValidBST(root, LONG_MIN, LONG_MAX);
    }
    bool isValidBST(TreeNode *root, long mn, long mx) {
        if (!root) return true;
        if (root->val <= mn || root->val >= mx) return false;
        return isValidBST(root->left, mn, root->val) && isValidBST(root->right, root->val, mx);
    }

};
```

➢ 方法二

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:

    bool isValidBST(TreeNode *root) {
        if (!root) return true;
        vector<int> vals;
        inorder(root, vals);
        for (int i = 0; i < vals.size() - 1; ++i) {
            if (vals[i] >= vals[i + 1]) return false;
        }
        return true;
    }
    void inorder(TreeNode *root, vector<int> &vals) {
        if (!root) return;
        inorder(root->left, vals);
        vals.push_back(root->val);
        inorder(root->right, vals);
    }

};
```

【PAT 原题】Is It a Binary Search Tree

Now given a sequence of integer keys, you are supposed to tell if it is the preorder traversal sequence of a BST or the mirror image of a BST(If we swap the left and right subtrees of every node, then the resulting tree is called the Mirror Image of a BST).

【核心代码】

```
vector<int> post,pre;
bool flag = true;;
void getPost(int preL, int preR) {
    if (preL > preR) return;
    int i = preL + 1 , j = preR;
    if (flag) {
        while (i <= preR && pre[preL] > pre[i]) i++;
        while (j > preL && pre[j] >= pre[preL]) j--;
    }
    else {
        while (i <= preR && pre[preL] <= pre[i]) i++;
        while (j > preL && pre[j] < pre[preL]) j--;
    }
    if (i - j != 1) return;
    getPost(preL + 1, j);
```

```
        getPost(i,preR);
        post.push_back(pre[preL]);
}
```

AVL 树

Now given a sequence of insertions, you are supposed to return the AVL tree.

【核心代码】

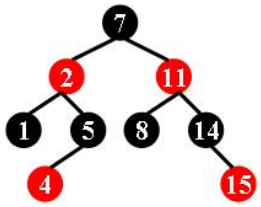
```
struct node{
        int val,height;
        node* lchild,rchild;
        node(int x): val(x), height(1),lchild(NULL),rchild(NULL) {}
};
int getHeight(node* root){
        if(root) return root->height;
        else return 0;
}
void updateHeight(node* root){
        root->height = max(getHeight(root->lchild), getHeight(root->rchild)) + 1;
}
Int getBalanceFactor(node* root){
        return getHeight(root->lchild) – getHeight(root->rchild);
}
void lRotation(node* &root){
        node* temp = root->rchild;
        root->rchild = temp->lchild;
        temp->lchild = root;
        updateHeight(root);
        updateHeight(temp);
        root = temp;
}
void rRotation(node* root){
        node* temp = root->lchild;
        root->lchild = temp->rchild;
        temp->rchild = root;
        updateHeight(root);
        updateHeight(temp);
        root = temp;
}
void insert(node* &root, int val){
        if(root ==NULL){
                root = new node(val);
                return ;
        }
        if(val < root->val){
                insert(root->lchild,val);
                updateHeight(root)
                if(getBalanceFactor(root) == 2){
```

```
                if(getBalanceFactor(root->lchild) == 1){//LL
                    rRotation(root);
                }else if(getBalanceFactor(root->lchild) == -1){//LR
                    lRotation(root->lchild);
                    rRotation(root);
                }
            }
        }else{
            insert(root->rchild,val);
            updateHeight(root)
            if(getBalanceFactor(root) == -2){
                if(getBalanceFactor(root->rchild) == -1){//RR
                    lRotation(root);
                }else if(getBalanceFactor(root->rchild) == 1){// RL
                    rRotation(root->rchild);
                    lRotation(root);
                }
            }
        }
}
```

红黑树

红黑树具有如下特点的树：

- 是一棵二叉搜索树；
- 所有结点不是红的就是黑的；
- 根结点是黑色的；
- 所有的的叶子结点都是黑的（由于叶子结点没有键值一般省略不画，可以看成空结点）；
- 如果一个结点是红的，那么其孩子结点一定是黑的；
- 从任意一个结点出发到达叶节点的简单路径中黑色结点数目相同。



堆

堆是一个完全二叉树，因此一般用层次遍历序列存储堆。堆又分为大顶堆，小顶堆。判断给定序列是否是堆？C++ STL 提供了 is_heap(first,last,cmp)默认<是大顶堆。C++STL 还提供了以下关于堆的操作 make_heap(first,last,cmp)，pop_heap(first,last,cmp)将堆顶值移到最后，push_heap(first,last,cmp)自动扩容。

并查集

```
int father[n];
//初始化并查集
for(int i = 0; i < n; i++){
```

```
        father[i] = i;
}
//查找和压缩路径
int findFather(int x){
        return x == father[x] ? x : (father[x] = findFather(father[x]));
}
//合并
void union(int a ,int b){
        int fa = findFather(a);
        int fb = findFather(b);
        if(fa != fb);{
                father[fa] = fb;
        }
}
```

【原创】Robber gang

Suppose there were n robbers, and each of them had several places they wanted to rob, so they could be divided into groups according to where they wanted to rob. Assuming that each gang can only take one place, how many groups will they form? What is the maximum value of each gang robbery? What is the maximum average value of each gang?

Example:

Input:

```
8 6    //    8 is the total numbers of the robbers , 6 is the total numbers of the places.
300 400 200 600 250 350 // the maximum value of each place.
3: 1 2 3
1: 4
2: 5 6
1: 4
1: 4
2: 3 2
1: 4
1: 1
```

Output:

```
3
4: 2 4 5 7 600 150
3: 1 6 8 400 133
1: 3 350 350
```

【核心代码】

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
#define _CRT_SECURE_NO_WARNINGS
#define inf INT_MAX
using namespace std;
int n, m;
vector<int> father;//并查集
```

```
vector<int> p, value;// 第一个想抢该地方的人编号,每个地方抢到的最大值价值
struct gang {
	int total;
	int value;
	vector<int> v;
	bool operator < (const gang &ano) const {return total > ano.total; }
};
void init() {
	for (int i = 1; i <= n; i++){
		father[i] = i;
	}
}
int findFather(int x) {
	return x == father[x] ? x : (father[x] = findFather(father[x]));
}
void unionset(int a, int b) {
	int fa = findFather(a), fb = findFather(b);
	if (fa != fb) {
		father[fa] = fb;
	}
}
int main() {
	scanf("%d %d", &n, &m);
	father.resize(n + 1);
	p.resize(m + 1,0);
	value.resize(m + 1);
	init();
	for (int i = 1; i <= m; i++) {
		scanf("%d", &value[i]);
	}
	vector<vector<int>> wr(n + 1);//每个人想抢劫的地方
	for (int i = 1; i <= n; i++) {
		int temp;
		scanf("%d:", &temp);
		for (int j = 0; j < temp; j++) {
			int k;
			scanf("%d", &k);
			wr[i].push_back(k);
			if (p[k] == 0) p[k] = i;
			unionset(i, findFather(p[k]));
		}
	}
	vector<vector<int>> v(n + 1); // 团伙编号
	for (int i = 1; i <= n; i++) {
		v[father[i]].push_back(i);
	}
```

```
    vector<gang> ans;//团伙抢到的价值
    int cnt = 0;
    for (int i = 1; i <= n; i++) {
        int max = -1;
        for (int j = 0; j < v[i].size(); j++) {
            int index = v[i][j];
            for (int k = 0; k < wr[index].size(); k++) {
                if (max < value[wr[index][k]]) {
                    max = value[wr[index][k]];
                }
            }
        }
        if (max != -1) ans.push_back({(int)v[i].size(),max,v[i]});
        if (v[i].size() != 0) cnt++;
    }
    cout << cnt << endl;
    sort(ans.begin(), ans.end());
    for (int i = 0; i < cnt; i++) {
        cout << ans[i].total << ":";
        for (int j = 0; j < ans[i].v.size(); j++) {
            cout << " " << ans[i].v[j];
        }
        cout << " " << ans[i].value << " " << ans[i].value / ans[i].total << endl;
    }
    system("pause");
    return 0;
}
```

【LeetCode 原题】Longest Consecutive Sequence

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.Your algorithm should run in O(n) complexity.

Example:

Input: [100, 4, 200, 1, 3, 2]

Output: 4

Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

【核心代码】

```
class Solution {
public:
    unordered_map<int, int> father;
    int longestConsecutive(vector<int>& nums) {
        unordered_set<int> num;
        for(auto it : nums) num.insert(it);
        unordered_map<int, int> mapp, m;//mapp  哈希映射; m  统计个数;
        for (auto it : num) {
            father[it] = it;
            mapp[it] = 1;
            if (mapp[it + 1]) {
```

```
                Union(it , it+1);
            }
            if (mapp[it - 1]) {
                Union(it, it - 1);
            }
        }
        int maxLen = 0;
        for (auto it : num) {
            int f = findFather(it);
            if (m.find(f) == m.end()) {
               m[f] = 1;
            }
            else {
               m[f]++;
            }
            maxLen = max(maxLen, m[f]);
        }
        return maxLen;
    }
    void Union(int a, int b) {//a > b
        int fa = findFather(a);
        int fb = findFather(b);
        if (fa != fb) father[fa] = fb;
    }
    int findFather(int x) {
        return (x == father[x] ? x : father[x] = findFather(father[x]));
    }
};
```

【思考】还有更简单方法吗？（提示：先找出最小的）

哈夫曼编码

总结

六、 图论

图的遍历和连通分量

- **图的 BFS**

图 bfs 模板

