Contents

1 math

1.1 公式

1. Most Divisor Number

Range	最多因數數	因數個數
10^9	735134400	1344
2^{31}	2095133040	1600
10^{18}	897612484786617600	103680
2^{64}	9200527969062830400	161280

2. Catlan Number

$$C_n = \frac{1}{n}\binom{2n}{n}, C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \ldots$$

3. Faulhaber's formula

$$\sum_{k=1}^{n} k^p = \frac{1}{p+1}\sum_{r=0}^{p}\binom{p+1}{r}B_r n^{p-r+1}$$

where $B_0 = 1,\ B_r = 1 - \sum_{i=0}^{r-1}\binom{r}{i}\frac{B_i}{r-i+1}$

也可用高斯消去法找 $deg(p+1)$ 的多項式，例：

$$\sum_{k=1}^{n} k^2 = a_3 n^3 + a_2 n^2 + a_1 n + a_0$$

$$\begin{bmatrix} 0^3 & 0^2 & 0^1 & 0^0 \\ 1^3 & 1^2 & 1^1 & 1^0 \\ 2^3 & 2^2 & 2^1 & 2^0 \\ 3^3 & 3^2 & 3^1 & 3^0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0^2 \\ 0^2+1^2 \\ 0^2+1^2+2^2 \\ 0^2+1^2+2^2+3^2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 & 5 \\ 27 & 9 & 3 & 1 & 14 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 4 & 6 & 7 & 3 \\ 0 & 0 & 6 & 11 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \\ 0 \end{bmatrix},\ \sum_{k=1}^{n} k^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

4. SG Function

$SG(x) = mex\{SG(y) | x \to y\}$

$mex(S) = min\{n | n \in \mathbb{N}, n \notin S\}$

5. Fibonacci

$$\begin{bmatrix} f_{n-1} & f_n \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} f_n & f_{n+1} \end{bmatrix}$$

$$\begin{bmatrix} f_n & f_{n+1} \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^p = \begin{bmatrix} f_{n+p} & f_{n+p+1} \end{bmatrix}, p \in \mathbb{N}$$

6. Pick's Theorem

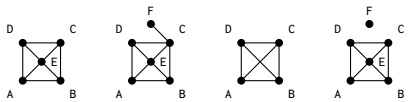
給定頂點座標均是整點（或正方形格子點）的簡單多邊形，其面積 A 和內部格點數目 i、邊上格點數目 b 的關係為

$$A = i + \frac{b}{2} - 1$$

7. Euler's Formula

對於有 V 個點、E 條邊、F 個面（含外部）的連通平面圖

$$F + V - E = 2$$



(1)、(2)○；(3)×，$\overline{AC}$ 與 $\overline{BD}$ 相交；(4)×，非連通圖

1.2 矩陣快速冪

```cpp
using ll = long long;
using mat = vector<vector<ll>>;
const int mod = 1e9 + 7;

mat operator*(mat A, mat B) {
  mat res(A.size(), vector<ll>(B[0].size()));
  for(int i=0; i<A.size(); i++) {
    for(int j=0; j<B[0].size(); j++) {
      for(int k=0; k<B.size(); k++) {
        res[i][j] += A[i][k] * B[k][j] % mod;
        res[i][j] %= mod;
      }
    }
  }
  return res;
}

mat I = ;
// compute matrix M^n
// 需先 init I 矩陣
mat mpow(mat& M, int n) {
  if(n <= 1) return n ? M : I;
  mat v = mpow(M, n>>1);
  return (n & 1) ? v*v*M : v*v;
}

// 迴圈版本
mat mpow(mat M, int n) {
  mat res(M.size(), vector<ll>(M[0].size()));
  for(int i=0; i<res.size(); i++)
    res[i][i] = 1;
  for(; n; n>>=1) {
    if(n & 1) res = res * M;
    M = M * M;
  }
  return res;
}
```

1.3 質數與因數

```cpp
歐拉篩O(n)
#define MAXN 47000 //sqrt(2^31)=46,340...
bool isPrime[MAXN];
int p[MAXN];
int pSize=0;
void getPrimes(){
  memset(isPrime,true,sizeof(isPrime));
  isPrime[0]=isPrime[1]=false;
  for(int i=2;i<MAXN;i++){
    if(isPrime[i]) p[pSize++]=i;
    for(int j=0;j<pSize&&i*p[j]<=MAXN;++j){
      isPrime[i*p[j]]=false;
      if(i%p[j]==0) break;
    }
  }
}

最大公因數 O(log(min(a,b)))
int GCD(int a, int b){
  if(b == 0) return a;
  return GCD(b, a%b);
}

質因數分解
void primeFactorization(int n){
  for(int i=0; i<p.size(); ++i) {
    if(p[i]*p[i] > n) break;
    if(n % p[i]) continue;
    cout << p[i] << ' ';
    while(n%p[i] == 0) n /= p[i];
  }
  if(n != 1) cout << n << ' ';
  cout << '\n';
}
```

```

擴展歐幾里得算法 ax + by = GCD(a, b)
int ext_euc(int a, int b, int &x, int &y) {
  if(b == 0){
    x = 1, y = 0;
    return a;
  }
  int d = ext_euc(b, a%b, y, x);
  y -= a/b*x;
  return d;
}
int main(){
  int a, b, x, y;
  cin >> a >> b;
  ext_euc(a, b, x, y);
  cout << x << ' ' << y << endl;
  return 0;
}



歌德巴赫猜想
解: 把偶數 N (6≤N≤10^6) 寫成兩個質數的和。
#define N 20000000
int ox[N], p[N], pr;
void PrimeTable(){
  ox[0] = ox[1] = 1;
  pr = 0;
  for(int i=2;i<N;i++){
    if(!ox[i]) p[pr++] = i;
    for(int j=0; i*p[j]<N&&j<pr; j++)
      ox[i*p[j]] = 1;
  }
}
int main(){
  PrimeTable();
  int n;
  while(cin>>n, n){
    int x;
    for(x=1;; x+=2)
      if(!ox[x] && !ox[n-x]) break;
    printf("%d = %d + %d\n", n, x, n-x);
  }
}

problem :
給定整數 N，求N最少可以拆成多少個質數的和。
如果N是質數，則答案為 1。
如果N是偶數(N!=2)，則答案為2(強歌德巴赫猜想)。
如果N是奇數且N-2是質數，則答案為2(2+質數)。
其他狀況答案為 3 (弱歌德巴赫猜想)。

bool isPrime(int n){
  for(int i=2;i<n;++i){
    if(i*i>n) return true;
    if(n%i==0) return false;
  }
  return true;
}
int main(){
  int n;
  cin>>n;
  if(isPrime(n)) cout<<"1\n";
  else if(n%2==0||isPrime(n-2)) cout<<"2\n";
  else cout<<"3\n";
}
```

1.4 歐拉函數

```
//計算閉區間 [1,n] 中有幾個正整數與 n 互質

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

1.5 乘法逆元、組合數

$$
\begin{aligned}
& x^{-1} mod\ m \\
&= \begin{cases} 1, & \text{if } x=1 \\ -\lfloor \frac{m}{x} \rfloor (m\ mod\ x)^{-1}, & \text{otherwise} \end{cases} \quad (mod\ m) \\
&= \begin{cases} 1, & \text{if } x=1 \\ (m-\lfloor \frac{m}{x} \rfloor)(m\ mod\ x)^{-1}, & \text{otherwise} \end{cases} \quad (mod\ m)
\end{aligned}
$$

若 $p \in prime$, 根據費馬小定理, 則

$$
\begin{aligned}
\because \quad ax &\equiv 1 \ (mod\ p) \\
\therefore \quad ax &\equiv a^{p-1} \ (mod\ p) \\
\therefore \quad x &\equiv a^{p-2} \ (mod\ p)
\end{aligned}
$$

```
using ll = long long;
const int maxn = 2e5 + 10;
const int mod = 1e9 + 7;

int fact[maxn] = {1, 1}; // x! % mod
int inv[maxn] = {1, 1};  // x^(-1) % mod
int invFact[maxn] = {1, 1};// (x!)^(-1) % mod

void build() {
  for(int x=2; x<maxn; x++) {
    fact[x] = (ll)x * fact[x-1] % mod;
    inv[x] = (ll)(mod-mod/x)*inv[mod%x]%mod;
    invFact[x] = (ll)invFact[x-1]*inv[x]%mod;
  }
}

// 前提：mod 為質數
void build() {
  auto qpow = [&](ll a, int b) {
    ll res = 1;
    for(; b; b>>=1) {
      if(b & 1) res = res * a % mod;
      a = a * a % mod;
    }
    return res;
  };

  for(int x=2; x<maxn; x++) {
    fact[x] = (ll)x * fact[x-1] % mod;
    invFact[x] = qpow(fact[x], mod-2);
  }
}

// C(a, b) % mod
int comb(int a, int b) {
  if(a < b) return 0;
  ll x = fact[a];
  ll y = (ll)invFact[b] * invFact[a-b] % mod;
  return x * y % mod;
}
```

1.6 大步小步

```
  題意
給定 B,N,P，求出 L 滿足 B^L N(mod P)。
 題解
餘數的循環節長度必定為 P 的因數，因此
     B^0 B^P,B^1 B^(P+1),…，
也就是說如果有解則 L<N，枚舉0,1,2,L-1
     能得到結果，但會超時。
將 L 拆成 mx+y，只要分別枚舉 x,y 就能得到答案，
設 m=√P 能保證最多枚舉 2√P 次 。
B^(mx+y) N(mod P)
B^(mx)B^y N(mod P)
B^y N(B^(-m))^x (mod P)
先求出 B^0,B^1,B^2,...,B^(m-1)，
再枚舉 N(B^(-m)),N(B^(-m))^2,… 查看是否有對應的
     B^y。
這種算法稱為大步小步演算法，
大步指的是枚舉 x (一次跨 m 步)，
小步指的是枚舉 y (一次跨 1 步)。
  複雜度分析
利用 map/unorder_map 存放
     B^0,B^1,B^2,…,B^(m-1)，
枚舉 x 查詢 map/unorder_map 是否有對應的 B^y，
存放和查詢最多 2√P 次，時間複雜度為
     O(√Plog√P)/O(√P)。

using LL = long long;
LL B, N, P;
LL fpow(LL a,LL b,LL c){
    LL res=1;
    for(;b;b >>=1){
        if(b&1)
            res=(res*a)%c;
        a=(a*a)%c;
    }
    return res;
}
LL BSGS(LL a,LL b,LL p){
    a%=p,b%=p;
    if(a==0)
        return b==0?1:-1;
    if(b==1)
        return 0;
    map<LL, LL> tb;
    LL sq=ceil(sqrt(p-1));
    LL inv=fpow(a,p-sq-1,p);
    tb[1]=sq;
    for(LL i=1,tmp=1;i<sq;++i){
        tmp=(tmp*a)%p;
        if(!tb.count(tmp))
            tb[tmp]=i;
    }
    for(LL i=0;i<sq;++i){
        if(tb.count(b)){
            LL res=tb[b];
            return i*sq+(res==sq?0:res);
        }
        b=(b*inv)%p;
    }
    return -1;
}
int main(){
    IOS;   //輸入優化
    while(cin>>P>>B>>N){
        LL ans=BSGS(B,N,P);
        if(ans==-1)
            cout<<"no solution\n";
        else
            cout<<ans<<'\n';
    }
}
```

2 字串

2.1 最長迴文子字串

