ACM常用模板(+模板题)(基础)

目录

- 大数
- 二分
- 枚举排列
- 子集生成
- n皇后回溯
- 并查集
- 树状数组
- KMP，Sunday，BM
- 01背包，完全背包
- 最长(不)上升或下降子序列
- 最长公共子序列
- 拓扑排序
- 欧拉路径和回路
- 搜索
- 最小生成树
- 最短路
- GCD和LCM
- 埃拉托斯特尼筛法
- 唯一分定理
- 扩展欧几里得
- 欧拉函数
- 快速幂
- 矩阵快速幂

说明

- 虽然只打了不到一年的ACM，但是在ACM中一些算法以后还是可能用到的，在这里进行一个小小的总结，总结了一些简单常见的算法模板（比较适合新手）
- 模板终究只是模板，最好还是自己真正掌握，经常依赖模板，只是自己还没掌握的表现；
- 本人只是一个ACM的小白，不是很全，代码也写的坑坑洼洼。。。，大佬勿喷，可能以后学习了新的算法再补过来，主要还是方便以后自己复习。

大数

加法，乘法模板

```
//题目链接 : http://poj.org/problem?id=2506
//题目大意 : 就是问你用2*1,1*2,2*2的砖拼成2*n的长方形,有多少种拼法
//解题思路 : 考虑n的时候,假设我们已经铺好了n-1块砖,第n块只能竖着放
             //假设我们已经铺好了n-2块砖,最后两列有3种方式,但是其中有一种方法和.
             //所以f[n] = 2* f[n-2] + f[n-1]
#include <stdio.h>
#include <iostream>
#include <string.h>
using namespace std;
const int maxn = 10000 + 10;

//加法
string bigIntegerAdd(string s1,string s2){
    int a[maxn],b[maxn];
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    int len1 = s1.size(),len2 = s2.size();
    int maxL = max(len1,len2);
    for(int i = 0; i < len1; i++)a[i] = s1[len1-1-i]-'0';
    for(int i = 0; i < len2; i++)b[i] = s2[len2-1-i]-'0';
    for(int i = 0; i < maxL; i++){
        if(a[i]+b[i] >= 10){
            int temp = a[i]+b[i];
            a[i] = temp%10;
            a[i+1] += (temp/10);
        }
        else a[i] += b[i];
```

```
    }
    string c = "";
    if(a[maxL] != 0)
        c += a[maxL] + '0';
    for(int i = maxL-1; i >= 0; i--)c += a[i] + '0';
    return c;
}

//乘法
string bigIntegerMul(string s1,string s2){
    int a[maxn],b[maxn],c[maxn*2 + 5];
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
    int len1 = s1.size(),len2 = s2.size();
    for(int i = 0; i < len1; i++)a[i] = s1[len1-1-i]-'0'; //倒置
    for(int i = 0; i < len2; i++)b[i] = s2[len2-1-i]-'0';
    for(int i = 0; i < len1; i++){
        for(int j = 0; j < len2; j++){
            c[i+j] += a[i]*b[j];
        }
    }
    for(int i = 0; i < maxn*2; i++){
        if(c[i] >= 10){
            c[i+1] += c[i]/10;
            c[i] %= 10;
        }
    }
    string ans = "";
    int i;
    for(i = maxn * 2; i >= 0; i--)
        if(c[i] != 0)
            break;
    for(;i >= 0; i--)ans += c[i] + '0';
    return ans;
}

int main(){
    //freopen("in.txt","r",stdin);
    int n;
    string s[255];
    s[0] = "1",s[1] = "1";  //注意0的时候是1
    for(int i = 2;i <= 255; i++){
```

```
        string temp = bigIntegerMul("2",s[i-2]);
        s[i] = bigIntegerAdd(s[i-1],temp);
    }
    while(~scanf("%d",&n))
        cout<<s[n]<<endl;
    return 0;
}
```

减法模板

```
#include <bits/stdc++.h>
const int maxn = 200 + 10;
using namespace std;
typedef long long LL;

//具体实现
string subInfo(char *s1,char *s2){
    int a[maxn],b[maxn];
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    int len1 = strlen(s1),len2 = strlen(s2);
    int maxLen = max(len1,len2);
    for(int i = 0; i < len1; i++) a[i] = s1[len1 - i - 1] - '0';
    for(int i = 0; i < len2; i++) b[i] = s2[len2 - i - 1] - '0';
    for(int i = 0; i < maxLen; i++){
        if(a[i]-b[i] < 0){
            a[i] = a[i]+10-b[i];
            a[i+1] -= 1;
        }
        else a[i] -= b[i];
    }
    string str = "";
    int i;
    for(i = maxLen-1; i >= 0; i--)if(a[i] != 0)break;
    for(;i >= 0; i--)str += a[i]+'0';
    return str;
}

//大数减法的模板
string bigIntegerSub(char *s1,char *s2){
    if(s1 == s2)
        return "0"; //相等
```

```
    int len1 = strlen(s1),len2 = strlen(s2);
    if(len1 > len2)
        return subInfo(s1,s2);
    else if(len1 < len2)
        return "-" + subInfo(s2,s1); //负数
    else {                          //长度相等时判断大小
        for(int i = 0; i < len1; i++){
            if(s1[i]-'0' > s2[i]-'0')
                return subInfo(s1,s2);
            else if(s1[i]-'0' < s2[i]-'0')
                return "-" + subInfo(s2,s1);
        }
    }
}

int main(){
    char s1[maxn],s2[maxn];
    scanf("%s\n%s",s1,s2);
    cout<<bigIntegerSub(s1,s2)<<endl;
    return 0;
}
```

求阶乘以及位数模板

```
//http://acm.nyist.edu.cn/JudgeOnline/problem.php?pid=28
//大数阶乘的模板
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100000 + 10;

//大数计算阶乘位数
//lg(N!)=[lg(N*(N-1)*(N-2)*.....*3*2*1)]+1 = [lgN+lg(N-1)+lg(N-2)+..
int factorialDigit(int n){
    double sum = 0;
    for(int i = 1; i <= n; i++){
        sum += log10(i);
    }
    return (int)sum+1;
}

//大数计算阶乘
string bigFactorial(int n){
```

```
    int ans[maxn],digit = 1;
    ans[0] = 1;
    for(int i = 2; i <= n; i++){
        int num = 0;
        for(int j = 0; j < digit; j++){
            int temp = ans[j]*i + num;
            ans[j] = temp%10;
            num = temp/10;
        }
        while(num != 0){
            ans[digit] = num%10;
            num /= 10;
            digit++;
        }
    }
    string str = "";
    for(int i = digit-1; i >= 0; i--)
        str += ans[i] + '0';
    return str;
}

int main(){
    int n;
    while(~scanf("%d",&n)){
        //cout<<factorialDigit(n)<<endl; //阶乘的位数
        cout<<bigFactorial(n)<<endl;  //求出阶乘
    }
    return 0;
}
```

二分

二分的写法可以有很多种，这里列举几个常见的，主要是上界确定，区间以及查找元素是否重复的问题，也可以看看这篇文章。

- 求最小的 `i`，使得 `a[i] = key`，若不存在，则返回 `-1` (`lowerbound` 函数)；
- 求最大的 `i` 的下一个元素的下标(`c++` 中的 `upperbound` 函数)，使得 `a[i] = key`，若不存在，则返回 `-1`；
- 求最大的 `i`，使得 `a[i] = key`，若不存在，则返回 `-1`；
- 求最小的 `i`，使得 `a[i] > key`，若不存在，则返回 `-1`；

- 求最大的 `i`，使得 `a[i] < key`，若不存在，则返回 `-1`；

```cpp
#include <bits/stdc++.h>

using namespace std;
const int maxn = 100 + 10;

int cmp(const void *a, const void *b) {
    return *(int *) a - *(int *) b;
}

//普通的二分查找
int bs(int *arr,int L,int R,int target){
    while( L <= R){
        int mid = (L) + (R-L)/2;
        if(arr[mid] == target)
            return mid;
        if(arr[mid] > target)
            R = mid - 1;
        else
            L = mid + 1;
    }
    return -1; // not find
}

//求最小的i，使得a[i] = target，若不存在，则返回-1
//返回 如果有重复的 下界(比如1,2,2,2,3,4)查找2,返回1
int firstEqual(int arr[], int L, int R, int target) {
    while (L < R) {
        int mid = L + (R - L) / 2;
        if (arr[mid] < target)
            L = mid + 1;
        else
            R = mid;
    }
    if (arr[L] == target)
        return L;
    return -1;
}

//求最大的i的下一个元素的下标(c++中的upperbound函数)，使得a[i] = target，若不
int lastEqualNext(int arr[], int L, int R, int target) {
```

```
    while (L < R) {
        int m = L + (R - L) / 2;
        if (arr[m] <= target)
            L = m + 1;
        else
            R = m;
    }
    if (arr[L - 1] == target)
        return L;
    return -1;
}

//求最大的i，使得a[i] = target，若不存在，则返回-1
int lastEqual(int arr[], int L, int R, int target) {
    while (L < R) {
        int mid = L + ((R + 1 - L) >> 1);//向上取整
        if (arr[mid] <= target)
            L = mid;
        else
            R = mid - 1;
    }
    if (arr[L] == target)
        return L;
    return -1;
}

//求最小的i，使得a[i] > target，若不存在，则返回-1
int firstLarge(int arr[], int L, int R, int target) {
    while (L < R) {
        int m = L + ((R - L) >> 1);//向下取整
        if (arr[m] <= target)
            L = m + 1;
        else
            R = m;
    }
    if (arr[R] > target)
        return R;
    return -1;
}

//求最大的i，使得a[i] < target，若不存在，则返回-1
int lastSmall(int arr[], int L, int R, int target) {
    while (L < R) {
```

```
        int m = L + ((R + 1 - L) >> 1);//向上取整
        if (arr[m] < target)
            L = m;
        else
            R = m - 1;
    }
    if (arr[L] < target)
        return L;
    return -1;
}

int main() {
    //freopen("in.txt", "r", stdin);
    int n, a[maxn], v;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)scanf("%d", &a[i]); //1 3 2 9 4 1 3 7
    scanf("%d", &v);   //input the number you need find
    qsort(a, n, sizeof(a[0]), cmp);                // 1 1 2 2 2 3 3 4
    printf("after sorted : \n");
    for (int i = 0; i < n; i++)printf("%d ", a[i]);

    printf("\n-------------test----------------");

    printf("\n%d\n", firstEqual(a, 0, n, v));  //output 2  第一个
    printf("%d\n", lastEqualNext(a, 0, n, v)); //output 4 + 1,最后一个
    printf("%d\n", lastEqual(a, 0, n, v));     //output 4  最后一个
    printf("%d\n", firstLarge(a, 0, n, v));    //output 5(第一个3大于2
    printf("%d\n", lastSmall(a, 0, n, v));     //output 1(不是0)
    return 0;
}
/*
测试数据:
10
1 3 2 9 4 1 3 7 2 2
2
*/
```

输出

```
10
1 3 2 9 4 1 3 7 2 2
2
after sorted :
1 1 2 2 2 3 3 4 7 9
-------------test----------------
2
5
4
5
1
```


枚举排列

枚举排列的算法也有几个，包括刘汝佳书上的和经典的，还有做过的几个题LeetCode47。LeetCode46。

```cpp
#include <bits/stdc++.h>

using namespace std;
const int maxn = 100 + 10;

void permutation(int *arr, int n, int cur){
    if(cur == n){  // 边界
        for(int i = 0; i < n; i++)
            printf("%d ",arr[i]);
        printf("\n");
    }
    else for(int i = 1; i <= n; i++){  //尝试在arr[cur]中填充各种整数
        bool flag = true;
        for(int j = 0; j < cur; j++)if(i == arr[j]){  // 如果i已经在ar
            flag = false;
            break;
        }
        if(flag){
            arr[cur] = i;  //把i填充到当前位置
```

```
            permutation(arr, n, cur+1);
        }
    }
}

// 求 1 ~ n 的全排列，arr数组作为中间打印数组
int main(int argc, char const **argv)
{
    int a[maxn], n;
    scanf("%d", &n);
    permutation(a, n, 0);
    return 0;
}

```

```
//可重集的全排列
#include <bits/stdc++.h>
const int maxn = 100 + 10;

void permutation(int *arr,int *p,int n,int cur){
    if(cur == n){
        for(int i = 0; i < n; i++)
            printf("%d ",arr[i]);
        printf("\n");
    }else for(int i = 0; i < n; i++)if(!i || p[i] != p[i-1]){
        int c1 = 0, c2 = 0;
        for(int j = 0; j < n; j++)
            if(p[j] == p[i]) // 重复元素的个数
                c1++;
        for(int j = 0; j < cur; j++)
            if(arr[j] == p[i]) // 前面已经排列的重复元素的个数
                c2++;
        if(c2 < c1){
            arr[cur] = p[i];
            permutation(arr, p, n, cur+1);
        }
    }
}

int main(){
    int a[maxn], p[maxn] = {5, 6, 7, 5};  //可以有重复元素的全排列
    std::sort(p, p+4);
```

```
    permutation(a, p, 4, 0);
    return 0;
}

