Standard Code Library

Three Pigeons

Shanghai University

November 1, 2019

Contents

模板不完整度极高

数据范围

$bool$: $1byte$　$0/1$

int : $4byte$　$-2147483648 \sim 2147483647$　2.1×10^9

$unsigendint$: $0 \sim 4294967295$

$long\ long$: $8byte$　$-9223372036854775808 \sim 9223372036854775807$　9.2×10^{18}

$unsigend/long/long$: $0 \sim 18446744073709551615$　1.8×10^{19}

$double$: $8byte$　$15 \sim 16$ 位有效数字

$long\ double$: $12byte?$ ← 由编译器决定 $18 \sim 19$ 位有效数字

$0x3f3f3f3f$: 1061109567　1×10^9

$0x3f3f3f3f3f3f3f3f$: 4557430888798830399　4.5×10^{18}

对拍程序

```cpp
/*
bf.cpp 是暴力代码
sol.cpp 是正解代码
random.cpp 生成随机数据
将三个文件都放进来跑以下代码对拍
*/
#include <bits/stdc++.h>
using namespace std;
int main() {
    for(int T = 0; T < 10000; ++T) {
        system("./random > ./data.in");
        double st = clock();
        system("./bf < ./data.in > ./data.ans");
        double et = clock();
        system("./sol < ./data.in > ./data.out");
        if(system("diff ./data.out ./ data.ans")) {
            puts("Wrong Answer\n");
            return 0;
        } else {
            printf("Accepted, 测试点 #%d, 用时 %.0lfms\n", T, et - st);
        }
    }
    return 0;
}
```

STL

怕忘，全部记一遍

set

```
set<int> s;
set<double> ss;
s.begin() // 返回指向第一一个元素的迭代器器
s.clear() // 清除所有元素
s.count() // 返回某个值元素的个数
s.empty() // 如果集合为空，返回 true(真)
s.end() // 返回指向最后一一个元素之后的迭代器器，不不是最后一一个元素
s.equal_range() // 返回集合中与给定值相等的上下限的两个迭代器器
s.erase() // 删除集合中的元素
s.find() // 返回一一个指向被查找到元素的迭代器器
s.get_allocator() // 返回集合的分配器器
s.insert() // 在集合中插入入元素
s.lower_bound() // 返回指向大大于（或等于）某值的第一一个元素的迭代器器
s.key_comp() // 返回一一个用用于元素间值比比较的函数
s.max_size() // 返回集合能容纳的元素的最大大限值
s.rbegin() // 返回指向集合中最后一一个元素的反向迭代器器
s.rend() // 返回指向集合中第一一个元素的反向迭代器器
s.size() // 集合中元素的数目目
s.swap() // 交换两个集合变量量
s.upper_bound() // 返回大大于某个值元素的迭代器器
s.value_comp() // 返回一一个用用于比比较元素间的值的函数
```

multiset.count() 有多少个集合返回几

vector

```
vector<int> s;
// 定义一一个空的 vector 对象，存储的是 int 类型的元素
vector<int> s(n);
// 定义一一个含有 n 个 int 元素的 vector 对象
vector<int> s(first, last);
// 定义一一个 vector 对象，并从由迭代器器 first 和 last 定义的序列列 [first, last) 中复制初
↪ 值
s[i] // 直接以下标方方式访问容器器中的元素
s.front() // 返回首首元素
s.back() // 返回尾元素
s.push_back(x) // 向表尾插入入元素 x
s.size() // 返回表🄵长
s.empty() // 表为空时，返回真，否则返回假
s.pop_back() // 删除表尾元素
s.begin() // 返回指向首首元素的随机存取迭代器器
```

```
s.end() // 返回指向尾元素的下一一个位置的随机存取迭代器器
s.insert(it, val) // 向迭代器器 it 指向的元素前插入入新元素 val
s.insert(it, n, val) // 向迭代器器 it 指向的元素前插入入 n 个新元素 val
s.insert(it, first, last)
// 将由迭代器器 first 和 last 所指定的序列列 [first, last) 插入入到迭代器器 it 指向的元素前面面
s.erase(it) // 删除由迭代器器 it 所指向的元素
s.erase(first, last) // 删除由迭代器器 first 和 last 所指定的序列列 [first, last)
s.reserve(n) // 预分配缓冲空间，使存储空间至至少可容纳 n 个元素
s.resize(n)
/*
改变序列列🄵长度，超出的元素将会全部被删除，如果序列列需要扩展（原空间小小于 n），元素默认值将填满扩展出的空间
*/
s.resize(n, val)
/*
改变序列列🄵长度，超出的元素将会全部被删除，如果序列列需要扩展（原空间小小于 n），val 将填满扩展出的空间
*/
s.clear() // 删除容器器中的所有元素
s.swap(v) // 将 s 与另一一个 vector 对象进行行行交换
s.assign(first, last)
// 将序列列替换成由迭代器器 first 和 last 所指定的序列列 [first, last)，[first, last) 不不能是原序列列中的一一部分
```

string

```
/*
不建议使用
支持 vector 的操作
再加上字符串的拼接
*/
```

stack

```
stack<int> s;
stack<string> ss;
s.push(x); // 入入栈
s.pop(); // 出栈
s.top(); // 访问栈顶
s.empty(); // 当栈空时，返回 true
s.size(); // 访问栈中元素个数
```

queue

```
queue<int> q;
queue<double> qq;
```

```
q.push(x); // 入入队列列
q.pop(); // 出队列列
q.front(); // 访问队首首元素
q.back(); // 访问队尾元素
q.empty(); // 判断队列列是否为空
q.size(); // 访问队列列中的元素个数
```

priority_queue

```
priority_queue<int> q;
priority_queue<pair<int, int> > qq; // 注意在两个尖括号之间一一定要留留空格，防止止误判
priority_queue<int, vector<int>, greater<int> > qqq; // 定义小的先出队列
priority_queue<int, vector<int>, less<int> > qqq; // 定义大的先出队列
q.empty() // 如果队列列为空，则返回 true，否则返回 false
q.size() // 返回队列列中元素的个数
q.pop() // 删除队首首元素，但不不返回其值
q.top() // 返回具有最高高优先级的元素值，但不不删除该元素
q.push(item) // 在基于优先级的适当位置插入入新元素
```

map

```
map<string, int> m;
m[key] = value;
/*
[key] 操作是 map 很有特色的操作，如果在 map 中存在键值为 key 的元素对，则返回该元素对的值域
 ↪ 部分，否则将会创建一一个键值为 key 的元素对，值域为默认值。所以可以用用该操作向 map 中插
 ↪ 入入元素对或修改已经存在的元素对的值域部分。
*/
m.insert(make_pair(key, value));
/*
也可以直接调用用 insert 方法插入入元素对,insert 操作会返回一一个 pair，当 map 中没有与 key
 ↪ 相匹配的键值时，其 first 是指向插入入元素对的迭代器器，其 second 为 true；若 map 中已经
 ↪ 存在与 key 相等的键值时，其 first 是指向该元素对的迭代器器,second 为 false。
*/
int i = m[key];
/*
要注意的是，当与该键值相匹配的元素对不存在时，会创建键值为 key（当另一一个元素是整形时，
 ↪ m[key]=0）的元素对。
*/
map<string, int>::iterator it = m.find(key);
/* 如果 map 中存在与 key 相匹配的键值时,find 操作将返回指向该元素对的迭代器，否则，返回的迭代
 ↪ 器等于 map 的 end()(参F见 vector 中提到的 begin() 和 end() 操作)。
*/
m.erase(key); // 删除与指定 key 键值相匹配的元素对，并返回被删除的元素的个数。
m.erase(it); // 删除由迭代器 it 所指定的元素对，并返回指向下一一个元素对的迭代器。
m.size(); // 返回元素个数
```

```
m.empty(); // 判断是否为空
m.clear(); // 清空所有元素
```