```
void bfs(int s){
    queue<int> q;
    q.push(s);
    inq[s] = true;
    while(!q.empty()){
        int t = q.front()
        q.pop();
        for( t 的邻接点  v){
            if(没有入队){
                q.push(v);
                inq[v] = true;
                ……
            }
        }
    }
}
```

- **图的 DFS**

图 dfs 模板

```
void dfs(int s){
    visit[s] = true;
    for(s 的邻接点  v){
        if(没有访问) dfs(v);
        ……
    }
}
```

- **连通分量**

求连通分量个数

➢ 方法一

```
for(所有结点){
    if(没有访问) dfs(s); // bfs(s);
    cnt++; 求连通分量
}
```

➢ 方法二

并查集：对于每条边的两个端点进行合并，当某个顶点父顶点是本身时，就是一个连通分量。

【思考】如何判断图是连通图？如何遍历所有的边？

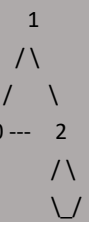
【LeetCode】Clone Graph

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

OJ's undirected graph serialization:

Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.
As an example, consider the serialized graph {0,1,2#1,2#2,2}.
The graph has a total of three nodes, and therefore contains three parts as separated by #.
1. First node is labeled as 0. Connect node 0 to both nodes 1 and 2.
2. Second node is labeled as 1. Connect node 1 to node 2.
3. Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.
Visually, the graph looks like the following:



【核心代码】

```
/**
 * Definition for undirected graph.
 * struct UndirectedGraphNode {
 *     int label;
 *     vector<UndirectedGraphNode *> neighbors;
 *     UndirectedGraphNode(int x) : label(x) {};
 * };
 */
class Solution {
public:
    unordered_map<int,UndirectedGraphNode*> m;
    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
        if(node == NULL) return NULL;
        UndirectedGraphNode* t = NULL;
        if(m.find(node->label) == m.end()){
            t = new UndirectedGraphNode(node->label);
            m[node->label] = t;
            for(auto it : node->neighbors){
                UndirectedGraphNode* temp = cloneGraph(it);
                if(temp) t->neighbors.push_back(temp);
            }
        }else{
            t = m[node->label];
        }
        return t;
    }
};
```

【注意】在遍历时有的 vertex 相关信息已经存在，因此可通过 map 进行查询使用。

【LintCode】Shortest Path in Undirected Graph

Give an undirected graph, in which each edge's length is 1, and give two nodes from the graph. We need to find the length of the shortest path between the given two nodes.

Example :

Given graph = {1,2,4#2,1,4#3,5#4,1,2#5,3}, and nodeA = 3, nodeB = 5.

1-------- 2 3

```
 \       |   |
  \      |   |
   \     |   |
    \    |   |
     \   |   |
       4     5
```

return 1.

【核心代码】

```
/**
 * Definition for Undirected graph.
 * struct UndirectedGraphNode {
 *        int label;
 *        vector<UndirectedGraphNode *> neighbors;
 *        UndirectedGraphNode(int x) : label(x) {};
 * };
 */
class Solution {
public:
    int shortestPath(vector<UndirectedGraphNode*> &graph, UndirectedGraphNode* A,
UndirectedGraphNode* B) {
        unordered_map<UndirectedGraphNode*,bool> inq;
        queue<UndirectedGraphNode*> q;
        q.push(A);
        inq[A] = true;
        UndirectedGraphNode* last = A;
        int ans = 0;
        while(!q.empty()){
            UndirectedGraphNode* f = q.front(); q.pop();
            for(auto it : f->neighbors){
                if(inq[it] == false){
                    if(it == B) return ans + 1;
                    q.push(it);
                    inq[it] = true;
                }
            }
            if(f == last) {
                ans ++;
                last = q.back();
            }
        }
        return 0;
    }
};
```

【PAT】Battle Over Cities

It is vitally important to have all the cities connected by highways in a war. If a city is occupied by the enemy, all the highways from/toward that city are closed. We must know immediately if we need to repair any other highways to keep the rest of the cities connected. Given the map of cities which have all the remaining highways marked, you are supposed to tell the number of highways need to be repaired, quickly.

For example, if we have 3 cities and 2 highways connecting $city_1$-$city_2$ and $city_1$-$city_3$. Then if $city_1$ is occupied by

the enemy, we must have 1 highway repaired, that is the highway $city_2$-$city_3$.

Input Specification:

Each input file contains one test case. Each case starts with a line containing 3 numbers *N* (<1000), *M* and *K*, which are the total number of cities, the number of remaining highways, and the number of cities to be checked, respectively. Then *M* lines follow, each describes a highway by 2 integers, which are the numbers of the cities the highway connects. The cities are numbered from 1 to *N*. Finally there is a line containing *K* numbers, which represent the cities we concern.

Output Specification:

For each of the *K* cities, output in a line the number of highways need to be repaired if that city is lost.

Sample Input:

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

Sample Output:

```
1
0
0
```

【核心代码】

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;
bool inq[1001];
vector <int> v[1001];
void bfs(int k){
	queue<int> q;
	q.push(k);
	inq[k] = true;
	while(!q.empty()){
		int l = q.front();q.pop();
		for(int i = 0 ; i < v[l].size();i ++){
			if(inq[v[l][i]] == false){
				q.push(v[l][i]);
				inq[v[l][i]] = true;
			}
		}
	}
}
int main(){
	ios::sync_with_stdio(false);
	int cityNum,roadNum,cn,cnt = 0;
	cin >> cityNum >> roadNum >> cn;
	for(int i = 0; i < roadNum; i++){
		int t,t1;
		cin >> t >> t1;
		v[t].push_back(t1);
		v[t1].push_back(t);
	}
```

```
    for(int i = 0; i < cn; i++){
        int t;
        cin >> t;
        cnt = 0;
        fill(inq,inq+1001,false);
        inq[t] = true;
        for(int j = 1;    j <= cityNum; j++){
            if(inq[j] == false){
                bfs(j);
                cnt++;
            }
        }
        cout << cnt - 1 << endl;
    }
    return 0;
}
```

【注意】当去除某个顶点时，可以将此顶点视为访问过；当去除某条边时且图以邻接矩阵存储，可以此边权值设为 0。

【LeetCode】Evaluate Division

Equations are given in the format A / B = k, where A and B are variables represented as strings, and k is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return -1.0.

Example:

Given a / b = 2.0, b / c = 3.0.

queries are: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .

return [6.0, 0.5, -1.0, 1.0, -1.0].

The input is: vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries , where equations.size() == values.size(), and the values are positive. This represents the equations. Return vector<double>.

According to the example above:

equations = [["a", "b"], ["b", "c"]],

values = [2.0, 3.0],

queries = [["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"]].

The input is always valid. You may assume that evaluating the queries will result in no division by zero and there is no contradiction.

【核心代码】

```
class Solution {
public:
    unordered_map<string, vector<pair<string, double>>> edge;
    unordered_map<string, bool> v;
    vector<double> res;
    vector<double> calcEquation(vector<pair<string, string>> equations, vector<double>& values,
vector<pair<string, string>> queries) {
        for (int i = 0; i < values.size(); i++) {
            edge[equations[i].first].push_back({ equations[i].second,values[i] });
            edge[equations[i].second].push_back({ equations[i].first,1 / values[i] });
        }
```

```
            for (auto it : queries) {
                double ans = 1;
                int sz = res.size();
                //the first and the second string are in edge.
                if (edge.find(it.first) != edge.end() && edge.find(it.second) != edge.end()) {
                    v[it.first] = true;
                    dfs(it.first, it.second, ans);
                    v[it.first] = false;
                }
                //the first or the second string is not in edge.
                //sz == res.size() -> the first and the second string is not connected.
                if (edge.find(it.first) == edge.end() || edge.find(it.second) == edge.end() || sz == res.size())
res.push_back(-1);
            }
            return res;
        }
        void dfs(string s, string e, double temp) {
            if (s == e) {
                res.push_back(temp);
                return;
            }
            for (auto it : edge[s]) {
                if (v[it.first] == false) {
                    v[it.first] = true;
                    dfs(it.first, e, temp * it.second);
                    v[it.first] = false;
                }
            }
        }
};
```

【LeetCode】Find Eventual Safe States

In a directed graph, we start at some node and every turn, walk along a directed edge of the graph. If we reach a node that is terminal (that is, it has no outgoing directed edges), we stop.

Now, say our starting node is *eventually safe* if and only if we must eventually walk to a terminal node. More specifically, there exists a natural number K so that for any choice of where to walk, we must have stopped at a terminal node in less than K steps.

Which nodes are eventually safe? Return them as an array in sorted order.

The directed graph has N nodes with labels 0, 1, ..., N-1, where N is the length of graph. The graph is given in the following form: graph[i] is a list of labels j such that (i, j) is a directed edge of the graph.

Example:

Input: graph = [[1,2],[2,3],[5],[0],[5],[],[]]

Output: [2,4,5,6]

【分析】安全的点是指从该点出发一定能到达终点。不能到达终点的情况只有出现环的情况。

【核心代码】

- **Approach 1 : DFS**

```
class Solution {
public:
    vector<int> eventualSafeNodes(vector<vector<int>>& graph) {
```

```
        vector<int> res;
        int size = graph.size();
        vector<int> color(size, 0);              // 0: not visited; 1: safe; 2: unsafe.
        for (int i = 0; i < size; ++i) {
            if (dfs(graph, i, color)) {          // the i-th node is safe
                res.push_back(i);
            }
        }
        return res;
    }
private:
    bool dfs(vector<vector<int>> &graph, int start, vector<int> &color) {
        if (color[start] != 0) {
            return color[start] == 1;
        }
        color[start] = 2;              // mark it as unsafe because it is on the path
        for (int next : graph[start]) {
            if (!dfs(graph, next, color)) {
                return false;
            }
        }
        color[start] = 1;              // mark it as safe because no loop is found
        return true;
    }
};
```

- **Approach 2 : we start with nodes that have no outgoing edges - those are eventually safe. Now, we can update any nodes which only point to eventually safe nodes - those are also eventually safe. Then, we can update again, and so on.**

【LeetCode】Keys and Rooms

There are N rooms and you start in room 0. Each room has a distinct number in 0, 1, 2, ..., N-1, and each room may have some keys to access the next room.