```

```
//全排列的非去重递归算法
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100 + 10;

void permutation(int arr[], int cur, int n){
    if( cur == n){
        for(int i = 0; i < n; i++)
            printf("%d ", arr[i]);
        printf("\n");
    }
    else for(int i = cur; i < n; i++){
        swap(arr[i], arr[cur]);
        permutation(arr, cur+1, n);
        swap(arr[i], arr[cur]);
    }
}
int main(){
    int n, a[maxn];
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    permutation(a, 0, n);
    return 0;
}

```

子集生成

增量构造法，位向量法，二进制法(常用)

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100 + 10;

```

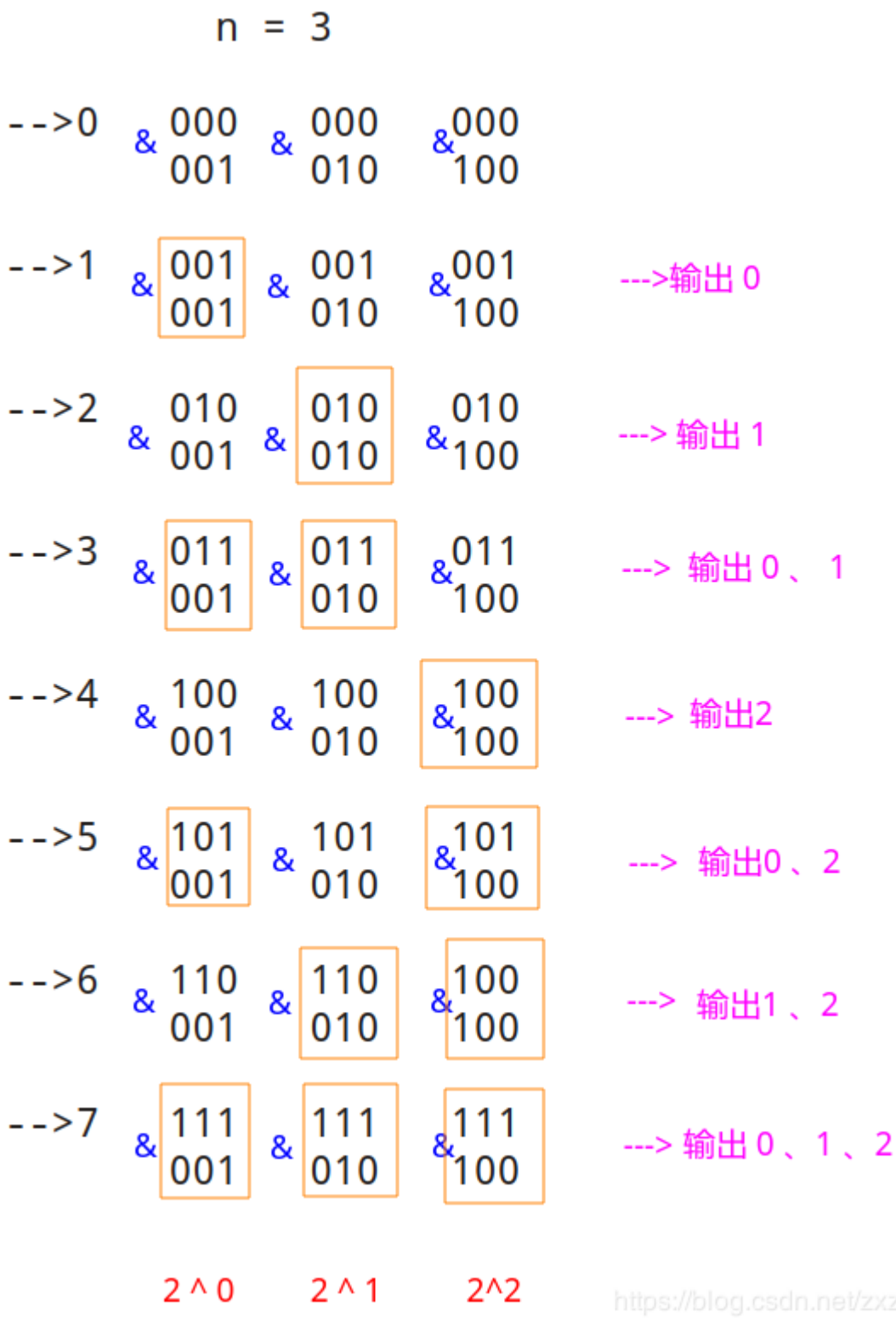
```
//打印0~n-1的所有子集
//按照递增顺序就行构造子集 防止子集的重复
void print_subset(int *arr, int n, int cur){
    for(int i = 0; i < cur; i++)
        printf("%d ", arr[i]);
    printf("\n");
    int s = cur ? arr[cur-1] + 1 : 0;  //确定当前元素的最小可能值
    for(int i = s; i < n; i++){
        arr[cur] = i;
        print_subset(arr, n, cur+1);
    }
}
int main(){
    int n, arr[maxn];
    scanf("%d", &n);
    print_subset(arr, n, 0);
    return 0;
}

```

这个其实很简单，就是枚举每个位置 0 和 1 两种情况即可。

```
// 1~n 的所有子集：位向量法
#include <bits/stdc++.h>
const int maxn = 100 + 10;
using namespace std;
int bits[maxn];//位向量bits[i] = 1,当且仅当i在子集 A 中

void print_subset(int n,int cur){
    if(cur == n){
        for(int i = 0; i < cur; i++)
            if(bits[i])
                printf("%d ",i);
        printf("\n");
        return;
    }
    bits[cur] = 1;
    print_subset(n,cur + 1);
    bits[cur] = 0;
    print_subset(n,cur + 1);
}

```

```
21 int main() {
22     int n;
23     scanf("%d", &n);
24     print_subset(n,0);
25     return 0;
26 }
27
```

二进制枚举子集用的多，这里举个例子 `n = 3`；则要枚举 `0 - 7` 对应的是有 `7` 个子集，每个子集去找有哪些元素 `print_subset` 中的 `1<< i`，也就是对应的那个位置是有元素的，例如 `1` 的二进制是 `0001` 也就是代表 `0` 位置有元素， `0010` 是 `2`，代表第一个位置是 `1`， `0100` 代表第 `2` 个位置上有元素，相应的 `1000 = 8` 对应第 `3` 个位置上有元素。
总结来说也就是对应 `1<< i` 对应 `i` 上是 `1` (从 `0` 开始)，其余位置是 `0` 。看图容易理解：




```cpp
// 0 ~ n-1的所有子集：二进制法枚举0 ~ n-1的所有子集
#include <bits/stdc++.h>
const int maxn = 100 + 10;

```

```
using namespace std;
void print_subset(int n,int cur){
    //这一步其实就是判断 cur 的二进制的各个位上是不是1，如果是1,就输出对应的那个
    for(int i = 0; i < n; i++)
        if(1 & (cur >> i))
            printf("%d ",i);
    printf("\n");
}
int main(int argc, char const** argv)
{
    int n;
    scanf("%d",&n);
    for(int i = 0; i < (1 << n); i++)
        print_subset(n,i);//枚举各子集对应的编码 0,1,2...pow(2,n) - 1
    return 0;
}
```

n皇后回溯

可以看这篇博客。

```
//n皇后问题：普通回溯法
#include <bits/stdc++.h>
const int maxn = 100 + 10;
using namespace std;
int sum,n,cnt; //解的个数，n皇后，递归次数
int C[maxn];

void Search(int cur){       //逐行放置皇后
    cnt++;
    if(cur == n)sum++;
    else for(int i = 0; i < n; i++){   //尝试在各列放置皇后
        bool flag = true;
        C[cur] = i;   //尝试把第cur行的皇后放在第i列//如果 等下不行的话 就下
        for(int j = 0; j < cur; j++){ //检查是否和已经放置的冲突
            if(C[cur] == C[j] || C[cur] + cur == C[j] + j || cur - C[
                flag = false;break;
            }
        }
        if(flag)Search(cur+1);
    }
}
```

```
int main(){
    scanf("%d",&n); //输入n皇后
    sum = cnt = 0;//解的个数 和 递归的次数
    Search(0);
    printf("%d %d\n",sum,cnt);
    return 0;
}
```

优化过的

```
// n皇后问题：优化的回溯法
#include <bits/stdc++.h>
const int maxn = 100 + 10;
using namespace std;

int sum,n,cnt;
int C[maxn];
bool vis[3][maxn];
int Map[maxn][maxn];//打印解的数组

//一般在回溯法中修改了辅助的全局变量，一定要及时把他们恢复原状
void Search(int cur){  //逐行放置皇后
    cnt++;
    if(cur == n){
        sum++;
        for(int i = 0; i < cur; i++)Map[i][C[i]] = 1;//打印解
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++)printf("%d ",Map[i][j]);
            printf("\n");
        }
        printf("\n");
        memset(Map,0,sizeof(Map)); //还原
    }
    else for(int i = 0; i < n; i++){ //尝试在 cur行的 各 列 放置皇后
        if(!vis[0][i] && !vis[1][cur+i] && !vis[2][cur-i+n]){//判断当前
            vis[0][i] = vis[1][cur+i] = vis[2][cur-i+n] = true;
            C[cur] = i;//cur 行的列是 i
            Search(cur+1);
            vis[0][i] = vis[1][cur+i] = vis[2][cur-i+n] = false;//切记
        }
    }
}
```

```
33
34  int main(){
35      scanf("%d",&n);
36      memset(vis,false,sizeof(vis));
37      memset(Map,0,sizeof(Map));
38      sum = cnt = 0;
39      Search(0);
40      printf("%d %d\n",sum,cnt);//输出 解决方案 和 递归次数
41      return 0;
42  }
```

并查集

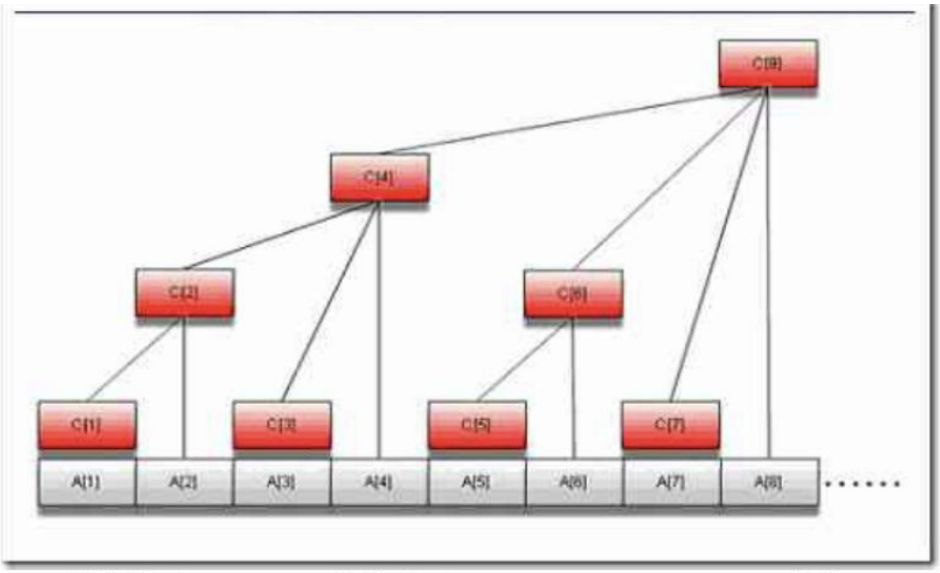
并查集详细讲解以及每一步的优化可以看这篇博客。

```
1   //题目连接 : http://poj.org/problem?id=1611
2   //题目大意 : 病毒传染，可以通过一些社团接触给出一些社团(0号人物是被感染的)问有多少
3   #include <stdio.h>
4   const int maxn = 100000 + 10;
5
6   int parent[maxn], rank[maxn];  //parent[]保存祖先,rank记录每个'树的高度'
7
8   void init(){
9       for(int i = 0; i < maxn; i++)parent[i] = i; //注意这里
10      for(int i = 0; i < maxn; i++)rank[i] = 1;
11  }
12
13  //int findRoot(int v){
14  //    return parent[v] == v ? v : parent[v] = findRoot(parent[v]);
15  //}
16
17  // 非递归
18  int findRoot(int v){
19      while(parent[v] != v){
20          parent[v] = parent[parent[v]]; // 路径压缩
21          v = parent[v];
22      }
23      return v;
24  }
25
26  void unions(int a, int b){
27      int aRoot = findRoot(a);
```

