ACM模板

by 鱼竿钓鱼干

E-mail:851892190@qq.com

参考：acwing板子（以这个为主+个人做题经验补充），洛谷题解，各类博客，各平台比赛，算法竞赛进阶指南

版本：2021/5/1

更新内容：

1. 组合数各种求法
2. 博弈论基础
3. 线性筛预处理加速
4. 删除不必要的东西与目录分支
5. 新增__int128
6. 新增扩展欧几里得解同余方程

复杂度反推

1. n<=30,指数级别
   dfs+剪枝，状压DP，指数型枚举（二进制）
2. n<=100 O(n^3)
   floyd,dp,高斯消元
3. n<=1000 O($n^{2),O(n}$2logn)
   dp,二分，朴素Dijkstra，朴素Prim，Bellman-Ford

4. n<=10000 O(n*sqrt(n))
   块状链表，分块，莫队
5. n<=1e5 O(nlogn)
   sort，线段树，树状数组，set/map，heap，拓扑排序，dijkstra+heap、prim+heap、spfa、求凸包、求半平面交、二分、CDQ分治、整体二分，map(超过4*1e5就别用了)
6. n<=1e6 O(n) 常数小的O(nlogn)
   输入输出100w的时候必须scanf

   hash，双指针，并查集，kmp，AC自动机
   常数小的O(nlogn)sort，树状数组，heap，dijkstra，spfa
7. n<=1e7 O(sqrt(n))
   判断质数
8. n<=1e18 O(logn)
   gcd,快速幂
9. n<=1e1000 O((logn)^2)
   高精加减乘除
10. n<=1e100000 O(logk*loglogk),k表示位数
    高精度加减，FFT/NTT

数学

质数

试除法

Pt.1

```
bool is_prime(long long x)
{
    if (x < 2) return false;
    for (int i = 2; i <= x / i; i ++ )//i*i<=x可能会溢出
        if (x % i == 0)
            return false;
    return true;
}
```

Pt.2

```
template<typename T>
bool Is_prime(T a){
    if(a <= 1)  return false;
    if(a == 2 || a == 3)    return true;
    if(a % 6 != 1 && a % 6 != 5)    return false;
    for(int i = 5 ; i * i <= n ; i += 6){
        if(n % i == 0 || n % (i + 2) == 0)  return 0;
    }
}
```

埃氏筛

```
int primes[N], cnt;     // primes[]存储所有素数
bool st[N];          // st[x]存储x是否被筛掉
//筛掉每个数的倍数，如果p没有被筛掉，那么说明p不是2~p-1任何一个数倍数即，2~p-1都不是p约数
//优化：只要筛1~n所有质数的倍数就行了，唯一分解定理
void get_primes(int n)
{
    for (int i = 2; i <= n; i ++ )
    {
        if (st[i]) continue;
        primes[cnt ++ ] = i;
        for (int j = i + i; j <= n; j += i)//筛倍数
            st[j] = true;
    }
}
```

线性筛

```
int primes[N], cnt;     // primes[]存储所有素数
bool st[N];             // st[x]存储x是否被筛掉，后面可以直接用来判是否为素数
                        //n只会被最小质因子筛掉
void get_primes(int n)
{
    memset(st,0,sizeof st);
    st[0]=st[1]=1;
    for (int i = 2; i <= n; i ++ )                  //不要忘记等号
    {
        if (!st[i]) primes[++cnt] = i;
        for (int j = 1; primes[j] <= n / i; j ++ )  //不要忘记等号
        {
            st[primes[j] * i] = true;
            //合数一定有最小质因数，用最小质因数的倍数筛去合数
            if (i % primes[j] == 0) break;
            //prime[j]一定是i最小质因子，也一定是prime[j]*i的最小质因子
        }
    }
}
```

线性筛加速质因数分解

```
//Author fishingrod
//CSDN:https://blog.csdn.net/qq_39354847
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define int long long

ll T,Q,n,m,k,p,ans,cnt,sum,tmp,num,last;
map<int,int>prime;
int pri[N],st[N];//st标记同时预处理除i对应最小质因子

inline void get_primes(int n)
{
    memset(st,0,sizeof st);
```

```
    st[0]=st[1]=1;
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i])
        {
             pri[++cnt]=i;
             st[i]=i;
        }
        for (int j = 1; pri[j] <= n / i; j ++ )
        {
            st[pri[j] * i]=pri[j];
            if (i % pri[j] == 0) break;
        }
    }
}
inline void divide(int x)
{
    prime.clear();
    while(x>1)
    {
        prime[st[x]]++;
        x/=st[x];
    }
}

signed main()
{
    n=scanf("%d",&n);
    get_primes(n);
    for(int i=2; i<=n; i++)divide(i);
    return 0;
}
```

线性筛预处理质因子个数

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll T;
const int N=2e7+10;
ll pr[N],m[N];
bool st[N];
ll cnt;
void get_prime()
{
    for(ll i=2;i<=N;i++)
    {
        if(st[i]==0)
        {
            pr[++cnt]=i;
            m[i]=1;
        }
        for(ll j=1;pr[j]<=N/i;j++)
        {
            st[i*pr[j]]=1;
            m[i*pr[j]]=m[i]+1;
```

```
            if(i%pr[j]==0)//i是pr[j]的倍数,所以就不增加了
            {
                m[i*pr[j]]=m[i];
                break;
            }
        }
    }

}
```

质因数

分解质因数

```
void divide(int n)
{
    for(int i = 2 ; i < = n / i ; i ++ )//不要忘记等号
        if(n%i==0)//i一定是质数
        {
            int s=0;
            while(n%i==0)
            {
                n/=i;
                s++;
            }
            printf("%d %d\n",i,s);
        }
    if(n>1)printf("%d %d\n",n,1);//处理唯一一个>sqrt(n)的
    puts(" ");
}
/*
给定两个数n，m,其中m是一个素数。
将n（0<=n<=2^31）的阶乘分解质因数，求其中有多少个m。
while(n/m) ans+=n/m,n/=m;
*/
```

试除法求约数

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

vector<int>get_divisors(int n)
{
    vector<int>res;
    for(int i=1;i<=n/i;i++)//从1开始，约数啊
        if(n%i==0)
        {
            res.push_back(i);
            if(i!=n/i)res.push_back(n/i);//约数通常成对出现，特判完全平方
        }
    sort(res.begin(),res.end());
    return res;
}

int main()
```

```
{
    int n;
    cin>>n;
    while(n--)
    {
        int x;
        cin>>x;
        auto res=get_divisors(x);
        for(auto t:res)cout<<t<<' ';
        cout<<endl;
    }
}
```

[约数个数(多个数相乘的)]

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

typedef long long LL;

const int mod=1e9+7;

int main()
{
    int n;
    cin>>n;
    unordered_map<int,int>primes;
    while(n--)
    {
        int x;
        cin>>x;
        for(int i=2;i<=x/i;i++)
            while(x%i==0)
            {
                x/=i;
                primes[i]++;
            }
        if(x>1)primes[x]++;
    }
    LL res=1;
    for(auto prime:primes)res=res*(prime.second+1)%mod;
    cout<<res<<endl;
    return 0;
}
```

约数个数和

```cpp
#include<bits/stdc++.h>
using namespace std;


int main()
{
    int res=0,n;
    cin>>n;
    for(int i=1;i<=n;i++)res+=n/i;
    cout<<res;
    return 0;
}
```

约数和

```cpp
#include<bits/stdc++.h>
using namespace std;

const int mod=1e9+7;
typedef long long LL;
int main()
{
    int n,x;
    unordered_map<int,int>primes;
    cin>>n;
    while(n--)
    {
        cin>>x;
        for(int i=2;i<=x/i;i++)
            while(x%i==0)
            {
                x/=i;
                primes[i]++;
            }
        if(x>1)primes[x]++;
    }
    LL res=1;
    for(auto prime:primes)
    {
        int p=prime.first,a=prime.second;
        LL t=1;
        while(a--)t=(t*p+1)%mod;
        res=res*t%mod;
    }
    cout<<res<<endl;
    return 0;
}
```

欧拉函数

欧拉函数

```
ll phi(ll x)
{
    ll res = x;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);

    return res;
}
```

筛法求欧拉函数（1~n,欧拉函数之和）

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

typedef long long ll;
const int N=1e6+10;

ll n,primes[N],phi[N],cnt;
bool st[N];

ll get_eulers(ll n)
{
    phi[1]=1;
    for(int i=2;i<=n/i;i++)
    {
        if(!st[i])
        {
            prime[cnt++]=i;
            phi[i]=i-1;
        }
        for(int j=0;prime[j]<=n/i;j++)
        {
            st[i*primes[j]]=1;
            if(i%primes[j]==0)
            {
                phi[primes[j]*i]=phi[i]*primes[j];
                break;
            }
            phi(primes[j]*i)=phi[i]*(primes[j]-1)
        }
    }
    ll res=0;
    for(int i=1;i<=n;i++)res+phi[i];
    return res;
}

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

```
    cout<<get_eulers(n)<<endl;

    return 0;
}
```

费马小定理

```
template<typename T>
const T MOD = 1e9 + 7;
T Q_Power(T a , T b ){
    T res = 1;
    while(b){
        if(b & 1) res = (res * a) % MOD;
        b >>= 1;
        a = (a * a) %  MOD;
    }
}

T X_1(T a){ return Q_Power(a,MOD - 2) ; }  //逆元
```

> 费马小定理主要用于求解在取模为素数的情况下，对于式子 $a \times x \equiv 1 \mod MOD$ 求解 a 的逆元 x ，与上述 扩展欧几里得算法拥有相当的时间复杂度，但却更好写些。

整除分块

> 以下为例题的解法
> 求:$\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$
> (待更新)

```
for(int l = 1 , r ; l <= n ; l = r + 1){
    r = n / (n / l);
    ans += (r - l + 1) * (n / l);
}
```

组合数

递推

```
for (int i = 0; i < N; i ++ )
    for (int j = 0; j <= i; j ++ )
        if (!j) c[i][j] = 1;
        else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
```

预处理逆元求组合数

```
首先预处理出所有阶乘取模的余数fact[N]，以及所有阶乘取模的逆元infact[N]
如果取模的数是质数，可以用费马小定理求逆元
int qmi(int a, int k, int p)    // 快速幂模板
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
```