Formally, each room i has a list of keys rooms[i], and each key rooms[i][j] is an integer in [0, 1, ..., N-1] where N = rooms.length. A key rooms[i][j] = v opens the room with number v.

Initially, all the rooms start locked (except for room 0).

You can walk back and forth between rooms freely.

Return true if and only if you can enter every room.

Example :

Input: [[1,3],[3,0,1],[2],[0]]

Output: false

Explanation: We can't enter the room with number 2.

【核心代码】

```
class Solution {
public:
    vector<bool> visited;
    bool canVisitAllRooms(vector<vector<int>>& rooms) {
        int n = rooms.size();
```

```
            visited.resize(n,false);
            int cnt = 0;
            for(int i = 0; i < n; i++){
                if(visited[i] == false){
                    cnt++;
                    if(cnt < 2) dfs(rooms, i);
                    else return false;
                }
            }
            return true;
        }
        void dfs(vector<vector<int>>& rooms, int s){
            visited[s] = true;
            for(auto it : rooms[s]){
                if(visited[it] == false) dfs(rooms, it);
            }
        }
};
```

【LeetCode】Reconstruct Itinerary

Given a list of airline tickets represented by pairs of departure and arrival airports [from, to], reconstruct the itinerary in order. All of the tickets belong to a man who departs from JFK. Thus, the itinerary must begin with JFK.

Note:

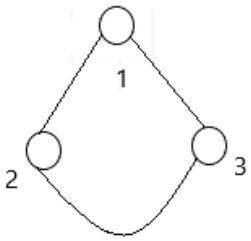
1. If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order
2. when read as a single string. For example, the itinerary ["JFK", "LGA"] has a smaller lexical order than ["JFK", "LGB"].
3. All airports are represented by three capital letters (IATA code).
4. You may assume all tickets form at least one valid itinerary.

【核心代码】

```
class Solution {
public:
    vector<string> findItinerary(vector<pair<string, string>> tickets) {
        vector<string> res;
        unordered_map<string, multiset<string>> m;
        for (auto a : tickets) {
            m[a.first].insert(a.second);
        }
        dfs(m, "JFK", res);
        return vector<string> (res.rbegin(), res.rend());
    }
    void dfs(unordered_map<string, multiset<string>>& m, string s, vector<string>& res) {
        while (m[s].size()) {
            string t = *m[s].begin();
            m[s].erase(m[s].begin());
```

```
                dfs(m, t, res);
            }
            res.push_back(s);
        }
};
```

环的判定



如何判断图是否存在环？有几种方法判定？

- DFS
- BFS
- 并查集
- 拓扑排序（有向图）

✧ DFS：对于某个 vertex 的邻接点，如果这个邻接点已被访问，说明存在环。或者对于某条边的两个端点，从任一端点出发能够通过其他路径到达另一端点，说明存在环。染色法，当进入某个顶点时染 1 号色，当退出时染 2 号色，在访问过程中出现 1 号色，则存在环，且环的顶点和非环的顶点可以记录。

✧

✧

【LeetCode】Redundant Connection

In this problem, a tree is an **undirected** graph that is connected and has **no cycles**.

The given input is a graph that started as a tree with N nodes (with distinct values 1, 2, ..., N), with one additional edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of edges. Each element of edges is a pair [u, v] with u < v, that represents an **undirected** edge connecting nodes u and v.

Return an edge that can be removed so that the resulting graph is a tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array. The answer edge [u, v] should be in the same format, with u < v.

Example 1:

Input: [[1,2], [1,3], [2,3]]

Output: [2,3]

Explanation: The given undirected graph will be like this:

1

/ \

2 - 3

Example 2:

Input: [[1,2], [2,3], [3,4], [1,4], [1,5]]

Output: [1,4]

Explanation: The given undirected graph will be like this:

5 - 1 - 2

| |

4 - 3

Note:

- The size of the input 2D-array will be between 3 and 1000.
- Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

【核心代码】

【LeetCode】Course Schedule

There are a total of *n* courses you have to take, labeled from 0 to n-1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite **pairs**, is it possible for you to finish all courses?

【核心代码】

```
class Solution {
public:
    bool canFinish(int numCourses, vector<pair<int, int>>& prerequisites) {
        vector<vector<int>> g(numCourses);
        vector<int> color(numCourses,0);
        for(auto it : prerequisites){
            g[it.second].push_back(it.first);
        }
        for(int i = 0; i < numCourses; i++){
            if(!dfs(i,g,color)) return false;
        }
        return true;
    }
    bool dfs(int s, vector<vector<int>> &g, vector<int> &color){
        if(color[s] != 0) return color[s] == 2;
        color[s] = 1;
        for(int i = 0; i < g[s].size(); i++){
            int u = g[s][i];
            if(!dfs(u,g,color)) return false;
        }
        color[s] = 2;
        return true;
    }
};
```

最短距离

- Dijkstra

➢ 邻接矩阵版本

```
vector<vector<int>> g(n,vector<int> (n,inf)), visited(n,vector<int> (n,false))
vector<int> d(n,inf);
void dijkstra(int s){
	d[s] = 0;
	visited[s] = true;
	while(1){
		int min = inf, v = -1;
		for(int i = 0; i < n; i++){
			if(visited[i] == false && min < d[i]){
				min = d[i];
				v = i;
			}
		}
		if(v == -1) break;
		visited[v] = true;
		for(int i = 0; i < n; i++){
			if(visited[i] == false && g[v][i] != inf){
				if(d[i] > d[v] +g[v][i]){//松弛条件
					d[i] = d[v] +g[v][i];
				}
			}
		}

}
}
```

➢ 优先队列版本

```
struct edge {int to,cost;};
struct dij{
	int x,d;
	dij(int _x,int _d):x(_x),d(_d){}
	bool operator < (const dij &ano) const { return d > ano.d; }
};
int V;//顶点个数
vector<edge> G[MAXV];
int d[MAXV];
void dijkstra(int s){
	priority_queue<dij > que;//小顶堆
	memset(d,INF,sizeof(d));
	d[s] = 0;
	que.push(dij(s,0)); //把起点推入队列
	while(!que.empty()){
		dij p = que.top(); que.pop();
		int v = p.x; //顶点的编号
```

```
            if (d[v] < p.d) continue;//说明，v 点已经通过其他路径变得松弛，距离更短。而 p.d 只是之前入队的旧元素
            for(int i = 0; i < G[v].size(); i++){
                edge e = G[v][i];
                if (d[e.to] > d[v] + e.cost) {
                    d[e.to] = d[v] + e.cost;
                    que.push(dij(e.to,d[e.to]));
                }
            }
        }
}
```

【LeetCode 原题】Reachable Nodes In Subdivided Graph

Starting with an **undirected** graph (the "original graph") with nodes from `0` to `N-1`, subdivisions are made to some of the edges.

The graph is given as follows: `edges[k]` is a list of integer pairs `(i, j, n)` such that `(i, j)` is an edge of the original graph,

and `n` is the total number of **new** nodes on that edge.

Then, the edge `(i, j)` is deleted from the original graph, `n` new nodes `(x_1, x_2, ..., x_n)` are added to the original graph,

and `n+1` new edges `(i, x_1), (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n), (x_n, j)` are added to the original graph.

Now, you start at node `0` from the original graph, and in each move, you travel along one edge.

Return how many nodes you can reach in at most `M` moves.

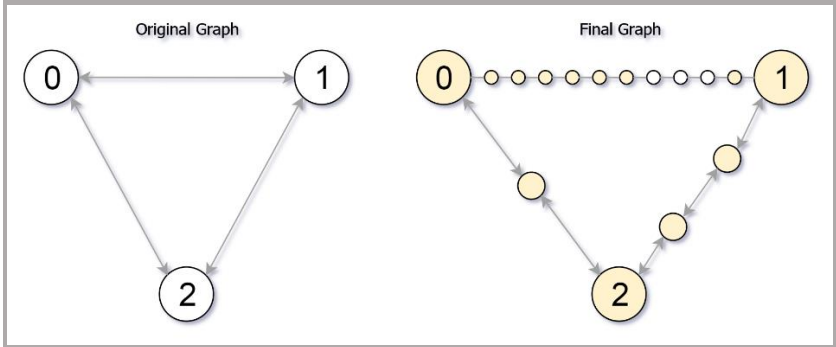
Example 1:

Input: `edges = [[0,1,10],[0,2,1],[1,2,2]], M = 6, N = 3`

Output: `13`

Explanation:

The nodes that are reachable in the final graph after M = 6 moves are indicated below.