```cpp
#include<bits/stdc++.h>
#define T(x) ((x)%2 ? s[(x)/2] : '.')
using namespace std;

string s;
int n;

int ex(int l,int r){
  int i=0;
  while(l-i>=0&&r+i<n&&T(l-i)==T(r+i)) i++;
  return i;
}

int main(){
  cin>>s;
  n=2*s.size()+1;
  int mx=0;
  int center=0;
  vector<int> r(n);
  int ans=1;
  r[0]=1;
  for(int i=1;i<n;i++){
    int ii=center-(i-center);
    int len=mx-i+1;
    if(i>mx){
      r[i]=ex(i,i);
      center=i;
      mx=i+r[i]-1;
    }
    else if(r[ii]==len){
      r[i]=len+ex(i-len,i+len);
      center=i;
      mx=i+r[i]-1;
    }
    else r[i]=min(r[ii],len);
    ans=max(ans,r[i]);
  }
  cout<<ans-1<<"\n";
  return 0;
}
```

2.2 KMP

```cpp
const int maxn = 1e6 + 10;

int n, m;             // len(a), len(b)
int f[maxn];          // failure function
char a[maxn], b[maxn];

void failureFuntion() { // f[0] = 0
    for(int i=1, j=0; i<m; ) {
        if(b[i] == b[j]) f[i++] = ++j;
        else if(j) j = f[j-1];
        else f[i++] = 0;
    }
}

int kmp() {
    int i = 0, j = 0, res = 0;
    while(i < n) {
        if(a[i] == b[j]) i++, j++;
        else if(j) j = f[j-1];
        else i++;
        if(j == m) {
            res++; // 找到答案
            j = 0; // non-overlapping
        }
    }
    return res;
}

// Problem: 所有在b裡，前後綴相同的長度
// b = ababcababababcabab
// f = 0012012345123456789
// 前9 = 後9
// 前4 = 前9的後4 = 後4
// 前2 = 前4的後2 = 前9的後2 = 後2
for(int j=m; j; j=f[j-1]) {
    // j 是答案
}
```

2.3 Z Algorithm

```cpp
const int maxn = 1e6 + 10;

int z[maxn];   // s[0:z[i]) = s[i:i+z[i])
string s;

void makeZ() {  // z[0] = 0
  for(int i=1, l=0, r=0; i<s.length(); i++) {
    if(i<=r && z[i-l]<r-i+1) z[i] = z[i-l];
    else {
      z[i] = max(0, r-i+1);
      while(i+z[i]<s.length() &&
            s[z[i]]==s[i+z[i]]) z[i]++;
    }
    if(i+z[i]-1 > r) l = i, r = i+z[i]-1;
  }
}
```

3 algorithm

3.1 三分搜

```cpp
  題意
給定兩射線方向和速度，問兩射線最近距離。
  題解
假設 F(t) 為兩射線在時間 t 的距離，F(t)
     為二次函數，
可用三分搜找二次函數最小值。
struct Point{
    double x, y, z;
    Point() {}
    Point(double _x,double _y,double _z):
        x(_x),y(_y),z(_z){}
    friend istream& operator>>(istream& is,
         Point& p) {
        is >> p.x >> p.y >> p.z;
        return is;
    }
    Point operator+(const Point &rhs) const{
        return Point(x+rhs.x,y+rhs.y,z+rhs.z);
    }
    Point operator-(const Point &rhs) const{
        return Point(x-rhs.x,y-rhs.y,z-rhs.z);
    }
    Point operator*(const double &d) const{
        return Point(x*d,y*d,z*d);
    }
    Point operator/(const double &d) const{
        return Point(x/d,y/d,z/d);
    }
    double dist(const Point &rhs) const{
        double res = 0;
        res+=(x-rhs.x)*(x-rhs.x);
        res+=(y-rhs.y)*(y-rhs.y);
        res+=(z-rhs.z)*(z-rhs.z);
        return res;
    }
};
int main(){
    IOS;    //輸入優化
    int T;
    cin>>T;
    for(int ti=1;ti<=T;++ti){
        double time;
        Point x1,y1,d1,x2,y2,d2;
        cin>>time>>x1>>y1>>x2>>y2;
        d1=(y1-x1)/time;
        d2=(y2-x2)/time;
        double L=0,R=1e8,m1,m2,f1,f2;
        double ans = x1.dist(x2);
        while(abs(L-R)>1e-10){
            m1=(L+R)/2;
            m2=(m1+R)/2;
            f1=((d1*m1)+x1).dist((d2*m1)+x2);
            f2=((d1*m2)+x1).dist((d2*m2)+x2);
            ans = min(ans,min(f1,f2));
            if(f1<f2) R=m2;
            else L=m1;
        }
        cout<<"Case "<<ti<<": ";
        cout << fixed << setprecision(4) <<
             sqrt(ans) << '\n';
    }
}
```

3.2 差分

```cpp
用途：在區間 [l, r] 加上一個數字v。
b[l] += v; (b[0~l] 加上v)
b[r+1] -= v; (b[r+1~n] 減去v (b[r] 仍保留v) )
給的 a[] 是前綴和數列，建構 b[]，
因為 a[i] = b[0] + b[1] + b[2] + … + b[i]，
所以 b[i] = a[i] - a[i-1]。
```

```
在 b[l] 加上 v，b[r+1] 減去 v，
最後再從 0 跑到 n 使 b[i] += b[i-1]。
這樣一來，b[] 是一個在某區間加上v的前綴和。
int a[1000], b[1000];
// a: 前綴和數列, b: 差分數列
int main(){
    int n, l, r, v;
    cin >> n;
    for(int i=1; i<=n; i++){
        cin >> a[i];
        b[i] = a[i] - a[i-1]; //建構差分數列
    }
    cin >> l >> r >> v;
    b[l] += v;
    b[r+1] -= v;
    for(int i=1; i<=n; i++){
        b[i] += b[i-1];
        cout << b[i] << ' ';
    }
}
```

3.3 greedy

```
刪數字問題
//problem
給定一個數字 N(≤10^100)，需要刪除 K 個數字，
請問刪除 K 個數字後最小的數字為何?
//solution
刪除滿足第 i 位數大於第 i+1 位數的最左邊第 i 位數，
扣除高位數的影響較扣除低位數的大。
//code
int main(){
    string s;
    int k;
    cin>>s>>k;
    for(int i=0;i<k;++i){
        if((int)s.size()==0) break;
        int pos =(int)s.size()-1;
        for(int j=0;j<(int)s.size()-1;++j){
            if(s[j]>s[j+1]){
                pos=j;
                break;
            }
        }
        s.erase(pos,1);
    }
    while((int)s.size()>0&&s[0]=='0')
        s.erase(0,1);
    if((int)s.size()) cout<<s<<'\n';
    else cout<<0<<'\n';
}
最小區間覆蓋長度
//problem
給定 n 條線段區間為 [Li,Ri]，
請問最少要選幾個區間才能完全覆蓋 [0,S]?
//solution
先將所有區間依照左界由小到大排序，
對於當前區間 [Li,Ri]，要從左界 >Ri 的所有區間中，
找到有著最大的右界的區間，連接當前區間。

//problem
長度 n 的直線中有數個加熱器，
在 x 的加熱器可以讓 [x-r,x+r] 內的物品加熱，
問最少要幾個加熱器可以把 [0,n] 的範圍加熱。
//solution
對於最左邊沒加熱的點a，選擇最遠可以加熱a的加熱器，
更新已加熱範圍，重複上述動作繼續尋找加熱器。
//code
int main(){
    int n, r;
    int a[1005];
    cin>>n>>r;
    for(int i=1;i<=n;++i) cin>>a[i];
    int i=1,ans=0;
    while(i<=n){
        int R=min(i+r-1,n),L=max(i-r+1,0)
        int nextR=-1;
        for(int j=R;j>=L;--j){
            if(a[j]){
                nextR=j;
                break;
            }
        }
        if(nextR==-1){
            ans=-1;
            break;
        }
        ++ans;
        i=nextR+r;
    }
    cout<<ans<<'\n';
}
最多不重疊區間
//problem
給你 n 條線段區間為 [Li,Ri]，
請問最多可以選擇幾條不重疊的線段(頭尾可相連)?
//solution
依照右界由小到大排序，
每次取到一個不重疊的線段，答案 +1。
//code
struct Line{
    int L,R;
    bool operator<(const Line &rhs)const{
        return R<rhs.R;
    }
};
int main(){
    int t;
    cin>>t;
    Line a[30];
    while(t--){
        int n=0;
        while(cin>>a[n].L>>a[n].R,a[n].L||a[n].R)
            ++n;
        sort(a,a+n);
        int ans=1,R=a[0].R;
        for(int i=1;i<n;i++){
            if(a[i].L>=R){
                ++ans;
                R=a[i].R;
            }
        }
        cout<<ans<<'\n';
    }
}
最小化最大延遲問題
//problem
給定 N 項工作，每項工作的需要處理時長為 Ti，
期限是 Di，第 i 項工作延遲的時間為
    Li=max(0,Fi-Di)，
原本Fi 為第 i 項工作的完成時間，
求一種工作排序使 maxLi 最小。
//solution
按照到期時間從早到晚處理。
//code
struct Work{
    int t, d;
    bool operator<(const Work &rhs)const{
        return d<rhs.d;
    }
};
int main(){
    int n;
    Work a[10000];
    cin>>n;
    for(int i=0;i<n;++i)
        cin>>a[i].t>>a[i].d;
    sort(a,a+n);
    int maxL=0,sumT=0;
    for(int i=0;i<n;++i){
        sumT+=a[i].t;
        maxL=max(maxL,sumT-a[i].d);
    }
    cout<<maxL<<'\n';
}
最少延遲數量問題
//problem
給定 N 個工作，每個工作的需要處理時長為 Ti，
期限是 Di，求一種工作排序使得逾期工作數量最小。
//solution
期限越早到期的工作越先做。
將工作依照到期時間從早到晚排序，
依序放入工作列表中，如果發現有工作預期，
就從目前選擇的工作中，移除耗時最長的工作。
上述方法為 Moore-Hodgson s Algorithm。

//problem
給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜?
//solution
和最少延遲數量問題是相同的問題，只要將題敘做轉換。
工作處裡時長 → 烏龜重量
工作期限 → 烏龜可承受重量
多少工作不延期 → 可以疊幾隻烏龜
//code
struct Work{
    int t, d;
    bool operator<(const Work &rhs)const{
        return d<rhs.d;
        }
};
int main(){
    int n=0;
    Work a[10000];
    priority_queue<int> pq;
    while(cin>>a[n].t>>a[n].d)
        ++n;
    sort(a,a+n);
    int sumT=0,ans=n;
    for(int i=0;i<n;++i){
        pq.push(a[i].t);
        sumT+=a[i].t;
        if(a[i].d<sumT){
            int x=pq.top();
            pq.pop();
            sumT-=x;
            --ans;
        }
    }
    cout<<ans<<'\n';
}

任務調度問題
//problem
給定 N 項工作，每項工作的需要處理時長為 Ti，
期限是 Di，如果第 i 項工作延遲需要受到 pi
    單位懲罰，
請問最少會受到多少單位懲罰。
//solution
依照懲罰由大到小排序，
每項工作依序嘗試可不可以放在
    Di-Ti+1,Di-Ti,...,1,0，
如果有空閒就放進去，否則延後執行。

//problem
給定 N 項工作，每項工作的需要處理時長為 Ti，
期限是 Di，如果第 i 項工作在期限內完成會獲得 ai
    單位獎勵，
請問最多會獲得多少單位獎勵。
//solution
和上題相似，這題變成依照獎勵由大到小排序。
//code
struct Work{
    int d,p;
    bool operator<(const Work &rhs)const{
        return p>rhs.p;
        }
};
int main(){
    int n;
    Work a[100005];
    bitset<100005> ok;
    while(cin>>n){
```

```
        ok.reset();
        for(int i=0;i<n;++i)
            cin>>a[i].d>>a[i].p;
        sort(a,a+n);
        int ans=0;
        for(int i=0;i<n;++i){
            int j=a[i].d;
            while(j--)
                if(!ok[j]){
                    ans+=a[i].p;
                    ok[j]=true;
                    break;
                }
        }
        cout<<ans<<'\n';
    }
}
```

3.4 dinic

