背包问题：（找最大值）

V(i,j)=max｛V(i-1,j)，V(i-1,j-w(i))+v(i)｝

eg：

```
#include<stdio.h>
int max_(int x,int y){
    if(x>y)
    return x;
    else
    return y;
}
int main(){
    int t;
    while(~scanf("%d",&t)){
        while(t--){
            int m,u;
            int array_n[1024],array_v[1024],result[1024]={0};//n 价值 v 体
积//result 当前背包容量下的已有的价值
            scanf("%d%d",&m,&u);
            for(int i=0;i<m;i++){
                scanf("%d",&array_n[i]);
            }for(int i=0;i<m;i++){
                scanf("%d",&array_v[i]);
            }
            for(int i=0;i<m;i++){//对前i个物品进行操作
                for(int j=u;j>=array_v[i];j--){
                    result[j]=max_(result[j],result[j-array_v[i]]+array_n[i]);
                }
            }int max=-1;

            for(int i=0;i<=u;i++){
                max=max_(max,result[i]);
            }
            printf("%d\n",max);
        }
    }return 0;
}
```

背包问题：（找最小值）

```
#include<stdio.h>
#include<string.h>
int dp[50024]={};
int min(int x,int y){
    if(x>y)
        return y;
    else
        return x;
}
int main(){
    int t;
    while(~scanf("%d",&t)){
        while(t--){
            int e,f;
            scanf("%d%d",&e,&f);
            int n;
```

```
            scanf("%d",&n);
            int m[100024];
            int v[100024];
            memset(dp,50001,50024*sizeof(int));
            dp[0]=0;
            for(int i=0;i<n;i++){
                scanf("%d%d",&m[i],&v[i]);
            }
            for(int i=0;i<n;i++){
                for(int j=v[i];j<=f-e;j++){
                    dp[j]=min(dp[j],dp[j-v[i]]+m[i]);
                }
            }
            if(dp[f-e]!=dp[50023])
                printf("The minimum amount of money in the piggy-bank is
%d.\n",dp[f-e]);
            else
                printf("This is impossible.\n");
        }
    }return 0;
}
```

错排：
d（n）=（n-1）*(d(n-1)+d(n-2))

匈牙利算法：
eg:

```
#include<cstdio>
#include<cstring>
using namespace std;
//有n只公牛和m只母牛，然后每只公牛都可以和几只的母牛配对。
//在每只公牛只能配对一只母牛的情况下，求能为牛们配对最多多少对？
int n,m,ans;
int match[210];//母牛i的配偶是公牛match[i]
bool chw[210];//在此趟询问中，母牛i是否被询问过
bool mp[210][210];//公牛i与母牛j是否有关系

bool find_ans(int x)
{
    for(int i=1;i<=m;i++)
    {
        if(mp[x][i]==true&&chw[i]==true)
        {
            chw[i]=false;
            if(match[i]==0||find_ans(match[i])==true)
            //母牛没有配偶||匹配该母牛的公牛能否换一头母牛匹配
            {
                match[i]=x;
                return true;
            }
        }
    }
    return false;
}

int main()
{
```

```
    while(scanf("%d %d",&n,&m)!=EOF)
    {
        memset(mp,false,sizeof(mp));
        for(int i=1;i<=n;i++)
        {
            int k,x;
            scanf("%d",&k);
            for(int j=1;j<=k;j++)
            {
                scanf("%d",&x);
                mp[i][x]=true;
            }
        }
        ans=0;
        memset(match,0,sizeof(match));
        for(int i=1;i<=n;i++)
        {
            memset(chw,true,sizeof(chw));
            if(find_ans(i)==true)
                ans++;
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

选数问题（和为素数）：

```
#include <stdio.h>

int sushu(int n);
void fun(int n, int m);

int a[22], b[21], n, k, t = 0;

int main()
{
    int i;
    scanf("%d%d", &n, &k);
    int b[k];
    for(i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    fun(n, k);
    printf("%d", t);
}

void fun(int n, int m)//从n里面选m个数字
{
    int i, sum;
    if(m == 0)
    {
        sum = 0;
        for(i = 0; i < k; i++)
            sum += b[i];
        if(sushu(sum))
            t++;
        return;
    }
    for(i = n ; i >= m; i--)
```

```
    {
        b[m - 1] = a[i];
        fun(i - 1, m - 1);
    }
}

int sushu(int n)
{
    int i;
    for(i = 2; i < n; i++)
        if(n % i == 0)
            break;
    if(i == n || n == 2)
        return 1;
    else
        return 0;
}
```

装箱问题：

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

int n;
int d[20005];
int a[35];
int main(){
    int w;
    scanf("%d%d", &w, &n);
    int i, j;
    for (i = 0; i < n; i++){
        scanf("%d", &a[i]);
    }
    memset(d, 0, sizeof(d));
    for (i = 0; i < n; i++){
        for (j = w; j >= a[i]; j--)
            d[j] = max(d[j], d[j - a[i]] + a[i]);

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

dp思想：

```
#include<stdio.h>
int max_(int x,int y){
    if(x<y)
        return y;
    else
        return x;
}
int main(){
    int x;
    while(~scanf("%d",&x)){
        if(x==0)
            return 0;
```