```
    int bRoot = findRoot(b);
    if (aRoot == bRoot)
        return;
    if (rank[aRoot] < rank[bRoot])
        parent[aRoot] = bRoot;
    else if(rank[aRoot] > rank[bRoot]){
        parent[bRoot] = aRoot;
    }else{
        parent[aRoot] = bRoot;
        rank[bRoot]++;
    }
}

int is_same(int x,int y){ //检查是不是在同一个集合中
    return findRoot(x) == findRoot(y);
}

int main(){
    int n,m,k,x,root;
    while(~scanf("%d%d",&n,&m) && (n||m)){
        init();
        for(int i = 0; i < m; i++){
            scanf("%d%d",&k,&root);
            for(int j = 1; j < k; j++){
                scanf("%d",&x);
                unions(root,x);
            }
        }
        int sum = 1;
        for(int i = 1; i < n; i++)
            if(findRoot(i) == findRoot(0))
                sum++; //找和0是一个集合的
        printf("%d\n",sum);
    }
    return 0;
}

```

树状数组

树状数组的主要用于需要频繁的修改数组元素，同时又要频繁的查询数组内任意区间元素之和的时候。具体一些解释看图。



图中 C[1]的值等于 A[1]，C[2]的值等于 C[1]+A[2]=A[1]+a[2],C[4]的值=C[2]+C[3]=A[1]+A[2]+A[3]+A[4],假设我们现在需要更改元素 a[2],那么它将只影响到得 c 数组中的元素有 c[2],c[4],c[8],我们只需要重新计算这几个值即可，减少了很多重复的操作。这就是树状结构大致的一个存贮示意图。

下面看看它的定义：假设 a[1...N]为原数组,定义 c[1...N]为对应的树状数组：

c[i] = a[i - 2^k + 1] + a[i - 2^k + 2] + ... + a[i]（其中 k 为 i 的二进制表示末尾 0 的个数）。

下面枚举出 i 由 1...5 的数据，可见正是因为上面的 a[i - 2^k + 1]...a[i]的计算公式保证了我们 C 数组的正确意义，至于证明过程，请读者自己查找资料。

i		K		
1	$(1)_2$	0	1-2^0+1=1…1	c[1]=a[1]
2	$(10)_2$	1	2-2^1+1=1…2	c[2]=a[1]+a[2]=c[1]+a[2]
3	$(11)_2$	0	3-2^0+1=3…3	c[3]=a[3]
4	$(100)_2$	2	4-2^2+1=1…4	c[4]=a[1]+a[2]+a[3]+a[4]=c[2]+c[3]+a[4]
5	$(101)_2$	0	5-2^0+1=5…5	c[5]=a[5]

1.4.1.1 基本操作

对于 C[i]=a[i - 2^k + 1]...a[i]的定义中，比较难以逐磨的 k，他的值等于 i 这个数的二进制表示末尾 0 的个数.如 4 的二进制表示 0100,此时 k 就等于 2，而实际上我们还会发现 2^k 就是前一位的权值,即 0100 中,2^2=4，刚好是前一位数 1 的权值.所以所以 2^k 可以表示为 n&(n^(n-1))或更简单的 n&(-n)，例如：

（为了表示简便，假设现在一个 int 型为 4 位,最高位为符号位。）

int i=3&(-3);　　此时 i=1，3 的二进制为 0011,-3 的二进制为 1101(负数存的是补码）所以 0011&1101=1

int j=4&(-4);　　此时 j=4，理由同上。

所以计算 2^k 次方我们可以用如下代码

```
int lowbit(int x){
    return x&(-x); //或者 return x&(x^(x-1));
}
```

求 sum[1..k],我们需查找 k 的二进制表示中 1 的个数次就能得到最终结果，具体为什么，请见代码 i-=lowbit(i)注释

```
int sum(int i)//求前 i 项和
{
    int s=0;
    while(i>0)
    {
        s += c[i];
        i -= lowbit(i);
    }
    return s;
}
```

求s[7] = c[7] + c[6] + c[4]

时间复杂度是 O(logN)。

代码中“i -= lowbit(i)”的解释，这一步实际上等价于将 i 的二进制表示的最后一个 1 剪去，再向前数当前 1 的权个数（例子在下面），而 n 的二进制里最多有 log(n)个 1，所以查询效率是 log(n)，在示意图上的操作即可理解为依次找到所有的子节点。

以求 sum[1..7]为例,二进制为 0111,右边第一个 1 出现在第 0 位上,也就是说要从 a[7]开始向前数 1 个元素(只有 a[7]),即 c[7];

然后将这个 1 舍掉,得到 6,二进制表示为 0110,右边第一个 1 出现在第 1 位上,也就是说要从 a[6]开始向前数 2 个元素(a[6],a[5]),即 c[6];

然后舍掉用过的 1,得到 4,二进制表示为 0100,右边第一个 1 出现在第 2 位上,也就是说要从 a[4]开始向前数 4 个元素(a[4],a[3],a[2],a[1]),即 c[4].

所以 s[7]=c[7]+c[6]+c[4]。

1.4.1.3 给源数组加值操作

在上面的示意图中，假设更改的元素是 a[2],那么它影响到得 c 数组中的元素有 c[2],c[4],c[8]，我们只需一层一层往上修改就可以了,这个过程的最坏的复杂度也不过 O(logN);

```
void add(int i,int val)
{
    while(i<=n)
    {
        c[i] += val;
        i += lowbit(i);

    }
}
```

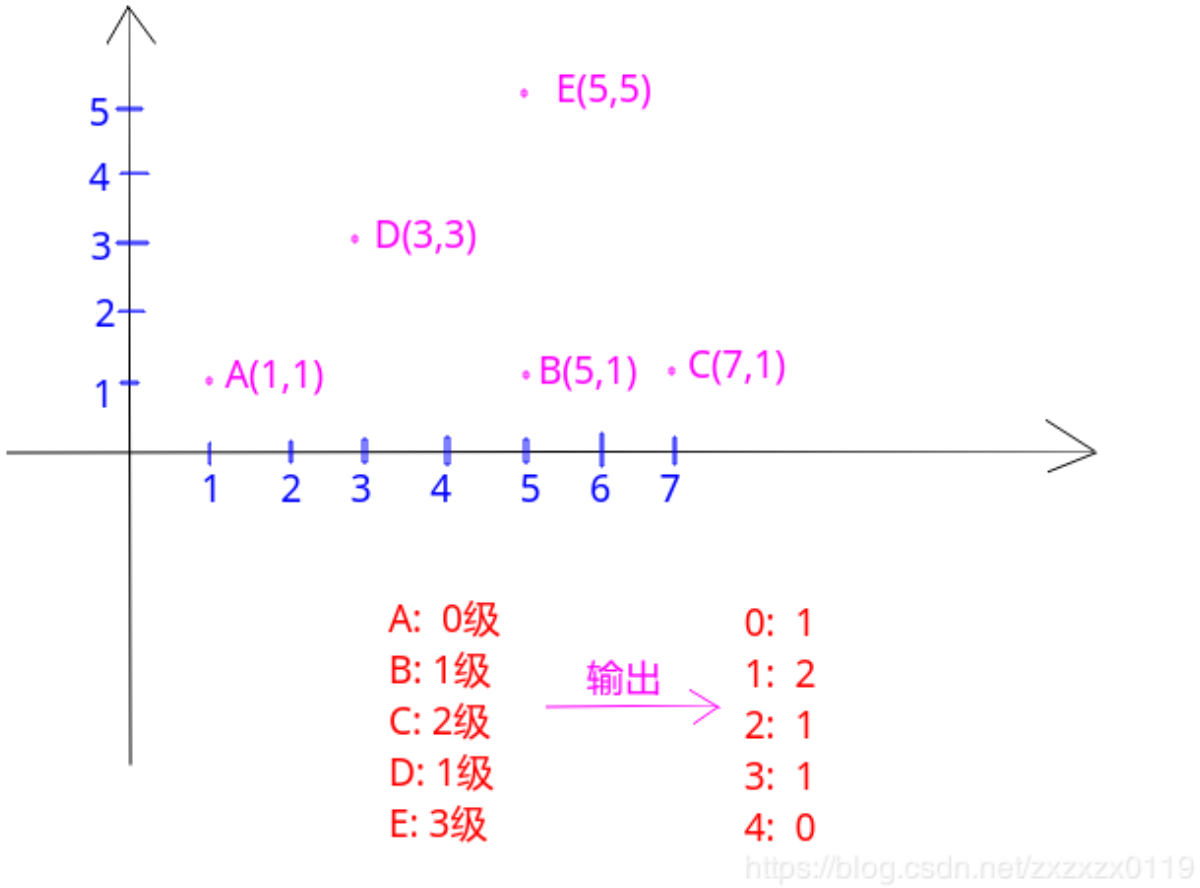
时间复杂度是O(logN)

代码中“i += lowbit(i)”的解释， i+（i 的二进制中最后一个 1 的权值,即 2^k）,在示意图上的操作即为提升一层，到上一层的节点，这个过程实际上也只是一个把末尾 1 后补 0 的过程（例子在下面）。

以修改 a[2]元素为例，需要修改 c[2],2 的二进制为 0010,末尾补 0 为 0100,即 c[4]，4 的二进制为 0100，在末尾补 0 为 1000 即 c[8]。所以我们需要修改的有 c[2],c[4],c[8]。

这里给出一个例题POJ2352Stars

题目意思就是给你一些星星的坐标，每个星星的级别是他左下方的星星的数量，要你求出各个级别的星星有多少个，看样例就懂了



题目中一个重要的信息就是输入是按照 y 递增，如果 y 相同则 x 递增的顺序给出的，所以，对于第 i 颗星星，它的 level 就是之前出现过的星星中，横坐标小于等于 i 的星星的数量。这里用树状数组来记录所有星星的 x 值。

- 代码中有个小细节就是 x++ 这是因为 lowbit 不能传值为 0，否则会陷入死循环。

```
#include <stdio.h>
#include <string.h>
const int maxn = 32000 + 10;
int n,c[maxn],level[15000+10];

//计算2^k
int lowbit(int x){
    return x&(-x);
}

//更新数组的值
void update(int i,int val){
```

```
    while(i < maxn){                //注意这里是最大的x,没有记录所以用maxn,
        c[i] += val;
        i += lowbit(i);             //不断的往上面更新
    }
}
//查询
int sum(int i){
    int s = 0;
    while(i > 0){
        s += c[i];
        i -= lowbit(i);             //不断的往下面加
    }
    return s;
}

int main(){
    int x,y;
    scanf("%d",&n);
    memset(c,0,sizeof(c));
    memset(level,0,sizeof(level));
    for(int i = 0; i < n; i++){
        scanf("%d%d",&x,&y);
        x++;                        //加入x+1，是为了避免0，X是可能为0的,LOl
        level[sum(x)]++;
        update(x,1);                //上面的层都要+1个
    }
    for(int i = 0; i < n; i++)
        printf("%d\n",level[i]);
    return 0;
}

```

KMP，Sunday，BM

这三个算法解决的问题都是：有一个文本串 S，和一个模式串 P，现在要查找 P 在 S 中的位置，三种算法的思路这里限于篇幅不多说，这篇博客对着三种算法讲解的比较详细。

KMP 的较直观的分析也可以看看我的另一篇博客

```
//题目连接 : http://acm.hdu.edu.cn/showproblem.php?pid=1711
//题目大意 : 找第二个数组在第一个数组中出现的位置，如果不存在，输出-1
```

```cpp
#include <stdio.h>
const int maxn = 1000000 + 10;
int n,m,a[maxn], b[10000 + 10],nexts[10000 + 10];

void getNext(int *p,int next[]) {   //优化后的求next数组的方法
    int len = m;
    next[0] = -1;    //next 数组中的 最大长度值(前后缀的公共最大长度) 的第一
    int k = -1,j = 0;
    while (j < len - 1) {
        if (k == -1 || p[j] == p[k]) { //p[k]表示前缀 p[j] 表示后缀
            k++; j++;
            if(p[j] != p[k])next[j] = k;
            else next[j] = next[k];   //因为不能出现p[j] = p[ next[j ]]
        }
        else k = next[k];
    }
}

int KMPSerach(int *s, int *p) {
    int sLen = n,pLen = m;
    int i = 0, j = 0;
    while (i < sLen && j < pLen) {
        if (j == -1 || s[i] == p[j])i++, j++;
        else j = nexts[j];
    }
    if (j == pLen)return i - j;
    else return -1;
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i++)scanf("%d", &a[i]);
        for(int i = 0; i < m; i++)scanf("%d", &b[i]);
        getNext(b,nexts); //获取next数组
        int ans = KMPSerach(a, b);
        if (ans != -1)printf("%d\n", ans + 1);
        else printf("-1\n");
    }
    return 0;
}
```

BM 算法，主要是根据两个规则去执行

BM 算法定义了两个规则：

- 坏字符规则：当文本串中的某个字符跟模式串的某个字符不匹配时，我们称文本串中的这个失配字符为坏字符，此时模式串需要向右移动，移动的位数 = 坏字符在模式串中的位置 - 坏字符在模式串中最右出现的位置。此外，如果"坏字符"不包含在模式串之中，则最右出现位置为 -1。
- 好后缀规则：当字符失配时，后移位数 = 好后缀在模式串中的位置 - 好后缀在模式串上一次出现的位置，且如果好后缀在模式串中没有再次出现，则为 -1。