bitset

```
const int MAXN = 32;
bitset<MAXN> bt; // bt 包括 MAXN 位，下标 0 ~ MAXN - 1，默认初始化为 0
bitset<MAXN> bt1(0xf); // 0xf 表示十十六进制数 f，对应二二进制 1111，将 bt1 低 4 位初始化
 ↪ 为 1
bitset<MAXN> bt2(012); // 012 表示八八进制数 12，对应二二进制 1010，
// 即将 bt2 低 4 位初始化为 1010
bitset<MAXN> bt3("1010"); // 将 bt3 低 4 位初始化为 1010
bitset<MAXN> bt4(s, pos, n); // 将 01 字符串串 s 的 pos 位开始的 n 位初始化 bt4
bt.any() // bt 中是否存在置为 1 的二二进制位？
bt.none() // bt 中不不存在置为 1 的二二进制位吗？
bt.count() // bt 中置为 1 的二二进制位的个数
bt.size() // bt 中二二进制位的个数
bt[pos] // 访问 bt 中在 pos 处的二二进制位
bt.test(pos) // bt 中在 pos 处的二二进制位是否为 1
bt.set() // 把 bt 中所有二二进制位都置为 1
bt.set(pos) // 把 bt 中在 pos 处的二二进制位置为 1
bt.reset() // 把 bt 中所有二二进制位都置为 0
bt.reset(pos) // 把 bt 中在 pos 处的二二进制位置为 0
bt.flip() // 把 bt 中所有二二进制位逐位取反
bt.flip(pos) // 把 bt 中在 pos 处的二二进制位取反
bt[pos].flip() // 同上
bt.to_ulong() // 用用 bt 中同样的二二进制位返回一一个 unsigned long 值
os << bt // 把 bt 中的位集输出到 os 流
```

数学

整除

1. 若 $a|b \Leftrightarrow -a|b \Leftrightarrow a|-b \Leftrightarrow |a||b|$

2. 若 $a|b$，$b|c \rightarrow a|c$

3. 若 a|b，a|c -> a|(bx+cy) 其中 x，y 为任意整数

4. 若 a|b -> am|bm 其中 m 为非零整数

5. 若 a|b，b|a -> b=±a <-> |b|=|a|

6. 若 a|bc，且 a 与 c 互质，则 a|b

7. 若 a|b，a|c，且 b 与 c 互质，则 a|bc

8. 若 a|b，c 为任意整数，则 b|ac

9. 对任意整数 a，b>0，存在唯一的数对 q，r，使 a=bq+r，其中 0 r<b，这个事实称为带余除法定理，是整除理论的基础

10. 若 c|a，c|b，则称 c 是 a，b 的公因数。若 d 是 a，b 的公因数，d 0，且 d 可被 a，b 的任意公因数整除，则 d 是 a，b 的最大公因数。若 a，b 的最大公因数等于 1，则称 a，b 互素，也称互质。累次利用带余除法可以求出 a，b 的最大公因数，这种方法常称为辗转相除法。又称欧几里得算法。

部分定理

1. 欧拉定理：对于互质的正整数 a 和 n，有 $a^{\varphi}(n) \equiv 1(mod\ n)$

2. 费马小定理：若 $gcd(a,b) = 1$，则 $a^{p-1} \equiv 1(mod\ n)$，当 n 为质数时，$inv(a) = a^{n-2} \mod n$

3. 威尔逊定理：当且仅当 p 为素数时，$(p-1)! \equiv -1(mod\ p)$

素数

欧拉筛 $O(n)$

```
/*
罗老师的代码
*/
const int maxn = 1e7 + 10;
int prime[maxn] = {0}, phi[maxn] = {0}, tot;

void euler()
{
    phi[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!phi[i])
        {
            prime[tot++] = i;
```

```
            phi[i] = i - 1;
        }
        for (int j = 0; j < tot && i * prime[j] < maxn; j++)
        {
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * phi[prime[j]];
        }
    }
}
```

莫比乌斯筛

不知道干嘛用的

```
const int maxn = 1e7 + 10;
int prime[maxn], tot = 0, mu[maxn];
bool check[maxn];

void mobius()
{
    mu[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!check[i])
        {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < tot && i * prime[j] < maxn; j++)
        {
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                mu[i * prime[j]] = 0;
                break;
            }
            mu[i * prime[j]] = -mu[i];
        }
    }
}
```

Lucas 定理

用来求 $C_n^m \bmod p$

```
/*
组合数取模
*/
ll PowMod(ll a,ll b,ll MOD){  //费马小定理求逆元，快速幂
    ll ret=1;
    while(b){
        if(b&1) ret=(ret*a)%MOD;
        a=(a*a)%MOD;
        b>>=1;
    }
    return ret;
}
ll fac[100005];
ll Get_Fact(ll p){
    fac[0]=1;
    for(ll i=1;i<=p;i++)
        fac[i]=(fac[i-1]*i)%p;  //预处理阶乘
}
ll Lucas(LL n,LL m,LL p){
    ll ret=1;
    while(n&&m){
        ll a=n%p,b=m%p;
        if(a<b) return 0;
        ret=(ret*fac[a]*PowMod(fac[b]*fac[a-b]%p,p-2,p))%p;
        n/=p;
        m/=p;
    }
    return ret;
}
```

欧拉函数

$\varphi(n)$ n n

性质

1. 对于质数 p，$\varphi(p) = p - 1$。注意 $\varphi(1) = 1$

2. 若 m,n 互质，$\varphi(mn) = \varphi(m)\varphi(n)$

3. 当 n 为奇数时，$\varphi(2n) = \varphi(n)$

4. 当 $n > 2$ 时，所有 $\varphi(n)$ 都是偶数

5. 当 $n > 6$ 时，所有 $\varphi(n)$ 都是合数

6. 欧拉定理：对于互质的正整数 a 和 n，有 $a^{\varphi(n)} \equiv 1 \mod n$

7. 欧拉定理推论：小于等于 n 的数中，与 n 互质数的总和为：$\varphi(n) \times \frac{n}{2}(n > 1)$

求一个数的欧拉函数 $O(\sqrt{n})$

```
//直接求解欧拉函数
  int euler(int n){ //返回 euler(n)
        int res=n,a=n;
        for(int i=2;i*i<=a;i++){
            if(a%i==0){
                res=res/i*(i-1);//先进行除法是为了防止中间数据的溢出
                while(a%i==0) a/=i;
            }
        }
       if(a>1) res=res/a*(a-1);
       return res;
}
```