```
        int array[1024];
        int group[1024];
        int count=0;
        while(x--){
            scanf("%d",&array[count++]);
            group[count-1]=array[count-1];
        }
        for(int i=0;i<count;i++){
            for(int j=0;j<i;j++){
                if(array[j]<array[i])
                    group[i]=max_(group[i],group[j]+array[i]);
            }

        }
        int sum=0;
        for(int i=0;i<count;i++){
            sum=max_(sum,group[i]);
        }
        printf("%d\n",sum);
    }return 0;
}
```

最长公共子序列(LCS)问题：
公式：
a[i-1]==b[i-1]-->dp[i-1][j-1]+1
a[i]!=b[i]-->max{dp[i-1][j],dp[i][j-1]}
eg：

```
#include<stdio.h>
#include<string.h>
int max(int x,int y){
    if(x<y)
        return y;
    else
        return x;
}
int dp[1024][1024];
int main(){
    char a[1024],b[1024];
    while(~scanf("%s%s",a,b)){
        int len=max(strlen(a),strlen(b));
        for(int i=0;i<=len;i++){
            for(int j=0;j<=len;j++){
                dp[i][j]=0;
            }
        }
        for(int i=1;i<=strlen(a);i++){
            for(int j=1;j<=strlen(b);j++){
                if(a[i-1]==b[j-1]){
                    dp[i][j]=dp[i-1][j-1]+1;
                }else{
                    dp[i][j]=max(dp[i][j-1],dp[i-1][j]);
                }
            }
        }
        printf("%d\n",dp[strlen(a)][strlen(b)]);
    }return 0;
}
```

LCS一维数组（慢）：

```
#include<cstdio>
#include<iostream>
int x,maxl,ans,f[100005];
int arr1[100005],arr2[100005];
using namespace std;
int main(){
    scanf("%d",&x);
    for(int i=0;i<x;i++){
        scanf("%d",&arr1[i]);
    }
    for(int i=0;i<x;i++){
        scanf("%d",&arr2[i]);
    }
    for(int i=0;i<x;i++){
        maxl=0;
        for(int j=0;j<x;j++){
            int ll=f[j];
            if(arr1[i]==arr2[j]&&f[j]<maxl+1)
                f[j]=maxl+1;
            maxl=max(maxl,ll);
        }
    }
    for(int i=0;i<x;i++){
        if(ans<f[i]){
            ans=f[i];
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

埃式塞：（快速求质数）

```
#include <iostream>
#include <cstdio>
#include <cmath>
using namespace std;

int main()
{
  // 埃氏筛

  // 0 表示质数，1 非质数（1 非质非合）
  int arr[101] = { 0 };  // 0 去掉，1~100
  arr[1] = 1;  // 1 既不是质数也不是合数

  for (int i = 2; i <= 100; ++i) {
    if (arr[i] == 1)
      continue;
    for (int i_times = i * 2; i_times <= 100; i_times += i) {
      arr[i_times] = 1;   // i 的 n 倍必然是合数
    }
  }

  int cnt = 0;
  for (int i = 1; i <= 100; ++i) {
    cnt += (arr[i] + 1) % 2;
  }
```

```
  cout << cnt << endl;
  return 0;
}
```

线性筛素数：

```
#include<bits/stdc++.h>
using namespace std;
int n;
map<int,int> Is_Prime;//保存每一个数是否为质数
vector<int> Prime;//保存全部质数
int main()
{
    scanf("%d",&n);//筛选2~n内的质数
    for(int i=2;i<=n;i++) Is_Prime[i]=1;//先默认全部都是质数
    for(int i=2;i<=n;i++)
    {
        if(Is_Prime[i]) Prime.push_back(i);//如果这个数是质数，就将其保存下来
        for(int j=0;j<Prime.size();j++)//将这个数与已有的质数进行操作
        {
            Is_Prime[i*Prime[j]]=0;//将这个数与该质数的积标记为非质数
            if(!(i%Prime[j])) break;//如果当前数是该质数的倍数，就退出循环
        }
    }
    //以下为输出部分
    printf("%d\n",Prime.size());
    for(int i=0;i<Prime.size();i++) printf("%d ",Prime[i]);
    return 0;
}
```

欧拉筛（上述筛，数组形式/速度更快）：

```
#include<cstdio>
#include<map>
#include<vector>
#include<iostream>
using namespace std;
int n,q;
int ls[100000005];
int prime[100000005];
int main(){
    scanf("%d%d",&n,&q);
    int count=0;
    for(int i=2;i<=n;i++){
        ls[i]=1;
    }for(int i=2;i<=n;i++){
        if(ls[i])
            prime[count++]=i;
        for(int j=0;j<count&&prime[j]*i<=n;j++){
            ls[i*prime[j]]=0;
            if(!(i%prime[j]))
                break;
        }
    }
    while(q--){
        int x;
        scanf("%d",&x);
        printf("%d\n",prime[x-1]);
    }
}
```

sort排序：

```
#include<iostream>
#include<algorithm>
using namespace std;
int array[1024];
int main(){
    int count=0;
    while(cin>>array[count]){
        count++;
        if(getchar()=='\n'){
            break;
        }
    }
    sort(array,array+count);
    for(int i=0;i<count;i++){
        cout<<array[i]<<endl;
    }
    return 0;
}
```