Example 2:

Input: `edges = [[0,1,4],[1,2,6],[0,2,8],[1,3,1]], M = 10, N = 4`

Output: `23`

【核心代码】

```
#define inf INT_MAX
struct dij {
    int x, d;
    dij(int _x, int _d) : x(_x), d(_d) {}
```

```
    bool operator < (const dij &ano) const {
        return d > ano.d;
    }
};
class Solution {
public:
    int reachableNodes(vector<vector<int>>& edges, int M, int N) {
        vector<vector<pair<int, int>>> adj(N); // {to,cost}
        vector<int> d(N, inf);
        map<pair<int, int>, int> lr, rl, cost;
        for (int i = 0; i < edges.size(); i++) {
            int x = edges[i][0], y = edges[i][1], c = edges[i][2];//x < y
            adj[x].push_back(make_pair(y, c));
            adj[y].push_back(make_pair(x, c));
            pair<int, int> p = make_pair(x, y);
            cost[p] = c;
            lr[p] = 0;
            rl[p] = 0;
        }
        priority_queue<dij> pq;
        d[0] = 0
        pq.push(dij(0, 0));
        while (!pq.empty()) {
            dij temp = pq.top();pq.pop();
            if (temp.d > d[temp.x] || temp.d > M) continue;
            int x = temp.x;
            for (int i = 0; i < adj[x].size(); i++) {
                int y = adj[x][i].first, c = adj[x][i].second;
                int nd = temp.d + c + 1;
                pair<int, int> p = ((x < y) ? make_pair(x, y) : make_pair(y, x));
                if (nd <= M) {
                    lr[p] = rl[p] = c;
                    if (d[y] > nd) {
                        d[y] = nd;
                        pq.push(dij(y, nd));
                    }
                }
                else {
                    int left = M - temp.d;
                    if (x < y) lr[p] = max(lr[p], left);
                    else rl[p] = max(rl[p], left);
                }
            }
        }
        int ans = 0;
        for (int i = 0; i < N; i++) {
```

```
                if (d[i] < inf) ans++;
            }
            for (auto it = lr.begin(); it != lr.end(); it++) {
                int l = it->second, r = rl[it->first];
                ans += min(l + r, cost[it->first]);
            }
            return ans;
        }
};
```

【PAT】Emergency

As an emergency rescue team leader of a city, you are given a special map of your country. The map shows several scattered cities connected by some roads. Amount of rescue teams in each city and the length of each road between any pair of cities are marked on the map. When there is an emergency call to you from some other city, your job is to lead your men to the place as quickly as possible, and at the mean time, call up as many hands on the way as possible.

Input

Each input file contains one test case. For each test case, the first line contains 4 positive integers: N (<= 500) - the number of cities (and the cities are numbered from 0 to N-1), M - the number of roads, C1 and C2 - the cities that you are currently in and that you must save, respectively. The next line contains N integers, where the i-th integer is the number of rescue teams in the i-th city. Then M lines follow, each describes a road with three integers c1, c2 and L, which are the pair of cities connected by a road and the length of that road, respectively. It is guaranteed that there exists at least one path from C1 to C2.

Output

For each test case, print in one line two numbers: the number of different shortest paths between C1 and C2, and the maximum amount of rescue teams you can possibly gather.\ All the numbers in a line must be separated by exactly one space, and there is no extra space allowed at the end of a line.

Sample Input

```
5 6 0 2
1 2 1 5 3
0 1 1
0 2 2
0 3 1
1 2 1
2 4 1
3 4 1
```

Sample Output

```
2 4
```

【核心代码】

```
struct node {
    int to, cost;
};
struct dij {
    int x, d;
    dij(int _x, int _d):x(_x),d(_d){}
```

```
	bool operator < (const dij &ano) const { return d > ano.d; }
};
vector<int> rt;
vector<vector<node>> edge;
vector<int> d;
vector<int> sp;
vector<int> mc;
int main() {
	int vn, en, s, e;
	cin >> vn >> en >> s >> e;
	d.resize(vn, inf);
	sp.resize(vn,0);
	mc.resize(vn, 0);
	rt.resize(vn);
	edge.resize(vn);
	for (auto &it : rt) {
		cin >> it;
	}
	for(int i = 0; i <en; i++){
		int c1, c2, cost;
		cin >> c1 >> c2 >> cost;
		edge[c1].push_back({ c2,cost });
		edge[c2].push_back({ c1,cost });
	}
	priority_queue<dij> q;
	q.push({ s,0 });
	d[s] = 0;
	sp[s] = 1;
	mc[s] = rt[s];
	while (!q.empty()) {
		dij temp = q.top(); q.pop();
		int u = temp.x;
		if (d[u] < temp.d) continue;
		for (int i = 0; i < edge[u].size(); i++) {
			int v = edge[u][i].to, cost = edge[u][i].cost;
			if (d[v] > d[u] + cost) {
				d[v] = d[u] + cost;
				sp[v] = sp[u];
				mc[v] = mc[u] + rt[v];
				q.push({ v,d[v] });
			}
			else if(d[u] + cost == d[v]) {
				sp[v] += sp[u];
				mc[v] = max(mc[u] + rt[v], mc[v]);
			}
		}
```

```
    }
    cout << sp[e] << " " << mc[e];
    system("pause");
    return 0;
}
```

- Floyd

Floyd 算法即全源最短路径算法，即给定任意起点、终点能给出最短路径，时间复杂度 $O(n^3)$。

```
int dis[maxv][maxv];
void floyd(){
    for(int k = 0; k < n; k++){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                if(dis[i][k] < inf && dis[k][j] < inf && dis[i][k] + dis[k][j] < dis[i][j]){
                    dis[i][j] = dis[i][k] + dis[k][j];
                }
            }
        }
    }
}
```

最小生成树（最小支撑树）

- Prim

从图中任选一顶点放入树顶点集中，然后不断从树顶点集到剩余图顶点中找距离最小的顶点加入，同时加入时会更新最短距离。

```
const int INF = 65535;
const int MAXV = 100000;
struct edge { int to, cost; };
struct Prim {
    int x, d;
    Prim(int _x, int _d) :x(_x), d(_d) {}
    bool operator < (const Prim &ano) const { return d > ano.d; }
};
vector<edge> G[MAXV];
vector<int> d(MAXV, INF);//树顶点集到各个顶点的最短距离
int prim(int s) {
    int ans = 0;
    priority_queue<Prim> que;//小顶堆
    d[s] = 0;
    que.push(Prim(s, 0)); //把起点推入队列
    while (!que.empty()) {
        Prim p = que.top(); que.pop();
        int v = p.x; //顶点的编号
```

```
        if (d[v] < p.d ) continue;//说明，v 点已经通过其他顶点变得松弛，距离更短。而 p.d 只是之前入队的旧元素
        ans += d[v];
        d[v] = 0;//说明 v 已经加入到树顶点集当中
        for (int i = 0; i < G[v].size(); i++) {
            edge e = G[v][i];
            if (d[e.to] > e.cost) {
                d[e.to] = e.cost;
                que.push(Prim(e.to, d[e.to]));
            }
        }
    }
    return ans;
}
```

- Kruskal

```
const int maxv = 110;
const int maxe = 1110;
int father[maxv];
struct edge {
    int u, v;
    int weight;
}e[maxe];
bool cmp(edge a, edge b) {
    return a.weight < b.weight;
}
int findFather(int x) {
    return (x == father[x] ? x : ( father[x] = findFather(father[x])));
}
int kruskal(int n, int m) {
    int ans = 0, num = 0;
    //初始化并查集
    for (int i = 0; i < n; i++)  {
        father[i] = i;
    }
    sort(e, e + n, cmp);
    for (int i = 0; i < m; i++) {
        //union
        int fa = findFather(e[i].u);
        int fb = findFather(e[i].v);
        if (fa != fb) {
            father[fa] = fb;
            num++;
            ans += e[i].weight;
            if (num == n - 1) break;
        }
    }
```

```
    if (num != n - 1) return -1;
    return ans;
}
```

拓扑排序

```
int indegree[maxn];
vector<vector<int>> g(n);
bool topOrder(){
    int num = 0;
    queue<int> q;
    for(int i = 0; i < n; i ++){
        if(indegree[i] == 0) q.push(i);
    }
    while(!q.empty()){
        int t = q.front();
        q.pop();
        for(int i = 0; i < v[t].size(); i++){
            int v = v[t][i];
            indegree[v] --;
            if(indegree[v] == 0) q.push(v);
        }
        num++;
    }
    if(n == num) return true;
    else return fasle;
}
```

【思考】如何求得所有的拓扑序列？

【PAT】Topological Order

This is a problem given in the Graduate Entrance Exam in 2018: Which of the following is NOT a topological order obtained from the given directed graph? Now you are supposed to write a program to test each of the options.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers N (≤ 1,000), the number of vertices in the graph, and M (≤ 10,000), the number of directed edges. Then M lines follow, each gives the start and the end vertices of an edge. The vertices are numbered from 1 to N. After the graph, there is another positive integer K (≤ 100). Then K lines of query follow, each gives a permutation of all the vertices. All the numbers in a line are separated by a space.

Output Specification:

Print in a line all the indices of queries which correspond to "NOT a topological order". The indices start from zero. All the numbers are separated by a space, and there must no extra space at the beginning or the end of the line. It is graranteed that there is at least one answer.

