# Contents


1 字串 1
1.1 最長迴文子字串 . . . 1
1.2 KMP . . . 1

2 STL 1
2.1 multiset . . . 1
2.2 unordered_set . . . 1

3 math 2
3.1 質數與因數 . . . 2
3.2 歐拉函數 . . . 2
3.3 atan . . . 2
3.4 大步小步 . . . 3

4 algorithm 3
4.1 basic . . . 3
4.2 三分搜 . . . 3
4.3 差分 . . . 4
4.4 greedy . . . 4
4.5 dinic . . . 5
4.6 Nim Game . . . 6
4.7 SCC Tarjan . . . 6
4.8 ArticulationPoints Tarjan . . . 7
4.9 最小樹狀圖 . . . 7
4.10 Astar . . . 9
4.11 JosephusProblem . . . 9
4.12 KM . . . 10
4.13 LCA 倍增法 . . . 10
4.14 LCA 樹壓平 RMQ . . . 11
4.15 MCMF . . . 12
4.16 莫隊 . . . 12
4.17 Dancing Links . . . 13

5 DataStructure 14
5.1 BIT . . . 14
5.2 ChthollyTree . . . 14
5.3 線段樹 1D . . . 14
5.4 線段樹 2D . . . 15
5.5 權值線段樹 . . . 16
5.6 Trie . . . 16
5.7 單調隊列 . . . 17

6 geometry 17
6.1 intersection . . . 17
6.2 半平面相交 . . . 17
6.3 凸包 . . . 18

7 DP 19
7.1 以價值為主的背包 . . . 19
7.2 抽屜 . . . 19
7.3 Barcode . . . 19
7.4 Deque 最大差距 . . . 20
7.5 LCS 和 LIS . . . 20
7.6 RangeDP . . . 20
7.7 stringDP . . . 20
7.8 TreeDP 有幾個 path 長度為 k . . . 20
7.9 TreeDP reroot . . . 21
7.10 WeightedLIS . . . 21


# 1 字串

## 1.1 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;
}
```

## 1.2 KMP

```cpp
#define maxn 1000005
int nextArr[maxn];
void getNextArr(const string& str) {
    nextArr[0] = 0;
    int prefixLen = 0;
    for (int i = 1; i < str.size(); ++i) {
        prefixLen = nextArr[i - 1];
        //如果不一樣就在之前算過的prefix中搜有沒有更短的前後綴
        while (prefixLen > 0 && str[prefixLen] !=
            str[i])
            prefixLen = nextArr[prefixLen - 1];
        //一樣就繼承之前的前後綴長度+1
        if (str[prefixLen] == str[i])
            ++prefixLen;
        nextArr[i] = prefixLen;
    }
    for (int i = 0; i < str.size() - 1; ++i) {
        vis[nextArr[i]] = true;
    }
}
```

# 2 STL

## 2.1 multiset

```
與 set 用法雷同，但會保留重複的元素。
資料由小到大排序。
宣告：
    multiset<int> st;
刪除資料：
    st.erase(val);
    //會刪除所有值為 val 的元素。
    st.erase(st.find(val));
    //只刪除第一個值為 val 的元素。
```

## 2.2 unordered_set

```
unordered_set 的實作方式通常是用雜湊表(hash table)，
資料插入和查詢的時間複雜度很低，為常數級別O(1)，
相對的代價是消耗較多的記憶體，空間複雜度較高，
無自動排序功能。

unordered_set 判斷元素是否存在
unordered_set<int> myunordered_set;
```

```
myunordered_set.insert(2);
myunordered_set.insert(4);
myunordered_set.insert(6);
cout << myunordered_set.count(4) << "\n"; // 1
cout << myunordered_set.count(8) << "\n"; // 0
```

# 3 math

## 3.1 質數與因數

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

## 3.2 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;
}
```