选择排序法：

```
#include<stdio.h>
int main()
{
    int array[5],n=5;
    for(int i=0;i<n;i++){
        scanf("%d",&array[i]);
    }
        int i,j,k,t;
        for(i=0;i<n-1;i++){
            k=i;
            for(j=i+1;j<n;j++){
                if(array[j]<array[k]){
                     k=j;
                     t=array[k];
                     array[k]=array[i];
                     array[i]=t;
                }
            }
        }
    for(int i=0;i<n;i++){
        printf("%d",array[i]);
    }

    return 0;
}
```

快速求因子的函数：
（个数）

```
long long f(long long a){
    long long sum = 0;
    for(long long i = 1;i <= a/i;i++){
        if(a % i == 0)
            if(i * i != a)
                sum += 2;
            else
```

```
                sum++;
    }
    return sum;
}
```

（数据+个数）

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
int array[1024];
int f(int x){
    int now=0;
    for(int i=1;i*i<=x;i++){
        if(x%i==0){
            array[now++]=i;
            if(i!=x/i){
                array[now++]=x/i;
            }
        }
    }
    sort(array,array+now);
    return now;
}
int main(){
    int x;
    scanf("%d",&x);
    int sum=f(x);
    for(int i=0;i<sum;i++){
        printf("%d ",array[i]);
    }return 0;
}
```

二分查找（仅递增）：

```
#include<stdio.h>
//二分查找-C语言实现
//基本思路：将排序好的数据存放到数组里（不能是链表）
//         这只前中后标签，与中间元素比，若小于就将后变为原来的中
//         继续计算中，比较，循环，直至等于中，或循环结束。
int binsearch(int *sortedSeq, int seqLength, int keyData);
int main(){
    int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int location;
    int target = 4;
    location = binsearch(array, 9, target);
    printf("%d\n", location);
    return 0;
}
int binsearch(int *sortedSeq, int seqLength, int keyData){
    int low = 0, mid, high = seqLength - 1;

    while (low <= high){
        mid = (low + high) / 2;//奇数，无论奇偶，有个值就行
        if (keyData < sortedSeq[mid]){
            high = mid - 1;//是mid-1，因为mid已经比较过了
        }
        else if (keyData > sortedSeq[mid]){
            low = mid + 1;
```

```
        }
        else{
            return mid;
        }
    }
    return -1;
}
```

二分查找（不递减）：

```
#include<stdio.h>
int array[1000005];
int group[1000005];
int main(){
    int m,n;
    scanf("%d %d",&m,&n);
    for(int i=0;i<m;i++){
        scanf("%d",&array[i]);
    }
    for(int i=0;i<n;i++){
        int x;
        scanf("%d",&x);
        int low=0,mid,high=m-1;
        while(low<high){
            mid=(low+high)/2;
            if(x<array[mid]){
                high=mid-1;
            }
            else if(x>array[mid]){
                low=mid+1;
            }
            else{
                high=mid;
            }
        }
        if(array[high]==x){
            group[i]=high+1;
        }
        else
            group[i]=-1;
    }
    for(int i=0;i<n;i++){
        printf("%d ",group[i]);
    }
    return 0;
}
```

十进制转换为二进制代码：

```
#include <iostream>
using namespace std;
int main()
{
    int num;
    int length = 0;
    int n[20];
    cout << "十进制: ";
    cin >> num;
    //循环除2，把余数存储在数组中
    while (num / 2)
```

```
    {
        n[length] = num % 2;
        length++;
        num = num / 2;
    }
    //存储最后一个余数
    n[length] = num;
    length++;
    cout << "二进制：";
    //将余数从下往上输出
    for (int i = length - 1; i >= 0; i--)
    {
        cout << n[i];
    }
    return 0;
}
```

（简易，不能重叠）朋友圈问题（给定人数，已知几对关系，求朋友圈的问题）：

```
#include <stdio.h>
#include <stdlib.h>
int root[100010];
int search(int men)
{
    int now;
    now=men;
    while(root[men]!=men)
    {
        men=root[men];
    }
    if(now!=men)
    {
        root[now]=root[men];
    }
    return men;
}
int main()
{
    int n,m,num,i;
    scanf("%d%d",&n,&m);
    num=0;
    for(i=1;i<=n;i++)
        root[i]=i;
    while(m--)
    {
        int men1,men2,root1,root2;
        scanf("%d%d",&men1,&men2);
        root1=search(men1);
        root2=search(men2);
        if(root1!=root2)
        {
            root[men1]=men2;

        }
    }
    for(i=1;i<=n;i++){
        if(root[i]==i)
        num++;
    }
```

```
    printf("%d",num);
    return 0;
}
```

二分法统计数组值（有序）：

```
#include <stdio.h>
int GetFirstKey(int arr[], int left, int right, int len, int key)
{
    int mid;
    if (left > right)
    {
        return -1;
    }
    mid = left - (left - right) / 2;
    if (key == arr[mid])
    {
        if ((mid > 0 && arr[mid - 1] != key) || mid == 0)
        {
            return mid;
        }
        else
        {
            right = mid - 1;
        }
    }
    else if (arr[mid] < key)
    {
        left = mid + 1;
    }
    else
    {
        right = mid - 1;
    }
    return GetFirstKey(arr, left, right, len, key);
}

int GetLastKey(int arr[], int left, int right, int len, int key)
{
    int mid;
    if (left > right)
    {
        return -1;
    }
    mid = left - (left - right) / 2;
    if (key == arr[mid])
    {
        if ((mid < len - 1 && arr[mid + 1] != key || mid == len - 1))
        {
            return mid;
        }
        else
        {
            left = mid + 1;
        }
    }
    else if (arr[mid] < key)
    {
        left = mid + 1;
```