```
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}

// 预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i ++ )
{
    fact[i] = (LL)fact[i - 1] * i % mod;
    infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
}
```

Lucas定理

若p是质数，则对于任意整数 1 <= m <= n，有：
    C(n, m) = C(n % p, m % p) * C(n / p, m / p) (mod p)

```
int qmi(int a, int k, int p)  // 快速幂模板
{
    int res = 1 % p;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}

int C(int a, int b, int p)  // 通过定理求组合数C(a, b)
{
    if (a < b) return 0;

    LL x = 1, y = 1;  // x是分子，y是分母
    for (int i = a, j = 1; j <= b; i --, j ++ )
    {
        x = (LL)x * i % p;
        y = (LL) y * j % p;
    }

    return x * (LL)qmi(y, p - 2, p) % p;
}

int lucas(LL a, LL b, int p)
{
    if (a < p && b < p) return C(a, b, p);
    return (LL)C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}
```

分解质因数法求组合数

当我们需要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：
    1. 筛法求出范围内的所有质数

```
    2. 通过 C(a, b) = a! / b! / (a - b)! 这个公式求出每个质因子的次数。 n! 中p的次数是 n / p + n / p^2 + n / p^3 + ...
    3. 用高精度乘法将所有质因子相乘

int primes[N], cnt;     // 存储所有质数
int sum[N];     // 存储每个质数的次数
bool st[N];     // 存储每个数是否已被筛掉


void get_primes(int n)      // 线性筛法求素数
{
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}


int get(int n, int p)       // 求n! 中的次数
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}


vector<int> mul(vector<int> a, int b)       // 高精度乘低精度模板
{
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i ++ )
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }

    while (t)
    {
        c.push_back(t % 10);
        t /= 10;
    }

    return c;
}

get_primes(a);  // 预处理范围内的所有质数

for (int i = 0; i < cnt; i ++ )     // 求每个质因数的次数
```

```
{
    int p = primes[i];
    sum[i] = get(a, p) - get(b, p) - get(a - b, p);
}

vector<int> res;
res.push_back(1);

for (int i = 0; i < cnt; i ++ )     // 用高精度乘法将所有质因子相乘
    for (int j = 0; j < sum[i]; j ++ )
        res = mul(res, primes[i]);
```

阶乘

阶乘位数

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    double n;
    while(cin>>n)
    {
        double s=0;
        for(int i=1;i<=n;i++)s+=log10(i);//n!取对数
        cout<<int(s)+1<<endl;
    }
    return 0;
}
```

斐波那契数列

矩阵快速幂

```
ll Fib_Q_M_Power(ll n){
    if(n <= 1) return n;
    if(n == 2) return 1;
    ll ans[2][2] = {1,0,0,1}, tmp[2][2] , a[2][2] = {1,1,1,0};
    while(n){
        if(n & 1){
            memset(tmp,0,sizeof(tmp));
            for(int i = 0 ; i < 2 ; i ++ ){
                for(int j = 0 ; j < 2 ; j ++ ){
                    for(int k = 0 ; k < 2 ; k ++ ){
                        tmp[i][j] = (tmp[i][j] + ans[i][k] * a[k][j] % MOD) %
MOD;
                    }
                }
            }

            for(int i = 0 ; i < 2 ; i ++ ){
                for(int j = 0 ; j < 2 ; j ++ ) ans[i][j] = tmp[i][j];
            }

        }
```

```
        n >>= 1;
        memset(tmp,0,sizeof(tmp));
        for(int i = 0 ; i < 2 ; i ++ ){
            for(int j = 0 ; j < 2 ; j ++ ){
                for(int k = 0 ; k < 2 ; k ++ ){
                    tmp[i][j] = (tmp[i][j] + a[i][k] * a[k][j] % MOD) % MOD;
                }
            }
        }

        for(int i = 0 ; i < 2 ; i ++ ){
            for(int j = 0 ; j < 2 ; j ++ ){
                a[i][j] = tmp[i][j];
            }
        }

    }
    return ans[1][0];
}
```

扩展欧几里得

扩欧求解同余方程ax+by=1（要求解最小正整数）

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

typedef long long LL;

int exgcd(int a,int b ,int &x,int &y)
{
    if(!b)
    {
        x=1,y=0;
        return a;
    }

    int d=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
}
int get_mod(int a,int b)
{
    return (a%b+b)%b;
}
int main()
{
    int a,b;
    cin>>a>>b;
    int x,y;
    exgcd(a,b,x,y);
    cout<<get_mod(x,b)<<endl;
    return 0;
}
```

求解线性同余方程ax+my=b（保证解在int范围内）

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

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

int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int a,b,m,x,y;
        cin>>a>>b>>m;
        //即求解ax+my=b
        int d=exgcd(a,m,x,y);
        if(b%d)puts("impossible");//gcd(a,b)|b看有没有解
        else cout<<(LL)x*b/d%m<<endl;
    }
    return 0;
}
```

小结

$x1$，$y1$为$ax+by=c$的特解
那么他的**最小正整数解**为
$d=gcd(a,b),p=abs(\frac{b}{d})$防止出现负数
$x=((x+p)\%p)\%p,$如果$x=0$那么$x=p$
输出int范围内的解
$x=x1\%b$

博弈论

SG函数（以NIM集合博弈为例题，每次可以拿走集合里的数）

```
#include<bits/stdc++.h>
using namespace std;
const int N=110,M=10010;
int n,m;
int s[N],f[M];
int sg(int x){//这是一棵搜索树
    if(f[x]!=-1)return f[x];//记忆化搜索 保证时间复杂度不是指数级别 每个状态只算一次 记忆化搜索是对搜索的一种优化，也是动态规划的一种实现方式
```

```
    unordered_set<int>S;//所有可以到的局面，函数内部声明，后面递归出来的和本次的S是不一样的
    for(int i=0;i<m;++i){
        int sum=s[i]; //当前的数的个数是s[i]
        if(x>=sum) S.insert(sg(x-sum)); //当前的数的个数大于sum 才能把它加进来 从x中取走s[i]个石子，要递归下去啊!
    }
    for(int i=0;;++i){//不属于当前集合的最小自然数 MEX
        if(!S.count(i)) return f[x]=i;
    }
}

int main(){
    cin>>m;
    for(int i=0;i<m;++i)cin>>s[i];
    cin>>n;
    memset(f,-1,sizeof(f));//初始化
    int res=0;
    for(int i=0;i<n;++i){
        int x;
        cin>>x;//对每一堆求sg然后异或起来
        res^=sg(x);
    }
    if(res)puts("Yes");
    else puts("No");
}
```

常见博弈结论

1. **尼姆游戏 Nim Game**

   给定 N 堆物品，第i堆物品有Ai个。两名玩家轮流行动，每次可以任选一堆，取走任意多个物品，可把一堆取光，但不能不取。**取走最后一件物品者获胜。**两人都采取最优策略，问先手是否必胜。

   结论:异或和为0，先手必败，反之先手必胜

2. **巴什博弈 Bash Game**

   有 1 堆石子，总个数是 n ，两名玩家轮流在石子堆中拿石子，每次至少取 1 个，至多取 m 个。**取走最后一个石子的玩家为胜者。**判定先手和后手谁胜。

   结论：若 (m + 1) | n （整除）则先手必败，否则先手必胜

3. **威佐夫博弈 Wythoff Game**

   有两堆石子，石子数可以不同。两人轮流取石子，每次可以在一堆中取，或者从两堆中取走相同个数的石子，数量不限，**取走最后一个石头的人获胜。**判定先手是否必胜。

   结论：假设两堆石子为(a,b)其中（a<b)，当且仅当$(b-a)*\frac{\sqrt{5}+1}{2}=a$,反之必胜

4. **斐波那契博弈 Fibonacci Game**

   有一堆个数为 n (n ≥ 2) 的石子，游戏双方轮流取石子，规则如下：

   先手不能在第一次把所有的石子取完，至少取 1 颗；

   之后每次可以取的石子数至少为 1 ，至多为对手刚取的石子数的 2 倍。

   约定**取走最后一个石子的人为赢家**，求必败态。

   结论:先手必败，当且仅当石子数为斐波那契数

   （有一个定理，任何正整数可以表示为若干个不连续斐波那契数之和）

进制与位运算

进制转换

10进制转K进制

```
string itoA(LL n,int radix)    //n是待转数字，radix是指定的进制
{
    string ans;
    do{
        int t=n%radix;
        if(t>=0&&t<=9)  ans+=t+'0';
        else ans+=t-10+'A';//注意大小写
        n/=radix;
    }while(n!=0);   //使用do{}while（）以防止输入为0的情况
    reverse(ans.begin(),ans.end());//逆序翻转
    return ans;
}
```

K进制转10进制

```
LL Atoi(string s,int radix)    //s是给定的radix进制字符串
{
    LL ans=0;
    for(int i=0;i<s.size();i++)
    {
        char t=s[i];
        if(t>='0'&&t<='9') ans=ans*radix+t-'0';
        else ans=ans*radix+t-'A'+10;
    }
    return ans;
}
```

二进制枚举

枚举子集 O(2^|S|),|S|表示集合中元素个数

```
// S 是一个二进制数（二进制每位数字对应一种状态），表示一个集合，i 枚举 S 的所有子集
//s=9=1001,有子集9=1001,8=1000,1=0001
for (int i = S; i; i = S & i - 1)
//[0,2^n-1]，n个元素的子集有2^n个
for(int i = 0;i<(1<<n);i++)
```

枚举方案

```
void binary_enum(int n)
{
    for(int i=0;i<(1<<n);i++)
        for(int j=0;j<n;j++)
        {
            if(i&(1<<j))
            {
                array[j]//别写成array[i]
            }
        }
}
```

枚举子集的子集O(3^n)

```
for (int S = 0; S < 1 << n; S ++ ) // 枚举集合 {0, ..., n - 1} 的所有子集
    for (int i = S; i; i = S & i - 1) // 枚举子集 S 的子集
        // blablabla
```