## 3.3 atan

```
說明
  atan() 和 atan2() 函數分別計算 x 和 y/x 的反正切。

回傳值
  atan() 函數會傳回介於範圍 -pi/2 到 pi/2
      弧度之間的值。
  atan2() 函數會傳回介於 -pi 至 pi 弧度之間的值。
  如果 atan2() 函數的兩個引數都是零，
  則函數會將 errno 設為 EDOM，並傳回值 0。

範例
int main(void){
    double a,b,c,d;
```

```
    c=0.45;
    d=0.23;
    a=atan(c);
    b=atan2(c,d);
    printf("atan(%lf)=%lf/n",c,a);
    printf("atan2(%lf,%lf)=%lf/n",c,d,b);

}
// atan(0.450000)=0.422854
// atan2(0.450000,0.230000)=1.098299
```

## 3.4 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';
    }
}
```

# 4 algorithm

## 4.1 basic

```
min_element：找尋最小元素
min_element(first, last)
max_element：找尋最大元素
max_element(first, last)
sort：排序，預設由小排到大。
sort(first, last)
sort(first, last, cmp)：可自行定義比較運算子 cmp 。
find：尋找元素。
find(first, last, val)
lower_bound：尋找第一個小於 x 的元素位置，
            如果不存在，則回傳 last 。
lower_bound(first, last, val)
upper_bound：尋找第一個大於 x 的元素位置，
            如果不存在，則回傳 last 。
upper_bound(first, last, val)
next_permutation：將序列順序轉換成下一個字典序，
                  如果存在回傳 true，反之回傳 false。
next_permutation(first, last)
prev_permutation：將序列順序轉換成上一個字典序，
                  如果存在回傳 true，反之回傳 false。
prev_permutation(first, last)
```

## 4.2 三分搜

```
  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;
    }
};
```

```cpp
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';
    }
}
```

## 4.3 差分

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。

```cpp
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] << ' ';
    }
}
```

## 4.4 greedy

貪心演算法的核心為，
採取在目前狀態下最好或最佳（即最有利）的選擇。
貪心演算法雖然能獲得當前最佳解，
但不保證能獲得最後（全域）最佳解，
提出想法後可以先試圖尋找有沒有能推翻原本的想法的反例，
確認無誤再實作。

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。

```cpp
//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*

```cpp
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*

```cpp
struct Line{
    int L,R;
    bool operator<(const Line &rhs)const{
        return R<rhs.R;
    }
};
```

```cpp
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
```

```cpp
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
```

```cpp
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
```

```cpp
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';
    }
}
```

## 4.5 dinic

```cpp
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;
}
```

## 4.6 Nim Game

```
//兩人輪流取銅板，每人每次需在某堆取一枚以上的銅板，
//但不能同時在兩堆取銅板，直到最後，
//將銅板拿光的人贏得此遊戲。
#define maxn 23+5
int SG[maxn];
int visited[1000+5];
int pile[maxn],ans;
void calculateSG(){
    SG[0]=0;
    for(int i=1;i<=maxn;i++){
        int cur=0;
        for(int j=0;j<i;j++)
            for(int k=0;k<=j;k++)
                visited[SG[j]^SG[k]]=i;
        while(visited[cur]==i) cur++;
        SG[i]=cur;
    }
}
int main(){
    calculateSG();
    int Case=0,n;
    while(cin>>n,n){
      ans=0;
      for(int i=1;i<=n;i++) cin>>pile[i];
      for(int i=1;i<=n;i++)
        if(pile[i]&1) ans^=SG[n-i];
      cout<<"Game "<<++Case<<": ";
      if(!ans) cout<<"-1 -1 -1\n";
      else{
        bool flag=0;
        for(int i=1;i<=n;i++){
          if(pile[i]){
            for(int j=i+1;j<=n;j++){
              for(int k=j;k<=n;k++){
                if((SG[n-i]^SG[n-j]^SG[n-k])==ans){
                  cout<<i-1<<" "<<j-1<<" "<<k-1<<endl;
                  flag=1;
                  break;
                }
              }
              if(flag) break;
            }
            if(flag) break;
          }
        }
      }
    }
    return 0;
}
/*
 input
4  1 0 1 100
3  1 0 5
2  2 1
0
 output
Game 1: 0 2 3
Game 2: 0 1 1
Game 3: -1 -1 -1
*/
```

## 4.7 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
//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;
}
```