求 $1 \sim n$ 的欧拉函数 $O(nlogn)$

好像没什么用，直接欧拉筛得到欧拉函数就好了

```
void Init(){
     euler[1]=1;
     for(int i=2;i<Max;i++)
       euler[i]=i;
     for(int i=2;i<Max;i++)
        if(euler[i]==i)
           for(int j=i;j<Max;j+=i)
              euler[j]=euler[j]/i*(i-1);//先进行除法是为了防止中间数据的溢出
}
```

积性函数

满足 $gcd(a,b) = 1$，且满足 $f(ab) = f(a)f(b)$，则 $f(x)$ 为积性函数

性质

1. 若 f(n),g(n) 均为积性函数，则函数 h(n)=f(n)g(n) 也是积性函数

2. 若 f(n) 为积性函数，则函数也是积性函数，反之一样

3. **任何积性函数都能应用线性筛，在 O(n) 时间内求出 1~n 项**

逆元（三种求法）

1. 费马小定理求逆元，要求**模数 p 为质数**则有

$inv(i) = i^{p-2} \mod p$

2. 扩展欧几里得，模数 p 可以不为质数

```
ll inv(ll a, ll p) //求 a 关于 p 的逆元
{
    ll x, y;
    ll d = exgcd(a, p, x, y);
    if (d != 1)
        return -1;
    return (x % p + p) % p;
}
```

3. 基于 $inv(a) = (p - \lfloor p/a \rfloor) * inv(p\%a)\%p$，在 $O(n)$ 时间复杂度下求出逆元表。(lls 模板)

```
void init()
{
    inv[1] = 1;
    for (int i = 2; i < maxn; i++)
        inv[i] = (mod - mod / i) * 1LL * inv[mod % i] % mod;
}
```

中国剩余定理

对于同余线性方程组

$x \equiv a_1 (mod\ m_1)$

$x \equiv a_2 (mod\ m_2)$

...

$x \equiv a_n (mod\ m_n)$

当 $m_1, m_2, \cdots, m_n$ 两两互质时，方程组有解，求其解 x

```
int CRT(int a[], int m[], int n) {
    int M = 1;
    int ans = 0;
    for(int i = 1; i <= n; i++) {
        M *= m[i];
    }
    for(int i = 1; i <= n; i++) {
        int x, y;
        int Mi = M / m[i];
        exgcd(Mi, m[i], x, y);
        ans = (ans + Mi * x * a[i]) % M;
    }
    if(ans < 0) ans += M;
    return ans;
}
```

扩展中国剩余定理

满足上式条件下 m 可以不互质

```
LL exgcd(LL a,LL b,LL &x,LL &y){
    if(!b){x=1,y=0;return a;}
    LL re=exgcd(b,a%b,x,y),tmp=x;
    x=y,y=tmp-(a/b)*y;
    return re;
}
LL m[maxn],a[maxn];      //m 为模数集, a 为余数集
LL exCRT(){
    LL M=m[1],A=a[1],t,d,x,y;int i;
    for(i=2;i<=n;i++){
        d=exgcd(M,m[i],x,y);//解方程
        if((a[i]-A)%d)return -1;//无解
        x*=(a[i]-A)/d,t=m[i]/d,x=(x%t+t)%t;//求 x
        A=M*x+A,M=M/d*m[i],A%=M;//日常膜一膜 (划掉) 模一模, 防止爆
    }
    A=(A%M+M)%M;
    return A;
}
```

矩阵快速幂

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int mod=123456789;
struct matrix{
    ll a[11][11]; //begin with 1
    int r,c;
    matrix(int n,int m):r(n),c(m){memset(a,0,sizeof(a));}
    ll* operator[](int x){return a[x];}
    friend matrix operator*(matrix A,matrix B)
    {
        matrix C(A.r,B.c);
        for(int i=1;i<=A.r;i++)
            for(int j=1;j<=B.c;j++)
                for(int k=1;k<=A.c;k++){
                    C[i][j]+=(A[i][k]*B[k][j])%mod;
                    C[i][j]+=mod;
                    C[i][j]%=mod;
                }
        return C;
    }
};
```

```
23
24  matrix qpow(matrix A,ll m)//方阵 A 的 m 次幂
25  {
26      matrix ans(A.r,A.c);
27      for(int i=1;i<=A.r;i++) ans.a[i][i]=1; //单位矩阵
28      while(m)
29      {
30          if(m&1)ans=ans*A;
31          A=A*A;
32          m>>=1;
33      }
34      return ans;
35  }
36
37  int main()
38  {
39      ll T,n;
40      for(cin>>T;T--;)
41      {
42          scanf("%lld",&n);
43          matrix A(6,6);
44          A[1][1]=1;
45          A[1][2]=2;
46          A[1][3]=1;
47          A[1][4]=3;
48          A[1][5]=3;
49          A[1][6]=1;
50          A[2][1]=1;
51          A[3][3]=1;
52          A[4][3]=1;
53          A[4][4]=1;
54          A[5][3]=1;
55          A[5][4]=2;
56          A[5][5]=1;
57          A[6][3]=1;
58          A[6][4]=3;
59          A[6][5]=3;
60          A[6][6]=1;
61          matrix X2(6,1);
62          X2[1][1]=2;
63          X2[2][1]=1;
64          X2[3][1]=1;
65          X2[4][1]=2;
66          X2[5][1]=4;
67          X2[6][1]=8;
68          matrix Xn=qpow(A,n-2)*X2;
```

```
69            printf("%lld\n",Xn[1][1]);
70        }
71    }
```

字符串

manacher（马拉车算法）

```
/*
求以任一点为中心的回文半径
O(n)
*/
const int maxl=1100005;
int p[2*maxl+5];   //p[i]-1 表示以 i 为中点的回文串长度
int Manacher(string s)
{
    string now;
    int len=s.size();
    for(int i=0;i<len;i++)      //将原串处理成%a%b%c% 形式，保证长度为奇数
    {
       now+='%';
       now+=s[i];
    }
    now+='%';
    int len=now.size();
    int pos=0,R=0;
    for (int i=0;i<len;i++)
    {
        if (i<R) p[i]=min(p[2*pos-i],R-i); else p[i]=1;
        while (0<=i-p[i]&&i+p[i]<len&&now[i-p[i]]==now[i+p[i]]) p[i]++;
        if (i+p[i]>R) {pos=i;R=i+p[i];}
    }
    int MAX=0;
    for (int i=0;i<len;i++)
    {
        cout<<i<<" : "<<p[i]-1<<endl;            //p[i]-1 为 now 串中以 i 为中点的回文半
        ↪ 径，即是 s 中最长回文串的长度
        cout<<now.substr(i-p[i]+1,2*p[i]-1)<<endl;
        MAX=max(MAX,p[i]-1);
    }
    return MAX;           //最长回文子串长度
}
```

```
/* 罗老师版马拉车 */
const int maxn = 2e5;

string Mnc(string &s)
{
    string t = "$#";
    for (int i = 0; i < s.length(); ++i) //构造辅助串
    {
```