```
    }
    else
    {
        right = mid - 1;
    }
    return GetLastKey(arr, left, right, len, key);
}

int  main()
{
    int brr[] = { 1, 2, 3, 3, 3, 3, 4, 5};
    int len = sizeof(brr) / sizeof(brr[0]);
    int first = GetFirstKey(brr, 0, len - 1, len - 1, 3);
    int last = GetLastKey(brr, 0, len - 1, len - 1, 3);
    printf("%d\n", last - first + 1);
    return 0;
}
```

并查集问题（朋友的朋友是朋友形式）：

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 1010

int qw[MAX];
int maxn;

void init() {//遍历所有同学，使他们拥有初始值。
    for (int i = 0; i < MAX; i++) {
        qw[i] = i;
    }
}

void update(int minab, int maxab) {//如果最小的那个值为-1 说明这个人使qw的粉丝
    for (int i = 0; i < MAX; i++) {
        if (qw[i] == maxab) {
            qw[i] = minab;// -1 一定小于所有遍历过的值，使那个maxab学生的值也为-1
        }
    }
}

void print() {//用于调试的函数
    for (int i = 0; i <= maxn; i++) {
        cout << qw[i] << ' ';
    }
    cout << endl;
}

int count() {//遍历所有学生统计qw的粉丝。
    int ans = 0;
    for (int i = 0; i < MAX; i++) {
        if (qw[i] == -1) {
            ans++;
        }
    }
    return ans;
}

int main() {
```

```
    int n, m, fans;
    while (cin >> n >> m) {
        maxn = 0;
        if (n == 0 && m == 0) {
            break;
        }
        init();
        for (int i = 0; i < n; i++) {
            cin >> fans;
            maxn = max(maxn, fans);
            qw[fans] = -1;
        }
        int a, b;
        for (int i = 0; i < m; i++) {
            cin >> a >> b;
            maxn = max(maxn, max(a, b));
            update(min(qw[a], qw[b]), max(qw[a], qw[b]));
            // print();
        }
    //print();
        cout << count() << endl;
    }
    return 0;
}
```

并查集改良版：

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 6000

int qw[MAX];
int maxn;
int n, m, p;

void init() {//遍历所有同学，使他们拥有初始值。
    for (int i = 0; i < MAX; i++) {
        qw[i] = i;
    }
}

void update(int minab, int maxab) {//使下标最小的数值为根，和根有关系的变成根的数值
    for (int i = 0; i < MAX; i++) {
        if (qw[i] == maxab) {
            qw[i] = minab;
        }
    }
}

void print() {//用于调试的函数
    for (int i = 0; i <= maxn; i++) {
        cout << qw[i] << ' ';
    }
    cout << endl;
}

int count() {//遍历所有学生统计qw的粉丝。
    int ans = 0;
    for (int i = 0; i < MAX; i++) {
```

```
        if (qw[i] == -1) {
            ans++;
        }
    }
    return ans;
}

int main() {
    cin >> n >> m>> p;
        maxn = 0;
        init();
        int a, b;
        for (int i = 0; i < m; i++) {
            cin >> a >> b;
            maxn = max(maxn, max(a, b));
            update(min(qw[a], qw[b]), max(qw[a], qw[b]));
            // print();
        }
        print();
    while(p--){
        int x,y;
        cin>>x>>y;
        if(qw[x]==qw[y]){
            printf("Yes\n");
        }
        else{
            printf("No\n");
        }
    }
    return 0;
}
```

寻找最长回文子串：
动态规划找：
当 i == j，dp[i][j]是回文子串（单字符都是回文子串）；
当j - i < 3，只要s[i] == s[j]，则dp[i][j]是回文子串（如 aa，aba），否则不是；
当j - i >= 3，如果s[i] == s[j] && dp[i+1][j-1]，则dp[i][j]是回文子串，否则不是。
中间扩散找：

```
#include <stdio.h>
#include <string.h>
int main()
{
    char w[1210];
    int i,len,max,j,k,x,y;
    while(~scanf("%s",w))
    {
        int a[1210]={1},b[1210]={0};//a全部为一 表示回文串至少为1 b全部置0
        len=strlen(w);//得到字符串长度
        for(i=0;i<len;i++)//找奇数回文子串
        {
            j=i-1;//子串是回文的话至少长度为三
            k=i+1;
            while(j>=0 && k<len && w[j]==w[k])//先找到最小的是回文的串 再判断更多的
            {
                    x=j;
                    y=k;
                    a[i]=k-j+1;
                j--;//找有五位时是否还为回文字符串
```