## 4.8 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;
}
```

## 4.9 最小樹狀圖

```
  定義
有向圖上的最小生成樹 (Directed Minimum Spanning Tree)
稱為最小樹形圖。
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];
    }
}
=======================================================
  Tarjan 的DMST 演算法
Tarjan 提出了一種能夠在
O(m+nlog n)時間內解決最小樹形圖問題的演算法。
  流程
Tarjan 的演算法分為收縮與伸展兩個過程。
接下來先介紹收縮的過程。
我們要假設輸入的圖是滿足強連通的，
如果不滿足那就加入 O(n) 條邊使其滿足，
並且這些邊的邊權是無窮大的。
我們需要一個堆存儲結點的入邊編號，入邊權值，
結點總代價等相關信息，由於後續過程中會有堆的合併操作，
這裡採用左偏樹 與並查集實現。
演算法的每一步都選擇一個任意結點v，
需要保證v不是根節點，並且在堆中沒有它的入邊。
再將v的最小入邊加入到堆中，
如果新加入的這條邊使堆中的邊形成了環，
那麼將構成環的那些結點收縮，
我們不妨將這些已經收縮的結點命名為超級結點，
再繼續這個過程，如果所有的頂點都縮成了超級結點，
那麼收縮過程就結束了。
整個收縮過程結束後會得到一棵收縮樹，
之後就會對它進行伸展操作。
堆中的邊總是會形成一條路徑v0 <- v1<- ... <- vk，
由於圖是強連通的，這個路徑必然存在，
並且其中的 vi 可能是最初的單一結點，
也可能是壓縮後的超級結點。
最初有 v0=a，其中 a 是圖中任意的一個結點，
每次都選擇一條最小入邊 vk <- u，
如果 u 不是v0,v1,...,vk中的一個結點，
那麼就將結點擴展到 v k+1=u。
如果 u 是他們其中的一個結點 vi，
那麼就找到了一個關於 vi <- ... <- vk <- vi的環，
再將他們收縮為一個超級結點c。
向隊列 P 中放入所有的結點或超級結點，
並初始選擇任一節點 a，只要佇列不為空，就進行以下步驟：
選擇 a 的最小入邊，保證不存在自環，
並找到另一頭的結點 b。
如果結點b沒有被記錄過說明未形成環，
令 a <- b，繼續目前操作尋找環。
如果 b 被記錄過了，就表示出現了環。
總結點數加一，並將環上的所有結點重新編號，對堆進行合併，
以及結點/超級結點的總權值的更新。
更新權值操作就是將環上所有結點的入邊都收集起來，
並減去環上入邊的邊權。
typedef long long ll;
#define maxn 102
#define INF 0x3f3f3f3f
```