Sample Input:

6 8

1 2

1 3

5 2

```
5 4
2 3
2 6
3 4
6 4
5
1 5 2 3 6 4
5 1 2 6 3 4
5 1 2 3 6 4
5 2 1 6 3 4
1 2 3 4 5 6
```

Sample Output:

```
3 4
```

【核心代码】

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;
int main() {
	int n, m, check;
	cin >> n >> m;
	vector<int> e[1001], res;
	vector<int> indegree(1001 ,0 ),tempdegree;
	for (int i = 0; i < m; i++) {
		int c, s;
		cin >> c >> s;
		e[c].push_back(s);
		indegree[s]++;
	}
	cin >> check;
	for (int i = 0; i < check; i++) {
		tempdegree = indegree;
		int flag = true;
		for (int j = 0; j < n; j++) {
			int temp;
			cin >> temp;
			if (tempdegree[temp] == 0) {
				for (int k = 0; k < e[temp].size(); k++) {
					tempdegree[e[temp][k]]--;
				}
			}
			else {
				flag = false;
			}
```

```
            }
            if (!flag) res.push_back(i);
        }
        cout << res[0];
        for (int i = 1; i < res.size(); i++) {
            cout << " " << res[i];
        }
        return 0;
}
```

【LeetCode】Course Schedule II

There are a total of *n* courses you have to take, labeled from 0 to n-1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite **pairs**, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

【核心代码】

```
class Solution {
public:
    vector<int> findOrder(int numCourses, vector<pair<int, int>>& prerequisites) {
        vector<vector<int>> g(numCourses);
        vector<int> indegree(numCourses,0);
        vector<int> order;
        for(auto it : prerequisites){
            g[it.second].push_back(it.first);
            indegree[it.first]++;
        }
        queue<int> q;
        for(int i = 0; i < numCourses; i++){
            if(indegree[i] == 0) q.push(i);
        }
        while(!q.empty()){
            int t = q.front(); q.pop();
            order.push_back(t);
            for(auto it : g[t]){
                if(indegree[it] > 0){
                    indegree[it]--;
                    if(indegree[it] == 0) q.push(it);
                }
            }
        }
        return ( order.size() == numCourses ? order : vector<int> {});
```

```
    }
};
```

【LeetCode】Sequence Reconstruction

Check whether the original sequence org can be uniquely reconstructed from the sequences in seqs. The org sequence is a permutation of the integers from 1 to n, with $1 \le n \le 10^4$. Reconstruction means building a shortest common supersequence of the sequences in seqs (i.e., a shortest sequence so that all sequences in seqs are subsequences of it). Determine whether there is only one sequence that can be reconstructed from seqs and it is the org sequence.

Example 1:

Given org = [1,2,3], seqs = [[1,2],[1,3]]

Return false

Explanation:

[1,2,3] is not the only one sequence that can be reconstructed, because [1,3,2] is also a valid sequence that can be reconstructed.

Example 2:

Given org = [1,2,3], seqs = [[1,2]]

Return false

Explanation:

The reconstructed sequence can only be [1,2].

Example 3:

Given org = [1,2,3], seqs = [[1,2],[1,3],[2,3]]

Return true

Explanation:

The sequences [1,2], [1,3], and [2,3] can uniquely reconstruct the original sequence [1,2,3].

Given org = [4,1,5,2,6,3], seqs = [[5,2,6,3],[4,1,5,2]]

Return true

【核心代码】

```
class Solution {
public:
    bool sequenceReconstruction(vector<int> &org, vector<vector<int>> &seqs) {
        int cnt = 0;
        vector<unordered_set<int>> g(org.size() + 1);
        vector<int> indegree(org.size() + 1, -1);
        queue<int> q;
        for (auto it : seqs) {//构造 Graph
            for (int i = 0; i < it.size(); i++) {
                if (it[i] > org.size() || it[i] < 1) return false;//出现不应出现的数字
                if (indegree[it[i]] == -1) indegree[it[i]] = 0;//初始化每个顶点入度
                if (i > 0) {
                    if (!g[it[i - 1]].count(it[i])) indegree[it[i]]++;
                    g[it[i - 1]].insert(it[i]);
                }
            }
```

```
            }
            for (int i = 1; i <= org.size(); i++) {
                if (indegree[i] == 0) {
                    q.push(i);
                    if (q.size() > 1) return false;//为了保证唯一，每次只能插入一个
                }
            }
            while (!q.empty()) {
                int t = q.front(); q.pop();
                if (t != org[cnt++]) return false;//不符合当前拓扑序列
                for (auto it : g[t]) {
                    if (--indegree[it] == 0) {
                        q.push(it);
                        if (q.size() > 1) return false;//为了保证唯一，每次只能插入一个
                    }
                }
            }
            return cnt == org.size();//判断构造的图是否能构唯一成要求的拓扑序列
        }
};
```

【另解】

```
class Solution {
public:
    bool sequenceReconstruction(vector<int>& org, vector<vector<int>>& seqs) {
        unordered_map<int, int> pos, pre;
        for (int i = 0; i < org.size(); ++i) pos[org[i]] = i;
        for (auto& seq : seqs) {
            for (int i = 0; i < seq.size(); ++i) {
                if (pos.find(seq[i]) == pos.end()) {
                    return false;
                }
                if (i > 0 && pos[seq[i - 1]] >= pos[seq[i]]) {
                    return false;
                }
                if (pre.find(seq[i]) == pre.end()) {
                    pre[seq[i]] = (i > 0) ? pos[seq[i - 1]] : -1;
                } else {
                    pre[seq[i]] = max(pre[seq[i]], (i > 0) ? pos[seq[i - 1]] : -1);
                }
            }
        }
        for (int i = 0; i < org.size(); ++i) {
            if (pre[org[i]] != i - 1)
                return false;
```

```
            }
            return true;
        }
};
```

【LintCode】Alien Dictionary

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where words are **sorted lexicographically by the rules of this new language**. Derive the order of letters in this language.

Example

Given the following words in dictionary,

["wrt","wrf","er","ett","rftt"]

The correct order is: "wertf"

Given the following words in dictionary,

["z","x"]

The correct order is: "zx".

Notice

1. You may assume all letters are in lowercase.
2. You may assume that if a is a prefix of b, then a must appear before b in the given dictionary.
3. If the order is invalid, return an empty string.
4. There may be multiple valid order of letters, return the smallest in lexicographical order

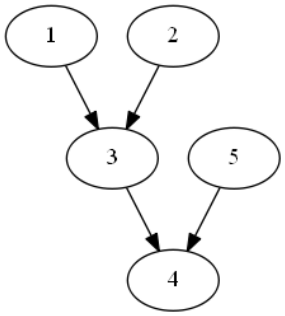
【核心代码】

```
class Solution {
public:
    string alienOrder(vector<string> &words) {
        unordered_map<char, vector<string>> origin;
        origin['0'] = words;
        unordered_map<char, set<char>> graph;
        unordered_map<char, int> indegree;
        buildGraph(origin, graph, indegree);
        string res = "";
        set<char> q;
        for (auto it : indegree) {
            if (it.second == 0) q.insert(it.first);
        }
        while (!q.empty()) {
            char f = *q.begin(); q.erase(q.begin());
            res += f;
            for (auto it : graph[f]) {
                if (--indegree[it] == 0) {
                    q.insert(it);
                }
            }
        }
```

```
        return res.size() == indegree.size() ? res : "";
    }
    void buildGraph(unordered_map<char, vector<string>> &w, unordered_map<char, set<char>> &graph,
unordered_map<char, int> &indegree) {
        unordered_map<char, vector<string>> next;
        for (auto it : w) {
            vector<string> &u = it.second;
            for (int i = 0; i < u.size(); i++) {
                if (indegree.find(u[i][0]) == indegree.end()) indegree[u[i][0]] = 0;
                if (u[i].length() > 1 && u.size() > 1) next[u[i][0]].push_back(u[i].substr(1));
                if (i > 0) {
                    if (u[i][0] != u[i - 1][0]) {
                        graph[u[i - 1][0]].insert(u[i][0]);
                        indegree[u[i][0]]++;
                    }
                }
            }
        }
        if (next.size() != 0) buildGraph(next, graph, indegree);
    }
};
```

有一个 n 个点，m 条边的有向无环图（即对于任意点 v，不存在从点 v 开始、点 v 结束的路径）。为了方便，点用 1,2,⋯,n 编号。设 count(x,y) 表示点 x 到点 y 不同的路径数量（规定 count(x,x)=0），求 $\sum_{i=1}^{n}\sum_{j=1}^{n} count(i,j)a_i b_j$ 除以 (10^9+7) 的余数。其中，a_i,b_j 是给定的数列。

【分析】



ans = ans + a[u] * b[v]

a[v] = a[u] + a[v]

【核心代码】

```
void solve() {
    for(int i = 1; i <= n; i++)
        if(in[i] == 0) que.push(i);
    while(!que.empty()) {
        int u = que.front(); que.pop();
        for(int i = head[u]; ~i; i = edge[i].nxt) {//遍历邻接点
            int v = edge[i].v;
            ans = (ans + a[u] * b[v] % MOD) % MOD;
            a[v] = (a[u] + a[v]) % MOD;
```

```
                --in[v];
                if(in[v] == 0) que.push(v);
            }
        }
}
```

关键路径

```
struct node {
    int v, w;
};
const int maxn = 1 << 20;
int indegree[maxn];
vector<vector<node>> g(n);
vector<int> ve(n), vl(n),temppath;//最早开始 ve,最迟开始 vl;
vector<vector<int>> criticalPath, criticalActivity(n);
stack<int> toporder;
bool topOrder() {
    queue<int> q;
    for (int i = 0; i < n; i++) {
        if (indegree[i] == 0) q.push(i);
    }
    while (!q.empty()) {
        int t = q.front();
        q.pop();
        toporder.push(t);
        for (int i = 0; i < g[t].size(); i++) {
            int v = g[t][i].v;
            indegree[v] --;
            if (indegree[v] == 0) q.push(v);
            ve[v] = max(ve[v], ve[t] + g[t][i].w);
        }
    }
    if (toporder.size() == n) return true;
    else return false;
}
void criticlaPath() {
    topOrder();
    fill(vl.begin(), vl.end(), ve[n - 1]);//已知汇点为 n，源点 0
        while (!toporder.empty()) {
            int u = toporder.top();
            toporder.pop();
            for (int i = 0; i < g[u].size(); i++) {
                int v = g[u][i].v, w = g[u][i].w;
                vl[u] = min(vl[u], vl[v] - w);//画图记忆
            }
        }
    for (int u = 0; u < n; u++) {
```