```
const int maxn = 1e5 + 10;
const int inf = 0x3f3f3f3f;
struct Edge {
    int s, t, cap, flow;
};
int n, m, S, T;
int level[maxn], dfs_idx[maxn];
vector<Edge> E;
vector<vector<int>> G;
void init() {
    S = 0;
    T = n + m;
    E.clear();
    G.assign(maxn, vector<int>());
}
void addEdge(int s, int t, int cap) {
    E.push_back({s, t, cap, 0});
    E.push_back({t, s, 0, 0});
    G[s].push_back(E.size()-2);
    G[t].push_back(E.size()-1);
}
bool bfs() {
    queue<int> q({S});
    memset(level, -1, sizeof(level));
    level[S] = 0;
    while(!q.empty()) {
        int cur = q.front();
        q.pop();
        for(int i : G[cur]) {
            Edge e = E[i];
            if(level[e.t]==-1 &&
                 e.cap>e.flow) {
                level[e.t] = level[e.s] + 1;
                q.push(e.t);
            }
        }
    }
    return ~level[T];
}
int dfs(int cur, int lim) {
    if(cur==T || lim==0) return lim;
    int result = 0;
    for(int& i=dfs_idx[cur]; i<G[cur].size()
         && lim; i++) {
        Edge& e = E[G[cur][i]];
        if(level[e.s]+1 != level[e.t])
            continue;
        int flow = dfs(e.t, min(lim,
            e.cap-e.flow));
        if(flow <= 0) continue;
        e.flow += flow;
        result += flow;
        E[G[cur][i]^1].flow -= flow;
        lim -= flow;
    }
    return result;
}
```

```
int dinic() {// O((V^2)E)
    int result = 0;
    while(bfs()) {
        memset(dfs_idx, 0, sizeof(dfs_idx));
        result += dfs(S, inf);
    }
    return result;
}
```

3.5 SCC Tarjan

```
//單純考SCC，每個SCC中找成本最小的蓋，如果有多個一樣小
//的要數出來，因為題目要方法數
//注意以下程式有縮點，但沒存起來，
//存法就是開一個array -> ID[u] = SCCID
#define maxn 100005
#define MOD 1000000007
long long cost[maxn];
vector<vector<int>> G;
int SCC = 0;
stack<int> sk;
int dfn[maxn];
int low[maxn];
bool inStack[maxn];
int dfsTime = 1;
long long totalCost = 0;
long long ways = 1;
void dfs(int u) {
    dfn[u] = low[u] = dfsTime;
    ++dfsTime;
    sk.push(u);
    inStack[u] = true;
    for (int v: G[u]) {
        if (dfn[v] == 0) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else if (inStack[v]) {
            //屬於同個SCC且是我的back edge
            low[u] = min(low[u], dfn[v]);
        }
    }
    //如果是SCC
    if (dfn[u] == low[u]) {
        long long minCost = 0x3f3f3f3f;
        int currWays = 0;
        ++SCC;
        while (1) {
            int v = sk.top();
            inStack[v] = 0;
            sk.pop();
            if (minCost > cost[v]) {
                minCost = cost[v];
                currWays = 1;
            }
            else if (minCost == cost[v]) {
                ++currWays;
            }
            if (v == u)
                break;
        }
        totalCost += minCost;
        ways = (ways * currWays) % MOD;
    }
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        scanf("%lld", &cost[i]);
    G.assign(n + 5, vector<int>());
    int m;
    scanf("%d", &m);
    int u, v;
    for (int i = 0; i < m; ++i) {
        scanf("%d %d", &u, &v);
        G[u].emplace_back(v);
    }
    for (int i = 1; i <= n; ++i) {
        if (dfn[i] == 0)
            dfs(i);
    }
    printf("%lld %lld\n", totalCost, ways %
         MOD);
    return 0;
}
```

3.6 SCC Kosaraju

```
//做兩次dfs, O(V + E)
//g 是原圖，g2 是反圖
//s是dfs離開的節點
void dfs1(int u) {
    vis[u] = true;
    for (int v : g[u])
        if (!vis[v]) dfs1(v);
    s.push_back(u);
}

void dfs2(int u) {
    group[u] = sccCnt;
    for (int v : g2[u])
        if (!group[v]) dfs2(v);
}

void kosaraju() {
    sccCnt = 0;
    for (int i = 1; i <= n; ++i)
        if (!vis[i]) dfs1(i);
    for (int i = n; i >= 1; --i)
        if (!group[s[i]]) {
            ++sccCnt;
            dfs2(s[i]);
        }
}
```

3.7 ArticulationPoints Tarjan

```
vector<vector<int>> G;
int N, timer;
bool visited[105];
int dfn[105]; // 第一次visit的時間
int low[105];
//最小能回到的父節點
//(不能是自己的parent)的visTime
int res;
//求割點數量
void tarjan(int u, int parent) {
    int child = 0;
    bool isCut = false;
    visited[u] = true;
    dfn[u] = low[u] = ++timer;
    for (int v: G[u]) {
        if (!visited[v]) {
            ++child;
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (parent != -1 && low[v] >=
                 dfn[u])
                isCut = true;
        }
        else if (v != parent)
            low[u] = min(low[u], dfn[v]);
    }
    //If u is root of DFS
        tree->有兩個以上的children
    if (parent == -1 && child >= 2)
        isCut = true;
    if (isCut) ++res;
}
int main() {
```

```
    char input[105];
    char* token;
    while (scanf("%d", &N) != EOF && N) {
        G.assign(105, vector<int>());
        memset(visited, false,
             sizeof(visited));
        memset(low, 0, sizeof(low));
        memset(dfn, 0, sizeof(visited));
        timer = 0;
        res = 0;
        getchar(); // for \n
        while (fgets(input, 105, stdin)) {
            if (input[0] == '0')
                break;
            int size = strlen(input);
            input[size - 1] = '\0';
            --size;
            token = strtok(input, " ");
            int u = atoi(token);
            int v;
            while (token = strtok(NULL, " "))
                 {
                v = atoi(token);
                G[u].emplace_back(v);
                G[v].emplace_back(u);
            }
        }
        tarjan(1, -1);
        printf("%d\n", res);
    }
    return 0;
}
```

3.8 最小樹狀圖

```
const int maxn = 60 + 10;
const int inf = 0x3f3f3f3f;
struct Edge {
    int s, t, cap, cost;
}; // cap 為頻寬 (optional)
int n, m, c;
int inEdge[maxn], idx[maxn], pre[maxn],
     vis[maxn];
// 對於每個點，選擇對它入度最小的那條邊
// 找環，如果沒有則 return;
// 進行縮環並更新其他點到環的距離。
int dirMST(vector<Edge> edges, int low) {
    int result = 0, root = 0, N = n;
    while(true) {
        memset(inEdge, 0x3f, sizeof(inEdge));
        // 找所有點的 in edge 放進 inEdge
        // optional: low 為最小 cap 限制
        for(const Edge& e : edges) {
            if(e.cap < low) continue;
            if(e.s!=e.t &&
                 e.cost<inEdge[e.t]) {
                inEdge[e.t] = e.cost;
                pre[e.t] = e.s;
            }
        }
        for(int i=0; i<N; i++) {
            if(i!=root && inEdge[i]==inf)
                return -1;//除了root 還有點沒有in
                     edge
        }
        int seq = inEdge[root] = 0;
        memset(idx, -1, sizeof(idx));
        memset(vis, -1, sizeof(vis));
        // 找所有的 cycle，一起編號為 seq
        for(int i=0; i<N; i++) {
            result += inEdge[i];
            int cur = i;
            while(vis[cur]!=i &&
                 idx[cur]==-1) {
                if(cur == root) break;
                vis[cur] = i;
                cur = pre[cur];
            }
            if(cur!=root && idx[cur]==-1) {
                for(int j=pre[cur]; j!=cur;
                     j=pre[j])
                    idx[j] = seq;
                idx[cur] = seq++;
            }
        }
        if(seq == 0) return result; // 沒有
             cycle
        for(int i=0; i<N; i++)
            // 沒有被縮點的點
            if(idx[i] == -1) idx[i] = seq++;
        // 縮點並重新編號
        for(Edge& e : edges) {
            if(idx[e.s] != idx[e.t])
                e.cost -= inEdge[e.t];
            e.s = idx[e.s];
            e.t = idx[e.t];
        }
        N = seq;
        root = idx[root];
    }
}
```

3.9 二分圖最大匹配

```
/* 核心：最大點獨立集 = |V| -
     |最大匹配數|，用匈牙利演算法找出最大匹配數 */
vector<Student> boys;
vector<Student> girls;
vector<vector<int>> G;
bool used[505];
int p[505];
bool match(int i) {
    for (int j: G[i]) {
        if (!used[j]) {
            used[j] = true;
            if (p[j] == -1 || match(p[j])) {
                p[j] = i;
                return true;
            }
        }
    }
    return false;
}
void maxMatch(int n) {
    memset(p, -1, sizeof(p));
    int res = 0;
    for (int i = 0; i < boys.size(); ++i) {
        memset(used, false, sizeof(used));
        if (match(i))
            ++res;
    }
    cout << n - res << '\n';
}
```

3.10 Astar

```
/*A*求k短路
  f(x) = g(x) + h(x)
  g(x) 是實際cost，h(x) 是估計cost
  在此h(x)用所有點到終點的最短距離，則當用Astar找點
  當該點cnt[u] == k時即得到該點的第k短路
*/
#define maxn 105
struct Edge {
    int u, v, w;
};
struct Item_pqH {
    int u, w;
    bool operator <(const Item_pqH& other)
         const {
        return this->w > other.w;
    }
};
struct Item_astar {
    int u, g, f;
    bool operator <(const Item_astar& other)
         const {
        return this->f > other.f;
    }
};
vector<vector<Edge>> G;
//反向圖，用於建h(u)
vector<vector<Edge>> invertG;
int h[maxn];
bool visited[maxn];
int cnt[maxn];
//用反向圖去求出每一點到終點的最短距離，並以此當作h(u)
void dijkstra(int s, int t) {
    memset(visited, 0, sizeof(visited));
    priority_queue<Item_pqH> pq;
    pq.push({s, 0});
    h[s] = 0;
    while (!pq.empty()) {
        Item_pqH curr = pq.top();
        pq.pop();
        visited[curr.u] = true;
        for (Edge& edge: invertG[curr.u]) {
            if (!visited[edge.v]) {
                if (h[edge.v] > h[curr.u] +
                     edge.w) {
                    h[edge.v] = h[curr.u] +
                         edge.w;
                    pq.push({edge.v,
                         h[edge.v]});
                }
            }
        }
    }
}
int Astar(int s, int t, int k) {
    memset(cnt, 0, sizeof(cnt));
    priority_queue<Item_astar> pq;
    pq.push({s, 0, h[s]});
    while (!pq.empty()) {
        Item_astar curr = pq.top();
        pq.pop();
        ++cnt[curr.u];
        //終點出現k次，此時即可得k短路
        if (cnt[t] == k)
            return curr.g;
        for (Edge& edge: G[curr.u]) {
            if (cnt[edge.v] < k) {
                pq.push({edge.v, curr.g +
                     edge.w, curr.g + edge.w
                     + h[edge.v]});
            }
        }
    }
    return -1;
}
int main() {
    int n, m;
    while (scanf("%d %d", &n, &m) && (n != 0
         && m != 0)) {
        G.assign(n + 5, vector<Edge>());
        invertG.assign(n + 5, vector<Edge>());
        int s, t, k;
        scanf("%d %d %d", &s, &t, &k);
        int u, v, w;
        for (int i = 0; i < m; ++i) {
            scanf("%d %d %d", &u, &v, &w);
            G[u].emplace_back(Edge{u, v, w});
            invertG[v].emplace_back(Edge{v,
                 u, w});
        }
        memset(h, 0x3f, sizeof(h));
        dijkstra(t, s);
        printf("%d\n", Astar(s, t, k));
    }
```