```
struct UnionFind {
  int fa[maxn << 1];
  UnionFind() { memset(fa, 0, sizeof(fa)); }
  void clear(int n) {
    memset(fa + 1, 0, sizeof(int) * n);
  }
  int find(int x) {
    return fa[x] ? fa[x] = find(fa[x]) : x;
  }
  int operator[](int x) { return find(x); }
};
struct Edge {
  int u, v, w, w0;
};
struct Heap {
  Edge *e;
  int rk, constant;
  Heap *lch, *rch;
  Heap(Edge *_e):
    e(_e),rk(1),constant(0),lch(NULL),rch(NULL){}
  void push() {
    if (lch) lch->constant += constant;
    if (rch) rch->constant += constant;
    e->w += constant;
    constant = 0;
  }
};
Heap *merge(Heap *x, Heap *y) {
  if (!x) return y;
  if (!y) return x;
  if(x->e->w + x->constant > y->e->w + y->constant)
    swap(x, y);
  x->push();
  x->rch = merge(x->rch, y);
  if (!x->lch || x->lch->rk < x->rch->rk)
    swap(x->lch, x->rch);
  if (x->rch)
    x->rk = x->rch->rk + 1;
  else
    x->rk = 1;
  return x;
}
Edge *extract(Heap *&x) {
  Edge *r = x->e;
  x->push();
  x = merge(x->lch, x->rch);
  return r;
}
vector<Edge> in[maxn];
int n, m, fa[maxn << 1], nxt[maxn << 1];
Edge *ed[maxn << 1];
Heap *Q[maxn << 1];
UnionFind id;
void contract() {
  bool mark[maxn << 1];
  //將圖上的每一個節點與其相連的那些節點進行記錄
  for (int i = 1; i <= n; i++) {
    queue<Heap *> q;
    for (int j = 0; j < in[i].size(); j++)
      q.push(new Heap(&in[i][j]));
    while (q.size() > 1) {
      Heap *u = q.front();
      q.pop();
      Heap *v = q.front();
      q.pop();
      q.push(merge(u, v));
    }
    Q[i] = q.front();
  }
  mark[1] = true;
  for(int a=1,b=1,p;Q[a];b=a,mark[b]=true){
    //尋找最小入邊以及其端點，保證無環
    do {
      ed[a] = extract(Q[a]);
      a = id[ed[a]->u];
    } while (a == b && Q[a]);
    if (a == b) break;
```

```
    if (!mark[a]) continue;
    //對發現的環進行收縮，以及環內的節點重新編號，
    //總權值更新
    for (a = b, n++; a != n; a = p) {
      id.fa[a] = fa[a] = n;
      if (Q[a]) Q[a]->constant -= ed[a]->w;
      Q[n] = merge(Q[n], Q[a]);
      p = id[ed[a]->u];
      nxt[p == n ? b : p] = a;
    }
  }
}
ll expand(int x, int r);
ll expand_iter(int x) {
  ll r = 0;
  for(int u=nxt[x];u!=x;u=nxt[u]){
    if (ed[u]->w0 >= INF)
      return INF;
    else
      r+=expand(ed[u]->v,u)+ed[u]->w0;
  }
  return r;
}
ll expand(int x, int t) {
  ll r = 0;
  for (; x != t; x = fa[x]) {
    r += expand_iter(x);
    if (r >= INF) return INF;
  }
  return r;
}
void link(int u, int v, int w) {
  in[v].push_back({u, v, w, w});
}
int main() {
  int rt;
  scanf("%d %d %d", &n, &m, &rt);
  for (int i = 0; i < m; i++) {
    int u, v, w;
    scanf("%d %d %d", &u, &v, &w);
    link(u, v, w);
  }
  //保證強連通
  for (int i = 1; i <= n; i++)
    link(i > 1 ? i - 1 : n, i, INF);
  contract();
  ll ans = expand(rt, n);
  if (ans >= INF)
    puts("-1");
  else
    printf("%lld\n", ans);
  return 0;
}
```

## 4.10 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;
}
```

## 4.11 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
```

```cpp
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;
}
```

## 4.12 KM

```cpp
/*題意: 給定一個W矩陣，現在分成row、column兩個1維陣列
    W[i][j]=k即代表column[i] + row[j]要>=k
    求row[] 與 column[]的所有值在滿足矩陣W的要求之下
    row[] + column[]所有元素相加起來要最小
    利用KM求二分圖最大權匹配
    Lx -> vertex labeling of X
    Ly -> vertex labeling of y
    一開始Lx[i] = max(W[i][j]), Ly = 0
    Lx[i] + Ly[j] >= W[i][j]
    要最小化全部的(Lx[i] + Ly[j])加總
    不斷的調整vertex
        labeling去找到一條交錯邊皆滿足Lx[i] + Ly[j]
        == W[i][j]的增廣路
    最後會得到正確的二分圖完美匹配中的最大權分配(先滿足最多
    意義是將最大化所有匹配邊權重和的問題改成最小化所有點的
#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;
}
```

## 4.13 LCA 倍增法

```cpp
//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;
}
```

## 4.14 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，
//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;
}
```