```
            for (int j = 0; j < g[u].size(); j++) {
                int v = g[u][j].v, w = g[u][j].w;
                int e = ve[u], l = ve[l] - w;
                if (e == l)    criticalActivity[u].push_back(v);
            }
        }
        dfs(0);
}
void dfs(int s) {
        temppath.push_back(s);
        if (s == n - 1) {
            criticalPath.push_back(temppath);
        }
        for (int i = 0; i < criticalActivity[s].size(); i++) dfs(criticalActivity[s][i]);
        temppath.pop_back();
}
```

dp 版本见第五章

【LeetCode 原题】Longest Increasing Path in a Matrix

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:

```
Input: nums =
[
  [9,9,4],
  [6,6,8],
  [2,1,1]
]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

Example 2:

```
Input: nums =
[
  [3,4,5],
  [3,2,6],
  [2,2,1]
]
Output: 4
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.
```

【核心代码】

```
typedef pair<int, int> pint;
class Solution {
public:
        map<pint, vector<pint>> mapp;
        map<pint,int> dp;
```

```
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        int ans = -1;
        int direction[][2] = { { 0,1 },{ 1,0 },{ 0,-1 },{ -1,0 } };
        int n = matrix.size();
        if(n == 0) return 0;
        int m =    m = matrix[0].size();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                for (int k = 0; k < 4; k++) {
                    dp[{i, j}] = 1;
                    int x = j + direction[k][0], y = i + direction[k][1];
                    if (isValid(y, x, n, m)) {
                        if (matrix[i][j] < matrix[y][x]) {
                            pint p = make_pair(i, j);
                            if (mapp.find(p) == mapp.end()) {
                                mapp[p] = vector<pint> { pint(y,x) };
                            }
                            else {
                                mapp[p].push_back(pint(y,x));
                            }
                        }
                    }
                }
            }
        }
        for (auto it : dp) ans = max (ans, DP(it.first));
        return ans;
    }
    bool isValid(int x, int y, int n, int m) {
        if (x < 0 || x >= n || y < 0 || y >= m) return false;
        return true;
    }
    int DP(pint i) {
        if (dp[i] > 1)    return dp[i];
        for (auto it : mapp[i]) {
            int temp = 1 + DP(it);
            if (temp > dp[i]) {
                dp[i] = temp;
            }
        }
        return dp[i];
    }
};
```

割集

连通图割集：S 是连通图 G 的一个子边集，若去掉 S 能将 G 分成两连通图，且少去掉任一条仍使 G 连通。

【原创】Cut Set

Given a undirectgraph with n edges and k queries, check each query is a cut set ? You may assume the graph is connected.

Example:

Input:

4

1-2 2-3 3-4 4-1

3

1-2 2-3

1-2

1-2 2-3 4-1

非连通图割集：S 是非连通图 G 的一个子边集，且 G 的连通分量数为 K，若去掉 S 能将 G 分成 K+1 连通图，且少去掉任一条仍使 G 的连通分量数为 K。

常见的图论模板题

- 二分图

【LeetCode】Is Graph Bipartite?

Given an undirected graph, return true if and only if it is bipartite.

Recall that a graph is bipartite if we can split it's set of nodes into two independent subsets A and B such that every edge in the graph has one node in A and another node in B.

The graph is given in the following form: graph[i] is a list of indexes j for which the edge between nodes i and j exists. Each node is an integer between 0 and graph.length - 1. There are no self edges or parallel edges: graph[i] does not contain i, and it doesn't contain any element twice.

Example 1:

Input: [[1,3], [0,2], [1,3], [0,2]]

Output: true

Explanation:

The graph looks like this:

0---- 1

| |

| |

3---- 2

We can divide the vertices into two groups: {0, 2} and {1, 3}.

Example 2:

Input: [[1,2,3], [0,2], [0,1,3], [0,2]]

Output: false

Explanation:

The graph looks like this:

0----1

| \ |

| \ |

3----2

We cannot find a way to divide the set of nodes into two independent subsets.

➢ 方法一：染色法。对于某个 vertex 来说，如果将其染色为 1 号色，那么其所有 adjacent vertex 都应染成 – 1 号色。当发现某个 vertex 的 adjacent vertex 染了相同色，则说明不能二分。

【核心代码】

```
class Solution {
public:
    bool isBipartite(vector<vector<int>>& graph) {
        int n = graph.size();
        vector<int> color(n,0);
        for(int i = 0; i < n; i++){
            if(color[i]) continue;
            color[i] = 1;
            if(!dfs(i,graph,color)) return false;
        }
        return true;
    }
```

```
    bool dfs(int u, vector<vector<int>> &g, vector<int> &color){
        for(auto &v : g[u]){
            if(color[v]){
                if(color[v] != -color[u]) return false;
            }
            else{
                color[v] = -color[u];
                if(!dfs(v,g,color)) return false;
            }
        }
        return true;
    }
};
```

- 方法二：匈牙利算法

- **最大流**

第五章 动态规划

1. 01 背包(0-1 Knapsack Problem)

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

2. 部分数的和（Subset Sum）

Given a array [a1, a2, a3, … ,an], find the subarray which has the target sum.

Example: Input: [1, 5, 11, 5], 11

Output: [[1, 5, 5] ,[11]]

Explanation: The array can be partitioned as [1, 5, 5] and [11].

【核心代码】

3. 组个数（Forming A Number）

Given n numbers {a1, a2, a3, … an}, we need to tell the total number of ways we can form a number 'N' using the sum of the given numbers. (allowing **repetitions** and **different arrangements**).

Example: Input: [1, 3, 5], 6

Output: 8

【核心代码】

```
class Solution{
public:
    int formingANumberWays(vector<int> &nums, int num) {
        vector<int> dp(num + 1, 0);
        dp[0] = 1;
        for (int i = 1; i <= num; i++) {
            for (auto it : nums) {
                if (i - it >= 0) dp[i] += dp[i - it];
            }
        }
        return dp[num];
    }
};
```

4. 最长公共子序列（Longest Common Subsequence）

Given a sequence X and a sequence Y, find the Longest Common Subsequence.

Example:

Input : [abbcdab], [addba]

Output:[aba]

Note: [adb],[ada] are also valid answers.

【分析】这是一道非常经典动态规划题，这个题有两个难点：1.如何求得最长公共子序列的长度 2.如何在求得如何求得最长公共子序列的长度过程中将最长公共子序列相关信息保存下来。

首先看第一个问题求最长公共子序列的长度，令 dp[i][j] 表示 X 序列前 i 位和 Y 序列前 j 位的 LCS 长

度，那么 dp[i][j]相关的递推式可以表达为 $dp[i][j]=\begin{cases} dp[i-1][j-1]+1, X[i\text{-}1]=Y[j\text{-}1] \\ \max(dp[i-1][j], dp[i][j\text{-}1]), X[i\text{-}1]\neq Y[j\text{-}1] \end{cases}$，

边界条件为 dp[0][j] = dp[i][0] = 0。

再看第二个问题，在动态规划决策过程中有三个决策量 dp[i-1][j-1] + 1，dp[i-1][j]，dp[i][j-1]，只要我们在决策过程中将此信息记录下来就能将 LCS 求出来。比如说：dp[i][j] = dp[i-1][j-1] + 1 这一决策记为 1，dp[i][j] = dp[i-1][j] (dp[i-1][j] > dp[i][j-1]) 这一决策记为 2，dp[i][j] = dp[i][j-1] (dp[i-1][j] < dp[i][j-1]) 这一决策记为 3，dp[i][j] = dp[i-1][j] (dp[i-1][j] == dp[i][j-1]) 这一决策记为 4。根据这一信息就能通过回溯得到 LCS。

LCS Dynamic Programing

	i	0	1	2	3	4	5	6	7
j			a	b	b	c	d	a	b
0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1	a	0,0	1,1	1,2	1,2	1,2	1,2	1,1	1,2
2	d	0,0	1,3	1,4	1,4	1,4	2,1	2,2	2,2
3	d	0,0	1,3	1,4	1.4	1,4	2,1	2,4	2,4
4	b	0,0	1,3	2,1	2,1	2,2	2,4	2,4	3,1
5	a	0,0	1,1	2,3	2,4	2,4	2,4	3,1	3,4
Result		adb							
		aba							
		ada							

【核心代码】

```
struct dplcs {
        int d ,r;
};
class Solution {
public:
        set<string> s;
        string lcs;
        set<string> getLcs(string x, string y) {
                int n = x.length(), m = y.length();
                vector<vector<dplcs> > dp(n + 1, vector<dplcs>(m + 1, { 0,0 }));
                for (int i = 1; i <= n; i++) {
                        for (int j = 1; j <= m; j++) {
                                if (x[i - 1] == y[j -1]) {
                                        dp[i][j].d = dp[i - 1][j - 1].d + 1;
                                        dp[i][j].r = 1;
                                }
                                else if (dp[i - 1][j].d > dp[i][j - 1].d) {
                                        dp[i][j].d = dp[i - 1][j].d;
                                        dp[i][j].r = 2;
                                }
                                else if(dp[i - 1][j].d < dp[i][j - 1].d){
                                        dp[i][j].d = dp[i][j - 1].d;
                                        dp[i][j].r = 3;
```

```
                }
                else {
                    dp[i][j].d = dp[i - 1][j].d;
                    dp[i][j].r = 4;
                }
            }
        }
        getLcs(dp, n, m, x);
        return s;
    }
    void getLcs(vector<vector<dplcs> > &dp, int i, int j, string &x) {
        if (i == 0 || j == 0) { s.insert(lcs); return; }
        if (dp[i][j].r == 1) {
            lcs += x[i - 1];
            getLcs(dp, i - 1, j - 1, x);
        }
        else if (dp[i][j].r == 2) {
            getLcs(dp, i - 1, j, x);
        }
        else if (dp[i][j].r == 3) {
            getLcs(dp, i, j - 1, x);
        }
        else {
            string temp = lcs;
            getLcs(dp, i - 1, j, x);
            //回溯
            lcs = temp;
            getLcs(dp, i, j - 1, x);
        }
    }
};
```

5. 最长递增/递减子序列(Longest Increasing/Non-Increasing Subsequence)

Given a array [a1, a2, a3, … ,an], find the longest order subarray.