```
        t += s[i];
        t += '#';
    }

    int ml = 0, p = 0, R = 0, M = 0;
    //最大长度，最长回文中心，当前最大回文串右端，当前最长回文中心

    int len = t.length();
    vector<int> P(len, 0); //回长度数组
    for (int i = 0; i < len; ++i)
    {
        P[i] = R > i ? min(P[2 * M - i], R - i) : 1; //转移方程

        while (t[i + P[i]] == t[i - P[i]]) //长度扩张
            ++P[i];

        if (i + P[i] > R) //更新右端和中心
        {
            R = i + P[i];
            M = i;
        }
        if (ml < P[i]) //记录极大
        {
            ml = P[i];
            p = i;
        }
    }

    return s.substr((p - ml) / 2, ml - 1); //返回回文串
}
```

KMP

```
char p[maxn],s[1000005];
int t;
int next1[maxn];
void getNext()
{
    int plen = strlen(p);
    int j = 0, k = -1;
    next1[0] = -1;
    while(j < plen)
    {
        if(k == -1 || p[k] == p[j])
        {
            ++j; ++k;
```

```
            next1[j] = k;
        }
        else
            k = next1[k];
    }
}
int KMP(int ans)
{
    int plen = strlen(p);
    int slen = strlen(s);
    int i = 0, j = 0;
    while(i < slen)
    {
        if(j == -1 || s[i] == p[j])
        {
            ++i; ++j;
        }
        else
            j = next1[j];
        if(j == plen)
        {
            ans++;
            j = next1[j];
        }
    }
    return ans;
}
```

扩展 KMP

```
char s[maxn];
int nxt[maxn],ex[maxn]; //ex 数组即为 extend 数组
//预处理计算 nxt 数组
void GETNEXT(char *str)
{
    int i=0,j,po,len=strlen(str);
    nxt[0]=len;//初始化 nxt[0]
    while(str[i]==str[i+1]&&i+1<len)//计算 nxt[1]
        i++;
    nxt[1]=i;
    po=1;//初始化 po 的位置
    for(i=2;i<len;i++)
    {
        if(nxt[i-po]+i<nxt[po]+po)//第一种情况，可以直接得到 nxt[i] 的值
            nxt[i]=nxt[i-po];
        else//第二种情况，要继续匹配才能得到 nxt[i] 的值
```

```
        {
            j=nxt[po]+po-i;
            if(j<0)j=0;//如果 i>po+nxt[po], 则要从头开始匹配
            while(i+j<len&&str[j]==str[j+i])//计算 nxt[i]
                j++;
            nxt[i]=j;
            po=i;//更新 po 的位置
        }
    }
}

//计算 extend 数组
void EXKMP(char *s1,char *s2)
{
    int i=0,j,po,len=strlen(s1),l2=strlen(s2);
    GETNEXT(s2);//计算子串的 nxt 数组
    while(s1[i]==s2[i]&&i<l2&&i<len)//计算 ex[0]
        i++;
    ex[0]=i;
    po=0;//初始化 po 的位置
    for(i=1;i<len;i++)
    {
        if(nxt[i-po]+i<ex[po]+po)//第一种情况，直接可以得到 ex[i] 的值
            ex[i]=nxt[i-po];
        else//第二种情况，要继续匹配才能得到 ex[i] 的值
        {
            j=ex[po]+po-i;
            if(j<0)j=0;//如果 i>ex[po]+po 则要从头开始匹配
            while(i+j<len&&j<l2&&s1[j+i]==s2[j])//计算 ex[i]
                j++;
            ex[i]=j;
            po=i;//更新 po 的位置
        }
    }
}
```

AC 自动机

```
#include <bits/stdc++.h>
typedef unsigned long long ull;
const int P = 1e9+7;
const int maxn = 5e5 + 200;
const int inf = 0x3f3f3f3f;
using namespace std;
struct trie
{
```

```
    trie *nxt[26];
    trie *fail;
    int cnt;//根据题意修改
    int flag;//根据提议修改
    trie()
    {
        cnt = 1;
        flag = 0;
        fail = NULL;
        memset(nxt,NULL,sizeof(nxt));
    }
};
trie *root;
int T,N,Q;
char S[maxn],s1[maxn],s2[maxn];
//插入字符串，根据题意修改函数中的 cnt、flag
void Insert(char *s)
{
    trie *p = root;
    int len = strlen(s);
    for(int i = 0; i < len; i++)
    {
        int id = s[i] - 'a';
        if(p->nxt[id] != NULL)
        {
            p = p->nxt[id];
            p -> cnt++;
        }
        else
        {
            p -> nxt[id] = new trie;
            p = p -> nxt[id];
        }
    }
    p -> flag++;
}
//获取 fail 指针，一般不用动
void getFail()
{
    queue<trie*> q;
    q.push(root);
    trie *temp,*p;
    while(!q.empty())
    {
        p = q.front();
        q.pop();
```

```
        for(int i = 0; i < 26; ++i)
        {
            if(p -> nxt[i])
            {
                if(p == root)
                    p -> nxt[i] -> fail = root;
                else
                {
                    temp = p -> fail;
                    while(temp)
                    {
                        if(temp -> nxt[i])
                        {
                            p -> nxt[i] -> fail = temp -> nxt[i];
                            break;
                        }
                        temp = temp -> fail;
                    }
                    if(!temp) p -> nxt[i] -> fail = root;
                }
                q.push(p -> nxt[i]);
            }
        }
    }
}
int query(char* s)
{
    int i = 0,res = 0;
    trie *p = root;
    trie *temp;
    while(s[i])
    {
        int id = s[i] - 'a';
        while(!p -> nxt[id] && p != root) p = p -> fail;
        p = p -> nxt[id];
        if(p == NULL) p = root;
        temp = p;
        //这里是匹配和计算的过程，根据题意修改
        while(temp != root && temp -> flag != 0)
        {
            res += temp -> flag;
            temp -> flag = 0;
            temp = temp -> fail;
        }
        i++;
    }
```

```
    return res;
}
void Free(trie *p)
{
    for(int i = 0; i < 26; ++i)
    {
        if(p -> nxt[i] != NULL) Free(p->nxt[i]);
    }
    delete(p);
    p = NULL;
}
```

字典树

```
#include <bits/stdc++.h>
typedef unsigned long long ull;
const int maxn = 11;
const int inf = 0x3f3f3f3f;
using namespace std;
struct trie
{
    trie *nxt[26];
    int cnt;
    trie()
    {
        cnt = 1;
        memset(nxt,NULL,sizeof(nxt));
    }
};
trie *root;//记得在函数开始前 new trie
int i,id;
char S[maxn],s1[maxn];
//插入字符串
void Insert(char *s)
{
    trie *p = root;
    i = 0;
    while(s[i]){
        id = s[i] - 'a';
        if(p->nxt[id])
        {
            p = p->nxt[id];
            p -> cnt++;
        }
        else
        {
```