```
    return 0;
}
```

3.11 JosephusProblem

```
//JosephusProblem，只是規定要先砍1號
//所以當作有n - 1個人，目標的13順移成12
//再者從0開始比較好算，所以目標12順移成11

// O(n)
int getWinner(int n, int k) {
    int winner = 0;
    for (int i = 1; i <= n; ++i)
        winner = (winner + k) % i;
    return winner;
}

int main() {
    int n;
    while (scanf("%d", &n) != EOF && n){
        --n;
        for (int k = 1; k <= n; ++k){
            if (getWinner(n, k) == 11){
                printf("%d\n", k);
                break;
            }
        }
    }
    return 0;
}

// O(k log(n))
int josephus(int n, int k) {
  if (n == 1) return 0;
  if (k == 1) return n - 1;
  if (k > n) return (josephus(n-1,k)+k)%n;
  int res = josephus(n - n / k, k);
  res -= n % k;
  if (res < 0)
    res += n; // mod n
  else
    res += res / (k - 1); // 还原位置
  return res;
}
```

3.12 KM

```
#define maxn 505
int W[maxn][maxn];
int Lx[maxn], Ly[maxn];
bool S[maxn], T[maxn];
//L[i] = j -> S_i配給T_j, -1 for 還沒匹配
int L[maxn];
int n;
bool match(int i) {
    S[i] = true;
    for (int j = 0; j < n; ++j) {
        // KM重點
        // Lx + Ly >= selected_edge(x, y)
        // 要想辦法降低Lx + Ly
        // 所以選Lx + Ly == selected_edge(x, y)
        if (Lx[i] + Ly[j] == W[i][j] &&
              !T[j]) {
            T[j] = true;
            if ((L[j] == -1) || match(L[j])) {
                L[j] = i;
                return true;
            }
        }
    }
    return false;
}
//修改二分圖上的交錯路徑上點的權重
//此舉是在通過調整vertex labeling看看
//能不能產生出新的增廣路
//(KM的增廣路要求Lx[i] + Ly[j] == W[i][j])
//在這裡優先從最小的diff調調看，才能保證最大權重匹配
void update()
{
    int diff = 0x3f3f3f3f;
    for (int i = 0; i < n; ++i) {
        if (S[i]) {
            for (int j = 0; j < n; ++j) {
                if (!T[j])
                    diff = min(diff, Lx[i] +
                         Ly[j] - W[i][j]);
            }
        }
    }
    for (int i = 0; i < n; ++i) {
        if (S[i]) Lx[i] -= diff;
        if (T[i]) Ly[i] += diff;
    }
}
void KM()
{
    for (int i = 0; i < n; ++i) {
        L[i] = -1;
        Lx[i] = Ly[i] = 0;
        for (int j = 0; j < n; ++j)
            Lx[i] = max(Lx[i], W[i][j]);
    }
    for (int i = 0; i < n; ++i) {
        while(1) {
            memset(S, false, sizeof(S));
            memset(T, false, sizeof(T));
            if (match(i))
                break;
            else
                update(); //去調整vertex
                     labeling以增加增廣路徑
        }
    }
}
int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                scanf("%d", &W[i][j]);
        KM();
        int res = 0;
        for (int i = 0; i < n; ++i) {
            if (i != 0)
                printf(" %d", Lx[i]);
            else
                printf("%d", Lx[i]);
            res += Lx[i];
        }
        puts("");
        for (int i = 0; i < n; ++i) {
            if (i != 0)
                printf(" %d", Ly[i]);
            else
                printf("%d", Ly[i]);
            res += Ly[i];
        }
        puts("");
        printf("%d\n", res);
    }
    return 0;
}
```

3.13 LCA 倍增法

```
//倍增法預處理O(nlogn)，查詢O(logn)，
//利用lca找樹上任兩點距離
#define maxn 100005
struct Edge {
  int u, v, w;
};
vector<vector<Edge>> G; // tree
int fa[maxn][31]; //fa[u][i] -> u的第2^i個祖先
long long dis[maxn][31];
int dep[maxn];//深度
void dfs(int u, int p) {//預處理fa
    fa[u][0] = p; //因為u的第2^0 = 1的祖先就是p
    dep[u] = dep[p] + 1;
    //第2^i的祖先是(第2^(i - 1)個祖先)的
    //第2^(i - 1)的祖先
    //ex: 第8個祖先是 (第4個祖先)的第4個祖先
    for (int i = 1; i < 31; ++i) {
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
        dis[u][i] = dis[fa[u][i - 1]][i - 1]
             + dis[u][i - 1];
    }
    //遍歷子節點
    for (Edge& edge: G[u]) {
        if (edge.v == p)
            continue;
        dis[edge.v][0] = edge.w;
        dfs(edge.v, u);
    }
}
long long lca(int x, int y) {
    //此函數是找lca同時計算x、y的距離 -> dis(x,
         lca) + dis(lca, y)
    //讓y比x深
    if (dep[x] > dep[y])
        swap(x, y);
    int deltaDep = dep[y] - dep[x];
    long long res = 0;
    //讓y與x在同一個深度
    for (int i = 0; deltaDep != 0; ++i,
          deltaDep >>= 1)
        if (deltaDep & 1)
            res += dis[y][i], y = fa[y][i];
    if (y == x) //x = y -> x、y彼此是彼此的祖先
        return res;
    //往上找，一起跳，但x、y不能重疊
    for (int i = 30; i >= 0 && y != x; --i) {
        if (fa[x][i] != fa[y][i]) {
            res += dis[x][i] + dis[y][i];
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    //最後發現不能跳了，此時x的第2^0 =
         1個祖先(或說y的第2^0 =
         1的祖先)即為x、y的lca
    res += dis[x][0] + dis[y][0];
    return res;
}
int main() {
  int n, q;
  while (~scanf("%d", &n) && n) {
    int v, w;
    G.assign(n + 5, vector<Edge>());
        for (int i = 1; i <= n - 1; ++i) {
      scanf("%d %d", &v, &w);
      G[i + 1].push_back({i + 1, v + 1, w});
      G[v + 1].push_back({v + 1, i + 1, w});
    }
        dfs(1, 0);
        scanf("%d", &q);
        int u;
        while (q--) {
            scanf("%d %d", &u, &v);
            printf("%lld%c", lca(u + 1, v +
                 1), (q) ? ' ' : '\n');
        }
  }
  return 0;
}
```

3.14 LCA 樹壓平 RMQ

```
//樹壓平求LCA RMQ(sparse table
     O(nlogn)建立，O(1)查詢)，求任意兩點距離，
```

```cpp
//如果用笛卡兒樹可以壓到O(n)建立，O(1)查詢
//理論上可以過，但遇到直鏈的case dfs深度會stack overflow
#define maxn 100005
struct Edge {
  int u, v, w;
};
int dep[maxn], pos[maxn];
long long dis[maxn];
int st[maxn * 2][32]; //sparse table
int realLCA[maxn * 2][32];
    //最小深度對應的節點，及真正的LCA
int Log[maxn]; //取代std::log2
int tp; // timestamp
vector<vector<Edge>> G; // tree
void calLog() {
  Log[1] = 0;
  Log[2] = 1;
  for (int i = 3; i < maxn; ++i)
    Log[i] = Log[i / 2] + 1;
}
void buildST() {
  for (int j = 0; Log[tp]; ++j) {
    for (int i = 0; i + (1 << j) - 1 < tp;
        ++i) {
      if (st[i - 1][j] < st[i - 1][j + (1 <<
          i - 1)]) {
        st[i][j] = st[i - 1][j];
        realLCA[i][j] = realLCA[i - 1][j];
      }
      else {
        st[i][j] = st[i - 1][j + (1 << i -
            1)];
        realLCA[i][j] = realLCA[i - 1][j + (1
            << i - 1)];
      }
    }
  }
} // O(nlogn)
int query(int l, int r) {// [l, r] min
    depth即為lca的深度
  int k = Log[r - l + 1];
  if (st[l][k] < st[r - (1 << k) + 1][k])
    return realLCA[l][k];
  else
    return realLCA[r - (1 << k) + 1][k];
}
void dfs(int u, int p) {//euler tour
  pos[u] = tp;
  st[tp][0] = dep[u];
  realLCA[tp][0] = dep[u];
  ++tp;
  for (int i = 0; i < G[u].size(); ++i) {
    Edge& edge = G[u][i];
    if (edge.v == p) continue;
    dep[edge.v] = dep[u] + 1;
    dis[edge.v] = dis[edge.u] + edge.w;
    dfs(edge.v, u);
    st[tp++][0] = dep[u];
  }
}
long long getDis(int u, int v) {
  if (pos[u] > pos[v])
    swap(u, v);
  int lca = query(pos[u], pos[v]);
  return dis[u] + dis[v] - 2 *
      dis[query(pos[u], pos[v])];
}
int main() {
  int n, q;
    calLog();
  while (~scanf("%d", &n) && n) {
    int v, w;
    G.assign(n + 5, vector<Edge>());
    tp = 0;
      for (int i = 1; i <= n - 1; ++i) {
      scanf("%d %d", &v, &w);
      G[i].push_back({i, v, w});
      G[v].push_back({v, i, w});
    }
      dfs(0, -1);
      buildST();
      scanf("%d", &q);
      int u;
      while (q--) {
          scanf("%d %d", &u, &v);
          printf("%lld%c", getDis(u, v),
              (q) ? ' ' : '\n');
      }
  }
  return 0;
}
```

3.15 LCA 樹鍊剖分

```cpp
#define maxn 5005
//LCA，用來練習樹鍊剖分
//題意：
    給定樹，找任兩點的中點，若中點不存在(路徑為even)
int dfn[maxn];
int parent[maxn];
int depth[maxn];
int subtreeSize[maxn];
//樹鍊的頂點
int top[maxn];
//將dfn轉成node編碼
int dfnToNode[maxn];
//重兒子
int hson[maxn];
int dfsTime = 1;
//tree
vector<vector<int>> G;
//處理parent、depth、subtreeSize、dfnToNode
void dfs1(int u, int p) {
    parent[u] = p;
    hson[u] = -1;
    subtreeSize[u] = 1;
    for (int v: G[u]) {
        if (v != p) {
            depth[v] = depth[u] + 1;
            dfs1(v, u);
            subtreeSize[u] += subtreeSize[v];
            if (hson[u] == -1 ||
                subtreeSize[hson[u]] <
                subtreeSize[v]) {
                hson[u] = v;
            }
        }
    }
}
//實際剖分 <- 參數t是top的意思
//t初始應為root本身
void dfs2(int u, int t) {
    top[u] = t;
    dfn[u] = dfsTime;
    dfnToNode[dfsTime] = u;
    ++dfsTime;
    //葉子點 -> 沒有重兒子
    if (hson[u] == -1)
        return;
    //優先對重兒子dfs，才能保證同一重鍊dfn連續
    dfs2(hson[u], t);
    for (int v: G[u]) {
        if (v != parent[u] && v != hson[u])
            dfs2(v, v);
    }
}
//不斷跳鍊，當跳到同一條鍊時，深度小的即為LCA
//跳鍊時優先鍊頂深度大的跳
int LCA(int u, int v) {
    while (top[u] != top[v]) {
        if (depth[top[u]] > depth[top[v]])
            u = parent[top[u]];
        else
            v = parent[top[v]];
    }
    return (depth[u] > depth[v]) ? v : u;
}
int getK_parent(int u, int k) {
    while (k-- && (u != -1))
        u = parent[u];
    return u;
}
int main() {
    int n;
    while (scanf("%d", &n) && n) {
        dfsTime = 1;
        G.assign(n + 5, vector<int>());
        int u, v;
        for (int i = 1; i < n; ++i) {
            scanf("%d %d", &u, &v);
            G[u].emplace_back(v);
            G[v].emplace_back(u);
        }
        dfs1(1, -1);
        dfs2(1, 1);
        int q;
        scanf("%d", &q);
        for (int i = 0; i < q; ++i) {
            scanf("%d %d", &u, &v);
            //先得到LCA
            int lca = LCA(u, v);
            //計算路徑長(經過的邊)
            int dis = depth[u] + depth[v] - 2
                * depth[lca];
            //讓v比u深或等於
            if (depth[u] > depth[v])
                swap(u, v);
            if (u == v) {
                printf("The fleas meet at
                    %d.\n", u);
            }
            else if (dis % 2 == 0) {
                //路徑長是even -> 有中點
                printf("The fleas meet at
                    %d.\n", getK_parent(v,
                    dis / 2));
            }
            else {
                //路徑長是odd -> 沒有中點
                if (depth[u] == depth[v]) {
                    int x = getK_parent(u, dis
                        / 2);
                    int y = getK_parent(v, dis
                        / 2);
                    if (x > y)
                        swap(x, y);
                    printf("The fleas jump
                        forever between %d
                        and %d.\n", x, y);
                }
                else {
                    //技巧: 讓深的點v往上dis /
                        2步 = y，
                    //這個點的parent設為x
                    //此時的x、y就是答案要的中點兩點
                    //主要是往下不好找，所以改用深的點用
                    int y = getK_parent(v, dis
                        / 2);
                    int x = getK_parent(y, 1);
                    if (x > y)
                        swap(x, y);
                    printf("The fleas jump
                        forever between %d
                        and %d.\n", x, y);
                }
            }
        }
    }
    return 0;
}
```

3.16 MCMF