```
                k++;
            }
        }
        for(i=0;i<len;i++)//偶数回文找法
        {
            j=i;
            k=i+1;
            while(j>=0 && k<len && w[j]==w[k])
            {
                    b[i]=k-j+1;
                j--;//先看中间是不是回文 再看两边
                k++;
            }
        }
        max=1;
        for(i=0;i<len;i++)
        {
            if(max<a[i])
                max=a[i];
            if(max<b[i])
                max=b[i];
        }
        printf("%d\n",max);
    }
    return 0;
}
```

全排类问题：

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

int arr[100010];

void printArr(int* arr, int size) {
    for (int i = 0; i < size; ++i) {
        if (i != 0) cout << ' ';
        cout << arr[i];
    }
    cout << endl;
}

int main()
{
    // sample input:
    // 4
    // 1 2 1 2

    // sample output:
    // 1 1 2 2
    // 1 2 1 2
    // 1 2 2 1
    // 2 1 1 2
    // 2 1 2 1
    // 2 2 1 1
    // P(4, 4) / (P(2, 2) * P(2, 2))

    int n;
```

```
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }


    sort(arr, arr + n);
    printArr(arr, n);

    // arr “改变了” 返回 true，否则返回 false
    while (next_permutation(arr, arr + n))
        printArr(arr, n);

    return 0;
}
```

迷宫问题：

```
#include<stdio.h>
int book[51][51],a[51][51];//book[][]等于0表示该点还没有走过 a[][]等于0表示该点不是障碍
int n,m,p,q,min=99999;
    int next[4][2]={
            {0,1}, //向右走一步
                    {1,0},//向下走一步
                    {0,-1},//向左走一步
                    {-1,0}};//向上走一步

void dfs(int x,int y,int step){   //step用来表示找到小红，小明走了多少步
    int tx,ty,k;
    if(x==p&& y==q){  //说明已经找到了小红
    /*
    还要说明一点：这里 为什么是（x，y），而不是（tx，xy）
    其实很简单 就是上一个dfs()函数传过来的坐标 ，做了这个dfs()函数的形参
    换句话说：就是判断点是否找到小红
    */

        if(step<min)
            min=step;
        return ;
        /*返回上一步，继续寻找其他路径（就是退回到上一个坐标，重新找其他路径）
           回到上一个dfs()函数

        */
        }

    for(k=0;k<=3;k++){   //下一步的坐标
        tx=x+next[k][0];
        ty=y+next[k][1];

            //判断是否越界,越界则重新进入for循环
            if(tx<1 || tx>n || ty<1 || ty>m)
                continue;
                //运行到这里，说明这条路，则需要换个方向，也就是重新进入for循环
            if(a[tx][ty]==0 && book[tx][ty]==0){
                book[tx][ty]=1; //标记这个点走过
                dfs(tx,ty,step+1); //进行下一步
                book[tx][ty]=0;   //重新尝试，退回到上一个点的位置
        }
```

```
        }
        return ;   //执行到这里，这层dfs()函数已经结束，则要回到上一层dfs()函数
}

int main(){
    int i,j,startx,starty;
    scanf("%d %d",&n,&m);    //输入迷宫的大小

    for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
            scanf("%d",&a[i][j]);  //输入迷宫的形状

    scanf("%d %d",&startx,&starty);  //小明的坐标
    scanf("%d %d",&p,&q);            //小红的坐标

    book[startx][starty]=1;          //起始点标记，就不会回到这个点了
    dfs(startx,starty,0);      //开始寻找最短路径

    printf("%d",min);         //输出最短路径
    return 0;
}
```

三角形路径求最大值（dp）：

```
#include<cstdio>
#include<iostream>
using namespace std;
int n,a[1002],i,j,ans,p;
int main(){
    scanf("%d",&n);
        for(i=n;i;i--){
            for(j=i;j<=n;j++){
               scanf("%d",&p);
                    a[j]=max(a[j],a[j+1])+p;
            }
        }
    for(i=1;i<=n;i++)
        ans=max(ans,a[i]);
    printf("%d",ans);
    return 0;
}
```

BFS（障碍为1，通路为0）：//(有些需要用数组标记)

```
#include <iostream>
#include <cstdio>
#include <queue>
using namespace std;
int Map[5][5];  //定义地图大小
int dir[4][2]= {{1,0},{-1,0},{0,-1},{0,1}};  //定义方向
int n,m,ans;
struct node
{
    int x,y,step;

} now,nextt;  //保存走步
int BFS(int x,int y)
{
    queue<node> q;
    int xx,yy,zz;
```

```
    Map[x][y]=2;  //走过初始点
    now.x=x;
    now.y=y;
    now.step=0;
    q.push(now);  //从当前点开始
    while(!q.empty())  //ture 非false的条件是非空
    {
        now=q.front();
        q.pop();
        for(int i=0; i<4; i++)  //遍历四个方向
        {
            xx=now.x+dir[i][0];
            yy=now.y+dir[i][1];  //走一步
            if(xx>=0&&xx<5&&yy>=0&&yy<5&&Map[xx][yy]!=1&&Map[xx][yy]!=2)  //可以
走
            {
                nextt.x=xx;
                nextt.y=yy;
                nextt.step=now.step+1;  //步数加一
                Map[now.x][now.y]=2;   //走过一个点
                if(Map[xx][yy]==3)  //到达终点
                    return nextt.step;
                q.push(nextt);

            }
            // for(int i=0; i<5; i++){      //打印地图
            //     for(int j=0; j<5; j++)
            //         cout << Map[i][j];
            //     cout << endl;
            // }
            // cout << endl;
        }
    }
    return -1;   //走不过去
}

int main()
{

    for(int i=0; i<5; i++)     //输入地图
        for(int j=0; j<5; j++)
            cin >> Map[i][j];
    Map[4][4]=3;  //定义终点
    ans=BFS(0,0);
    cout << ans<< endl;
    return 0;

}
```