```
1  //题目链接：http://acm.hrbust.edu.cn/index.php?m=ProblemSet&a=showProb
2  //题目大意：找一串字符中是否出现"bkpstor"这段字符
3
4  #include <stdio.h>
5  #include <string.h>
6  #include <algorithm>
7  using namespace std;
8  const int maxn = 1000;
9
10 int last(char *p, char c) { //找到文本串的  "坏字符"  在模式串中的位置
11     for (int i = strlen(p) - 1; i >= 0; i--)if (p[i] == c)return i;
12     return -1;
13 }
14
15 int BM_index(char *T, char *p) {
16     int n = strlen(T);
17     int m = strlen(p);
18     int i = m - 1, j = m - 1;
19     while (i <= n - 1) {
20         if (T[i] == p[j])
21             if (j == 0)return i;
22             else i--, j--;
23         else {
24             i = i + m - min(j, 1 + last(p, T[i]));//文本的不符合的那个 1
25             j = m - 1;//模式串的新位置
26         }
27     }
28     return -1;
29 }
30
31 int Sum(char *T, char *P, int s) {//输出文本串中包含模式串的数量
```

```
    int e = BM_index(T + s, P);
    return e == -1 ? 0 : 1 + Sum(T, P, s + e + 1);
}

//测试数据 : 123bkpstor456bkpstor67
int main() {
    char s[maxn];
    while (~scanf("%s",s)) {
        int cnt = BM_index(s, "bkpstor");
        //printf("%d\n",Sum(s,"bkpstor",0));出现次数输出2
        if (cnt == -1)printf("Safe\n");
        else printf("Warning\n");
    }
    return 0;
}
```

Sunday 算法也是两个规则

Sunday 算法由 Daniel M.Sunday 在 1990 年提出，它的思想跟 BM 算法很相似：

- 只不过 Sunday 算法是从前往后匹配，在匹配失败时关注的是文本串中参加匹配的最末位字符的下一位字符。
  - 如果该字符没有在模式串中出现则直接跳过，即移动位数 = 匹配串长度 + 1；
  - 否则，其移动位数 = 模式串中最右端的该字符到末尾的距离 +1。


```
//题目链接：http://acm.hrbust.edu.cn/index.php?m=ProblemSet&a=showProb
//题目大意：找一串字符中是否出现"bkpstor"这段字符
#include <stdio.h>
#include <string.h>
#include <algorithm>
const int maxn = 1000;
using namespace std;

int last(char *p, char c) {
    for (int i = strlen(p) - 1; i >= 0; i--)if (p[i] == c)return i;
    return -1;
}

int Sunday(char *s, char *p) {
    int sLen = strlen(s);
    int pLen = strlen(p);
    int i = 0, j = 0;
    while (i < sLen && j < pLen) {
```

```
        if (s[i] == p[j])i++, j++;
        else {
            int index = i + pLen - j;   // 字符串中右端对齐的字符
            if (last(p, s[index]) == -1) { i = index + 1; j = 0; }  /
            else {
                i = index - last(p, s[index]); j = 0; //否则  其移动步
            }
        }
    }
    if (j == pLen)return i - j;
    return -1;
}

int main() {
    char s[maxn];
    while (~scanf("%s",s)) {
        int cnt = Sunday(s,"bkpstor");
        if (cnt == -1)printf("Safe\n");
        else printf("Warning\n");
    }
    return 0;
}
```

01背包，完全背包

背包问题可以分为很多种，这里只简单讨论 01 背包和完全背包，
后来改写的 01 背包: 博客
参考博客:

https://blog.csdn.net/liusuangeng/article/details/38374405（很详细）
https://blog.csdn.net/xp731574722/article/details/70766804
https://blog.csdn.net/na_beginning/article/details/62884939#reply
https://blog.csdn.net/u013445530/article/details/40210587

```
//题目连接 : http://acm.hdu.edu.cn/showproblem.php?pid=2602
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;
```

```cpp
const int maxn = 1000+5;
int w[maxn],v[maxn],dp[maxn][maxn],vis[maxn];

//打印解
void print(int n,int C){
    for(int i = n; i > 1; i--){
        if(dp[i][C] == dp[i-1][C-w[i]] + v[i]){
            vis[i] = 1;
            C -= w[i];
        }
    }
    vis[1] = (dp[1][C] > 0) ? 1: 0;
}

int main(){
    freopen("in.txt","r",stdin);
    int n,C,T;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&C);
        for(int i = 1;i <= n; i++) scanf("%d",&v[i]);
        for(int i = 1;i <= n; i++) scanf("%d",&w[i]);
        memset(dp,0,sizeof(dp)); //dp[0][0~C]和dp[0~N][0]都为0，前者表示
        memset(vis,0,sizeof(vis));
        for(int i = 1; i <= n; i++){
            for(int j = 0;j <= C; j++){
                dp[i][j] = dp[i-1][j];  //如果j不大于v[i]的话就dp[i][j]
                if(j >= w[i])  dp[i][j] = max(dp[i][j],dp[i-1][j-w[i]
            }
        }
        printf("%d\n",dp[n][C]); //n个物品装入C容量的包能获得的最大价值
        //print(n,C);
        //for(int i = 1; i <= n; i++)if(vis[i])printf("%d ",i); //输出
    }
    return 0;
}
```

```cpp
#include <stdio.h>
#include <algorithm>
#include <string.h>
const int maxn = 1000+5;
using namespace std;
```

```

int w[maxn],v[maxn],dp[maxn][maxn];

int main(){
    int T,n,C;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&C);
        for(int i = 1; i <= n; i++)scanf("%d",&v[i]);
        for(int i = 1; i <= n; i++)scanf("%d",&w[i]);
        memset(dp,0,sizeof(dp));  //dp全部初始化为0
        for(int i = n;i >= 1; i--){
            for(int j = 0; j <= C; j++){
                dp[i][j] = (i == n ? 0 : dp[i+1][j]);
                if(j >= w[i])dp[i][j] = max(dp[i][j],dp[i+1][j-w[i]]+
            }
        }
        printf("%d\n",dp[1][C]);
    }
    return 0;
}
```

```
#include <stdio.h>
#include <algorithm>
#include <string.h>
const int maxn = 1000+5;
using namespace std;

int w[maxn],v[maxn],dp[maxn];

int main(){
    int T,n,C;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&C);
        for(int i = 1; i <= n; i++)scanf("%d",&v[i]);
        for(int i = 1; i <= n; i++)scanf("%d",&w[i]);
        memset(dp,0,sizeof(dp));
        for(int i = 1; i <= n; i++){
            for(int j = C;j >= 0; j--){
                if(j >= w[i])dp[j] = max(dp[j],dp[j-w[i]]+v[i]);
            }
```

```
        }
        printf("%d\n",dp[C]);
    }
    return 0;
}
```

完全背包和 01 背包的差别在于每个物品的数量有无数个，在考虑第 i 件物品的时候，不是考虑选不选这件，而是选多少件可以达到最大价值

$$f[i][j] = max\{f[i-1][v],f[i-1][v - k * c[i]] + k * w[i]\}$$
$$(0<=k*c[i]<=v)$$


```
for (int i = 1; i < n; i++){
    for (int j = 1; j <= v; j++){
        for (int k = 0; k*c[i] <= j; k++){
          if(c[i] <= j) dp[i][j] = max{dp[i-1][j],dp[i-1][j - k * c[
          else dp[i][j] = dp[i-1][j]/*继承前i个物品在当前空间大小时的价值*
        }
    }
}
```

优化:

所以说，对于第i件物品有放或不放两种情况，而放的情况里又分为放1件、2件、.....v/c[i]件

如果不放那么f[i][j]=f[i-1][j]；如果确定放，那么当前背包中应该出现至少一件第i种物品，所以f[i][j]中至少应该出现一件第i种物品,即f[i][j]=f[i][j-c[i]]+w[i]，为什么会是f[i][j-c[i]]+w[i]?

因为我们要把当前物品i放入包内，因为物品i可以无限使用，所以要用f[i][j-c[i]]；如果我们用的是f[i-1][j-c[i]]，f[i-1][j-c[i]]的意思是说，我们只有一件当前物品i，所以我们在放入物品i的时候需要考虑到第i-1个物品的价值(f[i-1][j-c[i]])；但是现在我们有无限件当前物品i，我们不用再考虑第i-1个物品了，我们所要考虑的是在当前容量下是否再装入一个物品i，而[j-c[i]]的意思是指要确保f[i][j]至少有一件第i件物品，所以要预留c[i]的空间来存放一件第i种物品。总而言之，如果放当前物品i的话，它的状态就是它自己"i"，而不是上一个"i-1"。

所以说状态转移方程为：

$$f[i][j]=max(f[i-1][j],f[i][j-c[i]]+w[i])$$

注意完全背包的顺序问题： 因为每种背包都是无限的。当我们把i从1到N循环时，dp[v]表示容量为v在前i种背包时所得的价值，这里我们要添加的不是前一个背包，而是当前背包。所以我们要考虑的当然是当前状态。为01背包中要按照v=V...0的逆序来循环。这是因为要保证

第i次循环中的状态dp[i][v]是由状态dp[i-1] [v-c[i]]递推而来。这正是为了保证每件物品只选一次，保证在考虑“选入第i件物品”这件策略时，依据的是一个绝无已经选入第i件物品的 子结果dp[i-1][v-c[i]]。而现在完全背包的特点恰是每种物品可选无限件，所以在考虑“加选一件第i种物品”这种策略时，却正需要一个可能已选入第i种物品的子结果dp[i][v-c[i]]，所以就可以并且必须采用v=0...V的顺序循环。
完全背包还有考虑是否要求恰好放满背包等问题，可以好好研究：

有的题目要求“恰好装满背包”时的最优解，有的题目则并没有要求必须把背包装满。一种区别这两种问法的实现方法是在初始化的时候有所不同。
如果是第一种问法，要求恰好装满背包，那么在初始化时除了f[0]为0其它f[1..V]均设为-∞，这样就可以保证最终得到的f[N]是一种恰好装满背包的最优解。
如果并没有要求必须把背包装满，而是只希望价格尽量大，初始化时应该将f[0..V]全部设为0。
为什么呢？可以这样理解：初始化的f数组事实上就是在没有任何物品可以放入背包时的合法状态。如果要求背包恰好装满，那么此时只有容量为0的背包可能被价值为0的nothing“恰好装满”，其它容量的背包均没有合法的解，属于未定义的状态，它们的值就都应该是-∞了。如果背包并非必须被装满，那么任何容量的背包都有一个合法解“什么都不装”，这个解的价值为0，所以初始时状态的值也就全部为0了。

第一种方法：转化为01背包问题求解既然01背包问题是最基本的背包问题，那么我们可以考虑把完全背包问题转化为01背包问题来解。最简单的想法是，考虑到第i种物品最多选V/c[i]件，于是可以把第i种物品转化为V/c[i]件费用及价值均不变的物品，然后求解这个01背包问题。这样完全没有改进基本思路的时间复杂度，但这毕竟给了我们将完全背包问题转化为01背包问题的思路：将一种物品拆成多件物品。

参考博客:
https://blog.csdn.net/Thousa_Ho/article/details/78156678
https://wenku.baidu.com/view/eea4a76b0b1c59eef8c7b497.html
https://www.cnblogs.com/shihuajie/archive/2013/04/27/3046458.html

```
#include <stdio.h>
#include <string.h>
#include <algorithm>

using namespace std;
const int maxn = 50000 + 10;
const int INF = -0X7ffff;
int w[maxn],v[maxn],dp[maxn];

int main(){
    int T,n,C;
    scanf("%d",&T);
```

```
    while(T--){
        scanf("%d%d",&n,&C);
        for(int i = 1; i <= C; i++)dp[i] = INF;
        dp[0] = 0;
        for(int i = 0; i < n; i++)scanf("%d%d",&w[i],&v[i]);
        for(int i = 0; i < n; i++){
            for(int j = w[i]; j <= C; j++){ //从前往后递推，这样才能保证-
                dp[j] = max(dp[j],dp[j-w[i]] + v[i]);
            }
        }
        if(dp[C] > 0)printf("%d\n",dp[C]);
        else printf("NO\n");
    }
    return 0;
}
```

最长(不)上升或下降子序列

Java编写与解释的博客。

先说 O(N*N) 的解法，第 i 个元素之前的最长递增子序列的长度要么是 1 （单独成一个序列），要么就是第 i-1 个元素之前的最长递增子序列加 1 ，这样得到状态方程：

```
LIS[i] = max{1,LIS[k]+1}  (∀k<i, arr[i] > arr[k])
```

这样 arr[i] 才能在 arr[k] 的基础上构成一个新的递增子序列。