```cpp
#define maxn 225
#define INF 0x3f3f3f3f
struct Edge {
    int u, v, cap, flow, cost;
};
//node size, edge size, source, target
int n, m, s, t;
vector<vector<int>> G;
vector<Edge> edges;
bool inqueue[maxn];
long long dis[maxn];
int parent[maxn];
long long outFlow[maxn];
void addEdge(int u, int v, int cap, int cost) {
    edges.emplace_back(Edge{u, v, cap, 0, cost});
    edges.emplace_back(Edge{v, u, 0, 0, -cost});
    m = edges.size();
    G[u].emplace_back(m - 2);
    G[v].emplace_back(m - 1);
}
//一邊求最短路的同時一邊MaxFLow
bool SPFA(long long& maxFlow, long long& minCost) {
    // memset(outFlow, 0x3f, sizeof(outFlow));
    memset(dis, 0x3f, sizeof(dis));
    memset(inqueue, false, sizeof(inqueue));
    queue<int> q;
    q.push(s);
    dis[s] = 0;
    inqueue[s] = true;
    outFlow[s] = INF;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inqueue[u] = false;
        for (const int edgeIndex: G[u]) {
            const Edge& edge = edges[edgeIndex];
            if ((edge.cap > edge.flow) && (dis[edge.v] > dis[u] + edge.cost)) {
                dis[edge.v] = dis[u] + edge.cost;
                parent[edge.v] = edgeIndex;
                outFlow[edge.v] = min(outFlow[u], (long long)(edge.cap - edge.flow));
                if (!inqueue[edge.v]) {
                    q.push(edge.v);
                    inqueue[edge.v] = true;
                }
            }
        }
    }
    //如果dis[t] > 0代表根本不賺還倒賠
    if (dis[t] > 0)
        return false;
    maxFlow += outFlow[t];
    minCost += dis[t] * outFlow[t];
    //一路更新回去這次最短路流完後要維護的
    //MaxFlow演算法相關(如反向邊等)
    int curr = t;
    while (curr != s) {
        edges[parent[curr]].flow += outFlow[t];
        edges[parent[curr] ^ 1].flow -= outFlow[t];
        curr = edges[parent[curr]].u;
    }
    return true;
}
long long MCMF() {
    long long maxFlow = 0;
    long long minCost = 0;
    while (SPFA(maxFlow, minCost))
        ;
    return minCost;
}
int main() {
    int T;
    scanf("%d", &T);
    for (int Case = 1; Case <= T; ++Case){
        //總共幾個月，囤貨成本
        int M, I;
        scanf("%d %d", &M, &I);
        //node size
        n = M + M + 2;
        G.assign(n + 5, vector<int>());
        edges.clear();
        s = 0;
        t = M + M + 1;
        for (int i = 1; i <= M; ++i) {
            int produceCost, produceMax, sellPrice, sellMax, inventoryMonth;
            scanf("%d %d %d %d %d", &produceCost, &produceMax, &sellPrice, &sellMax, &inventoryMonth);
            addEdge(s, i, produceMax, produceCost);
            addEdge(M + i, t, sellMax, -sellPrice);
            for (int j = 0; j <= inventoryMonth; ++j) {
                if (i + j <= M)
                    addEdge(i, M + i + j, INF, I * j);
            }
        }
        printf("Case %d: %lld\n", Case, -MCMF());
    }
    return 0;
}
```

3.17 莫隊

```cpp
/*利用prefix前綴XOR和
  如果要求[x, y]的XOR和只要回答prefix[y] ^ prefix[x - 1]即可在O(1)回答
  同時維護cnt[i]代表[x, y]XOR和 == i的個數
  如此我們知道[l, r]可以快速知道[l - 1, r], [l + 1, r], [l, r - 1], [l, r + 1]的答案
  就符合Mo's algorithm的思維O(N * sqrt(n))
  每次轉移為O(1)，具體轉移方法在下面*/
#define maxn 100005
//在此prefix[i]是[1, i]的XOR和
int prefix[maxn];
//log_2(1000000) = 19.931568569324174087221916576937...
//所以開到1 << 20
//cnt[i]代表的是有符合nums[x, y] such that nums[x] ^ nums[x + 1] ^ .. ^ nums[y] == i
//的個數
long long cnt[1 << 20];
//塊大小 -> sqrt(n)
int sqrtQ;
struct Query {
    int l, r, id;
    bool operator < (const Query& other) const {
        if (this->l / sqrtQ != other.l / sqrtQ)
            return this->l < other.l;
        //奇偶排序(優化)
        if (this->l / sqrtQ & 1)
            return this->r < other.r;
        return this->r > other.r;
    }
};
Query querys[maxn];
long long ans[maxn];
long long res = 0;
int k;
void add(int x) {
    res += cnt[k ^ prefix[x]];
    ++cnt[prefix[x]];
}
void sub(int x) {
    --cnt[prefix[x]];
    res -= cnt[k ^ prefix[x]];
}
int main() {
    int n, m;
    scanf("%d %d %d", &n, &m, &k);
    sqrtQ = sqrt(n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &prefix[i]);
        prefix[i] ^= prefix[i - 1];
    }
    for (int i = 1; i <= m; ++i) {
        scanf("%d %d", &querys[i].l, &querys[i].r);
        //減1是因為prefix[i]是[1, i]的前綴XOR和，所以題目問[l, r]我們要回答[l - 1, r]的答案
        --querys[i].l;
        querys[i].id = i;
    }
    sort(querys + 1, querys + m + 1);
    int l = 1, r = 0;
    for (int i = 1; i <= m; ++i) {
        while (l < querys[i].l) {
            sub(l);
            ++l;
        }
        while (l > querys[i].l) {
            --l;
            add(l);
        }
        while (r < querys[i].r) {
            ++r;
            add(r);
        }
        while (r > querys[i].r) {
            sub(r);
            --r;
        }
        ans[querys[i].id] = res;
    }
    for (int i = 1; i <= m; ++i){
        printf("%lld\n", ans[i]);
    }
    return 0;
}
```

3.18 Dancing Links

```cpp
struct DLX {
    int seq, resSize;
    int col[maxn], row[maxn];
    int U[maxn], D[maxn], R[maxn], L[maxn];
    int rowHead[maxn], colSize[maxn];
    int result[maxn];
    DLX(int r, int c) {
        for(int i=0; i<=c; i++) {
            L[i] = i-1, R[i] = i+1;
            U[i] = D[i] = i;
        }
        L[R[seq=c]=0]=c;
        resSize = -1;
        memset(rowHead, 0, sizeof(rowHead));
        memset(colSize, 0, sizeof(colSize));
```

```
    }
    void insert(int r, int c) {
        row[++seq]=r, col[seq]=c,
            ++colSize[c];
        U[seq]=c, D[seq]=D[c], U[D[c]]=seq,
            D[c]=seq;
        if(rowHead[r]) {
            L[seq]=rowHead[r],
                R[seq]=R[rowHead[r]];
            L[R[rowHead[r]]]=seq,
                R[rowHead[r]]=seq;
        } else {
            rowHead[r] = L[seq] = R[seq] =
                seq;
        }
    }
    void remove(int c) {
        L[R[c]] = L[c], R[L[c]] = R[c];
        for(int i=D[c]; i!=c; i=D[i]) {
            for(int j=R[i]; j!=i; j=R[j]) {
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                --colSize[col[j]];
            }
        }
    }
    void recover(int c) {
        for(int i=U[c]; i!=c; i=U[i]) {
            for(int j=L[i]; j!=i; j=L[j]) {
                U[D[j]] = D[U[j]] = j;
                ++colSize[col[j]];
            }
        }
        L[R[c]] = R[L[c]] = c;
    }
    bool dfs(int idx=0) { // 判斷其中一解版
        if(R[0] == 0) {
            resSize = idx;
            return true;
        }
        int c = R[0];
        for(int i=R[0]; i; i=R[i]) {
            if(colSize[i] < colSize[c]) c = i;
        }
        remove(c);
        for(int i=D[c]; i!=c; i=D[i]) {
            result[idx] = row[i];
            for(int j=R[i]; j!=i; j=R[j])
                remove(col[j]);
            if(dfs(idx+1)) return true;
            for(int j=L[i]; j!=i; j=L[j])
                recover(col[j]);
        }
        recover(c);
        return false;
    }
    void dfs(int idx=0) { // 判斷最小 dfs
        depth 版
        if(R[0] == 0) {
            resSize = min(resSize, idx); //
                注意init值
            return;
        }
        int c = R[0];
        for(int i=R[0]; i; i=R[i]) {
            if(colSize[i] < colSize[c]) c = i;
        }
        remove(c);
        for(int i=D[c]; i!=c; i=D[i]) {
            for(int j=R[i]; j!=i; j=R[j])
                remove(col[j]);
            dfs(idx+1);
            for(int j=L[i]; j!=i; j=L[j])
                recover(col[j]);
        }
        recover(c);
    }
};
```

4 DataStructure

4.1 BIT

```
template <class T> class BIT {
private:
  int size;
  vector<T> bit;
  vector<T> arr;

public:
  BIT(int sz=0): size(sz), bit(sz+1),
      arr(sz) {}

  /** Sets the value at index idx to val. */
  void set(int idx, T val) {
      add(idx, val - arr[idx]);
    }

  /** Adds val to the element at index idx.
      */
  void add(int idx, T val) {
    arr[idx] += val;
    for (++idx; idx<=size; idx+=(idx & -idx))
          bit[idx] += val;
  }

  /** @return The sum of all values in [0,
      idx]. */
  T pre_sum(int idx) {
    T total = 0;
    for (++idx; idx>0; idx-=(idx & -idx))
          total += bit[idx];
    return total;
  }
};
```

4.2 ChthollyTree