岛屿问题：

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 1005

int n;
char mapp[MAXN][MAXN];
int vis[MAXN][MAXN];
int wsad[4][2] = {0, 1, 0, -1, 1, 0, -1, 0};
```

```
struct node {
    int x, y;
};

void bfs(node x) {
    queue<node> q;
    q.push(x);
    node t, nx;
    vis[x.x][x.y] = 1;
    mapp[x.x][x.y] = 'o';
    while (!q.empty()) {
        t = q.front();
        q.pop();
        for (int i = 0; i < 4; i++) {
            nx.x = t.x + wsad[i][0];
            nx.y = t.y + wsad[i][1];

            if ((nx.x >= 0 && nx.x < n) && (nx.y >= 0 && nx.y < n) && \
                vis[nx.x][nx.y] == 0 && mapp[nx.x][nx.y] != 'o') {
                vis[nx.x][nx.y] = 1;
                mapp[nx.x][nx.y] = 'o';
                q.push(nx);
            }
        }
    }
}

void Print() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << mapp[i][j] << ' ';
        }
        cout << endl;
    }
    cout << endl;
}

int main() {
    while(cin >> n) {
        int ans = 0;
        memset(vis, 0, sizeof(vis));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cin >> mapp[i][j];
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (mapp[i][j] == 'x') {
                    node x;
                    x.x = i;
                    x.y = j;
                    bfs(x);
                    // Print();
                    ans++;
                }
            }
```

```
        }
        cout << ans << endl;
    }
    return 0;
}
```

删数问题取最小值（去除前导零）：

```
#include<cstdio>
#include<cstring>
#include<iostream>
int main(){
    char array[300];
    scanf("%s",array);
    int x;
    scanf("%d",&x);
    int len=strlen(array);
    while(x){
        int i=0;
        while(array[i]<=array[i+1]){
            i++;
        }
        while(i<len-1){
            array[i]=array[i+1];
            i++;
        }
        len--;
        x--;
    }
    int flag=1;
    for(int i=0;i<len;i++)
    {
        if(array[i]=='0'&&i<len-1&&flag==1)
            continue;
        else{
            printf("%c",array[i]);
            flag=0;
        }
    }return 0;
}
```

求第 k 小的数（nth_element）：

```
#include<cstdio>
#include<iostream>
#include<algorithm>
using namespace std;
int array[5000005];
int main(){
    int n,k;
    scanf("%d %d",&n,&k);
    for(int i=0;i<n;i++){
        scanf("%d",&array[i]);
    }
    nth_element(array,array+k,array+n);
    printf("%d",array[k]);
    return 0;
}
```

拼数（string类型的使用）：

```cpp
#include<cstdio>
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
string array[25];
int main(){
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        cin>>array[i];
    }
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(array[i]+array[j]<array[j]+array[i]){
                swap(array[i],array[j]);
            }
        }
    }
    for(int i=0;i<n;i++)
    {
        cout<<array[i];
    }
    return 0;
}
```

最大子串和：

```cpp
#include<cstdio>
#include<iostream>
using namespace std;
int array[200005];
int main(){
    int n;
    scanf("%d",&n);
    int max=-99999;
    int sum=0;
    for(int i=0;i<n;i++){
        scanf("%d",&array[i]);
        sum+=array[i];
        if(sum>max){
            max=sum;
        }
        if(sum<0){
            sum=0;
        }
    }
    printf("%d",max);
    return 0;
}
```

求最大矩形和（二维）：

```cpp
#include<stdio>
#include<iostream>
using namespace std;
int ans,a[155][155],n;
int maxn(int a,int b){return a>b?a:b;}
//自定义求最大值
```

```
int main(){
    scanf("%d",&n);
    int i,j,k;
    for(i=1;i<=n;++i){
        for(j=1;j<=n;++j){
            scanf("%d",&a[i][j]);
            a[i][j]+=a[i-1][j];
        }
    }//如上，前缀和处理
    for(i=1;i<=n;++i){
        for(k=1;k<=i;++k){
            int f[150]={0},dp[150]={0};//f[j]表示压缩的矩形第j列的值
            for(j=1;j<=n;++j){           //其实可以不开数组，一个f就可以
                f[j]=a[i][j]-a[i-k][j];//求压缩的矩形第j列的值
                dp[j]=maxn(dp[j-1]+f[j],f[j]);//动态规划
                ans=maxn(ans,dp[j]);//更新答案
            }
        }
    }
    cout<<ans<<endl;//愉快AC
    return 0;
}
```

st表模板：

```
//预处理复杂度同为O(nlogn),查询时间上，ST表为O(1),线段树为O(logn)
#include<bits/stdc++.h>
using namespace std;
const int N=5e4+10;
int a[N];//原始输入数组
//st表,表示需要查询的数组的从下标i到下标i+2^j-1的最值
int mx[N][30];//最大值
int mn[N][30];//最小值

//预处理
void init(int n)
{
    for(int i=1;i<=n;i++){
        mx[i][0]=a[i];
        mn[i][0]=a[i];
    }

    for(int j=1;(1<<j)<=n;j++)
    {
        for(int i=1;i+(1<<j)<=n+1;i++){
            mn[i][j] = min(mn[i][j-1],mn[i+(1<<(j-1))][j-1]);
            mx[i][j] = max(mx[i][j-1],mx[i+(1<<(j-1))][j-1]);
        }
    }
}