位运算

```
a & b按位与
a | b按位或
a ^ b按位异或//可以用来排除出现偶数次的数
~ a按位取反，用于表示负数 - x = ~ x +  1
a << b = a * 2^b
a >> b = a / 2 ^ b（去小数）
!a非

将 x 第 i 位取反：x ^= 1 << i
将 x 第 i 位制成 1：x |= 1 << i
将 x 第 i 位制成 0：x &= -1 ^ 1 << i 或 x &= ~(1 << i)
取 x 对 2 取模的结果：x & 1
取 x 的第 i 位是否为 1：x & 1 << i 或 x >> i & 1
取 x 的最后一位：x & -x
取 x 的绝对值：(x ^ x >> 31) - (x >> 31) （int 型）
判断 x 是否不为 2 的整次方幂：x & x - 1
判断 a 是否不等于 b：a != b,a - b,a ^ b
判断 x 是否不等于-1：x != -1,x ^ -1,x + 1,~x

异或，加法恒等式
a+b=a|b+2(a&b)  若a+b=a|b-->a&b=0
若a-b=a|b-->a-b每一位不能发生借位,即若a已知,对于a的每一位,若为1，则可以01若为0，则指南0
```

求二进制中1的个数

暴力O(logN)

```
int count(int x)
{
    int res = 0;
    while (x) res += x & 1, x >>= 1;
    return res;
}
```

位运算O(1)

```
int count(int x)
{
    x = (x >> 1 & 0x55555555) + (x & 0x55555555);
    x = (x >> 2 & 0x33333333) + (x & 0x33333333);
    x = (x >> 4 & 0x0f0f0f0f) + (x & 0x0f0f0f0f);
    return x % 255;
}
```

log2(x)取整

求log_2(x)（暴力）O（logx）

```
int log_2(int x)
{
    int res = 0;
    while (x >> 1) res ++ , x >>= 1;
    return res;
}
```

求log_2(x)（二分） O(loglogx)

```
int log_2(int x)
{
    int l = 0, r = 31, mid;
    while (l < r)
    {
        mid = l + r + 1 >> 1;
        if (x >> mid) l = mid;
        else    r = mid - 1;
    }
}
```

预处理log_2(x) 1~n中所有数 O（n）

```
int log_2[N];   // 存 log2(i) 的取整结果
void init(int n)
{
    for (int i = 0; 1 << i <= n; i ++ )
        log_2[1 << i] = i;
    for (int i = 1; i <= n; i ++ )
        if (!log_2[i])
            log_2[i] = log_2[i - 1];
}
```

龟速加(a*b%p，主要防爆)

```
//把b转为2进制，b=11=（1011）2=2^3+2^1+2^0
//a*b%p=8a%p+2a%p+a%p
//开long long
int add(int a, int b)
{
    while (b)
    {
        int x = a ^ b;
        b = (a & b) << 1;
        a = x;
    }
    return a;
}
```

快速幂

```
int quick(int a,int b)
{
    int res=1;
    a=a%mod;
    while(b)
    {
        if(b&1) res=(res*a)%mod;
        a=(a*a)%mod;
        b>>=1;
    }
    return res;
}
```

高精

A+B

```
#include<bits/stdc++.h>
using namespace std;
//如果k进制，那么10都改成k就行了，传进去的时候注意改A~10,F~15;
vector<int>add(vector<int> &A,vector<int> &B)
{
    if(A.size()<B.size())return add(B,A);
    vector<int>c;
    int t=0;//一定要初始化为0
    for(int i=0;i<A.size();i++)//A+B+t
    {
        t+=A[i];
        if(i<B.size())t+=B[i];
        c.push_back(t%10);
        t/=10;
    }
    if(t)c.push_back(t);//处理最高位
    return c;
}
int main()
{
    string a,b;
    vector<int>A,B;
```

```
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');//逆序输入,方便进位
    for(int i=b.size()-1;i>=0;i--)B.push_back(b[i]-'0');
    auto c=add(A,B);
    for(int i=c.size()-1;i>=0;i--)cout<<c[i];//逆序输出
    return 0;
}
```

A-B

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


void trimzero(vector<int> &A)//处理输入前的0和输出时的0
{
    while(A.size()>1&&A.back()==0)A.pop_back();
}
bool cmp(vector<int> &A,vector<int> &B)
{
    if(A.size()!=B.size())return A.size()>B.size();
    for(int i=A.size()-1;i>=0;i--)
        if(A[i]!=B[i])return A[i]>B[i];
    return 1;
}


vector<int>sub(vector<int> &A,vector<int> &B)
{
    vector<int>c;
    for(int i=0,t=0;i<A.size();i++)
    {
        t=A[i]-t;
        if(i<B.size())t-=B[i];
        c.push_back((t+10)%10);
        if(t<0)t=1;
        else t=0;
    }
    trimzero(c);
    return c;
}


int main()
{
    string a,b;
    cin>>a>>b;
    vector<int>A,B;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    for(int i=b.size()-1;i>=0;i--)B.push_back(b[i]-'0');
    trimzero(A);
    trimzero(B);
    if(cmp(A,B))
    {
        auto c=sub(A,B);
        for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    }
```

```
    else
    {
        auto c=sub(B,A);
        cout<<"-";
        for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    }
    return 0;
}
```

A*b

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


void trimzero(vector<int> &A)
{
    while(A.size()>1&&A.back()==0)A.pop_back();
}
vector<int> mul(vector<int> &A,int b)
{
    vector<int>c;
    for(int i=0,t=0;i<A.size()||t;i++)
    {
        if(i<A.size())t+=A[i]*b;
        c.push_back(t%10);
        t/=10;
    }
    trimzero(c);
    return c;
}
int main()
{
    string a;
    int b;
    cin>>a>>b;
    vector<int>A;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    trimzero(A);
    auto c=mul(A,b);
    for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    return 0;
}
```

A/b

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


void trimzero(vector<int> &A)
{
    while(A.size()>0&&A.back()==0)A.pop_back();
}
vector<int>div(vector<int> &A,int b,int &r)
{
```

```
    vector<int>c;
    r=0;
    for(int i=A.size()-1;i>=0;i--)//出发比较特别从高位开始搞
    {
        r=r*10+A[i];
        c.push_back(r/b);
        r%=b;
    }

    reverse(c.begin(),c.end());
    trimzero(c);
    return c;
}
int main()
{
    string a;
    int b;
    vector<int>A;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    int r;
    auto c=div(A,b,r);
    for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    cout<<endl<<r<<endl;
    return 0;
}
```

大数阶乘（vector太慢了，用数组）

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

int main()
{
    int n,ws;
    while(scanf("%d",&n)!=EOF)
    {
        double s=0;
        for(int i=1;i<=n;i++)s+=log10(i);
        ws=int(s)+1;//求位数
        int f[ws];
        memset(f,0,sizeof(f));
        int ans,jw,j;
        f[0]=1;
        for(int i=2;i<=n;i++)
        {
            int jw=0;
            for(j=0;j<ws;j++)
            {
                int ans=f[j]*i+jw;
                f[j]=ans%10;
                jw=ans/10;
            }
        }
        for(j=ws-1;j>=0;j--)printf("%d",f[j]);
        printf("\n");
    }
```

```
    return 0;
}
```

其他处理

去前导0

```
void trimzero(vector<int> &A)//处理输入前的0和输出时的0
{
    while(A.size()>1&&A.back()==0)A.pop_back();
}
```

多组输入初始化

```
多次输入或者累计运算
记得清空vector
vector<int>a;
a.clear();
```

排序

归并

排序

```
int q[N],tmp[N];
void merge_sort(int q[],int l,int r)//这里只有<=/>=没有</>
{
    if(l>=r)return;
    int mid=l+r>>1;
    merge_sort(q,l,mid),merge_sort(q,mid+1,r);
    int k=0,i=l,j=mid+1;
    while(i<=mid&&j<=r)
    {
        if(q[i]<=q[j])tmp[k++]=q[i++];
        else tmp[k++]=q[j++];
    }
    while(i<=mid)tmp[k++]=q[i++];
    while(j<=r)tmp[k++]=q[j++];


    for(int i=l,j=0;i<=r;i++,j++)q[i]=tmp[j];//不要写成i=1
}
```

求逆序对

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


typedef long long ll;
const int N=1e6+10;
```

```
int n;
int q[N],tmp[N];

//i不会等于1只有l
ll merge_sort(int l,int r)
{
    if(l>=r)return 0;
    int mid=l+r>>1;
    ll res=merge_sort(l,mid)+merge_sort(mid+1,r);
    int k=0,i=l,j=mid+1;//i是l别打成1
    while(i<=mid&&j<=r)
    {
        if(q[i]<=q[j])tmp[k++]=q[i++];
        else
        {
            tmp[k++]=q[j++];
            res+=mid-i+1;//q[i]>q[j]，左区间剩下的所有数与右区间当前数成为逆序对
        }
    }
    while(i<=mid)tmp[k++]=q[i++];      //扫尾
    while(j<=r)tmp[k++]=q[j++];
    for(int i=l,j=0;i<=r;i++,j++)q[i]=tmp[j];//不要写成i=1
    return res;
}


int main()
{
    int n;
    cin>>n;
    for(int i=0;i<n;i++)cin>>q[i];
    cout<<merge_sort(0,n-1);
}
```

逆序对应用

1. 交换重排，根据奇偶性判局面可达（可能是字符串，二维平面，数组序列）

输入输出

快读

```
inline int read()
{
    int x=0,y=1;char c=getchar();//y代表正负（1.-1），最后乘上x就可以了。
    while (c<'0'||c>'9') {if (c=='-') y=-1;c=getchar();}//如果c是负号就把y赋为-1
    while (c>='0'&&c<='9') x=x*10+c-'0',c=getchar();
    return x*y;//乘起来输出
}
```

c++关闭同步

```cpp
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
}
```

__int128

```cpp
__int128 read()
{
    __int128 f=1,w=0;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')
        f=-1;
        ch=getchar();
    }
    while(ch<='9'&&ch>='0')
    {
        w=w*10+ch-'0';
        ch=getchar();
    }
    return f*w;
}

void print(__int128 x)
{
    if(x<0)
    {
        putchar('-');
        x=-x;
    }
    if(x>9)print(x/10);
    putchar(x%10+'0');
}
```

数据结构

STL