```
//重點: 要求輸入資料隨機，否則可能被卡時間
struct Node {
    long long l, r;
    mutable long long val;
    Node(long long l, long long r, long long
        val)
        : l(l), r(r), val(val){}
    bool operator < (const Node& other)
        const{
        return this->l < other.l;
    }
};
set<Node> chthollyTree;
//將[l, r] 拆成 [l, pos - 1], [pos, r]
set<Node>::iterator split(long long pos) {
    //找第一個左端點大於等於pos的區間
    set<Node>::iterator it =
        chthollyTree.lower_bound(Node(pos,
        0, 0));
    //運氣很好直接找到左端點是pos的區間
    if (it != chthollyTree.end() && it->l ==
        pos)
        return it;
    //到這邊代表找到的是第一個左端點大於pos的區間
    //it - 1即可找到左端點等於pos的區間
    //(不會是別的，因為沒有重疊的區間)
    --it;
    long long l = it->l, r = it->r;
    long long val = it->val;
    chthollyTree.erase(it);
    chthollyTree.insert(Node(l, pos - 1,
        val));
    //回傳左端點是pos的區間iterator
    return chthollyTree.insert(Node(pos, r,
        val)).first;
}
//區間賦值
void assign(long long l, long long r, long
    long val) {
    //<注意>
    //end與begin的順序不能調換，因為end的split可能會改變
    //因為end可以在原本begin的區間中
    set<Node>::iterator end = split(r + 1),
        begin = split(l);
    //begin到end全部刪掉
    chthollyTree.erase(begin, end);
    //填回去[l, r]的區間
    chthollyTree.insert(Node(l, r, val));
}
//區間加值(直接一個個區間去加)
void add(long long l, long long r, long long
    val) {
    set<Node>::iterator end = split(r + 1);
    set<Node>::iterator begin = split(l);
    for (set<Node>::iterator it = begin; it
        != end; ++it)
        it->val += val;
}
//查詢區間第k小 -> 直接把每個區間丟去vector排序
long long getKthSmallest(long long l, long
    long r, long long k) {
    set<Node>::iterator end = split(r + 1);
    set<Node>::iterator begin = split(l);
    //pair -> first: val, second: 區間長度
    vector<pair<long long, long long>> vec;
    for (set<Node>::iterator it = begin; it
        != end; ++it) {
        vec.push_back({it->val, it->r - it->l
            + 1});
    }
    sort(vec.begin(), vec.end());
    for (const pair<long long, long long>&
        p: vec) {
        k -= p.second;
        if (k <= 0)
            return p.first;
    }
    //不應該跑到這
    return -1;
}
//快速冪
long long qpow(long long x, long long n,
    long long mod) {
    long long res = 1;
    x %= mod;
    while (n)
    {
        if (n & 1)
            res = res * x % mod;
        n >>= 1;
        x = x * x % mod;
    }
    return res;
}
//區間n次方和
long long sumOfPow(long long l, long long r,
    long long n, long long mod) {
    long long total = 0;
    set<Node>::iterator end = split(r + 1);
    set<Node>::iterator begin = split(l);
    for (set<Node>::iterator it = begin; it
        != end; ++it)
    {
        total = (total + qpow(it->val, n,
            mod) * (it->r - it->l + 1)) %
            mod;
    }
    return total;
}
```

4.3 線段樹 1D

```
#define MAXN 1000
int data[MAXN]; //原數據
```

```cpp
int st[4 * MAXN]; //線段樹
int tag[4 * MAXN]; //懶標
inline int pull(int l, int r) {
// 隨題目改變sum、max、min
// l、r是左右樹的index
    return st[l] + st[r];
}
void build(int l, int r, int i) {
// 在[l, r]區間建 樹，目前根的index為i
    if (l == r) {
        st[i] = data[l];
        return;
    }
    int mid = l + ((r - l) >> 1);
    build(l, mid, i * 2);
    build(mid + 1, r, i * 2 + 1);
    st[i] = pull(i * 2, i * 2 + 1);
}
int qry(int ql, int qr, int l, int r, int i){
// [ql,qr]是查詢區間，[l,r]是當前節點包含的區間
    if (ql <= l && r <= qr)
        return st[i];
    int mid = l + ((r - l) >> 1);
    if (tag[i]) {
        //如果當前懶標有值則更新左右節點
        st[i * 2] += tag[i] * (mid - l + 1);
        st[i * 2 + 1] += tag[i] * (r - mid);
        tag[i * 2] += tag[i];
        tag[i*2+1] += tag[i];
        tag[i] = 0;
    }
    int sum = 0;
    if (ql <= mid)
        sum+=query(ql, qr, l, mid, i * 2);
    if (qr > mid)
        sum+=query(ql, qr, mid+1, r, i*2+1);
    return sum;
}
void update(
    int ql,int qr,int l,int r,int i,int c) {
// [ql,qr]是查詢區間，[l,r]是當前節點包含的區間
// c是變化量
    if (ql <= l && r <= qr) {
        st[i] += (r - l + 1) * c;
            //求和,此需乘上區間長度
        tag[i] += c;
        return;
    }
    int mid = l + ((r - l) >> 1);
    if (tag[i] && l != r) {
        //如果當前懶標有值則更新左右節點
        st[i * 2] += tag[i] * (mid - l + 1);
        st[i * 2 + 1] += tag[i] * (r - mid);
        tag[i * 2] += tag[i];//下傳懶標至左節點
        tag[i*2+1] += tag[i];//下傳懶標至右節點
        tag[i] = 0;
    }
    if (ql <= mid) update(ql, qr, l, mid, i
         * 2, c);
    if (qr > mid) update(ql, qr, mid+1, r,
         i*2+1, c);
    st[i] = pull(i * 2, i * 2 + 1);
}
//如果是直接改值而不是加值，query與update中的tag與st
//改值從+=改成=
```

4.4 線段樹 2D

```cpp
//純2D segment tree 區間查詢單點修改最大最小值
#define maxn 2005 //500 * 4 + 5
int maxST[maxn][maxn], minST[maxn][maxn];
int N;
void modifyY(int index, int l, int r, int
     val, int yPos, int xIndex, bool
     xIsLeaf) {
    if (l == r) {
        if (xIsLeaf) {
            maxST[xIndex][index] =
                minST[xIndex][index] = val;
            return;
        }
        maxST[xIndex][index] =
             max(maxST[xIndex * 2][index],
             maxST[xIndex * 2 + 1][index]);
        minST[xIndex][index] =
             min(minST[xIndex * 2][index],
             minST[xIndex * 2 + 1][index]);
    }
    else {
        int mid = (l + r) / 2;
        if (yPos <= mid)
            modifyY(index * 2, l, mid, val,
                 yPos, xIndex, xIsLeaf);
        else
            modifyY(index * 2 + 1, mid + 1,
                 r, val, yPos, xIndex,
                 xIsLeaf);

        maxST[xIndex][index] =
             max(maxST[xIndex][index * 2],
             maxST[xIndex][index * 2 + 1]);
        minST[xIndex][index] =
             min(minST[xIndex][index * 2],
             minST[xIndex][index * 2 + 1]);
    }
}
void modifyX(int index, int l, int r, int
     val, int xPos, int yPos) {
    if (l == r) {
        modifyY(1, 1, N, val, yPos, index,
             true);
    }
    else {
        int mid = (l + r) / 2;
        if (xPos <= mid)
            modifyX(index * 2, l, mid, val,
                 xPos, yPos);
        else
            modifyX(index * 2 + 1, mid + 1,
                 r, val, xPos, yPos);
        modifyY(1, 1, N, val, yPos, index,
             false);
    }
}
void queryY(int index, int l, int r, int
     yql, int yqr, int xIndex, int& vmax,
     int &vmin) {
    if (yql <= l && r <= yqr) {
        vmax = max(vmax,
             maxST[xIndex][index]);
        vmin = min(vmin,
             minST[xIndex][index]);
    }
    else
    {
        int mid = (l + r) / 2;
        if (yql <= mid)
            queryY(index * 2, l, mid, yql,
                 yqr, xIndex, vmax, vmin);
        if (mid < yqr)
            queryY(index * 2 + 1, mid + 1, r,
                 yql, yqr, xIndex, vmax,
                 vmin);
    }
}
void queryX(int index, int l, int r, int
     xql, int xqr, int yql, int yqr, int&
     vmax, int& vmin) {
    if (xql <= l && r <= xqr) {
        queryY(1, 1, N, yql, yqr, index,
             vmax, vmin);
    }
    else {
        int mid = (l + r) / 2;
        if (xql <= mid)
            queryX(index * 2, l, mid, xql,
                 xqr, yql, yqr, vmax, vmin);
        if (mid < xqr)
            queryX(index * 2 + 1, mid + 1, r,
                 xql, xqr, yql, yqr, vmax,
                 vmin);
    }
}
int main() {
    while (scanf("%d", &N) != EOF) {
        int val;
        for (int i = 1; i <= N; ++i) {
            for (int j = 1; j <= N; ++j) {
                scanf("%d", &val);
                modifyX(1, 1, N, val, i, j);
            }
        }
        int q;
        int vmax, vmin;
        int xql, xqr, yql, yqr;
        char op;
        scanf("%d", &q);
        while (q--) {
            getchar(); //for \n
            scanf("%c", &op);
            if (op == 'q') {
                scanf("%d %d %d %d", &xql,
                     &yql, &xqr, &yqr);
                vmax = -0x3f3f3f3f;
                vmin = 0x3f3f3f3f;
                queryX(1, 1, N, xql, xqr,
                     yql, yqr, vmax, vmin);
                printf("%d %d\n", vmax, vmin);
            }
            else {
                scanf("%d %d %d", &xql, &yql,
                     &val);
                modifyX(1, 1, N, val, xql,
                     yql);
            }
        }
    }
    return 0;
}
```

4.5 權值線段樹

```cpp
//權值線段樹 + 離散化 解決區間第k小問題
//其他網路上的解法：2個heap，Treap，AVL tree
#define maxn 30005
int nums[maxn];
int getArr[maxn];
int id[maxn];
int st[maxn << 2];
void update(int index, int l, int r, int qx)
     {
    if (l == r)
    {
        ++st[index];
        return;
    }

    int mid = (l + r) / 2;
    if (qx <= mid)
        update(index * 2, l, mid, qx);
    else
        update(index * 2 + 1, mid + 1, r, qx);
    st[index] = st[index * 2] + st[index * 2
         + 1];
}
//找區間第k個小的
int query(int index, int l, int r, int k) {
    if (l == r)
        return id[l];
    int mid = (l + r) / 2;
    //k比左子樹小
    if (k <= st[index * 2])
```

```
        return query(index * 2, l, mid, k);
    else
        return query(index * 2 + 1, mid + 1,
             r, k - st[index * 2]);
}
int main() {
    int t;
    cin >> t;
    bool first = true;
    while (t--) {
        if (first)
            first = false;
        else
            puts("");
        memset(st, 0, sizeof(st));
        int m, n;
        cin >> m >> n;
        for (int i = 1; i <= m; ++i) {
            cin >> nums[i];
            id[i] = nums[i];
        }
        for (int i = 0; i < n; ++i)
            cin >> getArr[i];
        //離散化
        //防止m == 0
        if (m)
            sort(id + 1, id + m + 1);
        int stSize = unique(id + 1, id + m +
             1) - (id + 1);
        for (int i = 1; i <= m; ++i) {
            nums[i] = lower_bound(id + 1, id
                 + stSize + 1, nums[i]) - id;
        }
        int addCount = 0;
        int getCount = 0;
        int k = 1;
        while (getCount < n) {
            if (getArr[getCount] == addCount)
                 {
                printf("%d\n", query(1, 1,
                     stSize, k));
                ++k;
                ++getCount;
            }
            else {
                update(1, 1, stSize,
                     nums[addCount + 1]);
                ++addCount;
            }
        }
    }
    return 0;
}
```

4.6 Trie

```
const int maxc = 26;      // 單字字符數
const char minc = 'a';    // 首個 ASCII

struct TrieNode {
  int cnt;
  TrieNode* child[maxc];

  TrieNode() {
    cnt = 0;
    for(auto& node : child) {
      node = nullptr;
    }
  }
};

struct Trie {
  TrieNode* root;

  Trie() { root = new TrieNode(); }

  void insert(string word) {
    TrieNode* cur = root;
    for(auto& ch : word) {
      int c = ch - minc;
      if(!cur->child[c])
        cur->child[c] = new TrieNode();
      cur = cur->child[c];
    }
    cur->cnt++;
  }

  void remove(string word) {
    TrieNode* cur = root;
    for(auto& ch : word) {
      int c = ch - minc;
      if(!cur->child[c]) return;
      cur = cur->child[c];
    }
    cur->cnt--;
  }