Example:

Input : [1, 6, 3, 4, 7, 5, 8, 9, 2]

Output:[1, 3, 4, 7, 8, 9]

Note: [1, 3, 4, 5, 8, 9] is also a valid answer.

【分析】Let arr[0..n-1] be the input array and L(i) be the length of the LIS ending at index i such that arr[i] is the last element of the LIS.

Then, L(i) can be recursively written as:

L(i) = 1 + max(L(j)) where 0 < j < i and arr[j] < arr[i]; or

L(i) = 1, if no such j exists.

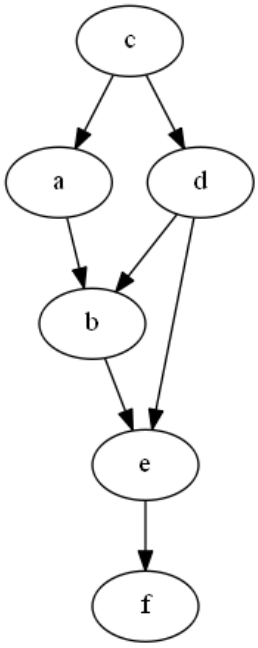
To find the LIS for a given array, we need to return max(L(i)) where 0 < i < n.

arr	1	6	3	4	7	5	8	9	2
LIS	1	2	2	3	3	4	5	6	2

【核心代码】

```
for (int i = 0; i < num; i++) {
        dp[i] = 1;
        for (int j = 0; j < i; j++)
                f (a[i] > a[j])
                        dp[i] = max(dp[i], dp[j] + 1);
        maxn = max(dp[i], maxn);
}
```

6. DAG 最长路



关键路径又被称为 DAG 最长路，DP 解决 DAG 最长路很简洁，也容易理解。

dp[i] 表示从 i 出发能够到达的最长有向路径长度，递推方程为

$$dp[i]=\begin{cases} dp[i], dp[i] >= dp[j]+e[i][j] \\ dp[j]+e[i][j], dp[i] < dp[j]+e[i][j] \end{cases}$$

而 dp[j]一般都需要递归实现。

左图中的最长路径为 c->a->b->e->f 和 c->d->b->e->f。

【核心代码】

```
int DP(int i){
        if(dp[i] > 0) return dp[i];
        for(int j = 0; j < n; j++){
                if(g[i][j] < inf){
                        int temp = DP(j) + g[i][j];
                        if(temp > dp[i] ){
                                dp[i] = temp;
                                pre[i].clear();
                                pre[i].push_back(j);
                        }else if(temp == dp[i]){
                                pre[i].push_back(j);
                        }
                }
        }
}
```

【注意】很多问题都能转化成 DAG 最长路问题，这要求我们思考过程中思路要灵活。

7. 最长的回文串 （Longest Palindromic Substring）

Given a string S, find the Longest Palindromic Substring.

Example:

Input : [abbcdab]

Output:[bb]

【核心代码】

```
class Solution {
public:
    string longestPalindrome(string s) {
        int n = s.length(), index = 0, max = 1;
        vector<vector<int>> dp(n,vector<int>(n,0));
        for(int i = 0; i < n ;i++){
            dp[i][i] = 1;
            if(i < n - 1 &&s[i] == s[i+1]){
                dp[i][i+1] = 1;
                max = 2;
                index = i;
            }
        }
        for(int l = 3; l <= n; l++){
            for(int i = 0; i + l - 1 < n; i++){
                int j = i + l -1;
                if(s[i] == s[j] && dp[i+1][j-1] == 1){
                    dp[i][j] = 1;
                    index = i;
                    max = l;
                }
            }
        }
        return s.substr(index,max);
    }
};
```

8. 最大连续子序列的和

Given a array [a1, a2, a3, … ,an], find the Maximum sum of the continutious subarray.

Example:

Input : [-2,1,-3,4,-1,2,1,-5,4]

Output:6

【核心代码】

```
class Solution {
public:
    int maximumSumOfSublist(vector<int> &nums) {
        int n = nums.size(), maxSum = INT_MIN;
        vector<int> dp(nums);
        dp[0] = nums[0];
        for (int i = 1; i < n; i++) {
```

```
                dp[i] = max(dp[i], dp[i - 1] + nums[i]);
                maxSum = max(maxSum, dp[i]);
            }
            return maxSum > dp[0] ? maxSum :dp[0];
        }
};
```

9. 不同的二叉树

Given n, how many structurally unique BST's (binary search trees) that store values 1 ... n?

10. 最长的斐波那契序列

【LeetCode 原题】Longest Fibonacci sequence

If the sequence `X_1, X_2, ..., X_n` satisfies the following conditions, it is said to be *Fibonacci* :

- `n >= 3`
- For all `i + 2 <= n`, there is `X_i + X_{i+1} = X_{i+2}`

Given a **strictly increasing** positive integer array formation sequence, found `A` that the length of the sub-strata longest Fibonacci sequence. Returns 0 if one does not exist.

(Recall that the original sequence is a sequence `A` derived out of it from the `A` delete any number of elements (or may not be deleted) without altering the order of the remaining elements, for example, `[3, 5, 8]` is `[3, 4, 5, 6, 7, 8]` a sub-sequence)

Example 1:

```
Input: [1,2,3,4,5,6,7,8]
 Output: 5
 Explanation: The longest Fibonacci subsequence is: [1,2,3,5,8].
```

Example 2:

```
Input: [1,3,7,11,12,14,18]
 Output: 3
 Explanation : The longest Fibonacci subsequences are: [1,11,12],[3,11,14] and
[7,11,18].
```

Prompt:

- `3 <= A.length <= 1000`
- `1 <= A[0] < A[1] < ... < A[A.length - 1] <= 10^9`
- *(For Java, C, C++, and C# submissions, the time limit is reduced by 50%)*

【核心代码】

```
class Solution {
public:
    int lenLongestFibSubseq(vector<int>& a) {
        int n = a.size();
        if (n < 1) return 0;
        vector<vector<int>> v = vector<vector<int>>(n + 1, vector<int>(n + 1, 0));
        int ret = 0;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                v[i][j] = 2;//以第 i 个数和第 j 个数作为最后两个数的斐波那契数列长度
                int u = lower_bound(a.begin(), a.begin() + j, a[i] - a[j]) - a.begin();
                if (u < j && a[u] == a[i] - a[j]) {
                    v[i][j] = max(v[i][j], v[j][u] + 1);
```

```
                    }
                    ret = max(ret, v[i][j]);
                }
            }
            return ret < 3 ? 0 : ret;
        }
};
```

11. 数位为 1 的个数

【LeetCode 原题】Counting Bits

Given a non negative integer number num. For every numbers i in the range 0 ≤ i ≤num calculate the number of 1's in their binary representation and return them as an array.

Example:For num = 5 you should return [0,1,1,2,1,2].

【核心代码】

```
class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> dp(num + 1,0);
        dp[0] = 0;
        if(num == 0) return dp;
        dp[1] = 1;
        for(int i = 2; i <= num; i++){
            if(i % 2 == 0) dp[i] = dp[i >> 1];
            else dp[i] = dp[i-1] + 1;
        }
        return dp;
    }
};
```

12. 编辑距离

Given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of edits (operations) required to convert ‘str1’ into ‘str2’.

a.Insert

b.Remove

c.Replace

All of the above operations are of equal cost.

【分析】The idea is process all characters one by one staring from either from left or right sides of both strings. Let us traverse from right corner, there are two possibilities for every pair of character being traversed.

1. If last characters of two strings are same, nothing much to do. Ignore last characters and get count for remaining strings. So we recur for lengths m-1 and n-1.
2. Else (If last characters are not same), we consider all operations on ‘str1’, consider all three operations on last character of first string, recursively compute minimum cost for all three operations and take minimum of three values.
    i. Insert: Recur for m and n-1
    ii. Remove: Recur for m-1 and n
    iii. Replace: Recur for m-1 and n-1

【核心代码】

```
int editDistDP(string str1, string str2, int m, int n){
    // Create a table to store results of subproblems
    int dp[m+1][n+1];
    // Fill d[][] in bottom up manner
    for (int i=0; i<=m; i++) {
        for (int j=0; j<=n; j++){
            // If first string is empty, only option is to
            // isnert all characters of second string
            if (i==0)
                dp[i][j] = j;    // Min. operations = j
            // If second string is empty, only option is to
            // remove all characters of second string
            else if (j==0)
                dp[i][j] = i; // Min. operations = i
            // If last characters are same, ignore last char
            // and recur for remaining string
            else if (str1[i-1] == str2[j-1])
                dp[i][j] = dp[i-1][j-1];
            // If the last character is different, consider all
            // possibilities and find the minimum
            else
                dp[i][j] = 1 + min(dp[i][j-1],    // Insert
                                   dp[i-1][j],    // Remove
                                   dp[i-1][j-1]); // Replace
        }
    }
    return dp[m][n];
}
```

13.