## 4.15 MCMF

```
#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;
//SPFA用
bool inqueue[maxn];
//SPFA用的dis[]
long long dis[maxn];
//maxFlow一路扣回去時要知道parent
//<注> 在這題因為G[][]中存的是edgeIndex in edges[]
//
    所以parent存的也是對應edges[]中的edgeIndex(主要是方便)
int parent[maxn];
//maxFlow時需要紀錄到node u時的bottleneck
//同時也代表著u該次流出去的量
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
    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;
}
```

## 4.16 莫隊

```
/*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;
}
```

## 4.17 Dancing Links

```
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);
    }
};
```

# 5 DataStructure

## 5.1 BIT

```cpp
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;
  }
};
```

## 5.2 ChthollyTree

```cpp
//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
    --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;
}
```

## 5.3 線段樹 1D

```cpp
#define MAXN 1000
int data[MAXN]; //原數據
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 query(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];//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;
    }
    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的
//改值從+=改成=
```

## 5.4 線段樹 2D

```
//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;
}
```

## 5.5 權值線段樹

```
//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;
}
```

## 5.6 Trie

```
const int maxn = 300000 + 10;
const int mod = 20071027;
int dp[maxn];
int mp[4000*100 + 10][26];
char str[maxn];
struct Trie {
    int seq;
    int val[maxn];
    Trie() {
        seq = 0;
        memset(val, 0, sizeof(val));
        memset(mp, 0, sizeof(mp));
    }
    void insert(char* s, int len) {
        int r = 0;
        for(int i=0; i<len; i++) {
            int c = s[i] - 'a';
            if(!mp[r][c]) mp[r][c] = ++seq;
            r = mp[r][c];
        }
        val[r] = len;
        return;
    }
    int find(int idx, int len) {
        int result = 0;
        for(int r=0; idx<len; idx++) {
            int c = str[idx] - 'a';
            if(!(r = mp[r][c])) return result;
            if(val[r])
                result = (result + dp[idx + 1]) % mod;
        }
        return result;
    }
};
int main() {
    int n, tc = 1;
    while(~scanf("%s%d", str, &n)) {
        Trie tr;
        int len = strlen(str);
```

```
        char word[100+10];
        memset(dp, 0, sizeof(dp));
        dp[len] = 1;
        while(n--) {
            scanf("%s", word);
            tr.insert(word, strlen(word));
        }
        for(int i=len-1; i>=0; i--)
            dp[i] = tr.find(i, len);
        printf("Case %d: %d\n", tc++, dp[0]);
    }
    return 0;
}
/****Input****
 * abcd
 * 4
 * a b cd ab
 *************
 ****Output***
 * Case 1: 2
 **************/
```

## 5.7 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。"--單調隊列

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;

void getmin() {
    // 得到這個隊列裡的最小值，直接找到最後的就行了
    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;
}

void getmax() {  // 和上面同理
    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;
}
```

# 6 geometry

## 6.1 intersection

```
using LL = long long;

struct Point2D {
    LL x, y;
};

struct Line2D {
    Point2D s, e;
    LL a, b, c;             // L: ax + by = c
    Line2D(Point2D s, Point2D e): s(s), e(e) {
        a = e.y - s.y;
        b = s.x - e.x;
        c = a * s.x + b * s.y;
    }
};

// 用克拉馬公式求二元一次解
Point2D intersection2D(Line2D l1, Line2D l2) {
    LL D  = l1.a * l2.b - l2.a * l1.b;
    LL Dx = l1.c * l2.b - l2.c * l1.b;
    LL Dy = l1.a * l2.c - l2.a * l1.c;

    if(D) {          // intersection
        double x = 1.0 * Dx / D;
        double y = 1.0 * Dy / D;
    } else {
        if(Dx || Dy) // Parallel lines
        else         // Same line
    }
}
```

## 6.2 半平面相交

```
// Q: 給定一張凸包(已排序的點)，
// 找出圖中離凸包外最遠的距離

const int maxn = 100 + 10;
const double eps = 1e-7;

struct Vector {
    double x, y;
    Vector(double x=0.0, double y=0.0): x(x), y(y) {}