```
            p -> nxt[id] = new trie;
            p = p -> nxt[id];
        }
        i++;
    }
}
//查询字符串，功能可以自己改
int query(char* s)
{
    trie *p = root;
    i = 0;
    while(s[i])
    {
        id = s[i] - 'a';
        if(p -> nxt[id]) p = p -> nxt[id];
        else return 0;
        i++;
    }
    return p -> cnt;
}
//递归释放字典树
void Free(trie *p)
{
    for(i = 0; i < 26; ++i) if(p -> nxt[i] != NULL) Free(p->nxt[i]);
    delete(p);
    p = NULL;
}
```

字符串哈希

$hash[i] = (hash[i-1] * seed + str[i]) \mod M$

```
#include <iostream>
#include <string>
#include <cstring>
#include <unordered_set>
using namespace std;
const int mod=1e6+13;
unordered_set<long long>hash_table;
int eid=0,p[mod];
struct node
{
    long long next;
    string s;
}e[100005];
void insert(int u,string str)
{
```

```
16      e[eid].s=str;
17      e[eid].next=p[u];
18      p[u]=eid++;
19  }
20  string str;
21  unsigned int BKDRHash(string str)   //unsigned int 在溢出时会自动取模 2^32??
22  {
23      unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
24      unsigned int hash = 0;
25      for(int i=0;i<str.size();i++)
26          hash = (hash * seed + (str[i]))%mod;
27      return (hash & 0x7FFFFFFF)%mod;
28  }
29  // AP Hash Function
30  unsigned int APHash(string str)
31  {
32      unsigned int hash = 0;
33      int i;
34      for (i=0;i<str.size(); i++)
35      {
36          if ((i & 1) == 0)
37              hash ^= ((hash << 7) ^ (str[i]) ^ (hash >> 3));
38          else
39              hash ^= (~((hash << 11) ^ (str[i]) ^ (hash >> 5)));
40      }
41      return (hash & 0x7FFFFFFF);
42  }
43  int hash_in(string s)
44  {
45      long long u=BKDRHash(s);
46      for(int i=p[u];i!=-1;i=e[i].next)    //使用链式前向星模拟链表进行冲突判断
47      {
48          string v=e[i].s;
49          if(v==s) return 0;
50      }
51      insert(u,s);
52      return 1;
53  }
54  int main()
55  {
56      memset(p,-1,sizeof(p));
57      while(cin>>str)
58      {
59          cout<<str<<endl;
60          cout<<BKDRHash(str)<<endl;
61          cout<<APHash(str)<<endl;
```

```
62          cout<<hash_in(str)<<endl;
63      }
64      return 0;
65  }
```

数据结构

并查集

```
const int maxn = 200;
int father[maxn];
int Find(int a){
    if(father[a]==a) return a;
    return father[a]=Find(father[a]);
}
void Union(int a,int b){
    int f1=Find(a),f2=Find(b);
    father[f2] = f1;
}
void init(){
    for(int i = 1; i <= M; ++i) father[i]=i;
}
```

ST 表

离线区间最值

```
//d 为数据，mx[i][j] 表示 [i,i+2^j] 区间内最大值
int mx[100][100],d[100] = {0},n;
//查询 [l,r] 内最大值
int askmx(int l,int r) {
    int k = log2(r-l+1);
    return max(mx[l][k],mx[r-(1<<k)+1][k]);
}
//初始化，数据输入完后调用
void init(){
    for(int i = 0; i <= n; ++i) mx[i][0] = d[i];
    for(int j = 1; (1<<j) <= n+1; ++j)
        for(int i = 0; i + (1<<j) <= n+1; ++i)
            mx[i][j] = max(mx[i][j-1],mx[i+(1 << (j-1))][j-1]);
}
```

线段树

求和版本

```
#include <bits/stdc++.h>
typedef long long ll;
const int maxn = 2e5+300;
const int inf = 0x3f3f3f3f;
using namespace std;
int a[maxn],Sum[maxn<<2],lazy[maxn<<2];
//更新当前节点
```

```
8  void pushup(int& rt,int l,int r){
9      rt = l+r;
10 }
11 //下传函数
12 void pushdown(int l,int r,int rt){
13     //区间改值
14     if(lazy[rt]){
15         int m = (l+r) >> 1;
16         lazy[rt<<1] = lazy[rt];
17         lazy[rt<<1|1] = lazy[rt];
18         Sum[rt<<1] = lazy[rt] * (m-l+1);
19         Sum[rt<<1|1] = lazy[rt] * (r-m);
20         lazy[rt] = 0;
21     }
22     //区间增减
23     /*if(lazy[rt]){
24         int m = (l+r) >> 1;
25         lazy[rt<<1] += lazy[rt];
26         lazy[rt<<1|1] += lazy[rt];
27         Sum[rt<<1] += lazy[rt] * (m-l+1);
28         Sum[rt<<1|1] += lazy[rt] * (r-m);
29         lazy[rt] = 0;
30     }*/
31 }
32 //l: 当前节点的左端点 r: 当前节点的右端点 rt: 当前节点的编号
33 void build(int l,int r,int rt){
34     if(l == r){
35         Sum[rt] = a[l];
36         return;
37     }
38     int m = (l+r) >> 1;
39     build(l,m,rt<<1);
40     build(m+1,r,rt<<1|1);
41     pushup(Sum[rt],Sum[rt<<1],Sum[rt<<1|1]);
42 }
43 //l: 当前节点的左端点 r: 当前节点的右端点 rt: 当前节点的编号 [L,R] 查询的区间
44 int query(int L,int R,int l,int r,int rt){
45     if(L <= l && R >= r) return Sum[rt];
46     int m = (l+r) >> 1;
47     pushdown(l,r,rt);
48     if(R <= m) return query(L,R,l,m,rt<<1);
49     else if(L >= m+1) return query(L,R,m+1,r,rt<<1|1);
50     else{
51         int res = 0;
52         pushup(res,query(L,R,l,m,rt<<1),query(L,R,m+1,r,rt<<1|1));
53         return res;
```

```
    }
}
//l: 当前节点的左端点 r: 当前节点的右端点 rt: 当前节点的编号 将 L 的值改为 V
void update(int L,int V,int l,int r,int rt){
    if(l==r){Sum[rt]=V;return;}
    int m = (l+r) >> 1;
    pushdown(l,r,rt);
    if(L <= m) update(L,V,l,m,rt<<1);
    else update(L,V,m+1,r,rt<<1|1);
    pushup(Sum[rt],Sum[rt<<1],Sum[rt<<1|1]);
}
void segupdate(int L,int R,int l,int r,int rt,int lzy){
    if(L <= l && R >= r){
        //区间改值
        lazy[rt]=lzy;
        Sum[rt] = (r-l+1) * lzy;
        //区间加减
        /*lazy[rt]+=lzy;
        sum[rt]+=(r-l+1) * lzy;*/
        return;
    }
    int m = (l+r) >> 1;
    pushdown(l,r,rt);
    if(L <= m) segupdate(L,R,l,m,rt<<1,lzy);
    if(R > m) segupdate(L,R,m+1,r,rt<<1|1,lzy);
    pushup(Sum[rt],Sum[rt<<1],Sum[rt<<1|1]);
    return;
}
```

最大值

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 200000 + 5;
const int inf = 0x3f3f3f3f;
//mark 是原始数据
int mark[maxn],Max[maxn<<2];
//更新当前节点
void pushup(int rt){
    Max[rt] = max(Max[rt<<1],Max[rt<<1|1]);
}
//l: 当前节点的左端点 r: 当前节点的右端点 rt: 当前节点的编号
void build(int l,int r,int rt){
    if(l == r){Max[rt] = mark[l];return;}
    int m = (l+r) >> 1;
    build(l,m,rt<<1);
    build(m+1,r,rt<<1|1);
```

```
    pushup(rt);
}
//l: 当前节点的左端点 r: 当前节点的右端点 rt: 当前节点的编号 [L,R] 查询的区间
int query(int L,int R,int l,int r,int rt){
    if(L <= l && R >= r) return Max[rt];
    int m = (l+r) >> 1;
    int res = 0;
    if(L <= m) res = max(res,query(L,R,l,m,rt<<1));
    if(R > m) res = max(res,query(L,R,m+1,r,rt<<1|1));
    return res;
}
//l: 当前节点的左端点 r: 当前节点的右端点 rt: 当前节点的编号 将 L 的值改为 V
void update(int L,int V,int l,int r,int rt){
    if(l==r){Max[rt]=V;return;};
    int m = (l+r) >> 1;
    if(L <= m) update(L,V,l,m,rt<<1);
    else update(L,V,m+1,r,rt<<1|1);
    pushup(rt);
}
```