```
//题目连接 : http://acm.nyist.edu.cn/JudgeOnline/problem.php?pid=17
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;
const int maxn = 10000 + 10;

int dp[maxn]; /* dp[i]记录到[0,i]数组的LIS */
int maxx;/* LIS长度，初始化为1 */

int LIS(char *arr, int n) {
    for (int i = 0; i < n; i++) {
        dp[i] = 1;
        for (int j = 0; j < i; j++) // 注意i只遍历比它小的元素
            if (arr[j] < arr[i]) dp[i] = max(dp[i], dp[j] + 1);  //改
```

```
        maxx = max(maxx, dp[i]);
    }
    return maxx;
}

/* 递归输出LIS，因为数组dp还充当了“标记”作用 */
void printLis(char *arr, int index) {
    bool isLIS = false;
    if (index < 0 || maxx == 0)return;
    if (dp[index] == maxx) {
        --maxx;
        isLIS = true;
    }
    printLis(arr, --index);
    if (isLIS) printf("%c ", arr[index + 1]);
}

int main() {
    char s[maxn];
    int n;
    scanf("%d\n",&n);
    while(n--){
        maxx = 1;
        scanf("%s",s);
        printf("%d\n",LIS(s,strlen(s)));
        //printLis(s,strlen(s)-1);printf("\n");
    }
    return 0;
}
```

然后就是 nlog(n) 的解法：

参考博客:

https://segmentfault.com/a/1190000002641054

https://blog.csdn.net/joylnwang/article/details/6766317

```
//题目连接 : http://poj.org/problem?id=3903
#include <stdio.h>
#include <algorithm>
#include <vector>
#include <string.h>
using namespace std;
```

```cpp

const int INF = 0x3f3f3f3f;
const int maxn = 100000 + 5;
int a[maxn], dp[maxn], pos[maxn], fa[maxn];
vector<int> ans;

//用于最长非递减子序列种的lower_bound函数
int cmp(int a,int b){
    return a <= b;
}

//最长上升子序列
//dp[i]表示长度为i+1的上升子序列的最末尾元素 找到第一个比dp末尾大的来代替
int LIS(int n){
    memset(dp, 0x3f, sizeof(dp));
    pos[0] = -1;
    int i,lpos;
    for (i = 0; i < n; ++i){
        dp[lpos = (lower_bound(dp, dp + n, a[i]) - dp)] = a[i];
        pos[lpos] = i;    // *靠后打印
        fa[i] = (lpos ? pos[lpos - 1] : -1);
    }
    n = lower_bound(dp, dp + n, INF) - dp;
    for (i = pos[n - 1]; ~fa[i]; i = fa[i]) ans.push_back(a[i]);
    ans.push_back(a[i]);
    return n;
}

//非递减的LIS
int LISS(int n){
    memset(dp, 0x3f, sizeof(dp));
    pos[0] = -1;
    int i,lpos;
    for (i = 0; i < n; i++){
        dp[lpos = (lower_bound(dp, dp + n, a[i],cmp) - dp)] = a[i]; /
        pos[lpos] = i;    // *靠后打印
        fa[i] = (lpos ? pos[lpos - 1] : -1);
    }
    n = lower_bound(dp, dp + n, INF) - dp;
    for (i = pos[n - 1]; ~fa[i]; i = fa[i]) ans.push_back(a[i]);
    ans.push_back(a[i]);
    return n;

```

```
}

int main(){
    // freopen("in.txt","r",stdin);
    int n;
    while(~scanf("%d",&n)){
        ans.clear();
        for(int i = 0;i < n; i++)scanf("%d",&a[i]);
        printf("%d\n",LIS(n));
       // for(int i = ans.size()-1; i >= 0; i--) printf("%d ",ans[i])
    }
    return 0;
}
```

最长公共子序列

最长公共子序列利用动态规划的方式解决，具有最优子结构性质，和重叠子问题性质， `dp[i][j]` 记录序列 `Xi` 和 `Yi` 的最长公共子序列的长度，当 `i = 0` 或 `j =0` 时，空序列时 `Xi` 和 `Yi` 的最长公共子序列，其余情况如下

$$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$$

这里也给出一篇讲的好的博客

```
//题目连接 : http://poj.org/problem?id=1458
#include <stdio.h>
#include <iostream>
#include <string.h>
using namespace std;
const int maxn = 1000 + 10;
int dp[maxn][maxn];
int path[maxn][maxn]; //记录路径

```

```
int LCS(char *s1,char *s2){
    int len1 = strlen(s1)-1,len2 = strlen(s2)-1;//注意例如 abcfbc的str
    for(int i = 0; i <= len1; i++) dp[i][0] = 0;
    for(int i = 0; i <= len2; i++) dp[0][i] = 0;
    for(int i = 1; i <= len1; i++){
        for(int j = 1; j <= len2; j++)
            if(s1[i] == s2[j]){
                dp[i][j] = dp[i-1][j-1] + 1;
                path[i][j] = 1;
            }
            else if(dp[i-1][j] >= dp[i][j-1]) {
                dp[i][j] = dp[i-1][j];
                path[i][j] = 2;
            }
            else {
                dp[i][j] = dp[i][j-1];
                path[i][j] = 3;
            }
    }
    return dp[len1][len2];
}

//打印解
void print(int i,int j,char *s){
    if(i == 0 || j == 0)return ;
    if(path[i][j] == 1){
        print(i-1,j-1,s);
        printf("%c ",s[i]);
    }else if(path[i][j] == 2){
        print(i-1,j,s);
    }else print(i,j-1,s);
}

int main(){
    char s1[maxn],s2[maxn];
    while(~scanf("%s%s",s1+1,s2+1)){ //注意s1[0]不取-注意例如 abcfbc的st
        memset(path,0,sizeof(path));
        printf("%d\n",LCS(s1,s2));
        //print(strlen(s1)-1,strlen(s2)-1,s1);
    }
    return 0;
}

```

拓扑排序

有向无环图上的一种排序方式，我的这篇博客也讲解了一下。可以两种写法:

```cpp
//题目链接 : https://vjudge.net/contest/169966#problem/O
#include <bits/stdc++.h>

using namespace std;
const int maxn = 100 + 10;

int n, m;
int in[maxn];
vector<int>G[maxn];

void topo(){
    queue<int>q;
    for(int i = 1; i <= n ; i++)
        if(in[i] == 0)
            q.push(i);
    bool flag = true;
    while(!q.empty()){
        int cur = q.front();
        q.pop();
        if(flag){
            printf("%d",cur);
            flag = false;
        }
        else
            printf(" %d",cur);
        for(int i = 0; i < G[cur].size(); i++){
            if(--in[G[cur][i]] == 0)
                q.push(G[cur][i]);
        }
    }
}

int main(int argc, char const **argv)
{
    int from, to;
    while(~scanf("%d%d", &n, &m) && (n || m)){
```

```
        memset(in, 0, sizeof(in));
        for(int i = 1; i <= n; i++)
            G[i].clear();
        for(int i = 0; i < m; i++){
            scanf("%d%d", &from, &to);
            in[to]++; //度
            G[from].push_back(to);
        }
        topo();
        printf("\n");
    }
    return 0;
}
```

```
#include <iostream>
#include <queue>
#include <string.h>
using namespace std;
#define maxn  5005

typedef struct { ///邻接表实现
    int next_arc;///下一条边
    int point;
} Arc; ///边的结构体,

Arc arc[maxn];///储存每条边
int node[maxn],vis[maxn],top[maxn];///储存每个顶点，node[i]表示第i个顶点指
int n, m, t;

void dfs(int v) {
    for (int i = node[v]; i != -1; i = arc[i].next_arc) {
        if (!vis[arc[i].point]) {
            dfs(arc[i].point);
        }
    }
    vis[v] = 1;
    top[--t] = v;
}


int main() {
```

```
    int a, b;
    while (cin >> n >> m && (m || n)) {
        memset(node, -1, sizeof(node));
        memset(vis, 0, sizeof(vis));
        for (int i = 1; i <= m; i++) {
            cin >> a >> b;
            arc[i].next_arc = node[a];///第一次是出发点，以后是下一个
            arc[i].point = b;
            node[a] = i;
            vis[arc[i].point] = 0;
        }
        t = n;
        for (int j = 1; j <= n; j++) if (!vis[j]) dfs(j);
        for (int i = 0; i < n - 1; i++)
            cout << top[i] << " ";
        cout << top[n - 1] << endl;
    }
    return 0;
}
```

欧拉路径和回路

首先看定义：

欧拉回路：

- (1) 图 G 是连通的，不能有孤立点存在。
- (2) 对于无向图来说度数为奇数的点个数为 0，对于有向图来说每个点的入度必须等于出度。

欧拉路径：

- (1) 图 G 是连通的，无孤立点存在。
- (2) 对于无向图来说，度数为奇数的的点可以有 2 个或者 0 个，并且这两个奇点其中一个为起点另外一个为终点。对于有向图来说，可以存在两个点，其入度不等于出度，其中一个入度比出度大 1，为路径的起点；另外一个出度比入度大 1，为路径的终点。判断连通的方式有 DFS 或者并查集，然后再判断一下是否满足条件即可，之前做的欧拉回路的几个好题在我的github仓库

```cpp
#include <bits/stdc++.h>

const int maxn = 1000 + 5;
using namespace std;

bool vis[maxn];
int in[maxn];
int n, m;
vector<int>G[maxn];

void dfs(int u){
    vis[u] = 1;
    for(int i = 0; i < G[u].size(); i++){
        if(!vis[G[u][i]])
            dfs(G[u][i]);
    }
}

int main(){
    //freopen("in.txt","r",stdin);
    int from, to;
    while(scanf("%d", &n) != EOF && n){
        scanf("%d", &m);
        memset(vis, 0, sizeof(vis));
        memset(in, false, sizeof(in));
        for(int i = 1; i <= n; i++)
            G[i].clear();
        bool flag = true;
        for(int i = 0; i < m; i++){
            scanf("%d%d",&from, &to);
            G[from].push_back(to);
            G[to].push_back(from);
            in[from]++;
            in[to]++;
        }
        dfs(1);
        for(int i = 1; i <= n; i++)
            if(in[i] % 2 != 0)
                flag = false;
        for(int i = 1; i <= n; i++)
            if(!vis[i])
                flag = false;
```

```
        cout << (flag ? 1 : 0) << endl;
    }
    return 0;
}
```

```

#include <bits/stdc++.h>

const int maxn = 1000 + 5;
using namespace std;

int n, m, parent[maxn],ranks[maxn],in[maxn];

void init(){
    for(int i = 0; i < maxn; i++)
        parent[i] = i;
    for(int i = 0; i < maxn; i++)
        ranks[i] = 1;
}

//int findRoot(int v){
//    return parent[v] == v ? v : parent[v] = findRoot(parent[v]);
//}

int findRoot(int v){
    while(parent[v] != v){
        parent[v] = parent[parent[v]];
        v = parent[v];
    }
    return v;
}

void unions(int a, int b){
    int aRoot = findRoot(a);
    int bRoot = findRoot(b);
    if (aRoot == bRoot)
        return;
    if (ranks[aRoot] < ranks[bRoot])
        parent[aRoot] = bRoot;
    else if(ranks[aRoot] > ranks[bRoot]){
        parent[bRoot] = aRoot;
```

```
    }else {
        parent[aRoot] = bRoot;
        ranks[bRoot]++;
    }
}


int main(){
    //freopen("in.tat","r",stdin);
    int u, v;
    while(scanf("%d", &n) != EOF && n){
        init();
        scanf("%d", &m);
        memset(in,0,sizeof(in));
        for(int i = 0; i < m; i++){
            scanf("%d%d",&u, &v);
            unions(u, v);
            in[u]++;
            in[v]++;
        }
        bool flag = true;
        int temp = findRoot(1);
        for(int i = 1; i <= n; i++)
            if(findRoot(i) != temp)
                flag = false;
        for(int i = 1; i <= n; i++)
            if(in[i] % 2 != 0)
                flag = false; //判断度
        cout << (flag ? 1 : 0) << endl;
    }
    return 0;
}

```

搜索

这里举几个比较好的题目: POJ3984，这个题目要求记录BFS最短路的路径，我这里使用三种方法:

- BFS+在结构体中记录路径
- BFS+记录路径

- DFS加回溯法

```cpp
//题目链接;http://poj.org/problem?id=3984
//题目大意：给你一个5*5的矩阵，让你找一条路，从左上点，到右下点
//解题思路:利用BFS求解最短路，利用vector记录路径