附录一 原创题

1. 现有一组数组 data[]，该数组的 data[i].b 的值前半段合法，后半段为不合法，现在要求你找出最后一个 data[i].b 的值合法的元素。系统为你提供了一个检测函数 bool isValid(int k, EleType* &head)，当原始存储结构为数组形式时，如果函数返回值为 false 时，存储结构就会变成以原始顺序存储的链表，链表的头元素指针为 head；而当该函数返回值为 true，存储结构依然为数组。一旦存储结构变成链表，使用检测函数就得小心，因为一旦返回 false，整个链表就会在某处断链，导致无法使用检测函数。现要求你求出合理的各种查找方案的最坏情况下的查找次数中最小的。

附录二 C++正则表达式

一、 语法规则

Special pattern characters

Special pattern characters are characters (or sequences of characters) that have a special meaning when they appear in a regular expression pattern, either to represent a character that is difficult to express in a string, or to represent a category of characters. Each of these *special pattern characters* is matched in the target sequence against a single character (unless a quantifier specifies otherwise).

characters	description	matches
.	not newline	any character except *line terminators* (LF, CR, LS, PS).
\t	tab (HT)	a horizontal tab character (same as \u0009).
\n	newline (LF)	a newline (line feed) character (same as \u000A).
\v	vertical tab (VT)	a vertical tab character (same as \u000B).
\f	form feed (FF)	a form feed character (same as \u000C).
\r	carriage return (CR)	a carriage return character (same as \u000D).
\c*letter*	control code	a control code character whose *code unit value* is the same as the remainder of dividing the *code unit value* of *letter* by 32. For example: \ca is the same as \u0001, \cb the same as \u0002, and so on...
\x*hh*	ASCII character	a character whose *code unit value* has an hex value equivalent to the two hex digits *hh*. For example: \x4c is the same as L, or \x23 the same as #.
\u*hhhh*	unicode character	a character whose *code unit value* has an hex value equivalent to the four hex digits *hhhh*.
\0	null	a null character (same as \u0000).
int	backreference	the result of the submatch whose opening parenthesis is the *int*-th (*int* shall begin by a digit other than 0). See groups below for more info.
\d	digit	a decimal digit character (same as [[:digit:]]).
\D	not digit	any character that is not a decimal digit character (same as [^[:digit:]]).
\s	whitespace	a whitespace character (same as [[:space:]]).
\S	not whitespace	any character that is not a whitespace character (same as [^[:space:]]).
\w	word	an alphanumeric or underscore character (same as [[:alnum:]]).

\W	not word	any character that is not an alphanumeric or underscore character (same as [^_[:alnum:]]).
character	character	the character *character* as it is, without interpreting its special meaning within a regex expression. Any *character* can be escaped except those which form any of the special character sequences above. Needed for: ^ $ \ . * + ? ( ) [ ] { } |
[*class*]	character class	the target character is part of the class (see character classes below)
[^*class*]	negated character class	the target character is not part of the class (see character classes below)

Notice that, in C++, character and string literals also escape characters using the backslash character (\), and this affects the syntax for constructing regular expressions from such types. For example:

```
std::regex e1 ("\\d");  // regular expression: \d -> matches a digit character
std::regex e2 ("\\\\"); // regular expression: \\ -> matches a single backslash (\) character
```

Quantifiers

Quantifiers follow a character or a *special pattern character*. They can modify the amount of times that character is repeated in the match:

characters	times	effects
*	0 or more	The preceding atom is matched 0 or more times.
+	1 or more	The preceding atom is matched 1 or more times.
?	0 or 1	The preceding atom is optional (matched either 0 times or once).
{*int*}	*int*	The preceding atom is matched exactly *int* times.
{*int*,}	*int* or more	The preceding atom is matched *int* or more times.
{*min*, *max*}	between *min* and *max*	The preceding atom is matched at least *min* times, but not more than *max*.

By default, all these quantifiers are greedy (i.e., they take as many characters that meet the condition as possible). This behavior can be overridden to *ungreedy* (i.e., take as few characters that meet the condition as possible) by adding a *question mark* (?) after the quantifier.
For example:
Matching "(a+).*" against "aardvark" succeeds and yields aa as the first submatch.
While matching "(a+?).*" against "aardvark" also succeeds, but yields a as the first submatch.

Groups

Groups allow to apply quantifiers to a sequence of characters (instead of a single character). There are two kinds of groups:

characters	description	effects
(*subpattern*)	Group	Creates a backreference.
(?:*subpattern*)	Passive group	Does not create a backreference.

When a group creates a backreference, the characters that represent the *subpattern* in the target sequence are stored as a *submatch*. Each submatch is numbered after the order of appearance of their opening parenthesis (the first submatch is number 1, the second is number 2, and so on...).

These *submatches* can be used in the regular expression itself to specify that the entire subpattern should appear again somewhere else (see *\int* in the special characters list). They can also be used in the *replacement string* or retrieved in the match results object filled by some regex operations.

Assertions

Assertions are conditions that do not consume characters in the target sequence: they do not describe a character, but a condition that must be fulfilled before or after a character.

characters	description	condition for match
^	Beginning of line	Either it is the beginning of the target sequence, or follows a *line terminator*.
$	End of line	Either it is the end of the target sequence, or precedes a *line terminator*.
\b	Word boundary	The previous character is a *word character* and the next is a *non-word character* (or vice-versa). Note: The beginning and the end of the target sequence are considered here as *non-word characters*.
\B	Not a word boundary	The previous and next characters are both *word characters* or both are *non-word characters*. Note: The beginning and the end of the target sequence are considered here as *non-word characters*.
(?=*subpattern*)	Positive lookahead	The characters following the assertion must match *subpattern*, but no characters are consumed.
(?!*subpattern*)	Negative lookahead	The characters following the assertion must not match *subpattern*, but no characters are consumed.

Alternatives

A pattern can include different alternatives:

character	description	effects
|	Separator	Separates two alternative patterns or subpatterns.

A regular expression can contain multiple alternative patterns simply by separating them with the *separator operator*(|): The regular expression will match if any of the alternatives match, and as soon as one does.

Subpatterns (in groups or assertions) can also use the *separator operator* to separate different alternatives.

Character classes

A character class defines a category of characters. It is introduced by enclosing its descriptors in square brackets ([and]).
The regex object attempts to match the entire character class against a single character in the target sequence (unless a quantifier specifies otherwise).
The character class can contain any combination of:

- **Individual characters:** Any character specified is considered part of the class (except the characters \, [,] and - when they have a special meaning as described in the following paragraphs).
  For example:
  [abc] matches a, b or c.
  [^xyz] matches any character except x, y and z.
- **Ranges:** They can be specified by using the *hyphen character* (-) between two valid characters.
  For example:
  [a-z] matches any lowercase letter (a, b, c, ... until z).
  [abc1-5] matches either a, b or c, or a digit between 1 and 5.
- **POSIX-like classes:** A whole set of predefined classes can be added to a custom character class. There are three kinds:

class	description	notes
[:*classname*:]	character class	Uses the *regex traits*' isctype member with the appropriate type gotten from applying lookup_classname member on *classname* for the match.
[.*classname*.]	collating sequence	Uses the *regex traits*' lookup_collatename to interpret *classname*.
[=*classname*=]	character equivalents	Uses the *regex traits*' transform_primary of the result of regex_traits::lookup_collatename for *classname* to check for matches.

- The choice of available classes depend on the regex_traits type and on its selected locale. But at least the following character classes shall be recognized by any regex_traits type and locale:

class	description	equivalent (with regex_traits, default locale)
[:alnum:]	alpha-numerical character	isalnum
[:alpha:]	alphabetic character	isalpha
[:blank:]	blank character	isblank

[:cntrl:]	control character	iscntrl
[:digit:]	decimal digit character	isdigit
[:graph:]	character with graphical representation	isgraph
[:lower:]	lowercase letter	islower
[:print:]	printable character	isprint
[:punct:]	punctuation mark character	ispunct
[:space:]	whitespace character	isspace
[:upper:]	uppercase letter	isupper
[:xdigit:]	hexadecimal digit character	isxdigit
[:d:]	decimal digit character	isdigit
[:w:]	word character	isalnum
[:s:]	whitespace character	isspace

- Please note that the brackets in the class names are additional to those opening and closing the class definition.
  For example:
  [[:alpha:]] is a character class that matches any alphabetic character.
  [abc[:digit:]] is a character class that matches a, b, c, or a digit.
  [^[:space:]] is a character class that matches any character except a whitespace.
- **Escape characters:** All escape characters described above can also be used within a character class specification. The only change is with \b, that here is interpreted as a backspace character (\u0008) instead of a word boundary.
  Notice that within a class definition, those characters that have a special meaning in the regular expression (such as *, ., $) don't have such a meaning and are interpreted as normal characters (so they do not need to be escaped). Instead, within a class definition, the hyphen (-) and the brackets ([and]) do have special meanings under some circumstances, in which case they should be placed within the class in other locations where they do not have such special meaning, or be escaped with a backslash (\).

二、 应用

Example 1

查找重复出现 K 次的数字字母以及下划线字符
正则表达式为：(\w)\1{K-1}
比如匹配的字符串为 caseee1:___this___a___teeest
匹配的结果：caseee1:___this___a___teeest

Example 2

查找一句话中的单词
正则表达式为：\b(\w)+(?=\s)?\b
比如匹配的字符串为 case1: this a test p^ee.
匹配的结果：case1: this a test p^ee.

Example 3

将一组数字按照每三位用逗号分隔，比如 123456789 变成 123,456,789
正则表达式为：(\d)(?=(\d{3})+(?!\d))
比如匹配的字符串为 123456789
匹配的结果：123456789

Example 4

匹配数字(5.　.234　都是合法数字)
正则表达式为：(-?\d*)(\.\d*)?
给定一些输入：5 -3.2 aaaa 9999 -1000
匹配结果：5 -3.2 aaaa 9999 -1000 2.3.7