    Vector operator+(Vector v) {
        return Vector(x+v.x, y+v.y);
    }
    Vector operator-(Vector v) {
        return Vector(x-v.x, y-v.y);
    }
    Vector operator*(double val) {
        return Vector(x*val, y*val);
    }
    double dot(Vector v) { return x*v.x + y*v.y; }
    double cross(Vector v) { return x*v.y - y*v.x; }
    double length() { return sqrt(dot(*this)); }
    Vector unit_normal_vector() {
        double len = length();
        return Vector(-y/len, x/len);
    }
};

using Point = Vector;

struct Line {
    Point p;
    Vector v;
```

```
    double 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;
    }
    Point intersection(Line l) {
        Vector u = p - l.p;
        double t = l.v.cross(u) / v.cross(l.v);
        return p + v*t;
    }
};

int n, m;
Line narrow[maxn];     // 要判斷的直線
Point poly[maxn];      // 能形成半平面交的凸包邊界點

// return true if point p is on the left of line l
bool onLeft(Point p, Line l) {
    return l.v.cross(p-l.p) > 0;
}

int halfplaneIntersection() {
    int l, r;
    Line L[maxn];          // 排序後的向量隊列
    Point P[maxn];         // s[i] 跟 s[i-1] 的交點

    L[l=r=0] = narrow[0]; // notice: narrow is sorted
    for(int i=1; i<n; i++) {
        while(l<r && !onLeft(P[r-1], narrow[i])) r--;
        while(l<r && !onLeft(P[l], narrow[i])) l++;

        L[++r] = narrow[i];
        if(l < r) P[r-1] = L[r-1].intersection(L[r]);
    }

    while(l<r && !onLeft(P[r-1], L[l])) r--;
    if(r-l <= 1) return 0;

    P[r] = L[r].intersection(L[l]);

    int m=0;
    for(int i=l; i<=r; i++) {
        poly[m++] = P[i];
    }

    return m;
}

Point pt[maxn];
Vector vec[maxn];
Vector normal[maxn];// normal[i] = vec[i] 的單位法向量

double bsearch(double l=0.0, double r=1e4) {
    if(abs(r-l) < eps) return l;

    double mid = (l + r) / 2;

    for(int i=0; i<n; i++) {
        narrow[i] = Line(pt[i]+normal[i]*mid, vec[i]);
    }

    if(halfplaneIntersection())
        return bsearch(mid, r);
    else return bsearch(l, mid);
}

int main() {
    while(~scanf("%d", &n) && n) {
        for(int i=0; i<n; i++) {
            double x, y;
            scanf("%lf%lf", &x, &y);
            pt[i] = {x, y};
        }
        for(int i=0; i<n; i++) {
            vec[i] = pt[(i+1)%n] - pt[i];
            normal[i] = vec[i].unit_normal_vector();
        }

        printf("%.6lf\n", bsearch());
    }
    return 0;
}
```

## 6.3 凸包

```
// Q：平面上給定多個區域，由多個座標點所形成，再給定
// 多點(x,y)，判斷有落點的區域(destroyed)的面積總和。
#include <bits/stdc++.h>
using namespace std;

const int maxn = 500 + 10;
const int maxCoordinate = 500 + 10;

struct Point {
    int x, y;
};

int n;
bool destroyed[maxn];
Point arr[maxn];
vector<Point> polygons[maxn];

void scanAndSortPoints() {
    int minX = maxCoordinate, minY = maxCoordinate;
    for(int i=0; i<n; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        arr[i] = (Point){x, y};
        if(y < minY || (y == minY && x < minX)) {
    //  If there are floating points, use:
    //  if(y<minY || (abs(y-minY)<eps && x<minX)) {
            minX = x, minY = y;
        }
    }
    sort(arr, arr+n, [minX, minY](Point& a, Point& b){
        double theta1 = atan2(a.y - minY, a.x - minX);
        double theta2 = atan2(b.y - minY, b.x - minX);
        return theta1 < theta2;
    });
    return;
}