```
vector, 变长数组，倍增的思想
    size()  返回元素个数
    empty()  返回是否为空
    clear()  清空
    front()/back()
    push_back()/pop_back()
    begin()/end()
    []
    支持比较运算，按字典序


pair<int, int>
    first, 第一个元素
```

```
    second，第二个元素
    支持比较运算，以first为第一关键字，以second为第二关键字（字典序）


string，字符串
    size()/length()  返回字符串长度
    empty()
    clear()
    substr(起始下标，(子串长度))  返回子串
    c_str()  返回字符串所在字符数组的起始地址


queue，队列
    size()
    empty()
    push()  向队尾插入一个元素
    front()  返回队头元素
    back()  返回队尾元素
    pop()  弹出队头元素


priority_queue，优先队列，默认是大根堆
    size()
    empty()
    push()  插入一个元素
    top()  返回堆顶元素
    pop()  弹出堆顶元素
    定义成小根堆的方式：priority_queue<int, vector<int>, greater<int>> q;


stack，栈
    size()
    empty()
    push()  向栈顶插入一个元素
    top()  返回栈顶元素
    pop()  弹出栈顶元素


deque，双端队列
    size()
    empty()
    clear()
    front()/back()
    push_back()/pop_back()
    push_front()/pop_front()
    begin()/end()
    []


set，map，multiset，multimap，基于平衡二叉树（红黑树），动态维护有序序列
    size()
    empty()
    clear()
    begin()/end()
    ++，-- 返回前驱和后继，时间复杂度 O(logn)


    set/multiset
```

```
        insert()  插入一个数
        find()  查找一个数
        count()  返回某一个数的个数(由于set不重复原则所以只返回01)
        erase()
            (1) 输入是一个数x，删除所有x   O(k + logn)
            (2) 输入一个迭代器，删除这个迭代器
        lower_bound()/upper_bound()
            lower_bound(x)  返回大于等于x的最小的数的迭代器
            upper_bound(x)  返回大于x的最小的数的迭代器
    map/multimap
        insert()  插入的数是一个pair
        erase()  输入的参数是pair或者迭代器
        find()
        []  注意multimap不支持此操作。 时间复杂度是 O(logn)
        lower_bound()/upper_bound()


unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
    和上面类似，增删改查的时间复杂度是 O(1)
    不支持 lower_bound()/upper_bound()， 迭代器的++，--


bitset, 压位
    bitset<10000> s;
    ~, &, |, ^
    >>, <<
    ==, !=
    []


    count()  返回有多少个1


    any()  判断是否至少有一个1
    none()  判断是否全为0


    set()  把所有位置成1
    set(k, v)  将第k位变成v
    reset()  把所有位变成0
    flip()  等价于~
    flip(k) 把第k位取反
```

栈

表达式

后缀表达式

```cpp
#include<bits/stdc++.h>
using namespace std;

const int N=1e5+10;
stack<int> num;
stack<char> op;
```

```
void eval()
{
    auto b=num.top();num.pop();
    auto a=num.top();num.pop();
    auto c=op.top();op.pop();
    int x;
    if(c=='+')x=a+b;
    else if(c=='-')x=a-b;
    else if(c=='*')x=a*b;
    else x=a/b;
    num.push(x);
}
int main()
{
    unordered_map<char,int>pr{{'+',1},{'-',1},{'*',2},{'/',2}};//运算符优先级
    string s;
    cin>>s;
    for(int i=0;i<s.size();i++)
    {
        auto c=s[i];
        if(isdigit(c))//转数字
        {
            int x=0,j=i;
            while(j<s.size()&&isdigit(s[j]))
                x=x*10+s[j++]-'0';
            i=j-1;
            num.push(x);
        }
        else if(c=='(')op.push(c);
        else if(c==')')//括号直接算
        {
            while(op.top()!='(')eval();
            op.pop();
        }
        else
        {
            while(op.size()&&pr[op.top()]>=pr[c])eval();//如果当前优先级比前面的低就
算
            op.push(c);//操作符入栈
        }
    }
    while(op.size())eval();//剩余计算
    printf("%d",num.top());
    return 0;
}
```

单调栈

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

const int N=1e6+5;

int n;
int stk[N],tt;
```

```
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
    {
        int x;
        cin>>x;
        while(tt&&stk[tt]>=x)tt--;//1~i-1单调递增的栈,出栈了就不会再回来了
        if(tt)cout<<stk[tt]<<" ";
        else cout<<-1<<" ";
        stk[++tt]=x;
    }
}
```

队列

单调队列(滑动窗口)

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


const int N=1e6+5;


int a[N],q[N],n,k;//q存下标


int main()
{
    scanf("%d%d",&n,&k);
    for(int i=0;i<n;i++)scanf("%d",&a[i]);
    int hh=0,tt=-1;
    for(int i=0;i<n;i++)
    {
        //判断队头是已经滑出窗口
        if(hh<=tt&&i-k+1>q[hh])hh++;
        while(hh<=tt&&a[q[tt]]>=a[i])tt--;
        q[++tt]=i;
        if(i>=k-1)printf("%d ",a[q[hh]]);
    }
    puts("");
    hh=0,tt=-1;
    memset(q,0,sizeof(q));
    for(int i=0;i<n;i++)
    {
        if(hh<=tt&&i-k+1>q[hh])hh++;
        while(hh<=tt&&a[q[tt]]<=a[i])tt--;
        q[++tt]=i;
        if(i>=k-1)printf("%d ",a[q[hh]]);
    }
    return 0;
}
```

Trie树

存储查找字符串集合

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


const int N=1e6+10;


int son[N][26],cnt[N],idx;//下标是0的点既是根节点又是空节点，cnt是对应每个停止符的数量。
char str[N];


void insert(char *str)
{
    int p=0;
    for(int i=0;str[i];i++)
    {
        int u=str[i]-'a';
        if(!son[p][u])son[p][u]=++idx;//p是根u是儿子，如果没有儿子，idx只是查询有没有
        p=son[p][u];
    }
    cnt[p]++;
}


int query(char *str)
{
    int p=0;
    for(int i=0;str[i];i++)
    {
        int u=str[i]-'a';
        if(!son[p][u])return 0;
        p=son[p][u];
    }
    return cnt[p];
}
int main()
{
    int n;
    char op[2];
    scanf("%d",&n);
    while(n--)
    {
        scanf("%s%s",op,str);
        if(op[0]=='I')insert(str);
        else printf("%d\n",query(str));
    }
    return 0;
}
```

并查集

普通并查集

```
int find(int x){ return f[x] == x ? x : find(f[x]) ; }

void merge_set(int x,int y){
    int fx = find(x) , fy = find(y);
    if(fx!=fy) siz[fy]+=siz[fx] , f[fx]=fy;  // 联通的大小。
}

void init(){ for(int i = 1 ; i <= n ; i ++ ) f[i] = i , siz[i] = 1 ; }
```

维护距离(向量本质，有向图)

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


const int N=1e6+10;
int n,k,cnt;
int f[N],d[N];
int find(int x)
{
    if(f[x]!=x)
    {
        int t=find(f[x]);
        d[x]+=d[f[x]];
        f[x]=t;
    }
    return f[x];
}
int main()
{
    cin>>n>>k;
    for(int i=1;i<=n;i++)f[i]=i;
    while(k--)
    {
        int op,x,y;
        cin>>op>>x>>y;
        if(x>n||y>n)cnt++;
        else
        {
            int fx=find(x),fy=find(y);
            if(op==1)//同类
            {
                if(fx==fy&&(d[x]-d[y])%3)cnt++;
                else if(fx!=fy)
                {
                    f[fx]=fy;
                    d[fx]=d[y]-d[x];
                }
            }
            else
            {
                if(fx==fy&&(d[x]-d[y]-1)%3)cnt++;
                else if(fx!=fy)
```

```
                {
                    f[fx]=fy;
                    d[fx]=d[y]+1-d[x];
                }
            }
        }
    }
    cout<<cnt;
    return 0;
}
```

维护大小

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;


int n,m;
int f[N],siz[N];
int find(int x)
{
    if(f[x]!=x)f[x]=find(f[x]);
    return f[x];
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)//看清题目可能从0开始
    {
        f[i]=i;
        siz[i]=1;
    }
    while(m--)
    {
        int a,b;
        char op[2];
        scanf("%s",op);
        if(op[0]=='C')
        {
            scanf("%d%d",&a,&b);
            if(find(a)==find(b))continue;//判断一下有没有在同一集合里了
            siz[find(b)]+=siz[find(a)];//要在合并之前
            f[find(a)]=find(b);
        }
        else if(op[1]=='1')
        {
            scanf("%d%d",&a,&b);
            if(find(a)==find(b))puts("Yes");
            else puts("No");
        }
        else
        {
            scanf("%d",&a);
            printf("%d\n",siz[find(a)]);
        }
    }
```

```
    return 0;
}
```

扩展域

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


const int N=1e6+10;
int f[N],enem[N];//enem存p的敌人
int find(int x)
{
    if(f[x]!=x)f[x]=find(f[x]);
    return f[x];
}
void merge_set(int x,int y)
{
    int fx=find(x),fy=find(y);
    if(fx==fy)return;//不要忘记
    else f[fx]=fy;
}
int main()
{
    int n,m,cnt;
    char op[2];
    scanf("%d%d",&n,&m);
    cnt=0;
    for(int i=1;i<=2*n;i++)f[i]=i;
    for(int i=1;i<=m;i++)
    {
        int p,q;
        scanf("%s%d%d",op,&p,&q);
        int fp=find(p),fq=find(q);
        if(op[0]=='F')merge_set(p,q);
        else
        {
            if(!enem[p])enem[p]=q;
            else merge_set(q,enem[p]);
            if(!enem[q])enem[q]=p;
            else merge_set(p,enem[q]);
        }
    }
    for(int i=1;i<=n;i++)
    {
        if(f[i]==i)cnt++;
    }
    printf("%d",cnt);
    return 0;
}
```

哈希表

字符串hash

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

typedef unsigned long long ULL;
const int P=131,N=1e5+10;

char str[N];
ULL p[N],has[N];
int n,m;

ULL get_hash(int l,int r)
{
    if(l>r||l<1||r>n)return 0;
    return has[r]-has[l-1]*p[r-l+1];
}
int main()
{
    scanf("%d%d",&n,&m);
    scanf("%s",str+1);
    p[0]=1;
    for(int i=1;i<=n;i++)
    {
        has[i]=has[i-1]*P+str[i];
        p[i]=p[i-1]*P;
    }
    int l1,r1,l2,r2;

    while(m--)
    {
        scanf("%d%d%d%d",&l1,&r1,&l2,&r2);
        if(get_hash(l1,r1)==get_hash(l2,r2))puts("Yes");
        else puts("No");
    }
    return 0;
}
```