主席树

```
#include <iostream>
#include <map>
#include <set>
const int maxn = 1e9+100;
using namespace std;
struct node{
    node *lson;
    node *rson;
    int tcnt;
    bool valid;
    node(){
        lson = NULL;
        rson = NULL;
        tcnt = 0;
        valid = 1;
    }


};

node* root[200005];
int T,tot,cnt,n,m,a[200005],ID[200005];
map<int,int> M;
set<int> S;
```

```
25
26
27  inline void pushup(node *&rt){
28      rt -> tcnt = rt->lson->tcnt+rt->rson->tcnt;
29  }
30
31  void build(int l,int r,node *&rt){
32          rt = new node;
33          if(l==r) return;
34          int m = (l+r)>>1;
35          build(l,m,rt->lson);
36          build(m+1,r,rt->rson);
37      }
38
39  void add(int x,int l,int r,node *&rt,node *&lst){
40      rt = new node;
41      rt->tcnt=lst->tcnt+1;
42      if(l==r) return;
43      int m = (l+r)>>1;
44      if(x <= m){
45          rt->rson=lst->rson;
46          add(x,l,m,rt->lson,lst->lson);
47      }
48      else{
49          rt->lson=lst->lson;
50          add(x,m+1,r,rt->rson,lst->rson);
51      }
52  }
53
54  int findkth(int k,int l,int r,node *rt,node *lst){
55      if(l==r) return ID[l];
56      int m = (l+r) >> 1;
57      int s = rt->lson->tcnt-lst->lson->tcnt;
58      if(s>=k) return findkth(k,l,m,rt->lson,lst->lson);
59      else return findkth(k-s,m+1,r,rt->rson,lst->rson);
60  }
61
62
63  void Free(node *rt){
64      if(rt->lson && rt->lson->valid) Free(rt->lson);
65      if(rt->rson && rt->rson->valid) Free(rt->rson);
66      rt->valid=0;
67      delete(rt);
68      rt=NULL;
69  }
70
```

```cpp
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cnt = 0;tot = 0;
    cin >> n >> m;
    for(int i = 0; i < n; ++i){
        cin >> a[i];
        S.insert(a[i]);
    }
    for(set<int>::iterator it = S.begin(); it != S.end(); ++it){
        M[*it]=++cnt;
        ID[cnt] = *it;
    }
    build(1,cnt,root[tot]);
    for(int i = 0; i < n; ++i){
        add(M[a[i]],1,cnt,root[tot+1],root[tot]);
        tot++;
    }
    int l,r,k;
    while(m--){
        cin >> l >> r >> k;
        cout << findkth(k,1,cnt,root[r],root[l-1]) << endl;
    }
    return 0;
}
```

逆序对（归并排序）

```cpp
#include <bits/stdc++.h>
typedef long long ll;
const int maxn = 1e5 + 200;
using namespace std;
//acnt 为逆序对的数量
int a[maxn],b[maxn],acnt,i,j,cnt;

void Merge(int l,int m,int r){
    cnt = l,i=l,j=m+1;
    while(i  <= m && j <= r){
        if(a[i] <= a[j])
            b[cnt++] = a[i++];
        else{
            b[cnt++] = a[j++];
            //统计逆序对
            acnt += m-i+1;
        }
    }
```

```
19      while(i <= m) b[cnt++] = a[i++];
20      while(j <= r) b[cnt++] = a[j++];
21      for(int i = l; i <= r; i++) a[i]=b[i];
22  }
23  void Mergesort(int l,int r){
24      if(l < r - 1){
25          Mergesort(l,(l+r)>>1);
26          Mergesort(((l+r)>>1)+1,r);
27      }
28      Merge(l,(l+r)>>1,r);
29      return;
30  }
```

图论

最小生成树

Prim

```cpp
#include <bits/stdc++.h>
const int maxn = 200;
const int inf = 0x3f3f3f3f;
using namespace std;
int N,dis[maxn][maxn],vis[maxn],ans,mndis[maxn];
void prim(){
    //初始化，将 1 号点加入到生成树中
    ans = 0;
    int cnt = 1;
    vis[1] = 1;
    for(int i = 1;i <= N;++i) mndis[i]=dis[1][i];
    while(cnt != N){
        int mn = inf,id;
        //找出所有点中距离生成树最近的点
        for(int i = 1; i <= N; i++){
            if(!vis[i] && mndis[i] < mn){
                mn = mndis[i];
                id = i;
            }
        }
        //将找到的点加入生成树
        vis[id] = 1;
        ans += mn;
        cnt++;
        //更新剩余点到树的距离
        for(int i = 1; i <= N; i++) mndis[i] = min(mndis[i],dis[id][i]);

    }
}
```

Kruskal

```cpp
#include <bits/stdc++.h>
const int maxn = 200;
using namespace std;
struct edge{
    int from;
    int to;
    int val;
    edge(int a = 0,int b = 0,int c = 0){from=a;to=b;val=c;}
    friend bool operator > (edge a,edge b){
```

```
        return a.val > b.val;
    }
};
int N,M,father[maxn];
priority_queue< edge,vector<edge>,greater<edge> > Q;
int Find(int a){
    if(father[a]==a) return a;
    return father[a]=Find(father[a]);
}
void Union(int a,int b){
    int f1=Find(a),f2=Find(b);
    father[f2] = f1;
}
void init(){
    for(int i = 1; i <= M; ++i) father[i]=i;
    while(!Q.empty()) Q.pop();
}
void kruscal(){
    int ans = 0,cnt = 0;
    while(!Q.empty() && cnt != M-1){
        edge temp = Q.top();
        Q.pop();
        int f = temp.from,t = temp.to,v = temp.val;
        if(Find(t)!=Find(f)){
            Union(f,t);
            cnt++;
            ans+=v;
        }
    }
    //cnt 小于 M-1 则没有连通，否则 ans 为最小生成树大小
    if(cnt != M-1) puts("?");
    else printf("%d\n",ans);
}
```