//BFS+在结构体中记录路径
#include <stdio.h>
#include <iostream>
#include <stack>
#include <string.h>
#include <string>
#include <queue>
#pragma warning(disable : 4996)
using namespace std;
const int maxn = 100 + 10;

int n;
int map[maxn][maxn];
bool vis[maxn][maxn];
int dir[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};//对应上，右，下，左

struct Road {//路径记录
    int x,y,d;//d表示方向
    Road() {}//默认的构造函数
    Road(int a,int b,int c):x(a),y(b),d(c) {}//带参数的构造函数
};

struct node {//节点类型
    int x,y;
    vector<Road> v;//记录路径的结构体
};

bool check(node u) {
    if (!vis[u.x][u.y] && u.x >= 0 && u.x < n && u.y >= 0 && u.y < n
        return true;
    else
        return false;
}

void BFS(int x, int y) {
    queue<node>q;
```

```
    node now,next;
    now.x = x;
    now.y = y;
    now.v.push_back(Road(x,y,-1));
    q.push(now);
    while (!q.empty()) {
        now = q.front();
        q.pop();
        if (now.x == n - 1 && now.y == n-1) {
            for(int i = 0; i < now.v.size(); i++){
                printf("(%d, %d)\n",now.v[i].x,now.v[i].y);
            }
            break;
        }
        for (int i = 0; i < 4; i++) {
            next.x = now.x + dir[i][0];
            next.y = now.y + dir[i][1];
            if (check(next)) {
                vis[next.x][next.y] = true;
                next.v = now.v;
                next.v[now.v.size()-1].d = i;
                next.v.push_back(Road(next.x,next.y,-1));
                q.push(next);
            }
        }
    }
}

int main() {
    //freopen("in.txt", "r", stdin);
    flag = false;
    n = 5;
    memset(vis, 0, sizeof(vis));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &map[i][j]);
        }
    }
    BFS(0, 0);
    return 0;
}
```

```cpp
//BFS+记录路径
#include <stdio.h>
#include <iostream>
#include <string.h>

using namespace std;
const int maxn = 100 + 10;

int dir[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};
bool vis[maxn][maxn];
int map[maxn][maxn];

struct Node {
    int x,y,pre;
};

Node m_queue[maxn];

bool Check(int x,int y){
    if(!map[x][y] && !vis[x][y] && x >= 0 && x < 5&&y >= 0 && y < 5)r
    else return false;
}

void print(int u){
    if(m_queue[u].pre != -1){
        print(m_queue[u].pre);
        printf("(%d, %d)\n",m_queue[u].x,m_queue[u].y);
    }
}

void BFS(int x,int y){
    int head = 0,rear = 1;
    m_queue[head].x = x;
    m_queue[head].y = y;
    m_queue[head].pre = -1;
    vis[x][y] = true;
    while(head < rear){
        for(int i = 0; i < 4; i++){
            int xx = m_queue[head].x + dir[i][0];
            int yy = m_queue[head].y + dir[i][1];
            if(Check(xx,yy)){
                vis[xx][yy] = 1;
```

```
                m_queue[rear].x = xx;
                m_queue[rear].y = yy;
                m_queue[rear].pre = head;
                rear++;
            }
            if(xx == 4&&yy == 4)print(rear - 1);
        }
        head++;//出队
    }
}


int  main(){
    //freopen("in.txt","r",stdin);
    int n = 5;
    memset(vis,0,sizeof(vis));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            scanf("%d",&map[i][j]);
        }
    }
    printf("(0, 0)\n");
    BFS(0,0);
    return 0;
}
```

```
//DFS加回溯法
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <string>
#include <vector>
#include <stack>

using namespace std;
const int maxn = 100+10;
const int INF = 0x3f3f3f3f;

int n,k,ans = INF;
int map[maxn][maxn];
bool vis[maxn][maxn];
int dir[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};
```

```
17
18  struct node {
19      int x,y;
20  };
21  vector<node>dict;
22  stack<node>s;
23
24  void memcpy(stack<node>s){
25      dict.clear();
26      while(!s.empty()){
27          dict.push_back(s.top());
28          s.pop();
29      }
30  }
31
32  bool Check(int x,int y){
33      if(!vis[x][y] && map[x][y] != 1 && x >= 0 && x < n && y >= 0 && y
34      else return false;
35  }
36
37  void DFS(int x,int y,int step){//深刻理解递归的含义
38      node now;
39      now.x = x;
40      now.y = y;
41      s.push(now);
42      if(now.x == n - 1 &&now.y == n - 1){
43          if(step < ans){//记录最短的路径
44              ans = step;
45              memcpy(s);
46          }
47      }
48      for(int i = 0; i < 4; i++){
49          int xx = now.x + dir[i][0];
50          int yy = now.y + dir[i][1];
51          if(Check(xx,yy)){
52              vis[xx][yy] = true;
53              DFS(xx,yy,step + 1);
54              vis[xx][yy] = false;//回溯
55          }
56      }
57      s.pop();//反弹的时候重新还原栈
58  }
59
```

```
int main(){
    //freopen("in.txt","r",stdin);
    n = 5;
    memset(vis,false,sizeof(vis));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            scanf("%d",&map[i][j]);
        }
    }
    DFS(0,0,0);
    for(int i = dict.size() - 1; i >= 0; i--){
        printf("(%d, %d)\n",dict[i].x,dict[i].y);
    }
    return 0;
}
```

这里再给出DFS的一个好题，印象比较深刻的

```
//题目链接：https://vjudge.net/contest/169966#problem/V
//题目大意：有一个正方形的 战场看，边长为1000，西南方向在坐标原点，战场上有n 个敌
            //要你从西边界出发，东边界离开，求西边界和东边界上的坐标，坐标要尽量
//解题思路：按照 刘汝佳书上的思路，先判断有无解，从上边界开始DFS如果可以一直到达
          //否则，也是从上边界开始，dfs和东西边界 相连的圆(如果没有就是1000),
          //主要还是DFS的运用
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <math.h>

const int maxn = 1000 + 10;
const double W = 1000.0;
using namespace std;

double x[maxn], y[maxn], r[maxn],Left,Right;
bool vis[maxn];
int n;

bool isinterect(int u, int v) {
    return sqrt((x[u] - x[v])*(x[u] - x[v]) + (y[u] - y[v])*(y[u] - y
}

```

```
void check_circle(int u) {
    if (x[u] <= r[u])Left = min(Left, y[u] - sqrt(r[u] * r[u] - x[u]
    if (x[u] + r[u] >= W)Right = min(Right, y[u] - sqrt(r[u] * r[u] -
}

bool dfs(int u) {// 检查是否有解
    if (vis[u] == true)return false;
    vis[u] = true;
    if (y[u] <= r[u])return true;  //刚好碰到边界  以及  相切都可以
    for (int i = 0; i < n; i++)if (isinterect(u,i) && dfs(i))return t
    check_circle(u);
    return false;
}

int main() {
    //freopen("in.txt", "r", stdin);
    while (scanf("%d", &n) != EOF) {
        memset(vis, false, sizeof(vis));
        Left = Right = W;
        bool flag = true;
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf%lf", &x[i], &y[i], &r[i]);
        }
        for (int i = 0; i < n; i++) {
            if (y[i] + r[i] >= W) {  //从上往下  DFS
                if (dfs(i)) { flag = false; break; }//如果能够从最上面 
            }
        }
        if (flag) printf("0.00 %.2lf 1000.00 %.2lf\n", Left,Right);
        else printf("IMPOSSIBLE\n");
    }
    return 0;
}
```

最小生成树

直接给出模板 prim 和 kruskal (prim 是堆优化的)

另外，我的另外一篇博客也总结了这两个算法。

```
//Prim模板
//题目链接 : http://poj.org/problem?id=1287
```

```cpp
#include <stdio.h>
#include <string.h>
#include <vector>
#include <queue>

using namespace std;
const int maxn = 100 + 10;
const int INF = 1<<30;

struct Node{
    int v,w;
    Node (){}
    Node (int v,int w):v(v),w(w){}
    bool operator < (const Node&rhs ) const {
        return rhs.w < w;
    }
};

int n,d[maxn];
bool vis[maxn];
int Map[maxn][maxn];

void init(){
    for(int i = 0; i < maxn; i++)vis[i] = false;
    for(int i = 0; i < maxn; i++)d[i] = INF;
}

int Prim(int s){
    priority_queue<Node>q;
    q.push(Node(s,0));
    int ans = 0;
    while(!q.empty()){
        Node Now = q.top(); q.pop();
        int v = Now.v;
        if(vis[v]) continue;
        ans += Now.w;
        vis[v] = true;
        for(int i = 1; i <= n; i++){
            int w2 = Map[v][i];
            if(!vis[i] && d[i] > w2){
                d[i] = w2;
                q.push(Node(i,d[i]));
            }
```

```
        }
    }
    return ans;
}

int main(){
    //freopen("in.txt","r",stdin);
    int m,a,b,c;
    while(~scanf("%d",&n)&&n){
        scanf("%d",&m);
        init();
        for(int i = 1; i <= n; i++){
            Map[i][i] = INF;
            for(int j = 1; j < i; j++)Map[i][j] = Map[j][i] = INF;
        }
        for(int i = 0; i < m; i++){
            scanf("%d%d%d",&a,&b,&c);
            if(c < Map[a][b])Map[a][b] = Map[b][a] = c; //注意重边，取小
        }
        printf("%d\n",Prim(1));
    }
    return 0;
}
```

```
//题目链接 : http://poj.org/problem?id=1287
//kruskal模板
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <queue>

using namespace std;
const int maxn = 1e5 + 10;

int Fa[maxn],Rank[maxn];

void init(){
    for(int i = 0; i <= maxn; i++)Fa[i] = i;
    for(int i = 0; i <= maxn; i++)Rank[i] = 1;
}

int Find(int v){
```

```
19      return Fa[v] == v ? v : Fa[v] = Find(Fa[v]);
20  }
21
22  void Union(int x, int y){
23      int fx = Find(x);
24      int fy = Find(y);
25      if (fx == fy)return;
26      if (Rank[fx] < Rank[fy])
27          Fa[fx] = fy;
28      else{
29          Fa[fy] = fx;
30          if (Rank[fx] == Rank[fy])Rank[fy]++;
31      }
32  }
33
34  struct Edge{
35      int u,v,w;
36      Edge(){}
37      Edge(int u,int v,int w):u(u),v(v),w(w){}
38  }edge[maxn*2];
39
40
41  int cmp(const void *a,const void *b){
42      Edge *aa = (Edge*)a;
43      Edge *bb = (Edge*)b;
44      return aa->w > bb->w ? 1 : -1; //降序
45  }
46
47  int krusal(int n,int m){
48      qsort(edge,m,sizeof(edge[0]),cmp);
49      int ans = 0;
50      int cnt = 0;
51      for(int i = 0; i < m; i++){
52          int u = edge[i].u;
53          int v = edge[i].v;
54          if(Find(u) != Find(v)){
55              Union(u,v);
56              cnt++;
57              ans += edge[i].w;
58          }
59          if(cnt == n-1)break;
60      }
61      return ans;
```

```
}

int main(){
    //freopen("in.txt","r",stdin);
    int n,m;
    while(~scanf("%d",&n)&&n){
        scanf("%d",&m);
        init();
        for(int i = 0; i < m; i++) scanf("%d%d%d",&edge[i].u,&edge[i]
        printf("%d\n",krusal(n,m));
    }
    return 0;
}
```

最短路

dijkstra， bellman_ford， floyd， SPFA

最经典的Hdu1874畅通工程续

```
//堆优化dijkstra
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e4 + 10;
const int INF = 1e9;

struct Node{
    int v,w;
    Node(int v,int w):v(v),w(w){}
    bool operator < (const Node&rhs) const {
        return rhs.w < w;
    }
};

vector<Node>G[maxn];
bool vis[maxn];
int d[maxn];
int n,m;

void init(){
    for(int i = 0; i < maxn; i++)G[i].clear();
    for(int i = 0; i < maxn; i++)vis[i] = false;
    for(int i = 0; i < maxn; i++)d[i] = INF;
```

```
}

int dijkstra(int s,int e){ //传入起点终点
    priority_queue<Node>q;
    q.push(Node(s,0));
    d[s] = 0;
    while(!q.empty()){
        Node now = q.top(); q.pop();
        int v = now.v;
        if(vis[v])continue;
        vis[v] = true;
        for(int i = 0; i < G[v].size(); i++){
            int v2 = G[v][i].v;
            int w = G[v][i].w;
            if(!vis[v2] && d[v2] > w+d[v]){
                d[v2] = w+d[v];
                q.push(Node(v2,d[v2]));
            }
        }
    }
    return d[e];
}

int main(){
    //freopen("in.txt","r",stdin);
    int s,e;
    while(~scanf("%d%d",&n,&m)){
        init();
        for(int i = 0; i < m; i++){
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            G[x].push_back(Node(y,z));
            G[y].push_back(Node(x,z));
        }
        scanf("%d%d",&s,&e);
        int ans = dijkstra(s,e);
        if(INF != ans)printf("%d\n",ans);
        else printf("-1\n");
    }
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
const int maxn = 1000;
#define INF 0x1f1f1f1f
using namespace std;
bool flag[maxn];
int graph[maxn][maxn],low[maxn];

void DIJ(int n, int s) {
    memset(flag, false, sizeof(flag));
    flag[s] = true;
    for (int i = 0; i < n; i++)low[i] = graph[s][i];
    for (int i = 1; i < n; i++) {
        int min = INF, p;
        for (int j = 0; j < n; j++)
            if (!flag[j] && min > low[j]) {
                min = low[j];
                p = j;
            }
        flag[p] = true;
        for (int j = 0; j < n; j++)
            if (!flag[j] && low[j] > graph[p][j] + low[p])
                low[j] = graph[p][j] + low[p];
    }
}

int main() {
    int n, m, begin, t;
    while (~scanf("%d%d",&n,&m)) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i == j)
                    graph[i][j] = 0;
                else
                    graph[i][j] = INF;
        for (int i = 1; i <= m; i++) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            if (z < graph[x][y]) graph[x][y] = graph[y][x] = z;
        }
        cin >> begin >> t;
        DIJ(n, begin);
        if (low[t] < INF) cout << low[t] << endl;
```