ST（区间查询）

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
int a[N];
int f[N][50];
int n,m;

void ST_prework()
{
    for(int i=1;i<=n;i++)f[i][0]=a[i];//递推边界
    int t=log(n)/log(2)+1;
    for(int j=1;j<t;j++)//先区间长度
        for(int i=1;i<=n-(1<<j)+1;i++)//然后遍历起点
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
```

```
}
int ST_query(int l,int r)
{
    int k=log(r-l+1)/log(2);
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)scanf("%d",&a[i]);
    ST_prework();
    scanf("%d",&m);
    while(m--)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        printf("%d\n",ST_query(l,r));
    }

    return 0;
}
```

查找

二分

二分的应用

1. 数据范围大，数据量小，把数据存到普通数组，排序，二分查找下标，可以得到区间数据数量
2. 最大求最小，最小求最大
3. 区间gcd具有单调性

整数二分

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

const int N=1e5+5;
int a[N],x;

bool check_1(int mid)//判定条件if中的式子要考虑会不会溢出或者因为整型除法失效
{
    if(a[mid]>=x)return 1;//不要写成a[x]=x
    else return 0;
}


bool check_2(int mid)
{
    if(a[mid]<=x)return 1;
    else return 0;
}
int bsearch_1(int l,int r)//第一个满足条件的值，即右半段
{
```

```
    while(l<r)
    {
        int mid=l+r>>1;
        if(check_1(mid))r=mid;//方便记忆，右边第一个
        else l=mid+1;//别忘记else
    }
    if(a[l]!=x)return -1;
    else return l;//不要写成return 1;
}


int bsearch_2(int l,int r)//最后一个满足条件的值，即左半段
{
    while(l<r)
    {
        int mid=l+r+1>>1;
        if(check_2(mid))l=mid;//方便记忆，左边最后一个
        else r=mid-1;//别忘记else
    }
    if(a[l]!=x)return -1;
    else return l;
}


int main()
{
    int n,q;
    scanf("%d%d",&n,&q);
    for(int i=0;i<n;i++)scanf("%d",&a[i]);
    while(q--)
    {
        scanf("%d",&x);
        printf("%d %d\n",bsearch_1(0,n-1),bsearch_2(0,n-1));
    }

}
```

浮点数二分

```
bool check(double x) {/* ... */} // 检查x是否满足某种性质


double bsearch_3(double l, double r)//输入l和r的时候保证l<r不要输入一个负数就反过来了不能写(-n,n)
{
    const double eps = 1e-6;   // eps 表示精度，取决于题目对精度的要求，一般比要求的两位有效数字
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

```
#include<bits/stdc++.h>
using namespace std;
const double eps=1e-7;
double y;
double f(double x)
{
    return 0.0001*x*x*x*x*x*x+0.003*x*x*x*x+0.5*x-3;
}
int main()
{
    while(scanf("%lf",&y)!=EOF)
    {
        double mid;
        double l=-20.0,r=20.0;
        while(l<=r)
        {
            mid=(l+r)/2.0;
            if(fabs(f(mid)-y)<1e-5)break;//如果直接数值型的可以这样处理保证精度
            if(f(mid)<y)l=mid;
            else r=mid;
        }
        printf("%.4lf\n",mid);
    }
    return 0;
}
```

整数三分（凸函数）

```
LL check(LL mid)//带入函数
{
    return ;
}
void solve()
{
    cin>>x>>y;
    LL l=0,r=min(x/2,y/3);
    while(l<r)
    {
        LL lmid=l+(r-l)/3;
        LL rmid=r-(r-l)/3;
        if(check(lmid)>check(rmid))r=rmid-1;
        else l=lmid+1;
    }
    cout<<check(l)<<endl;
}
```

搜索与图论

图

图的存储

邻接表

```
// 对于每个点k，开一个单链表，存储k所有可以走到的点。h[k]存储这个单链表的头结点，双向图开双倍2*N!
int h[N], e[N], ne[N], idx;

// 添加一条边a->b
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

// 初始化
idx = 0;
memset(h, -1, sizeof h);
```

邻接矩阵

用一个二维数组表示所建的图。通常查询修改的复杂度为 O(1) 十分方便快捷，但由于其空间复杂度为 $O(n^2)$，二维数组过大的时候会采用邻接表。

一般无向图相关于主对角线对称，主对角线为 0

有向带权图，使用领接矩阵会好很多。

DFS

有多少个连通块(Flood Fill)

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

int n,m,cnt;
char mp[505][505];
int xx[]={0,0,1,-1};
int yy[]={1,-1,0,0};

void dfs(int x,int y)
{
    for(int i=0;i<4;i++)
    {
        int dx=x+xx[i];
        int dy=y+yy[i];
        if(dx>=0&&dx<=n+1&&dy>=0&&dy<=m+1&&mp[dx][dy]!='*')
        {
            mp[dx][dy]='*';//直接标记
            dfs(dx,dy);
        }
    }
}

int main()
```

```
{
        cin>>n>>m;
        for(int i=1;i<=n;i++)
        {
                for(int j=1;j<=m;j++)
                {
                        char c;
                        cin>>c;
                        mp[i][j]=c;
                }
        }
        dfs(0,0);//方式开局就是这*
        for(int i=1;i<=n;i++)
            for(int j=1;j<=m;j++)
                if(mp[i][j]=='0')
                    cnt++;

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

> 求联通块大小的题目若涉及到查询次数较多的题目，用并查集为更优选项。

DFS遍历图(树的重心)

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


const int N=1e6+10;
const int M=N*2;//无向图两条边

int h[N],e[M],ne[M],idx;
bool st[N];
int ans=N;
int n;
void add(int a,int b)//a指向b
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
//以u为根的子树大小（点数量）
int dfs(int u)
{
    st[u]=1;//标记一下
    int sum=1,res=0;//sum
    for(int i=h[u];i!=-1;i=ne[i])//遍历与u连通的点
    {
        int j=e[i];
        if(!st[j])
        {
            int s=dfs(j);//当前子树大小
            res=max(res,s);
            sum+=s;//s是u为根子数大小一部分
        }
    }
    res=max(res,n-sum);//n-sum为，子树上面一坨
    ans=min(ans,res);
```

```
    return sum;//以u为根子节点大小
}
int main()
{
    cin>>n;
    memset(h,-1,sizeof(h));//初始化
    for(int i=1;i<n;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b),add(b,a);//无向图两条边
    }
    dfs(1);//图当中的编号开始搜，随便从哪个点开始都可以
    cout<<ans<<endl;
    return 0;
}
```

列出所有解

全排列

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


int n;
bool vis[30];
int a[20];


void pr()
{
    for(int i=1;i<=n;i++)
    {
        cout<<setw(5)<<a[i];
    }
    cout<<endl;
}
void dfs(int x)//x是层数
{
    if(x>n)
    {
        pr();//超出了n就结束了
    }
    for(int i=1;i<=n;i++)
    {
        if(!vis[i])
        {
            a[x]=i;//第x层是i;
            vis[i]=1;
            dfs(x+1);
            vis[i]=0;//释放回到上一个节点，消去访问记录，其实a[x]也要消去只不过新的值会覆盖
        }

    }
}
```

```
int main()
{
    cin>>n;
    dfs(1);
    return 0;
}
```

STL

```
//prev_permutation函数可以制造前一个排列，如果已经为第一个，则返回false。
#include<bits/stdc++.h>
using namespace std;
int n,a[10000];
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)   //读入数据
        cin>>a[i];
    if(prev_permutation(a,a+n))  //如果为真就输出数组
        for(int i=0;i<n;i++)
            cout<<a[i]<<" ";
    else cout<<"ERROR";   //否则输出ERROR
    cout<<endl;
    return 0;
}
//next_permutation同理
int main()
{
    string str = "abcde";
    int num = 1;
    while(next_permutation(str.begin(),str.end()))
    {
        num++;
        cout<<str<<endl;
        if(num==5)
            break;
    }
    return 0;
}
```

组合输出

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


int n,r;
int a[50];
bool vis[50];


void pr()
{
    for(int i=1;i<=r;i++)
    cout<<setw(3)<<a[i];
```

```
    cout<<endl;
}


void dfs(int x)
{
    if(x>r)
    {
        pr();
        return;
    }
    for(int i=1;i<=n;i++)
    {
        if(!vis[i]&&(i>a[x-1]||x==1))
        {
            a[x]=i;
            vis[i]=1;
            dfs(x+1);
            vis[i]=0;
        }
    }
}
int main()
{
    cin>>n>>r;
    dfs(1);
    return 0;
}
```

BFS

最短步数（边的权值均为1，STL写法）

```
#include<bits/stdc++.h>
using namespace std;
int l,r,c;
int xx[]={1,-1,0,0,0,0};
int yy[]={0,0,1,-1,0,0};
int zz[]={0,0,0,0,1,-1};
int sx,sy,sz,ex,ey,ez;
char mp[40][40][40];
bool vis[40][40][40];
bool flag;
struct node
{
    int x,y,z,s;//s存步数
};

void bfs(int z,int x,int y)
{
    queue<node>q;
    q.push((node){x,y,z,0});//创建结构体队列
    vis[z][x][y]=1;//不要忘记
    while(!q.empty())
    {
        if(q.front().x==ex&&q.front().y==ey&&q.front().z==ez)
        {
```

```
            flag=1;
            printf("Escaped in %d minute(s).",q.front().s);
            break;
        }
        for(int i=0;i<6;i++)
        {
            int dx=q.front().x+xx[i];//是队头的xyz不是x+xx[i]
            int dy=q.front().y+yy[i];
            int dz=q.front().z+zz[i];
            //看清地图范围0~n-1还是1~n，看清是n*n还是n*m，哪个是行哪个是列也要看清楚
            if(dx>=1&&dy>=1&&dz>=1&&dz<=l&&dx<=r&&dy<=c&&!vis[dz][dx]
[dy]&&mp[dz][dx][dy]!='#')
            {
                q.push((node){dx,dy,dz,q.front().s+1});
                vis[dz][dx][dy]=1;
            }
        }
        q.pop();
    }
}
int main()
{
    cin>>l>>r>>c;
    for(int i=1;i<=l;i++)
    {
        for(int j=1;j<=r;j++)
        {
            for(int k=1;k<=c;k++)
            {
                cin>>mp[i][j][k];
                if(mp[i][j][k]=='S')
                {
                    sz=i;sx=j;sy=k;
                }
                if(mp[i][j][k]=='E')
                {
                    ez=i;ex=j;ey=k;
                }
            }
        }
    }
    bfs(sz,sx,sy);
    if(!flag)printf("Trapped!");
    return 0;
}
```