最短路

dijkstra(单源最短路)

```
#include <iostream>
#include <queue>
#include <cstring>
#include <algorithm>
const int maxn = 1000+50;
const int inf = 0x3f3f3f3f;
using namespace std;
struct edge{
    int to;
```

```
10      int val;
11      edge(int a = 0,int b = 0){to=a;val=b;}
12  };
13  struct nod{
14      int pos;
15      int d;
16      nod(int a = 0,int b = 0){pos=a;d=b;}
17      //优先队列重载大于符号
18      friend bool operator > (nod a, nod b){
19          return a.d>b.d;
20      }
21  };
22  //邻接表
23  vector<edge> E[maxn];
24
25  int T,N,dis[maxn];//dis[i] 为从 X 到 i 的最短距离, 可以根据情况扩充为 d[i][j][k].....
26
27  //加边, 无向图时添加两条边
28  void add(int f,int t,int v){
29      E[f].push_back(edge(t,v));
30      E[t].push_back(edge(f,v));
31      return;
32  }
33
34  void dij(){
35      memset(dis,inf,sizeof(dis));
36      priority_queue< nod,vector<nod>,greater<nod> > Q;
37      dis[N]=0;
38      Q.push(nod(N,0));//初始态,N 为出发点
39      while(!Q.empty()){
40          nod temp = Q.top();
41          int pos = temp.pos;
42          int d = temp.d;
43          Q.pop();
44          if(d > dis[pos]) continue;
45          //遍历邻接表更新相邻点的最短距离
46          for(int i = 0; i < E[pos].size(); ++i){
47              int to = E[pos][i].to;
48              int val = E[pos][i].val;
49              int nd = d + val;
50              if(nd < dis[to]){
51                  dis[to] = nd;
52                  Q.push(nod(to,nd));
53              }
54          }
55      }
```

```
}
```

SPFA(单源最短路)

可以判负环

```
const int maxn = 1e5 + 5;
const int INF = 0x3f3f3f3f;
struct edge
{
    int to;
    int val;
    edge(int a = 0, int b = 0)
    {
        to = a; val = b;
    }
};
vector<edge> mp[maxn];  //邻接表存图
int d[maxn];            //最短距离
bool vis[maxn];         //记录节点是否在队列里
int EnqueNum[maxn];     //记录进队次数
int PointNum;           //节点数
bool spfa(int s)        //s 为起点
{
    memset(d,INF,sizeof(d));
    d[s] = 0;
    queue<int> que;     //记录路径改变的点
    que.push(s);
    vis[s] = true;
    EnqueNum[s]++;
    while(!que.empty())
    {
        int temp = que.front();
        que.pop();
        vis[temp] = false;
        for(int i = 0; i < mp[temp].size(); i++)
        {
            edge e = mp[temp][i];
            //如果可以松弛
            if(d[e.to] > d[temp] + e.val)
            {
                d[e.to] = d[temp] + e.val;
                //且该点不在队列里
                if(!vis[e.to])
                {
                    vis[e.to] = true;
                    que.push(e.to);
```

```
                    EnqueNum[e.to]++;
                    if(EnqueNum[e.to] >= PointNum)  //判负环
                        return false;
                }
            }
        }
    }
    return true;
}
```

floyd(多源最短路)

```
//注意 k,i,j!!!
for(int k=1;k<=n;k++)
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
```

匈牙利算法 (最大匹配)

最小点覆盖 = 最大匹配，最大独立集 = 点 - 最小点覆盖

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5;

char mp[55][55];
int R,C,ans,lnum,rnum,match[maxn],a[55][55],b[55][55];
bool vis[maxn];
vector<int> E[maxn];

bool dfs(int u){
    for(int i = 0; i < E[u].size(); ++i){
        int v = E[u][i];
        if(!vis[v]){
            vis[v]=1;
            if(match[v]==-1 || dfs(match[v])){
                match[v]=u;
                return 1;
            }
        }
    }
    return 0;
}

int xyl(){
    memset(match,-1,sizeof(match));
```

```
    ans = 0;
    for(int i = 0; i < lnum; ++i){
        memset(vis,0,sizeof(vis));
        ans += dfs(i);
    }
    return ans;
}
```

KM 算法 (带权最大匹配)

三份板子不知道哪份是正确实现

```
//不确定对不对的 O(n^3)
const int AX = 3e2+6;
bool visx[AX];
bool visy[AX];
int w[AX][AX];
int lx[AX] , ly[AX];  //可行性顶标
int linker[AX];  //记录匹配的边
int slack[AX];   //记录每个 j 相连的 i 的最小的 lx[i]+ly[j]-w[i][j]
int n ;
bool dfs( int x ){
    visx[x] = true;
    for( int y = 1 ; y <= n ; y ++ ){
        if( !visy[y] && lx[x] + ly[y] == w[x][y] ){
            visy[y] = true;
            if( linker[y] == -1 || dfs( linker[y] ) ){
                linker[y] = x ;
                return true;
            }
        }else if( slack[y] > lx[x] + ly[y] - w[x][y] ){//x,y 不在相等子图且 y 不在增广路
            slack[y] = lx[x] + ly[y] - w[x][y];
        }
    }
    return false;
}

void KM(){
    memset( linker , -1 , sizeof(linker) );
    memset( ly , 0 , sizeof(ly) );
    for( int i = 1 ; i <= n ; i++ ){
        lx[i] = -INF;
        for( int j = 1 ; j <= n ; j++ ){
            if( lx[i] < w[i][j] ) lx[i] = w[i][j];
        }
    }
    for( int x = 1 ; x <= n ; x++ ){
```

```
        for( int i = 1 ; i <= n ; i++ ) slack[i] = INF;//每次匹配 x 都要更新 slack
        while(1){
            memset( visx , false , sizeof(visx) );
            memset( visy , false , sizeof(visy) );
            if( dfs(x) ){
                break;
            }else{ // 匹配失败后 x 一定在增广路，寻找不在增广路的 j
                int delta = INF;
                for( int j = 1 ; j <= n ; j++ ){
                    if( !visy[j] && delta > slack[j] ){
                        delta = slack[j];
                    }
                }

                for( int i = 1 ; i <= n ; i++ ){
                    if( visx[i] ) lx[i] -= delta;
                }
                for( int i = 1 ; i <= n ; i++ ){
                    if( visy[i] ) ly[i] += delta;
                    else slack[i] -= delta;
                    //修改顶标后，要把所有的 slack 值都减去 delta
                     //slack[j] = min(lx[i] + ly[j] -w[i][j])
                     //在增广路的 lx[i] 减少，所以不在增广路的 slack[j] 减小
                }
            }
        }
    }
}
```

```
void bfs(int k)
{
    int px,py=0,yy=0,d;
    memset(pre,0,sizeof pre);
    memset(slack,0x3f,sizeof slack);
    linky[py]=k;

    do
    {
        px=linky[py],d=0x3f3f3f3f,vis[py]=1;
        for(int i=1;i<=n;i++)
            if(vis[i]==0)
            {
                if(slack[i]>dbx[px]+dby[i]-C[px][i])
                    slack[i]=dbx[px]+dby[i]-C[px][i],pre[i]=py;
                if(slack[i]<d) d=slack[i],yy=i;
            }
        for(int i=0;i<=n;i++)
```