```
        else cout << "-1" << endl;
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
#define INF 0x1f1f1f1f
using namespace std;
const int maxn = 1000;
vector<pair<int, int> >Edge[maxn];

int n, m, dist[maxn];
void init() {
    for (int i = 0; i < maxn; i++)Edge[i].clear();
    for (int i = 0; i < maxn; i++)dist[i] = INF;
}

int main() {
    int s, t;
    while (cin >> n >> m) {
        init();
        for (int i = 0; i < m; i++) {
            int x, y, z;
            cin >> x >> y >> z;
            Edge[x].push_back(make_pair(y, z));
            Edge[y].push_back(make_pair(x, z));
        }
        cin >> s >> t;
        priority_queue<pair<int, int>>q;
        dist[s] = 0;
        q.push(make_pair(-dist[s], s));
        while (!q.empty()) {
            int now = q.top().second;
            q.pop();
            for (int i = 0; i < Edge[now].size(); i++) {
                int v = Edge[now][i].first;
                if (dist[v] > dist[now] + Edge[now][i].second) {
                    dist[v] = dist[now] + Edge[now][i].second;
                    q.push(make_pair(-dist[v], v));
                }
            }
        }
```

```
        if (dist[t] < INF)cout << dist[t] << endl;
        else cout << "-1" << endl;
    }
    return 0;
}
```

```
//(3)bellman_ford
#include <bits/stdc++.h>
#define maxn 1000
#define INF 0x1f1f1f1f
using namespace std;

struct Edge {
    int u, v;
    int weight;
};
Edge edge[maxn * 2];

int dist[maxn];

void bellman_ford(int s, int n, int m) {
    for (int i = 0; i < n; i++) dist[i] = INF;
    dist[s] = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < 2 * m; j++)
            if (dist[edge[j].u] > dist[edge[j].v] + edge[j].weight)
                dist[edge[j].u] = dist[edge[j].v] + edge[j].weight;
}

int main() {
    int n, m, s, t;
    while (cin >> n >> m) {
        for (int i = 0; i < m; i++) {
            scanf("%d%d%d", &edge[i].u, &edge[i].v, &edge[i].weight);
            edge[i + m].v = edge[i].u;
            edge[i + m].u = edge[i].v;
            edge[i + m].weight = edge[i].weight;
        }
        cin >> s >> t;
        bellman_ford(s, n, m);
        if (dist[t] < INF) cout << dist[t] << endl;
        else cout << "-1" << endl;
```

```
    }
    return 0;
}
```

```
//(4)floyd
#include <bits/stdc++.h>
#define maxn 1000
#define INF 0x1f1f1f1f
using namespace std;

int graph[maxn][maxn];
int low[maxn];

void floyd(int n) {
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                graph[i][j] = min(graph[i][j], graph[i][k] + graph[k]
}

int main() {
    int n, m, s, t;
    while (cin >> n >> m) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i == j)
                    graph[i][j] = 0;
                else
                    graph[i][j] = INF;
        for (int i = 1; i <= m; i++) {
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            if (graph[x][y] > z) graph[x][y] = graph[y][x] = z;
        }
        floyd(n);
        scanf("%d%d", &s, &t);
        if (graph[s][t] < INF) cout << graph[s][t] << endl;
        else cout << "-1" << endl;
    }
    return 0;
}
```

```cpp
//(5)SPFA
#include <bits/stdc++.h>
#define maxn  1000
#define INF 0x1f1f1f1f
using namespace std;

int graph[maxn][maxn],flag[maxn],dist[maxn];

void SPFA(int s, int n) {
    queue<int>q;
    memset(flag, false, sizeof(flag));
    for (int i = 0; i < n; i++) dist[i] = INF;
    q.push(s);
    dist[s] = 0;
    flag[s] = true;
    while (!q.empty()) {
        int temp = q.front();
        q.pop();
        flag[temp] = false;
        for (int i = 0; i < n; i++) {
            if (dist[i] > dist[temp] + graph[temp][i]) {
                dist[i] = dist[temp] + graph[temp][i];
                if (!flag[i]) {
                    q.push(i);
                    flag[i] = true;
                }
            }
        }
    }
}

int main() {
    int n, m, s, t;
    while (cin >> n >> m) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i == j)
                    graph[i][j] = 0;
                else
                    graph[i][j] = INF;
        for (int i = 0; i < m; i++) {
            int x, y, z;
```

```
            cin >> x >> y >> z;
            if (z < graph[x][y]) graph[x][y] = graph[y][x] = z;
        }
        cin >> s >> t;
        SPFA(s, n);
        if (dist[t] < INF) cout << dist[t] << endl;
        else cout << "-1" << endl;

    }
    return 0;
}

```

```
//(6)pair+SPFA
#include <bits/stdc++.h>
#define INF 0x1f1f1f1f
using namespace std;

const int maxn = 1000;
vector<pair<int, int> >Edge[maxn];

int n, m,dist[maxn];
bool flag[maxn];

void init() {
    for (int i = 0; i < maxn; i++)Edge[i].clear();
    for (int i = 0; i < maxn; i++)flag[i] = false;
    for (int i = 0; i < maxn; i++)dist[i] = INF;
}

int main() {
    int s, t;
    while (cin >> n >> m) {
        init();
        for (int i = 0; i < m; i++) {
            int x, y, z;
            cin >> x >> y >> z;
            Edge[x].push_back(make_pair(y, z));
            Edge[y].push_back(make_pair(x, z));
        }
        cin >> s >> t;
        queue<int>q;
```

```
        dist[s] = 0;
        flag[s] = true;
        q.push(s);
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            flag[now] = false;
            for (int i = 0; i < Edge[now].size(); i++) {
                int v = Edge[now][i].first;
                if (dist[v] > dist[now] + Edge[now][i].second) {
                    dist[v] = dist[now] + Edge[now][i].second;
                    if (!flag[Edge[now][i].first]) {
                        q.push(Edge[now][i].first);
                        flag[Edge[now][i].first] = true;
                    }
                }
            }
        }
        if (dist[t] < INF)cout << dist[t] << endl;
        else cout << "-1" << endl;
    }
    return 0;
}
```

这里给出一个比较特殊的最短路题目

> 1.最短路的条数
> 2.多条时选择消防员(最短路经过的点的那些权值)多的那一条
> 3.记录路径

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e4 + 10;
const int INF = 1e9;
typedef long long LL;

struct Node{
    int v,w;
    Node(int v,int w):v(v),w(w){}
    bool operator < (const Node&rhs) const {
```

```
        return rhs.w < w;
    }
};

vector<Node>G[maxn];
bool vis[maxn];
int Pa[maxn],Weight[maxn],TotalWeight[maxn],TotalRoad[maxn];
int d[maxn];
int n,m,s,e;

void init(){
    for(int i = 0; i < maxn; i++)G[i].clear();
    for(int i = 0; i < maxn; i++)vis[i] = false;
    for(int i = 0; i < maxn; i++)d[i] = INF;
    for(int i = 0; i < maxn; i++)Pa[i] = -1; //记录路径
    for(int i = 0; i < maxn; i++)TotalRoad[i] = 0;
    for(int i = 0; i < maxn; i++)TotalWeight[i] = 0;
}

void PrintPath(int x){
    if(Pa[x] == -1) printf("%d",x);
    else {
        PrintPath(Pa[x]);
        printf(" %d",x);
    }
}

int dijkstra(int s,int e){ //传入起点终点
    priority_queue<Node>q;
    q.push(Node(s,0));
    d[s] = 0, Pa[s] = -1, TotalWeight[s] = Weight[s],TotalRoad[s] = 1
    while(!q.empty()){
        Node now = q.top(); q.pop();
        int v = now.v;
        if(vis[v])continue;
        vis[v] = true;
        for(int i = 0; i < G[v].size(); i++){
            int v2 = G[v][i].v;
            int w = G[v][i].w;
            if(!vis[v2] && d[v2] > w+d[v]){
                d[v2] = w + d[v];
                TotalWeight[v2] = TotalWeight[v] + Weight[v2];
                Pa[v2] = v;
```

```
                TotalRoad[v2] = TotalRoad[v];
                q.push(Node(v2,d[v2]));
            }
            else if(!vis[v2] && d[v2] == w+d[v]){
                if(TotalWeight[v2] < TotalWeight[v] + Weight[v2]){ //
                    TotalWeight[v2] = TotalWeight[v] + Weight[v2];
                    Pa[v2] = v;
                    //q.push(Node(v2,d[v2]));不需要入队了
                }
                TotalRoad[v2] += TotalRoad[v]; //加上之前的条数
            }
        }
    }
    return d[e];
}

int main(){
    //freopen("in.txt","r",stdin);
    int a,b,c;
    scanf("%d%d%d%d",&n,&m,&s,&e);
    for(int i = 0; i < n; i++)scanf("%d",&Weight[i]);
    init();
    for(int i = 0; i < m; i++){
        scanf("%d%d%d",&a,&b,&c);
        G[a].push_back(Node(b,c));
        G[b].push_back(Node(a,c));
    }
    int ans = dijkstra(s,e);
    //if(INF != ans)printf("%d\n",ans); else printf("-1\n");
    printf("%d %d\n",TotalRoad[e],TotalWeight[e]);
    PrintPath(e);
    printf("\n");
    return 0;
}
```

GCD和LCM

注意一下 n 个数的 GCD 和 LCM

```
//题目连接 : http://acm.hdu.edu.cn/showproblem.php?pid=1019
#include <bits/stdc++.h>
using namespace std;
```

```cpp
const int maxn = 100 + 10;

int gcd(int a,int b){
    return b == 0?a:gcd(b,a%b);
}

int lcm(int a,int b){
    return a/gcd(a,b)*b;
}

int ngcd(int v[],int n){
    if(n == 1) return v[0];
    return gcd(v[n-1],ngcd(v,n-1));
}

int nlcm(int v[],int n){
    if(n == 1) return v[0];
    return lcm(v[n-1],nlcm(v,n-1));
}
int main(){
    int n,m,a[maxn];
    scanf("%d",&n);
    while(n--){
        scanf("%d",&m);
        for(int i = 0; i < m; i++)scanf("%d",&a[i]);
        //printf("%d\n",ngcd(a,m));
        printf("%d\n",nlcm(a,m));
    }
    return 0;
}
```

埃拉托斯特尼筛法

素数和分解定理等看另一篇博客总结。

比较经典的快速筛素数，给个例题，用 BFS 和筛素数解决

```cpp
#include <stdio.h>
#include <string.h>
#include <queue>
using namespace std;
const int maxn = 10000 + 10;
int prime[maxn],s,e,vis[maxn];
```

```cpp
bool is_prime[maxn],flag;

//素数筛选的模板 :    |埃式筛法|
int Sieve(int n) {
    int k = 0;
    for(int i = 0; i <= n; i++)is_prime[i] = true;
    is_prime[0] = false,is_prime[1] = false;
    for(int i = 2; i <= n; i++) {
        if(is_prime[i]) {
            prime[k++] = i;
            for(int j = i*i; j <= n; j += i)is_prime[j] = false; // 
        }
    }
    return k;
}

struct Node {
    int v,step;
    Node(){}
    Node(int v,int step):v(v),step(step){}
};

void cut(int n,int *a){  //将各位存到a数组中
    int index = 3;
    while(n > 0){
        a[index--] = n%10;
        n /= 10;
    }
}

int rev(int *a){  //还原
    int s = 0;
    for(int i = 0; i < 4; i++)s = s*10 + a[i];
    return s;
}

void BFS(int s,int step) {
    queue<Node>q;
    Node now,next;
    q.push(Node(s,step));
    vis[s] = 1;
    while(!q.empty()) {
        now = q.front();q.pop();
```

```
50          if(now.v == e) {
51              flag = true;
52              printf("%d\n",now.step);
53              return ;
54          }
55          int a[4],b[4];
56          cut(now.v,a);
57          for(int i = 0; i < 4; i++){
58              memcpy(b,a,sizeof(b));
59              for(int j = 0; j <= 9; j++){
60                  b[i] = j;
61                  next.v = rev(b);
62                  if(next.v < 10000 && next.v > 1000 && is_prime[next.v
63                      vis[next.v] = 1;
64                      next.step = now.step+1;
65                      q.push(next);
66                  }
67              }
68          }
69      }
70  }
71  int main() {
72      int T;
73      Sieve(maxn); //先把素数筛选出来
74      scanf("%d",&T);
75      while(T--) {
76          flag = false;
77          memset(vis,0,sizeof(vis));
78          scanf("%d%d",&s,&e);
79          BFS(s,0);
80          if(flag == 0)printf("Impossible\n");
81      }
82      return 0;
83  }
```