//查询函数
int search(int l,int r)//l和r范围为1~n
{
    int k=log2(r-l+1);
    int t1=min(mn[l][k],mn[r-(1<<k)+1][k]);//区间最小值
    int t2=max(mx[l][k],mx[r-(1<<k)+1][k]);//区间最大值
    return t2-t1;
}
```

```
int main(){
//  freopen("in.txt","r",stdin);
    int n,q;scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    init(n);
    while(q--){
        int l,r;
        scanf("%d%d",&l,&r);
        printf("%d\n",search(l,r));
    }
    return 0;
}
```

lower_bound()与upper_bound()的用法：

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main()
{
    vector<int> ve(8, 0);
    for (int i = 0; i < 8; i++)
    {
        ve[i] = i;
    }
    auto low = lower_bound(ve.begin(), ve.end(), 3)-ve.begin();
    int upp = upper_bound(ve.begin(), ve.end(), 3) - ve.begin();
    cout<<low<<endl;
    cout << upp << endl;
}
```

```
3
4
```

kmp板子：

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <algorithm>
#include <cstdlib>
const int maxn=1000005;
using namespace std;
string s,p;
int nxtpos[maxn],lcpp[maxn],nxtpos0[maxn];//nxtpos0为未优化版nxtpos数组
void getnxt()
{
    nxtpos[0]=-1;
    nxtpos0[0]=-1;
    int len = p.length();
    int j=0,k=-1;
    while(j <= len - 1)
    {
        if(k==-1||p[j]==p[k])
        {
```

```
            j++;
            k++;
            nxtpos0[j]=k;
            if(p[j]!=p[k])
            {
                nxtpos[j] = k;
            }
            else
            {
                nxtpos[j] = nxtpos[k];
            }
        }
        else
        {
            k = nxtpos0[k];
        }
    }
}
void kmp()
{
    int i=0,j=0;
    int slen=s.length();
    int plen=p.length();
    while(i < slen)
    {
        if(j == -1||p[j] == s[i])
        {
            i++;
            j++;
        }
        else
        {
            j=nxtpos0[j];
        }
        if(j==plen)
        {
            printf("%d\n",i-j+1);
            i=i-j+1;
            j=0;
        }
    }
}
int main()
{
    cin>>s>>p;
    getnxt();
    kmp();
    for(int i=0;i<p.length();i++)
    {
        lcpp[i]=nxtpos0[i+1];
        printf("%d ",lcpp[i]);
    }
    return 0;
}
```

AC自动机模板：

```
#include <queue>
```

```cpp
#include <cstdlib>
#include <cmath>
#include <cstdio>
#include <string>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long ll;
const int maxn =  2*1e6+9;

int trie[maxn][26]; //字典树
int cntword[maxn];  //记录该单词出现次数
int fail[maxn];     //失败时的回溯指针
int cnt = 0;

void insertWords(string s){
    int root = 0;
    for(int i=0;i<s.size();i++){
        int next = s[i] - 'a';
        if(!trie[root][next])
            trie[root][next] = ++cnt;
        root = trie[root][next];
    }
    cntword[root]++;       //当前节点单词数+1
}
void getFail(){
    queue <int>q;
    for(int i=0;i<26;i++){       //将第二层所有出现了的字母扔进队列
        if(trie[0][i]){
            fail[trie[0][i]] = 0;
            q.push(trie[0][i]);
        }
    }

//fail[now]     ->当前节点now的失败指针指向的地方
//tire[now][i] -> 下一个字母为i+'a'的节点的下标为tire[now][i]
    while(!q.empty()){
        int now = q.front();
        q.pop();

        for(int i=0;i<26;i++){       //查询26个字母
            if(trie[now][i]){
                //如果有这个子节点为字母i+'a',则
//让这个节点的失败指针指向(((他父亲节点)的失败指针所指向的那个节点)的下一个节点)
                //有点绕,为了方便理解特意加了括号

                fail[trie[now][i]] = trie[fail[now]][i];
                q.push(trie[now][i]);
            }
            else//否则就让当前节点的这个子节点
                //指向当前节点fail指针的这个子节点
                trie[now][i] = trie[fail[now]][i];
        }
    }
}
```

```
int query(string s){
    int now = 0,ans = 0;
    for(int i=0;i<s.size();i++){    //遍历文本串
        now = trie[now][s[i]-'a'];  //从s[i]点开始寻找
        for(int j=now;j && cntword[j]!=-1;j=fail[j]){
            //一直向下寻找,直到匹配失败(失败指针指向根或者当前节点已找过).
            ans += cntword[j];
            cntword[j] = -1;    //将遍历国后的节点标记,防止重复计算
        }
    }
    return ans;
}

int main() {
    int n;
    string s;
    cin >> n;
    for(int i=0;i<n;i++){
        cin >> s ;
        insertWords(s);
    }
    fail[0] = 0;
    getFail();
    cin >> s ;
    cout << query(s) << endl;
    return 0;
}
```

数列分段（要求分段结果的最大值最小）：

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,m,x[100005],sum=0,max1=0;
int judge(int sum1)
{
    int num=1;
    int temp=0;
    for(int i=0;i<n;i++) //分段
    {
        temp+=x[i];
        if(temp>sum1)
        {
            num++;
            temp=x[i];
        }
    }
    if(num>m) //分出的段数大于该有的段数
        return 1;
    return 0;
}
int main()
{
    cin>>n>>m;
    for(int i=0;i<n;i++)
    {
        cin>>x[i];
        sum+=x[i];
        if(x[i]>max1)
```

```
            max1=x[i];
    }
    int l=max1,r=sum;  //查阅的满足条件的数介于数列的最大值和数列的中间值
    while(l<r)
    {
        int mid=(l+r)/2;
        if(judge(mid))
            l=mid+1;
        else
            r=mid;
    }
    cout<<r<<endl;
    return 0;
}
```