```
            if(vis[i]) dbx[linky[i]]-=d,dby[i]+=d;
            else slack[i]-=d;
        py=yy;
    }while(linky[py]!=0);
    while(py) linky[py]=linky[pre[py]],py=pre[py];
}
void km()
{
    memset(dbx,0,sizeof dbx);
    memset(dby,0,sizeof dby);
    memset(linky,0,sizeof linky);
    for(int i=1;i<=n;i++)
        memset(vis,0,sizeof vis),bfs(i);
}
```

```
const int maxn = 300;
const int inf = 0x3f3f3f3f;

int n;
bool vis_l[maxn],vis_r[maxn];
int match[maxn],slack[maxn],ex_l[maxn],ex_r[maxn];
int E[maxn][maxn];


bool dfs(int u){
    vis_l[u]=1;
    for(int v = 0; v < n; ++v){
        int w = E[u][v];
        if(!vis_r[v] && ex_l[u]+ex_r[v]==w){
            vis_r[v]=1;
            if(match[v]==-1||dfs(match[v])){
                match[v]=u;
                return 1;
            }
        }
        else slack[v] = min(slack[v],ex_l[u]+ex_r[v]-w);
    }
    return 0;
}

int km(){
    memset(match,-1,sizeof(match));
    memset(ex_r,0,sizeof(ex_r));
    for(int i = 0; i < n; ++i){
        ex_l[i] = -inf;
        for(int j = 0; j < n; ++j)
            ex_l[i]=max(ex_l[i],E[i][j]);
```

```
    }
    for(int i = 0; i < n; ++i){
        memset(slack,inf,sizeof(slack));
        while(true){
            memset(vis_l,0,sizeof(vis_l));
            memset(vis_r,0,sizeof(vis_r));
            if(dfs(i)) break;
            else{
                int d = inf;
                for(int j = 0; j < n; ++j)
                    if(!vis_r[j]) d = min(d,slack[j]);
                for(int j = 0; j < n; ++j){
                    if(vis_l[j]) ex_l[j]-=d;
                    if(vis_r[j]) ex_r[j]+=d;
                    else slack[j] -= d;
                }
            }
        }
    }
    int res = 0;
    for(int i = 0; i < n; ++i)
        if(match[i] > -1)
            res += E[match[i]][i];
    return res;
}
```

网络流

DINIC

```
const int maxn = 300;
const int inf = 0x3f3f3f3f;

int n;
bool vis_l[maxn],vis_r[maxn];
int match[maxn],slack[maxn],ex_l[maxn],ex_r[maxn];
int E[maxn][maxn];


bool dfs(int u){
    vis_l[u]=1;
    for(int v = 0; v < n; ++v){
        int w = E[u][v];
        if(!vis_r[v] && ex_l[u]+ex_r[v]==w){
            vis_r[v]=1;
            if(match[v]==-1||dfs(match[v])){
                match[v]=u;
```

```
                return 1;
            }
        }
        else slack[v] = min(slack[v],ex_l[u]+ex_r[v]-w);
    }
    return 0;
}

int km(){
    memset(match,-1,sizeof(match));
    memset(ex_r,0,sizeof(ex_r));
    for(int i = 0; i < n; ++i){
        ex_l[i] = -inf;
        for(int j = 0; j < n; ++j)
            ex_l[i]=max(ex_l[i],E[i][j]);
    }
    for(int i = 0; i < n; ++i){
        memset(slack,inf,sizeof(slack));
        while(true){
            memset(vis_l,0,sizeof(vis_l));
            memset(vis_r,0,sizeof(vis_r));
            if(dfs(i)) break;
            else{
                int d = inf;
                for(int j = 0; j < n; ++j)
                    if(!vis_r[j]) d = min(d,slack[j]);
                for(int j = 0; j < n; ++j){
                    if(vis_l[j]) ex_l[j]-=d;
                    if(vis_r[j]) ex_r[j]+=d;
                    else slack[j] -= d;
                }
            }
        }
    }
    int res = 0;
    for(int i = 0; i < n; ++i)
        if(match[i] > -1)
            res += E[match[i]][i];
    return res;
}
```

费用流

```
const int maxn = 1e3 + 7;
const int INF = 0x3f3f3f3f;
struct Edge
```

```
{
    int from, to, cap, flow, cost;
};
struct MCMF
{
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[maxn];
    int inq[maxn];
    int d[maxn]; //最短路数组
    int p[maxn]; //记录路径
    int a[maxn]; //记录流量
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++)
            G[i].clear();
        edges.clear();
    }
    void addedge(int from, int to, int cap, int cost)
    {
        edges.push_back(Edge{from, to, cap, 0, cost});
        edges.push_back(Edge{to, from, 0, 0, -cost});
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    bool spfa(int s, int t, int &flow, int &cost)
    {
        for (int i = 0; i < n; i++)
            d[i] = INF;
        memset(inq, 0, sizeof(inq));
        d[s] = 0;
        inq[s] = 1;
        p[s] = 0;
        a[s] = INF;
        queue<int> q;
        q.push(s);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (int i = 0; i < G[u].size(); i++)
            {
                Edge &e = edges[G[u][i]];
```

```
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost)
                {
                    d[e.to] = d[u] + e.cost;             //松弛
                    p[e.to] = G[u][i];                   //记录上一个点
                    a[e.to] = min(a[u], e.cap - e.flow); //流量控制
                    if (!inq[e.to])
                    {
                        q.push(e.to);
                        inq[e.to] = 1;
                    }
                }
            }
        }
        if (d[t] == INF)
            return false; //不存在最短路
        flow += a[t];
        cost += d[t] * a[t];
        int u = t;
        while (u != s)
        {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
            u = edges[p[u]].from;
        }
        return true;
    }
    int Mincost(int s, int t)
    {
        int flow = 0, cost = 0;
        while (spfa(s, t, flow, cost))
            ;
        return cost;
    }
};
```

计算几何

几何板子以后再补先补点骚操作

模拟退火

```
struct Point{
    double x,y,z;
};
double dist(Point a,Point b){
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+(a.z-b.z)*(a.z-b.z));
}
double solve(int n){
    double ans=1e30;
    Point tmp;
    tmp.x=tmp.y=tmp.z=0;;
    int s=1;
    double step=100;
    while(step>eps){
        for(int i=1;i<=n;i++){
            if(dist(tmp,p[s])<dist(tmp,p[i])) s=i;
        }
        double Dist=dist(tmp,p[s]);
        ans=min(ans,Dist);
        tmp.x+=(p[s].x-tmp.x)/Dist*step;
        tmp.y+=(p[s].y-tmp.y)/Dist*step;
        tmp.z+=(p[s].z-tmp.z)/Dist*step;
        step*=0.98;
    }
    return ans;
}
```

三分

解决凹凸函数的最值问题

```
while (r-l>eps)
    {
        double lmid=l+(r-l)/3.0,rmid=r-(r-l)/3.0;
        if (f(lmid)<f(rmid)) l=lmid; else r=rmid;
    }
```