唯一分定理

任何大于1的自然数，都可以唯一分解成有限个质数的乘积

例如对于大于1的自然数n，

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} = \prod_{i=1}^{k} p_i^{\alpha_i}$$

这里P_i均为质数，其指数a_i是正整数。

这样的分解称为的**标准分解式**。

具体的不细说，看一个比较简单的模板题其余还有题目在我的代码仓库。

```
1  //题目连接 : https://acm.sdut.edu.cn/onlinejudge2/index.php/Home/Index
2  #include <stdio.h>
3  using namespace std;
4  const int maxn = 30000;
5
6  int prime[maxn];
7  bool is_prime[maxn];
8
9  //素数筛选的模板 :    |埃式筛法|
10 int Sieve(int n) {
11     int k = 0;
12     for(int i = 0; i <= n; i++)is_prime[i] = true;
13     is_prime[0] = false,is_prime[1] = false;
14     for(int i = 2; i <= n; i++) {
15         if(is_prime[i]) {
16             prime[k++] = i;
17             for(int j = i*i; j <= n; j += i)is_prime[j] = false; // 
18         }
19     }
20     return k;
21 }
22
23 int main(){
24     int T,n;
```

```
    scanf("%d",&T);
    int len = Sieve(maxn);
    while(T--){
        scanf("%d",&n);
        int ans[maxn],k = 0;
        for(int i = 0; i < len; i++){ //唯一分解定理
            while(n % prime[i] == 0){
                ans[k++] = prime[i];
                n /= prime[i];
            }
        }
        for(int i = 0; i < k; i++)printf(i == 0 ? "%d": "*%d",ans[i])
        printf("\n");
    }
    return 0;
}
```

扩展欧几里得

基本算法：对于不完全为 0 的非负整数 a，b， gcd（a，b） 表示 a，b 的最大公约数，必然存在整数对 x，y，使得 gcd（a，b）=ax+by 。，具体用处在于:

- 求解不定方程；
- 求解模线性方程（线性同余方程）；
- 求解模的逆元；

给出一个讲的不错的博客。
这里给出一个求解不定方程组解的测试程序：

```
//扩展欧几里得的学习
#include <stdio.h>

int exgcd(int a,int b,int &x,int &y){
    if(b == 0){
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b,a%b,x,y);  //d 就是gcd(a,b)
```

```
    int t = x;
    x = y;
    y = t-(a/b)*y;
    return d;
}

bool linear_equation(int a,int b,int c,int &x,int &y){
    int d = exgcd(a,b,x,y);  //d是gcd(a,b)//求出a*x+b*y = gcd(a,b)的一

    printf("%d*x + %d*y = %d(gcd(a,b))的一些解是\n",a,b,d);
    printf("%d %d\n",x,y); //第一组
    for(int t = 2; t < 10; t++){   //输出 a*x + b*y = gcd(a,b)的其他一
        int x1 = x + b/d*t;
        int y1 = y - a/d*t;
        printf("%d %d\n",x1,y1); //其余组
    }
    printf("\n\n"); //第一组


    if(c%d) return false;
    int k = c/d;    //上述解乘以 c/gcd(a,b)就是 a*x +b*y = c的解
    x *= k; y *= k;    //求得的只是其中一组解
    return true;
}

int main(){
    //freopen("in.txt","r",stdin);
    int a = 6,b = 15,c = 9;  //求解 6*x + 15*y = 9的解
    int x,y;
    int d = exgcd(a,b,x,y);

    if(!linear_equation(a,b,c,x,y))printf("无解\n");

    printf("%d*x + %d*y = %d的一些解是\n",a,b,c);
    printf("%d %d\n",x,y); //第一组
    for(int t = 2; t < 10; t++){  //输出其他的一些解
        int x1 = x + b/d*t;
        int y1 = y - a/d*t;
        printf("%d %d\n",x1,y1); //其余组
    }
    return 0;
}
```

再给出一个比较简单的模板题

```
//题目链接：http://poj.org/problem?id=1061
//题目大意：中文题，给出青蛙A，B的起点坐标,以及跳跃距离m,n以及环的长度L，求什么时
/*解题思路：  假设跳了T次以后，青蛙1的坐标便是x+m*T,青蛙2的坐标为y+n*T。它们能够
              得到ax+by==c，直接利用欧几里得扩展定理可以得到一组x,y但是这组x,y
              首先,当gcd(a,b)不能整除c的时候是无解的，当c能整除gcd(a,b)时，把
              我们知道它的通解为x0+b/gcd(a,b)*t要保证这个解是不小于零的最小的整
              我们先计算当x0+b/gcd(a,b)*t=0时的t值,
              此时的t记为t0=-x0/b/gcd(a,b)（整数除法），代入t0如果得到的x小于
#include <stdio.h>
using namespace std;
typedef long long LL;

LL exgcd(LL a,LL b,LL &x,LL &y){
    if(b == 0){
        x = 1;
        y = 0;
        return a;
    }
    LL d = exgcd(b,a%b,x,y);
    LL t = x;
    x = y;
    y = t-(a/b)*y;
    return d;
}

int main(){
    //freopen("in.txt","r",stdin);
    LL x,y,m,n,L,T,P;
    while(cin>>x>>y>>m>>n>>L){
        LL a,b,c;
        a = n-m;
        b = L;
        c = x-y;
        LL d = exgcd(a,b,T,P);
        if(c%d!=0){ cout<<"Impossible"<<endl; continue; }
        T = T*(c/d);
        P = P*(c/d);

        LL k = T*d/b;
        k = T-k*b/d;
```

```
        if(k < 0) k = k+b/d;
        printf("%lld\n",k);
    }
    return 0;
}
```

欧拉函数

欧拉函数只是工具：

- 提供 1 到 N 中与 N 互质的数；
- 对于一个正整数 N 的素数幂分解为 `N = P1^q1 * P2^q2 * ... * Pn^qn`，则欧拉函数 `γ(N) = N*(1-1/P1) * (1-1/P2) * ... * (1-1/Pn)`。

这里也给出一个博客讲解。

```
//欧拉函数模板
//题目连接 : https://vjudge.net/contest/185827#problem/G
//题目意思 : 就是要你求1~n互素的对数(注意1~1不要重复)
#include <stdio.h>
#include <math.h>
const int maxn = 50000;
int phi[maxn+1], phi_psum[maxn+1];

int euler_phi(int n){
    int m = sqrt(n+0.5);
    int ans = n;
    for(int i = 2; i < m; i++)if(n % i == 0){
        ans = ans/i*(i-1);
        while(n % i == 0)n /= i;
    }
    if(n > 1)ans = ans/n*(n-1);
    return ans;
}


//筛素数的方法,求解1~n所有数的欧拉phi函数值
void phi_table(int n) {
    phi[1] = 0; //这里不计算第一个1,1和1,1是重复的，等下直接2*phi_psum[n] +
    for(int i = 2; i <= n; i++) if(phi[i] == 0)
```

```
        for(int j = i; j <= n; j += i) {
            if(phi[j] == 0) phi[j] = j;
            phi[j] = phi[j] / i * (i-1);
        }
}

int main() {
    int n;
    phi_table(maxn);
    phi_psum[0] = 0;
    for(int i = 1; i <= maxn; i++)phi_psum[i] = phi_psum[i-1] + phi[i
    while(scanf("%d", &n) == 1 && n) printf("%d\n",2*phi_psum[n] + 1)

    return 0;
}
```

这里再贴上大牛的模板：

```
const int MAXN = 10000000;
bool check[MAXN+10];
int phi[MAXN+10];
int prime[MAXN+10];
int tot;//素数的个数

// 线性筛（同时得到欧拉函数和素表）
void phi_and_prime_table(int N) {
    memset(check,false,sizeof(check));
    phi[1] = 1;
    tot = 0;
    for(int i = 2; i <= N; i++) {
        if( !check[i] ) {
            prime[tot++] = i;
            phi[i] = i-1;
        }
        for(int j = 0; j < tot; j++) {
            if(i * prime[j] > N)break;
            check[i * prime[j]] = true;
            if( i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            } else {
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
```

```
            }
        }
    }
}
```

快速幂

乘法快速幂看这里。

```
#include <stdio.h>

//计算(a*b)%c
long long mul(long long a,long long b,long long mod) {
    long long res = 0;
    while(b > 0){
        if( (b&1) != 0)  // 二进制最低位是1 --> 加上 a的 2^i 倍, 快速幂是
            res  = ( res + a) % mod;
        a = (a << 1) % mod;    // a = a * 2    a随着b中二进制位数而扩大
        b >>= 1;               // b -> b/2     右移  去掉最后一位 因为当
    }
    return res;
}

//幂取模函数
long long pow1(long long a,long long n,long long mod){
    long long res = 1;
    while(n > 0){
        if(n&1)
            res = (res * a)%mod;
        a = (a * a)%mod;
        n >>= 1;
    }
    return res;
}


// 计算 ret = (a^n)%mod
long long pow2(long long a,long long n,long long mod) {
    long long res = 1;
    while(n > 0) {
        if(n & 1)
```

```
            res = mul(res,a,mod);
        a = mul(a,a,mod);
        n >>= 1;
    }
    return res;
}

//递归分治法求解
int pow3(int a,int n,int Mod){
    if(n == 0)
        return 1;
    int halfRes = pow1(a,n/2,Mod);
    long long res = (long long)halfRes * halfRes % Mod;
    if(n&1)
        res = res * a % Mod;
    return (int)res;
}

int main(){
    printf("%lld\n",mul(3,4,10));
    printf("%lld\n",pow1(2,5,3));
    printf("%lld\n",pow2(2,5,3));
    printf("%d\n",pow3(2,5,3));
    return 0;
}
```

矩阵快速幂

这个也是比较常用的，主要还是在于常数矩阵的求解和递推公式的求解，这里给出一个博客讲解和自己写过的一道模板题，**以及后面自己的总结。**

$$F_n = \begin{cases} x, & x < 10 \\ \sum_{i=0}^{9} ai * F_{x-i-1}, & x \geq 10 \end{cases}$$

$$\begin{pmatrix} F_n \\ F_{n-1} \\ F_{n-2} \\ F_{n-3} \\ F_{n-4} \\ F_{n-5} \\ F_{n-6} \\ F_{n-7} \\ F_{n-8} \\ F_{n-9} \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}^{n-9} \begin{pmatrix} F_9 \\ F_8 \\ F_7 \\ F_6 \\ F_5 \\ F_4 \\ F_3 \\ F_2 \\ F_1 \\ F_0 \end{pmatrix}$$

转换之后，直接把问题转化为求矩阵的 n-9 次幂。

```
//题目链接：https://vjudge.net/contest/172365#problem/A
//题目大意：函数满足——If x < N f(x) = x.
//If x >= N f(x) = a0 * f(x - 1) + a1 * f(x - 2) + a2 * f(x - 3) + ……
//ai(0<=i<=9) can only be 0 or 1 .
//要求f(k) % mod;
//解题思路：构造矩阵求解N*N矩阵

#include <stdio.h>
#include <string.h>
const int N = 10;
using namespace std;

int Mod;
struct Matrix  {
    int m[N][N];
    Matrix(){}
};

void Init(Matrix &matrix){
    for(int i = 0;i < N; i++)scanf("%d",&matrix.m[0][i]);
    for(int i = 1;i < N; i++){
        for(int j = 0;j < N; j++){
            if(i == (j+1))matrix.m[i][j] = 1;
            else matrix.m[i][j] = 0;
```

```
        }
    }
}

//矩阵相乘
Matrix Mul(Matrix &a,Matrix &b){
    Matrix c;
    for(int i = 0; i < N; i++){
        for(int j = 0;j < N; j++){
            c.m[i][j] = 0;
            for(int k = 0; k < N; k++)c.m[i][j] += a.m[i][k]*b.m[k][j
            c.m[i][j] %= Mod;
        }
    }
    return c;
}

//矩阵幂
Matrix Pow(Matrix& matrix, int k) {
    Matrix res;
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            if (i == j) res.m[i][j] = 1;
            else res.m[i][j] = 0;
    while (k) {
        if (k & 1) res = Mul(matrix,res);
        k >>= 1;
        matrix = Mul(matrix,matrix);
    }
    return res;
}

int main() {
    int k,a[N];
    while (~scanf("%d%d",&k,&Mod)) {
        Matrix matrix;
        Init(matrix);
        if (k < 10) { printf("%d\n",k % Mod); continue; }
        matrix = Pow(matrix,k-9);
        int sum = 0;
        for (int i = 0; i < N; i++)sum += matrix.m[0][i] * (9 - i) %
        printf("%d\n",sum%Mod);
    }
```

```
68      return 0;
69  }
```