BFS+路径保存

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

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

int n,m;
```

```
int xx[]={0,0,1,-1};
int yy[]={1,-1,0,0};
const int N=105;
node pre[N][N];//保存路径
int g[N][N];//保存图
bool vis[N][N];
void bfs(int sx,int sy)
{
    queue<node>q;
    q.push((node){1,1,0});
    vis[sx][sy]=1;//不要忘记
    pre[sx][sy]=(node){1,1,0};
    while(!q.empty())
    {
        if(q.front().x==n&&q.front().y==m)
        {
            cout<<q.front().s<<endl;
            cout<<n<<" "<<m<<endl;
            while(n!=sx||m!=sy)//输出路径
            {
                cout<<pre[n][m].x<<" "<<pre[n][m].y<<endl;
                n=pre[n][m].x;
                m=pre[n][m].y;
            }
            break;
        }
        for(int i=0;i<4;i++)
        {
            int dx=q.front().x+xx[i];
            int dy=q.front().y+yy[i];
            if(dx>=1&&dy>=1&&dx<=n&&dy<=m&&!g[dx][dy]&&!vis[dx][dy])
            {
                pre[dx][dy]=(node){q.front().x,q.front().y,q.front().s};//保留从哪
里转移过来的就行
                q.push((node){dx,dy,q.front().s+1});
                vis[dx][dy]=1;
            }
        }
        q.pop();
    }
}

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            cin>>g[i][j];
    bfs(1,1);
}
```

BFS遍历图(边权1最短路，手动模拟队列写法)

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

```
const int N=1e6+10;
int h[N],e[N],ne[N],idx;
int n,m;
int d[N],q[N];//d距离，q队列


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


int bfs()
{
    int hh=0,tt=0;
    q[0]=1;//0号节点是编号为1的点，q[0]=v可以做v开始搜的广搜
    memset(d,-1,sizeof(d));
    d[1]=0;//存储每个节点离起点的距离，这个不要忘记了
    while(hh<=tt)
    {
        int t=q[hh++];//t=q[hh]队头同时hh+1弹出队头
        for(int i=h[t];i!=-1;i=ne[i])
        {
            int j=e[i];
                        //如果j没被扩展过
            if(d[j]==-1)
            {
                d[j]=d[t]+1;//d[j]存储j离起点距离，并标记访问过
                q[++tt]=j;//压入j
            }
        }
    }
    return d[n];
}
int main()
{
    cin>>n>>m;
    memset(h,-1,sizeof(h));
    for(int i=1;i<=m;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
    }
    cout<<bfs()<<endl;
    return 0;
}
```

连通块里多少块

```
#include<bits/stdc++.h>
using namespace std;
int xx[]={0,0,1,-1};
int yy[]={1,-1,0,0};
int vis[1005][1005];
bool mp[1005][1005];
int ans[1000005];
```

```
int color,cnt,n,m;


void bfs(int x, int y)
{
    queue<int>h;
    queue<int>l;
    h.push(x);l.push(y);
    vis[x][y]=color;
    while(!h.empty())
    {
        for(int i=0;i<4;i++)
        {
            int dx=h.front()+xx[i];
            int dy=l.front()+yy[i];
            if(dx>=1&&dx<=n&&dy>=1&&dy<=n&&mp[dx][dy]!=mp[h.front()]
[l.front()]&&!vis[dx][dy])
            {
                h.push(dx);l.push(dy);
                vis[dx][dy]=color;//颜色标记区分不同的连通块
            }
        }
        h.pop();l.pop();//弹出多少次就有多少格子
        cnt++;
    }
    return;
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            char ch;
            cin>>ch;
            if(ch=='1')mp[i][j]=1;
            else mp[i][j]=0;
        }
    }
    //如果是确定的起点那下面的直接bfs(sx,sy)即可
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(!vis[i][j])//排除已经搜索过的连通块
            {
                color++;
                bfs(i,j);
                ans[color]=cnt;
                cnt=0;//初始化cnt
            }
        }
    }
    for(int i=1;i<=m;i++)
    {
        int x,y;
        cin>>x>>y;
```

```
        cout<<ans[vis[x][y]]<<endl;
    }
    return 0;
}
```

拓扑

拓扑序列（有向无环图DAG)

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int n,m;
int h[N],e[N],ne[N],idx;
int q[N],d[N];//q队列存储层次遍历序列，d存储i号节点入度

void add(int a,int b)
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
//返回布尔序列是否存在，若存在，则存储在q数组中
bool topsort()
{
    int hh=0,tt=-1;
    //遍历每个节点，入队为0则入队
    for(int i=1;i<=n;i++)
        if(!d[i])
            q[++tt]=i;

    while(hh<=tt)
    {
        //队列不为空则取出头节点
        int t=q[hh++];//出队的顺序就是拓扑序
        //遍历头节点每个出边
        for(int i=h[t];i!=-1;i=ne[i])
        {
            int j=e[i];
            //出边能到的节点入度减1
            d[j]--;
            if(d[j]==0)q[++tt]=j;//如果节点j，入度0则入队
        }
    }

    return tt==n-1;//不要打成=，所有点都入队了说明存在拓扑序列
}
int main()
{
    cin>>n>>m;
    memset(h,-1,sizeof(h));
    for(int i=0;i<m;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
        d[b]++;//b节点入度增加1
    }
    if(topsort())
```

```
    {
        for(int i=0;i<n;i++)printf("%d ",q[i]);
        puts("");
    }
    else puts("-1");
    return 0;
}
```

最短路

dijkstra朴素稠密图O（n^2)

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


const int N=510;
int n,m;
int g[N][N];//邻接矩阵处理稠密图
int dist[N];
bool st[N];
int dijkstra()
{
    memset(dist,0x3f,sizeof(dist));//距离初始化为正无穷
    memset(st,0,sizeof st);
    dist[1]=0;//一号点初始化为0

    for(int i=0;i<n;i++)//迭代n次
    {
        int t=-1;//t开始为-1表示还没确定最短路
        for(int j=1;j<=n;j++)
            if(!st[j]&&(t==-1||dist[t]>dist[j]))//所有st[j]=0的点中找到距离最小的点
                t=j;
        st[t]=1;

        for(int j=1;j<=n;j++)//用t更新其他点到1的距离，遍历边有效更新m次
            dist[j]=min(dist[j],dist[t]+g[t][j]);
    }

    if(dist[n]==0x3f3f3f3f)return -1;
    return dist[n];
}
int main()
{
    scanf("%d%d",&n,&m);

    memset(g,0x3f,sizeof(g));//初始化点位无穷

    while(m--)
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        g[a][b]=min(g[a][b],c);//处理重边保留距离最短的即可
    }

    int t=dijkstra();
```

```
    printf("%d\n",t);

    return 0;
}
```

dijkstra堆优化稀疏图O(mlogn)

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int> PII;
int n,m,s;
const int N=5e5+10;//邻接表N看边数啊啊啊
int h[N],e[N],ne[N],w[N],idx;
int dist[N];
bool vis[N];
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
void dijkstra(int st)
{
    memset(dist,0x3f,sizeof dist);//你开0X3f就要对于int的数组，不要用long long数组开memset 0x3f
    memset(vis,0,sizeof vis);
    dist[st]=0;
    priority_queue<PII,vector<PII>,greater<PII>>heap;//小根堆，顺序不能换，因为pair按first排序
    heap.push({0,st});//first距离，second编号
    while(heap.size())
    {
        //维护当前未被st标记且离源点最近的点
        auto t=heap.top();
        heap.pop();
        int ver=t.second,distance=t.first;
        if(vis[ver])continue;
        vis[ver]=1;
        for(int i=h[ver];i!=-1;i=ne[i])//用t更新其他点
        {
            int j=e[i];
            if(dist[j]>distance+w[i])
            {
                dist[j]=distance+w[i];//松弛
                heap.push({dist[j],j});
            }
        }
    }
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m>>s;
    memset(h,-1,sizeof h);
    while(m--)
    {
```

```
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    dijkstra(s);
    for(int i=1;i<=n;i++)
    {
        if(dist[i]!=0x3f3f3f3f)cout<<dist[i]<<" ";
        else cout<<"2147483647 ";
    }
    return 0;
}
```

dijkstra反向建图求多个点到起点的最短路

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

const int N=1e4+10,M=2e5+10;
int h[N],e[M],w[M],ne[M],idx;
int n,m;
int dist[N];
bool vis[N];
typedef pair<int,int> PII;
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
void dijkstra(int st)
{
    memset(dist,0x3f,sizeof dist);
    memset(vis,0,sizeof vis);
    dist[st]=0;
    priority_queue<PII,vector<PII>,greater<PII>>heap;
    heap.push({0,st});
    while(heap.size())
    {
        auto t=heap.top();
        int ver=t.second,distance=t.first;
        heap.pop();
        if(vis[ver])continue;
        vis[ver]=1;
        for(int i=h[ver];i!=-1;i=ne[i])
        {
            int j=e[i];
            if(dist[j]>distance+w[i])
            {
                dist[j]=distance+w[i];
                heap.push({dist[j],j});
            }
        }
    }

}
int main()
{
    ios::sync_with_stdio(0);
```

```
    cin.tie(0);
    cout.tie(0);
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
        add(b+n,a+n,c);