  // 字典裡有出現 word
  bool search(string word, bool prefix=0) {
    TrieNode* cur = root;
    for(auto& ch : word) {
      int c = ch - minc;
      if(!(cur=cur->child[c])) return false;
    }
    return cur->cnt || prefix;
  }

  // 字典裡有 word 的前綴為 prefix
  bool startsWith(string prefix) {
    return search(prefix, true);
  }
};
```

4.7 AC Trie

```
const int maxn = 1e4 + 10; // 單字字數
const int maxl = 50 + 10; // 單字字長
const int maxc = 128;     // 單字字符數
const char minc = ' ';    // 首個 ASCII

int trie[maxn*maxl][maxc]; // 原字典樹
int val[maxn*maxl];       // 結尾(單字編號)
int cnt[maxn*maxl];       // 結尾(重複個數)
int fail[maxn*maxl];      // failure link
bool vis[maxn*maxl];      // 同單字不重複

struct ACTrie {
  int seq, root;

  ACTrie() {
    seq = 0;
    root = newNode();
  }

  int newNode() {
    for(int i=0; i<maxc; i++) trie[seq][i]=0;
    val[seq] = cnt[seq] = fail[seq] = 0;
    return seq++;
  }

  void insert(char* s, int wordId=0) {
    int p = root;
    for(; *s; s++) {
      int c = *s - minc;
      if(!trie[p][c]) trie[p][c] = newNode();
      p = trie[p][c];
    }
    val[p] = wordId;
    cnt[p]++;
  }

  void build() {
    queue<int> q({root});
    while(!q.empty()) {
      int p = q.front();
      q.pop();
      for(int i=0; i<maxc; i++) {
        int& t = trie[p][i];
        if(t) {
          fail[t] = p?trie[fail[p]][i]:root;
          q.push(t);
        } else {
          t = trie[fail[p]][i];
        }
      }
    }
  }

  // 要存 wordId 才要 vec
  // 同單字重複match要把所有vis取消掉
  int match(char* s, vector<int>& vec) {
    int res = 0;
    memset(vis, 0, sizeof(vis));
    for(int p=root; *s; s++) {
      p = trie[p][*s-minc];
      for(int k=p; k && !vis[k]; k=fail[k]) {
        vis[k] = true;
        res += cnt[k];
        if(cnt[k]) vec.push_back(val[k]);
      }
    }
    return res;    // 匹配到的單字量
  }
};

ACTrie ac;     // 建構，初始化
ac.insert(s); // 加字典單字
// 加完字典後
ac.build();    // !!! 建 failure link !!!
ac.match(s);   // 多模式匹配(加vec存編號)
```

4.8 單調隊列

```
//單調隊列
"如果一個選手比你小還比你強，你就可以退役了。"

example

給出一個長度為 n 的數組，
輸出每 k 個連續的數中的最大值和最小值。

#include <bits/stdc++.h>
#define maxn 1000100
using namespace std;
int q[maxn], a[maxn];
int n, k;
//得到這個隊列裡的最小值，直接找到最後的就行了
void getmin() {
    int head=0,tail=0;
    for(int i=1;i<k;i++) {
        while(head<=tail&&a[q[tail]]>=a[i])
            tail--;
        q[++tail]=i;
    }
    for(int i=k; i<=n;i++) {
        while(head<=tail&&a[q[tail]]>=a[i])
            tail--;
        q[++tail]=i;
        while(q[head]<=i-k) head++;
        cout<<a[q[head]]<<" ";
    }
    cout<<endl;
}
// 和上面同理
void getmax() {
    int head=0,tail=0;
    for(int i=1;i<k;i++) {
        while(head<=tail&&a[q[tail]]<=a[i])tail--;
        q[++tail]=i;
    }
    for(int i=k;i<=n;i++) {
```

```
        while(head<=tail&&a[q[tail]]<=a[i])tail--;
        q[++tail]=i;
        while(q[head]<=i-k) head++;
        cout<<a[q[head]]<<" ";
    }
    cout<<endl;
}

int main(){
    cin>>n>>k; //每k個連續的數
    for(int i=1;i<=n;i++) cin>>a[i];
    getmin();
    getmax();
    return 0;
}
```

5 Geometry

5.1 Template

```
using DBL = double;
using TP = DBL; // 存點的型態

const DBL pi = acos(-1);
const DBL eps = 1e-8;
const TP inf = 1e30;
const int maxn = 5e4 + 10;

struct Vector {
  TP x, y;
  Vector(TP x=0, TP y=0): x(x), y(y) {}
  DBL length();
};
using Point = Vector;
using Polygon = vector<Point>;

Vector operator+(Vector a, Vector b) {
  return Vector(a.x+b.x, a.y+b.y); }
Vector operator-(Vector a, Vector b) {
  return Vector(a.x-b.x, a.y-b.y); }
Vector operator*(Vector a, DBL b) {
  return Vector(a.x*b, a.y*b); }
Vector operator/(Vector a, DBL b) {
  return Vector(a.x/b, a.y/b); }

TP dot(Vector a, Vector b) {
  return a.x*b.x + a.y*b.y;
}
TP cross(Vector a, Vector b) {
  return a.x*b.y - a.y*b.x;
}
DBL Vector::length() {
  return sqrt(dot(*this, *this));
}
DBL dis(Point a, Point b) {
  return sqrt(dot(a-b, a-b));
}
Vector unit_normal_vector(Vector v) {
  DBL len = v.length();
  return Vector(-v.y/len, v.x/len);
}

struct Line {
  Point p;
  Vector v;
  DBL ang;
  Line(Point _p={}, Vector _v={}) {
    p = _p;
    v = _v;
    ang = atan2(v.y, v.x);
  }
  bool operator<(const Line& l) const {
    return ang < l.ang;
  }
};

struct Segment {
    Point s, e;
    Segment(): s({0, 0}), e({0, 0}) {}
    Segment(Point s, Point e): s(s), e(e) {}
    DBL length() { return dis(s, e); }
};

struct Circle {
  Point o;
  DBL r;
  Circle(): o({0, 0}), r(0) {}
  Circle(Point o, DBL r=0): o(o), r(r) {}
  Circle(Point a, Point b) { // ab 直徑
    o = (a + b) / 2;
    r = dis(o, a);
  }
  Circle(Point a, Point b, Point c) {
    Vector u = b-a, v = c-a;
    DBL c1=dot(u, a+b)/2, c2=dot(v, a+c)/2;
    DBL dx=c1*v.y-c2*u.y, dy=u.x*c2-v.x*c1;
    o = Point(dx, dy) / cross(u, v);
    r = dis(o, a);
  }
  bool cover(Point p) {
    return dis(o, p) <= r;
  }
};
```

5.2 Polygon

```
// 判斷點 (point) 是否在凸包 (p) 內
bool pointInConvex(Polygon& p, Point point) {
  // 根據 TP 型態來寫，沒浮點數不用 dblcmp
  auto dblcmp=[](DBL v){return (v>0)-(v<0);};
  // 不包含線上，改 '>=' 為 '<'
  auto test = [&](Point& p0, Point& p1) {
    return dblcmp(cross(p1-p0, point-p0))>=0;
  };
  p.push_back(p[0]);
  for(int i=1; i<p.size(); i++) {
    if(!test(p[i-1], p[i])) {
      p.pop_back();
      return false;
    }
  }
  p.pop_back();
  return true;
}

// 計算簡單多邊形的面積
// ! p 為排序過的點 !
DBL polygonArea(Polygon& p) {
  DBL sum = 0;
  for(int i=0, n=p.size(); i<n; i++)
    sum += cross(p[i], p[(i+1)%n]);
  return abs(sum) / 2.0;
}
```

5.3 Intersection

```
// 除 intersection(Line a, Line b) 之外，
// 皆尚未丟 online judge

int dcmp(DBL a, DBL b=0.0) {
  return (a > b) - (a < b);
}

bool hasIntersection(Point p, Segment s) {
  return dcmp(cross(p-s.s, s.s-s.e))==0&&
         dcmp(dot(p.x-s.s.x, p.x-s.e.x))<=0&&
         dcmp(dot(p.y-s.s.y, p.y-s.e.y))<=0;
}

bool hasIntersection(Point p, Line l) {
  return dcmp(cross(p-l.p, l.v)) == 0;
}

DBL dis(Line l, Point p) {
  DBL t = cross(p, l.v) + cross(l.v, l.p);
  return abs(t) / sqrt(dot(l.v, l.v));
}

Point intersection(Line a, Line b) {
  Vector u = a.p - b.p;
  DBL t = 1.0*cross(b.v, u)/cross(a.v, b.v);
  return a.p + a.v*t;
}
```

5.4 最小圓覆蓋

```cpp
vector<Point> p(3);   // 在圓上的點
Circle MEC(vector<Point>& v, int n, int d=0){
  Circle mec;
  if(d == 1) mec = Circle(p[0]);
  if(d == 2) mec = Circle(p[0], p[1]);
  if(d == 3) return Circle(p[0], p[1], p[2]);
  for(int i=0; i<n; i++) {
    if(mec.cover(v[i])) continue;
    p[d] = v[i];
    mec = MEC(v, i, d+1);
  }
  return mec;
}
```

5.5 旋轉卡尺

```cpp
// 回傳凸包內最遠兩點的距離
int longest_distance(Polygon& p) {
  auto test = [&](Line l, Point a, Point b) {
   return cross(l.v,a-l.p)<=cross(l.v,b-l.p);
  };
  if(p.size() <= 2) {
    return cross(p[0]-p[1], p[0]-p[1]);
  }
  int mx = 0;
  for(int i=0, j=1, n=p.size(); i<n; i++) {
    Line l(p[i], p[(i+1)%n] - p[i]);
    for(;test(l,p[j],p[(j+1)%n]);j=(j+1)%n);
    mx = max({
      mx,
      dot(p[(i+1)%n]-p[j], p[(i+1)%n]-p[j]),
      dot(p[i]-p[j], p[i]-p[j])
    });
  }
  return mx;
}
```

5.6 凸包

- TP 為 Point 裡 x 和 y 的型態
- struct Point 需要加入並另外計算的 variables:
  1. ang, 該點與基準點的 atan2 值
  2. d2, 該點與基準點的 (距離)^2
- 注意計算 d2 的型態範圍限制

```cpp
using TP = long long;
using Polygon = vector<Point>;

const TP inf = 1e9;   // 座標點最大值

Polygon convex_hull(Point* p, int n) {
  auto dblcmp = [](DBL a, DBL b=0.0) {
    return (a>b) - (a<b);
  };
  auto rmv = [&](Point a, Point b, Point c) {
    return cross(b-a, c-b) <= 0; // 非浮點數
    return dblcmp(cross(b-a, c-b)) <= 0;
  };

  // 選最下裡最左的當基準點，可在輸入時計算
  TP lx = inf, ly = inf;
  for(int i=0; i<n; i++) {
    if(p[i].y<ly || (p[i].y==ly&&p[i].x<lx)){
      lx = p[i].x, ly = p[i].y;
    }
  }

  for(int i=0; i<n; i++) {
    p[i].ang=atan2(p[i].y-ly,p[i].x-lx);
    p[i].d2 = (p[i].x-lx)*(p[i].x-lx) +
              (p[i].y-ly)*(p[i].y-ly);
  }
  sort(p, p+n, [&](Point& a, Point& b) {
    if(dblcmp(a.ang, b.ang))
      return a.ang < b.ang;
    return a.d2 < b.d2;
  });

  int m = 1;    // stack size
  Point st[n] = {p[n]=p[0]};
  for(int i=1; i<=n; i++) {
    for(;m>1&&rmv(st[m-2],st[m-1],p[i]);m--);
    st[m++] = p[i];
  }
  return Polygon(st, st+m-1);
}
```

5.7 半平面相交

```cpp
using DBL = double;
using TP = DBL; // 存點的型態
using Polygon = vector<Point>;

const int maxn = 5e4 + 10;