在一定区域内改变数值(dfs):

```
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int a[31][31],vis[31][31],n;
void dfs(int x,int y){
    if(x<0||x>n-1||y<0||y>n-1||a[x][y]==1||vis[x][y]==1) return;//出界或撞墙
    vis[x][y]=1;//标记
    dfs(x,y+1);//上下左右
    dfs(x,y-1);
    dfs(x+1,y);
    dfs(x-1,y);
}
int main(){
    int i,j;
    memset(vis,0,sizeof(vis));
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    for(i=0;i<n;i++) dfs(0,i);//四面开始遍历
    for(i=0;i<n;i++) dfs(i,0);
    for(i=n-1;i>=0;i--) dfs(i,n-1);
    for(i=n-1;i>=0;i--) dfs(n-1,i);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(vis[i][j]==1)
                printf("%d ",a[i][j]);//有标记，还输出0
            else if(vis[i][j]==0&&a[i][j]==1)
                printf("%d ",a[i][j]);//墙当然不变了
            else
                printf("2 ");//剩下的就是圈内的了
        }//分两部分是因为每行最后数字的后面没有空格
        printf("\n");
    }
    return 0;
```

```
}
```

最大正方形：

```
#include<cstdio>
#include<iostream>
using namespace std;
int array[105][105];
int f[105][105];
int main(){
    int m,n;
    int ans=0;
    scanf("%d%d",&m,&n);
    for(int i=1;i<=m;i++){
        for(int j=1;j<=n;j++){
            scanf("%d",&array[i][j]);
            if(array[i][j]==1){
                f[i][j]=min(min(f[i-1][j],f[i][j-1]),f[i-1][j-1])+1;
                ans=max(ans,f[i][j]);
            }
        }
    }
    printf("%d",ans);
    return 0;
}
```

```
#include<cstdio>
#include<iostream>
#include<string>
using namespace std;
const int maxn=11000000+5;
string ma="@#";
int mp[2*maxn]; //存放回文串长度+1
void manacher(string s){
    int len=s.size();
    for (int i=0; i<len; i++){
        ma+=s[i];
        ma+='#';
    }
    int nlen=ma.size();
    int mx=0,id=0;  //mx为最右端位置，id为能延伸到最右边的字符串的中心位置
    for (int i=0; i<nlen; i++){
        mp[i]=(i>mx)?1:min(mp[2*id-i],mx-i); //每次取小赋值，防止超过mx
        while (ma[i+mp[i]]==ma[i-mp[i]]) mp[i]++;
        if (i+mp[i]>mx){
            mx=i+mp[i];
            id=i;
        }
    }
}


int main(){

    ios::sync_with_stdio(false);
    string s;
    cin>>s;
    int len=s.size();
    manacher(s);
```

```
    int ans=0;
    for (int i=0; i<2*len+2; i++)
        ans=max(ans,mp[i]-1);
    cout<<ans;
    return 0;
}
```

逆元：（结果为1-n）
扩展欧几里得算法
扩展欧几里得算法是求整数x、y 使得 ax+by=1
单点查找素数快，在a与p互质而p不是质数也可以使用

```
#include <iostream>
using namespace std;
typedef long long ll;
void Exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0)    x=1,y=0;
    else
    {
        Exgcd(b, a%b, y, x);
        y-=a/b*x;
    }
}
int main()
{
    ll x,y,n,p;
    cin>>n>>p;
    for(int i=1;i<=n;i++)
    {
        Exgcd(i,p,x,y);
        x=(x%p+p)%p;
        cout<<x<<endl;
    }
    return 0;
}
```

快速幂（费马小定理）
费马小定理 ：若pp为素数，aa为正整数，且aa、pp互质。 则有a(p-1) ≡1(modp)。
代入式子化简得 x≡ap−2(modp)

```
#include <iostream>
using namespace std;
typedef long long ll;
ll fpm(ll x,ll power, ll p)
{
    x%=p;
    ll ans=1;
    for(;power;power>>=1)
    {
        if(power&1)
        {
            ans*=x;
            ans%=p;
        }
        x*=x;
        x%=p;
    }
    return ans;
}
```

```
int main()
{
    ll n,p;
    cin>>n>>p;
    for(int i=1;i<=n;i++)
    {
        ll x = fpm(i, p-2, p); //x为a在mod p意义下的逆元
        cout<<x<<endl;
    }
    return 0;
 }
```

线性算法
适用于求一连串的逆元

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=4e6+5;
ll inv[maxn];
int main()
{
    ll n,p;
    cin>>n>>p;
    inv[1]=1;
    cout<<inv[1]<<endl;
    for(int i=2;i<=n;i++)
    {
        inv[i]=(p - p / i) * inv[p % i] % p;
        printf("%lld\n",inv[i]);
    }
    return 0;
}
```