    }
    dijkstra(1);
    int res=0;
    for(int i=2;i<=n;i++)res+=dist[i];
    dijkstra(n+1);
    for(int i=n+2;i<=2*n;i++)res+=dist[i];
    cout<<res;
}
```

bellman_ford()(处理边数限制)

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


const int N=510,M=10010;


int n,m,k;
int dist[N],backup[N];


struct Edge
{
    int a,b,w;
}edges[M];


int bellman_ford()
{
    memset(dist,0x3f,sizeof dist);
    dist[1]=0;//初始化
    for(int i=0;i<k;i++)
    {
        memcpy(backup,dist,sizeof dist);//防止串联更新
        for(int j=0;j<m;j++)
        {
            int a=edges[j].a,b=edges[j].b,w=edges[j].w;
            dist[b]=min(dist[b],backup[a]+w);//用备份更新
        }
    }
    if(dist[n]>0x3f3f3f3f/2)return -1;//
    return dist[n];
}
int main()
{
```

```
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m>>k;

    for(int i=0;i<m;i++)
    {
        int a,b,w;
        cin>>a>>b>>w;
        edges[i]={a,b,w};
    }
    int t=bellman_ford();
    if(t==-1)puts("impossible");
    else cout<<t<<endl;
    return 0;
}
```

SPFA

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int h[N],e[N],w[N],ne[N],idx;
bool st[N];
int dist[N];
int n,m;
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}


int spfa()
{
    memset(dist,0x3f,sizeof dist);
    dist[1]=0;
    queue<int>q;
    q.push(1);
    st[1]=1;
    while(q.size())
    {
        int t=q.front();
        q.pop();
        st[t]=0;
        for(int i=h[t];i!=-1;i=ne[i])
        {
            int j=e[i];
            if(dist[j]>dist[t]+w[i])
            {
                dist[j]=dist[t]+w[i];
                if(!st[j])
                {
                    q.push(j);
                    st[j]=1;
                }
            }
        }
    }
```

```
    if(dist[n]>0x3f3f3f3f/2)return -1;
    return dist[n];
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    int t=spfa();
    if(t==-1)puts("impossible");
    else cout<<t<<endl;
    return 0;
}
```

SPFA（判有无负权环）

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


const int N=2010,M=10010;
int h[N],e[M],ne[M],w[M],idx;
int dist[N],cnt[N];
bool st[N];
int n,m;
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
bool spfa()
{
    queue<int>q;
    for(int i=1;i<=n;i++)q.push(i),st[i]=1;
    while(q.size())
    {
        int t=q.front();
        q.pop();
        st[t]=0;
        for(int i=h[t];i!=-1;i=ne[i])
        {
            int j=e[i];
            if(dist[j]>dist[t]+w[i])
            {
                dist[j]=dist[t]+w[i];
                cnt[j]=cnt[t]+1;//不要写++cnt[j]，重边会影响的
                if(cnt[j]>=n)return 1;
                if(!st[j])
                {
                    q.push(j);
                    st[j]=1;
```

```
                }
            }
        }
    }
    return 0;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    if(spfa())cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
    return 0;
}
```

Floyd(多源汇最短路)

```
#include<bits/stdc++.h>
using namespace std;
int n,m,q;
const int N=210,INF=1e9;
int d[N][N];

void floyd()
{
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m>>q;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(i==j)d[i][j]=0;//处理自环
            else d[i][j]=INF;

    while(m--)
    {
        int a,b,w;
        cin>>a>>b>>w;
        d[a][b]=min(d[a][b],w);
    }
    floyd();
    while(q--)
    {
```

```
        int a,b;
        cin>>a>>b;
        if(d[a][b]>INF/2)cout<<"impossible"<<endl;
        else cout<<d[a][b]<<endl;
    }
    return 0;
}
```

Floyd求最短环

```
#include<bits/stdc++.h>
using namespace std;
const int N=200;
typedef long long ll;
const int INF=0x7f7f7f7f;
ll g[N][N],dist[N][N];
int n,m;
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(i!=j)dist[i][j]=g[i][j]=INF;
    while(m--)
    {
        ll a,b,c;
        cin>>a>>b>>c;
        g[a][b]=g[b][a]=min(g[a][b],c);
        dist[a][b]=dist[b][a]=min(g[a][b],c);
    }
    ll ans=INF;
    for(int k=1;k<=n;k++)
    {
        for(int i=1;i<k;i++)
            for(int j=i+1;j<k;j++)
                ans=min(ans,dist[i][j]+g[i][k]+g[k][j]);
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
            {
                dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
                dist[j][i]=dist[i][j];
            }
    }
    if(ans==INF)cout<<"No solution.";
    else cout<<ans;
    return 0;
}
```

最小生成树

Kruskal（稀疏图）(O(mlogm))

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e6+10;
int n,m;
int f[N];
struct Edge
{
    int a,b,w;
}edges[N];
bool cmp(Edge a,Edge b)
{
   return a.w<b.w;//如果写成if的最后别忘记加return 0
}

int find(int x)
{
    if(f[x]!=x)f[x]=find(f[x]);
    return f[x];
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    for(int i=0;i<m;i++)
    {
        int a,b,c;
        cin>>a>>b>>c;
        edges[i]={a,b,c};
    }
    sort(edges,edges+m,cmp);//对边排序
    int res=0,cnt=0;//res最小生成树边权重之和，cnt记录最小生成树中的边数
    //并查集看有没有在集合里
    for(int i=0;i<=m;i++)f[i]=i;
    for(int i=0;i<m;i++)
    {
        int a=edges[i].a,b=edges[i].b,w=edges[i].w;
        a=find(a),b=find(b);
        if(a!=b)
        {
            f[a]=b;
            res+=w;
            cnt++;
        }
    }
    if(cnt<n-1)cout<<"impossible"<<endl;
    else cout<<res<<endl;
    return 0;
}
```

Prim(稠密图)(O(n^2))

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e3,INF=0x3f3f3f3f;
int dist[N],g[N][N];//dist是顶点到任意一个树顶点的最短距离
bool st[N];
int n,m;
int prim()
{
    memset(dist,0x3f,sizeof dist);
    int res=0;//存最小生成树所有边长度之和
    for(int i=0;i<n;i++)
    {
        int t=-1;
        for(int j=1;j<=n;j++)//找集合外所有点中到集合距离最小的点
            if(!st[j]&&(t==-1||dist[t]>dist[j]))
                t=j;
        if(i&&dist[t]==INF)return INF;//不是第一个点而且最短距离都为INF，就不存在最小生成树
        if(i)res+=dist[t];//只要不是第一个点
        st[t]=1;
        //扫描顶点t的所有边，在以t为中心更新其他点到树的距离（这时候t已经在生成树里了，其他点到t距离就是到生成树距离）
        for(int j=1;j<=n;j++)dist[j]=min(dist[j],g[t][j]);
    }
    return res;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    memset(g,0x3f,sizeof g);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        g[a][b]=g[b][a]=min(g[a][b],c);
    }
    int t=prim();
    if(t==INF)cout<<"impossible";
    else cout<<t<<endl;
    return 0;
}
```

二分图

染色法判二分图(O(m+n))

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

const int N=1e6+10,M=2e6+10;

int n,m;
int h[N],e[M],ne[M],idx;
int color[N];
```

```
bool dfs(int u,int c)//u为点编号，c为染色
{
    color[u]=c;
    for(int i=h[u];i!=-1;i=ne[i])//遍历和点连接的点
    {
        int j=e[i];
        if(!color[j])//没染色,那就染（3-c实现1染2，2染1）
        {
            if(!dfs(j,3-c))return 0;
        }
        else if(color[j]==c)return 0;//已经染色
    }
    return 1;
}
void add(int a,int b)
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
        add(b,a);
    }
    bool flag=1;//染色是否有矛盾发生
    for( int i=1;i<=n;i++)
        if(!color[i])
        {
            if(!dfs(i,1))//dfs false有矛盾发生
            {
                flag=0;
                break;
            }
        }

    if(flag)puts("Yes");
    else puts("No");
}
```

匈牙利算法二分图最大匹配图

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

const int N=510,M=100010;

int n1,n2,m;
int h[N],e[M],ne[M],idx;
int match[N];
bool st[N];
```

```
void add(int a,int b)
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
bool find(int x)
{
    for(int i=h[x];i!=-1;i=ne[i])//x是男的，j是妹子，遍历男的看上的所有妹子
    {
        int j=e[i];
        if(!st[j])
        {
            st[j]=1;
            if(match[j]==0||find(match[j]))//妹子没有匹配或者妹子原本匹配的男的有备胎
            {
                match[j]=x;
                return 1;
            }
        }
    }
    return 0;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n1>>n2>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
    }
    int res=0;
    for(int i=1;i<=n1;i++)
    {
        memset(st,0,sizeof st);
        if(find(i))res++;
    }
    cout<<res<<endl;
    return 0;
}
```

前缀和/差分

前缀和

一维前缀和

```
S[i] = a[1] + a[2] + ... a[i]
a[l] + ... + a[r] = S[r] - S[l - 1]
```

二维前缀和

```
S[i, j] = 第i行j列格子左上部分所有元素的和
以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为：
S[x2, y2] - S[x1 - 1, y2] - S[x2, y1 - 1] + S[x1 - 1, y1 - 1]
```

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

const int N=1e3+5;
int mp[N][N],dp[N][N];
int main()
{
    int n,m,q,x1,x2,y1,y2;
    cin>>n>>m>>q;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            cin>>mp[i][j];
        }
    }
    memset(dp,0,sizeof(dp));

//预处理二位前缀和数组dp
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1]+mp[i][j];
        }
    }


    while(q--)
    {
        cin>>x1>>y1>>x2>>y2;
        cout<<dp[x2][y2]-dp[x2][y1-1]-dp[x1-1][y2]+dp[x1-1][y1-1]<<endl;
    }
}
```

Tip:前缀和一些注意点(激光炸弹为例)

1. 卡内存，直接累加读入，开const in N的时候别太浪

   ```
   cin>>s[i][j];
   s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1];
   ```

2. 卡边界，为了方便处理前缀和，最好把前面的s[0]留出来，所以预处理

   ```
   s[++x][++y]+=w;
   //s[x++][y++]+=w;错误
   ```

3. 覆盖范围处理(r不出界)