// Return: 能形成半平面交的凸包邊界點
Polygon halfplaneIntersect(vector<Line>&nar){
  sort(nar.begin(), nar.end());
  // DBL 跟 0 比較, 沒符點數不用
  auto dblcmp=[](DBL v){return (v>0)-(v<0);};
  // p 是否在 l 的左半平面
  auto lft = [&](Point p, Line l) {
    return dblcmp(cross(l.v, p-l.p)) > 0;
  };

  int ql = 0, qr = 0;
  Line L[maxn] = {nar[0]};
  Point P[maxn];

  for(int i=1; i<nar.size(); i++) {
    for(; ql<qr&&!lft(P[qr-1],nar[i]); qr--);
    for(; ql<qr&&!lft(P[ql],nar[i]); ql++);
    L[++qr] = nar[i];
    if(dblcmp(cross(L[qr].v,L[qr-1].v))==0) {
      if(lft(nar[i].p,L[--qr])) L[qr]=nar[i];
    }
    if(ql < qr)
      P[qr-1] = intersection(L[qr-1], L[qr]);
  }
  for(; ql<qr && !lft(P[qr-1], L[ql]); qr--);
  if(qr-ql <= 1) return {};
  P[qr] = intersection(L[qr], L[ql]);
  return Polygon(P+ql, P+qr+1);
}
```

6 DP

6.1 以價值為主的背包

```cpp
/*w 變得太大所以一般的01背包解法變得不可能
  觀察題目w變成10^9
  而v_i變成10^3
  N不變10^2
  試著湊湊看dp狀態
  dp[maxn][maxv]是可接受的複雜度
  剩下的是轉移式，轉移式變成
  dp[i][j] = w ->
        當目前只考慮到第i個商品時，達到獲利j時最少的weigh
        = w
  所以答案是dp[n][1 ~ maxv]找價值最大且裝的下的*/
#define maxn 105
#define maxv 100005
long long dp[maxn][maxv];
long long weight[maxn];
long long v[maxn];
int main() {
    int n;
    long long w;
    scanf("%d %lld", &n, &w);
    for (int i = 1; i <= n; ++i) {
        scanf("%lld %lld", &weight[i], &v[i]);
    }
    memset(dp, 0x3f, sizeof(dp));
    dp[0][0] = 0;
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j <= maxv; ++j) {
            if (j - v[i] >= 0)
                dp[i][j] = dp[i - 1][j -
                    v[i]] + weight[i];
            dp[i][j] = min(dp[i - 1][j],
                 dp[i][j]);
        }
    }
    long long res = 0;
    for (int j = maxv - 1; j >= 0; --j) {
        if (dp[n][j] <= w) {
            res = j;
            break;
        }
    }
    printf("%lld\n", res);
    return 0;
}
```

6.2 抽屜

```cpp
long long dp[70][70][2];
// 初始條件
dp[1][0][0] = dp[1][1][1] = 1;
for (int i = 2; i <= 66; ++i){
    // i個抽屜0個安全且上方0 =
    // (底下i - 1個抽屜且1個安全且最上面L) +
    // (底下n - 1個抽屜0個安全且最上方為0)
    dp[i][0][0]=dp[i-1][1][1]+dp[i-1][0][0];
    for (int j = 1; j <= i; ++j) {
        dp[i][j][0] =
          dp[i-1][j+1][1]+dp[i-1][j][0];
        dp[i][j][1] =
          dp[i-1][j-1][1]+dp[i-1][j-1][0];
    }
} //答案在 dp[n][s][0] + dp[n][s][1];
```

6.3 Barcode

```cpp
int N, K, M;
long long dp[55][55];
// n -> 目前剩多少units
// k -> 目前剩多少bars
```

```cpp
// m -> 1 bar最多多少units
long long dfs(int n, int k) {
    if (k == 1) {
        return (n <= M);
    }
    if (dp[n][k] != -1)
        return dp[n][k];
    long long result = 0;
    for (int i = 1; i < min(M + 1, n); ++i)
         { // < min(M + 1, n)是因為n不能==0
        result += dfs(n - i, k - 1);
    }
    return dp[n][k] = result;
}
int main() {
    while (scanf("%d %d %d", &N, &K, &M) !=
         EOF) {
        memset(dp, -1, sizeof(dp));
        printf("%lld\n", dfs(N, K));
    }
    return 0;
}
```

6.4 Deque 最大差距

```cpp
/*定義dp[l][r]是l ~ r時與先手最大差異值
  轉移式：dp[l][r] = max{a[l] - solve(l + 1,
       r), a[r] - solve(l, r - 1)}
  裡面用減的主要是因為求的是相減且會一直換手，
  所以正負正負...*/
#define maxn 3005
bool vis[maxn][maxn];
long long dp[maxn][maxn];
long long a[maxn];
long long solve(int l, int r) {
    if (l > r) return 0;
    if (vis[l][r]) return dp[l][r];
    vis[l][r] = true;
    long long res = a[l] - solve(l + 1, r);
    res = max(res, a[r] - solve(l, r - 1));
    return dp[l][r] = res;
}
int main() {
    ...
    printf("%lld\n", solve(1, n));
}
```

6.5 LCS 和 LIS

```cpp
//LCS 和 LIS 題目轉換
LIS 轉成 LCS
    1. A 為原序列， B=sort(A)
    2. 對 A,B 做 LCS
LCS 轉成 LIS
    1. A, B 為原本的兩序列
    2. 最 A 序列作編號轉換，將轉換規則套用在 B
    3. 對 B 做 LIS
    4. 重複的數字在編號轉換時後要變成不同的數字，
       越早出現的數字要越小
    5. 如果有數字在 B 裡面而不在 A 裡面，
       直接忽略這個數字不做轉換即可
```

6.6 RangeDP

```cpp
//區間dp
int dp[55][55];
// dp[i][j] -> [i,j] 切割區間中最小的cost
int cuts[55];
int solve(int i, int j) {
    if (dp[i][j] != -1)
        return dp[i][j];
    //代表沒有其他切法，只能是cuts[j] - cuts[i]
    if (i == j - 1)
        return dp[i][j] = 0;
    int cost = 0x3f3f3f3f;
    for (int m = i + 1; m < j; ++m) {
        //枚舉區間中間切點
        cost = min(cost, solve(i, m) +
          solve(m, j) + cuts[j] - cuts[i]);
    }
    return dp[i][j] = cost;
}
int main() {
    int l,n;
    while (scanf("%d", &l) != EOF && l){
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i)
            scanf("%d", &cuts[i]);
        cuts[0] = 0;
        cuts[n + 1] = l;
        memset(dp, -1, sizeof(dp));
        printf("ans = %d.\n", solve(0,n+1));
    }
    return 0;
}
```

6.7 stringDP

Edit distance S_1 最少需要經過幾次增、刪或換字變成 S_2

$$dp[i,j] = \begin{cases} i+1, & \text{if } j = -1 \\ j+1, & \text{if } i = -1 \\ dp[i-1,j-1], & \text{if } S_1[i] = S_2[j] \\ \min\begin{Bmatrix} dp[i,j-1] \\ dp[i-1,j] \\ dp[i-1,j-1] \end{Bmatrix} + 1, & \text{if } S_1[i] \neq S_2[j] \end{cases}$$

Longest Palindromic Subsequence

$$dp[l,r] = \begin{cases} 1 & \text{if} \quad l = r \\ dp[l+1,r-1] & \text{if} \quad S[l] = S[r] \\ \max\{dp[l+1,r], dp[l,r-1]\} & \text{if} \quad S[l] \neq S[r] \end{cases}$$

6.8 樹 DP 有幾個 path 長度為 k

```cpp
#define maxn 50005
#define maxk 505
//dp[u][u的child且距離u長度k的數量]
long long dp[maxn][maxk];
vector<vector<int>> G;
int n, k;
long long res = 0;
void dfs(int u, int p) {
    //u自己
    dp[u][0] = 1;
    for (int v: G[u]) {
        if (v == p)
            continue;
        dfs(v, u);
        for (int i = 1; i <= k; ++i) {
            //子樹v距離i - 1的等於對於u來說距離i的
            dp[u][i] += dp[v][i - 1];
        }
    }
    //統計在u子樹中距離u為k的數量
    res += dp[u][k];
    long long cnt = 0;
    for (int v: G[u]) {
      if (v == p)
        continue; //重點算法
      for (int x = 0; x <= k - 2; ++x) {
        cnt +=
          dp[v][x]*(dp[u][k-x-1]-dp[v][k-x-2]);
      }
    }
    res += cnt / 2;
}
int main() {
    ...
    dfs(1, -1);
    printf("%lld\n", res);
    return 0;
}
```

6.9 TreeDP reroot

```cpp
/*re-root dp on tree O(n + n + n) -> O(n)*/
class Solution {
public:
    vector<int> sumOfDistancesInTree(int n,
         vector<vector<int>>& edges) {
        this->res.assign(n, 0);
        G.assign(n + 5, vector<int>());
        for (vector<int>& edge: edges) {
            G[edge[0]].emplace_back(edge[1]);
            G[edge[1]].emplace_back(edge[0]);
        }
        memset(this->visited, 0,
             sizeof(this->visited));
        this->dfs(0);
        memset(this->visited, 0,
             sizeof(this->visited));
        this->res[0] = this->dfs2(0, 0);
        memset(this->visited, 0,
             sizeof(this->visited));
        this->dfs3(0, n);
        return this->res;
    }
private:
    vector<vector<int>> G;
    bool visited[30005];
    int subtreeSize[30005];
    vector<int> res;
    //求subtreeSize
    int dfs(int u) {
        this->visited[u] = true;
        for (int v: this->G[u])
            if (!this->visited[v])
                this->subtreeSize[u] +=
                     this->dfs(v);
        //自己
        this->subtreeSize[u] += 1;
        return this->subtreeSize[u];
    }
    //求res[0], 0到所有點的距離
    int dfs2(int u, int dis) {
        this->visited[u] = true;
        int sum = 0;
        for (int v: this->G[u])
            if (!visited[v])
                sum += this->dfs2(v, dis + 1);
        //要加上自己的距離
        return sum + dis;
    }
    //算出所有的res
    void dfs3(int u, int n) {
        this->visited[u] = true;
        for (int v: this->G[u]) {
            if (!visited[v]) {
                this->res[v] = this->res[u] +
                     n - 2 *
                     this->subtreeSize[v];
                this->dfs3(v, n);
            }
        }
    }
};
```

6.10 WeightedLIS

```
#define maxn 200005
long long dp[maxn];
long long height[maxn];
long long B[maxn];
long long st[maxn << 2];
void update(int p, int index, int l, int r,
     long long v) {
    if (l == r) {
        st[index] = v;
        return;
    }
    int mid = (l + r) >> 1;
    if (p <= mid)
        update(p, (index << 1), l, mid, v);
    else
        update(p, (index << 1)+1,mid+1,r,v);
    st[index] =
      max(st[index<<1],st[(index<<1)+1]);
}
long long query(int index, int l, int r, int
     ql, int qr) {
    if (ql <= l && r <= qr)
        return st[index];
    int mid = (l + r) >> 1;
    long long res = -1;
    if (ql <= mid)
        res =
          max(res,query(index<<1,l,mid,ql,qr));
    if (mid < qr)
        res =
          max(res,query((index<<1)+1,mid+1,r,ql,qr));
    return res;
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        scanf("%lld", &height[i]);
    for (int i = 1; i <= n; ++i)
        scanf("%lld", &B[i]);
    long long res = B[1];
    update(height[1], 1, 1, n, B[1]);
    for (int i = 2; i <= n; ++i) {
        long long temp;
        if (height[i] - 1 >= 1)
            temp =
              B[i]+query(1,1,n,1,height[i]-1);
        else
            temp = B[i];
        update(height[i], 1, 1, n, temp);
        res = max(res, temp);
    }
    printf("%lld\n", res);
    return 0;
}
```