```
r=min(5001,r);
//5001为最大可能边界
```

```
#include<bits/stdc++.h>
using namespace std;
const int N=5e3+10;
typedef long long ll;
int s[N][N];
int n,r,x,y,w;
int main()
{
    cin>>n>>r;
    r=min(5001,r);//预处理半径
    while(n--)
    {
        cin>>x>>y>>w;
        s[++x][++y]+=w;
    }
    for(int i=1;i<=5001;i++)
        for(int j=1;j<=5001;j++)
            s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1];//直接累加节省内存

    int res=0;
    //枚举右下角
    for(int i=r;i<=5001;i++)//直接开地图最大
        for(int j=r;j<=5001;j++)
            res=max(res,s[i][j]-s[i][j-r]-s[i-r][j]+s[i-r][j-r]);

    cout<<res;
    return 0;
}
```

差分

一维差分

```
void init()
{
    memset(b,0,sizeof b);
    for(int i=1;i<=n;i++)b[i]=a[i]-a[i-1];
}
void edit(int l,int r,int c)
{
    //区间预处理非常规情况
    if(l>r)swap(l,r);
    if(l<st)l=st;
    if(r<st)r=st;
    if(l>ed)l=ed;
    if(r>ed)r=ed;
    b[l]+=c;b[r+1]-=c;
}
void build()
{
    for(int i=1;i<=n;i++)a[i]=a[i-1]+b[i];
```

```
}
```

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

//给区间[l, r]中的每个数加上c：B[l] += c, B[r + 1] -= c
//要确保修改的时候r+1>=l,l和r在区间范围内
const int N=1e6+5;
int a[N],b[N];
int main()
{
    int n,l,r,c;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        b[i]=a[i]-a[i-1];//构造差分数组
    }
    while(cin>>l>>r>>c)
    {
        b[l]+=c;
        b[r+1]-=c;
        for(int i=1;i<=n;i++)a[i]=a[i-1]+b[i];
        for(int i=1;i<=n;i++)cout<<a[i]<<" ";//如果是连续差分求最终值，把这条放到while外面就可以了，一般差分数据量比较大建议快速读入
        cout<<endl;
    }
    return 0;
}
```

二维差分

```
给以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵中的所有元素加上c：
S[x1, y1] += c, S[x2 + 1, y1] -= c, S[x1, y2 + 1] -= c, S[x2 + 1, y2 + 1] += c
```

例题

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

const int N=1e4+5;
int a[N][N],b[N][N];

inline int read()
{
    int x=0,y=1;char c=getchar();//y代表正负（1.-1），最后乘上x就可以了。
    while (c<'0'||c>'9') {if (c=='-') y=-1;c=getchar();}//如果c是负号就把y赋为-1
    while (c>='0'&&c<='9') x=x*10+c-'0',c=getchar();
    return x*y;//乘起来输出
}

int main()
{
    int n,m,q,x1,x2,y1,y2,c;
```

```
    n=read(),m=read(),q=read();
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            a[i][j]=read();
            b[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1];//预处理差分
        }
    }
    while(q--)
    {
        x1=read(),y1=read(),x2=read(),y2=read(),c=read();
        b[x1][y1]+=c;
        b[x1][y2+1]-=c;
        b[x2+1][y1]-=c;
        b[x2+1][y2+1]+=c;
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            a[i][j]=a[i-1][j]+a[i][j-1]-a[i-1][j-1]+b[i][j];
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

注意事项

1. 前缀和，差分尽量使用快读
2. 涉及最大最小到时候min，max初值设为0，以免遗漏开头的
3. 前缀和左上角，差分右下角，两者互为逆运算，容斥定理可推公式
4. 前缀和区间快速求和，差分区间增减修改

字符串

KMP

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

const int N=1e5+10,M=1e6+10;
int n,m;
char p[N],s[M];
int ne[N];//最长公共前缀后缀

int main()
{
    cin>>n>>p+1>>m>>s+1;
    //预处理 ne数组
    for(int i=2,j=0;i<=n;i++)//从第二个开始处理，第一个肯定0啊
    {
        while(j&&p[i]!=p[j+1])j=ne[j];//j+1和i试探一下看一不一样，不一样就j=ne[i]直到开
头
```

```
        if(p[i]==p[j+1])j++;
        ne[i]=j;
    }
    //kmp 匹配
    for(int i=1,j=0;i<=m;i++)
    {
        while(j&&s[i]!=p[j+1])j=ne[j];
        if(s[i]==p[j+1])j++;
        if(j==n)
        {
            printf("%d ",i-n);
            j=ne[j];
        }
    }
    return 0;
}
```

DP

DP思考方式

状态表示

集合

1. 维度的确定是最少要用几个维度来唯一确定如：背包有容量和价值表状态，最原始两维度。最长子序列以i结尾，就只要一个。最长公共子序列两个序列，就要两个维度
2. 所有满足+（题目条件+状态表示的条件）+的集合

属性

1. max(开long long)
2. min（开long long）
3. 方案数（直接开unsigned long long防爆）
4. 具体方案（记录状态转移）
5. 可达性（用|）

状态计算

划分

原则：补充不漏

1. 以i-1为倒数第二个（LIS）
2. 转移来源(数字三角)
3. 选i与不选i/选几个（背包）
4. a[i],b[j]是否包含在子序列当中

计算过程

要保证前面的已经计算好了，状态转移不能成环

背包

01背包

二维

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

const int N=1e4+10;

int n,m;
int v[N],w[N];
int f[N][N];


int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
    //所有状态f[0~n][0~m]
    //f[0][0~m]=0所以i就不从0开始了
    for(int i=1;i<=n;i++)
    {
        for(int j=0;j<=m;j++)
        {
            f[i][j]=f[i-1][j];//左边不含i，最大值就是f[i-1][j]
            if(j>=v[i])f[i][j]=max(f[i][j],f[i-1][j-v[i]]+w[i]);//装得下v[i]才有这
种情况，第一个就是左边最大值，第二个就是右边最大值，>=不要打成>
        }
    }
    cout<<f[n][m]<<endl;
    return 0;

}
```

一维

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

const int N=1e4+10;
int n,m;
int v[N],w[N];
int f[N];//有时候要开long long不然会爆

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];

    for(int i=1;i<=n;i++)
        for(int j=m;j>=v[i];j--)
            f[j]=max(f[j],f[j-v[i]]+w[i]);
    cout<<f[m]<<endl;
    return 0;
```

```
}
```

完全背包

二维

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

const int N=1e4+10;

int n,m;
int v[N],w[N];
int f[N][N];

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];

    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++)
            for(int k=0;k*v[i]<=j;k++)
                f[i][j]=max(f[i][j],f[i-1][j-v[i]*k]+w[i]*k);
    cout<<f[n][m]<<endl;
    return 0;
```

二维优化

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1010;
int f[N][N];
int v[N],w[N];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i = 1 ; i <= n ;i ++)cin>>v[i]>>w[i];


    for(int i = 1 ; i<=n ;i++)
        for(int j = 0 ; j<=m ;j++)
        {
            f[i][j] = f[i-1][j];
            if(j>=v[i])f[i][j]=max(f[i][j],f[i][j-v[i]]+w[i]);
        }
    cout<<f[n][m]<<endl;
}
```

一维优化

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

const int N=1e4+10;

int n,m;
int v[N],w[N];//有时候要开long long不然会爆
int f[N];


int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];


    for(int i=1;i<=n;i++)
        for(int j=v[i];j<=m;j++)
            f[j]=max(f[j],f[j-v[i]]+w[i]);
    cout<<f[m]<<endl;
    return 0;
}
```

多重背包

暴力朴素

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


const int N=1e4+10;


int n,m;
int v[N],w[N],s[i];
int f[N][N];


int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i]>>s[i];


    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++)
            for(int k=0;k<=s[i]&&k*v[i]<=j;k++)
                f[i][j]=max(f[i][j],f[i][j-v[i]*k]+w[i]*k)
    cout<<f[n][m]<<endl;
    return 0;


}
```

二进制优化

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

const int N=25000,M=2010;//N要拆出来所以1000*log2000

int n,m;
int v[N],w[N];
int f[N];
int main()
{
    cin>>n>>m;
    int cnt=0;
    for(int i=1;i<=n;i++)
    {
        int a,b,s;//体积，价值，个数
        cin>>a>>b>>s;
        int k=1;
        while(k<=s)
        {
            cnt++;
            v[cnt]=a*k;//k个物品打包
            w[cnt]=b*k;//k个物品打包
            s-=k;
            k*=2;
        }
        if(s>0)//补上c
        {
            cnt++;
            v[cnt]=a*s;
            w[cnt]=b*s;
        }
    }
    //01背包
    n=cnt;
    for(int i=1;i<=n;i++)
         for(int j=m;j>=v[i];j--)
             f[j]=max(f[j],f[j-v[i]]+w[i]);

    cout<<f[m]<<endl;
    return 0;
}
```

分组背包

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

const int N=110;

int n,m;
int v[N][N],w[N][N],s[N];//s表示第i组物品种类
int f[N];

int main()
{
```

```
    cin>>n>>m;//n组物品，m容量
    for(int i=1;i<=n;i++)
    {
        cin>>s[i];
        for(int j=0;j<s[i];j++)
            cin>>v[i][j]>>w[i][j];
    }

    for(int i=1;i<=n;i++)
        for(int j=m;j>=0;j--)//i-1推i逆序
            for(int k=0;k<s[i];k++)//有点像完全背包，k就是下标，注意自己是0开始还是1开始的。选第i组的第k件物品
                if(v[i][k]<=j)
                    f[j]=max(f[j],f[j-v[i][k]]+w[i][k]);

    cout<<f[m]<<endl;
    return 0;
}
```

背包方案数

二维

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

const int N=1e4+10;
int w[N],f[N][N];
int main(){
        int n,m;
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++)scanf("%d",&w[i]);
        for(int i=0;i<=n;i++)f[i][0]=1;//从0开始

        for(int i=1;i<=n;i++)
                for(int j=1;j<=m;j++)
                {
                        f[i][j]+=f[i-1][j];
                        if(j>=w[i])f[i][j]+=f[i-1][j-w[i]];
                }

        printf("%d",f[n][m]);
        return 0;
}
```

一维

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


const int N=1e4+10;
int w[N],f[N];
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
```

```
    for(int i=1;i<=n;i++)scanf("%d",&w[i]);
    f[0]=1;
    for(int i=1;i<=n;i++)
        for(int j=m;j>=w[i];j--)
            f[j]+=f[j-w[i]];

    printf("%d",f[m]);
    return 0;
}
```

####