//  returns cross product of u(AB) x v(AC)
int cross(Point& A, Point& B, Point& C) {
    int u[2] = {B.x - A.x, B.y - A.y};
    int v[2] = {C.x - A.x, C.y - A.y};
    return (u[0] * v[1]) - (u[1] * v[0]);
}

// size of arr = n >= 3
// st = the stack using vector, m = index of the top
vector<Point> convex_hull() {
    vector<Point> st(arr, arr+3);
    for(int i=3, m=2; i<n; i++, m++) {
        while(m >= 2) {
            if(cross(st[m], st[m-1], arr[i]) < 0)
                break;
            st.pop_back();
            m--;
        }
        st.push_back(arr[i]);
    }
    return st;
}

bool inPolygon(vector<Point>& vec, Point p) {
    vec.push_back(vec[0]);
    for(int i=1; i<vec.size(); i++) {
        if(cross(vec[i-1], vec[i], p) < 0) {
            vec.pop_back();
```

```
            return false;
        }
    }
    vec.pop_back();
    return true;
}

       1 | x1   x2   x3   x4   x5          xn |
   A = - |    x    x    x    x    x ... x    |
       2 | y1   y2   y3   y4   y5          yn |
double calculateArea(vector<Point>& v) {
    v.push_back(v[0]);          // make v[n] = v[0]
    double result = 0.0;
    for(int i=1; i<v.size(); i++)
        result += v[i-1].x*v[i].y - v[i-1].y*v[i].x;
    v.pop_back();
    return result / 2.0;
}

int main() {
    int p = 0;
    while(~scanf("%d", &n) && (n != -1)) {
        scanAndSortPoints();
        polygons[p++] = convex_hull();
    }

    int x, y;
    double result = 0.0;
    while(~scanf("%d%d", &x, &y)) {
        for(int i=0; i<p; i++) {
            if(inPolygon(polygons[i], (Point){x, y}))
                destroyed[i] = true;
        }
    }
    for(int i=0; i<p; i++) {
        if(destroyed[i])
            result += calculateArea(polygons[i]);
    }
    printf("%.2lf\n", result);
    return 0;
}
```

# 7 DP

## 7.1 以價值為主的背包

```
/*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weight總
      = 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;
}
```

## 7.2 抽屜

```
// dp[n][s][t] n: 幾個抽屜 s: 幾個是安全的 t: (0 or
    1) 最上面的抽屜是U or L
// 分兩種case
// case 1: dp[n][s][0] = dp[n - 1][s + 1][1] + dp[n -
    1][s][0]
// 此時最上面放U，則
// dp[n - 1][s + 1][1]: 現在要放的U會導致底下n -
    1個抽屜最上面L變不安全，為了得到n個抽屜s個安全，所以要s
    + 1
// dp[n - 1][s][0]: n -
    1個抽屜有s個安全，現在在其上面再放一個U不影響S的數量
// case 2: dp[n][s][1] = dp[n - 1][s - 1][1] + dp[n -
    1][s - 1][0]
// 在最上面放L，底下n - 1個抽屜有s -
    1個安全，無論上方是U、L皆不影響
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]);
```

## 7.3 Barcode

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

## 7.4 Deque 最大差距

```
/*定義dp[l][r]是l ~ r時與先手最大差異值
  Deque可以拿頭尾
  所以轉移式中dp[l][r]與dp[l + 1][r]、dp[l][r - 1]有關
  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));
}
```

## 7.5 LCS 和 LIS

```
//最長共同子序列(LCS)
給定兩序列 A,B ，求最長的序列 C ，
 C 同時為 A,B 的子序列。
//最長遞增子序列 (LIS)
 給你一個序列 A ，求最長的序列 B ，
  B 是一個（非）嚴格遞增序列，且為 A 的子序列。
//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
```

## 7.6 RangeDP

```
//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;
    int 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("The minimum cutting is %d.\n",
            solve(0, n + 1));
    }
    return 0;
}
```

## 7.7 stringDP

- Edit distance

  $S_1$ 最少需要經過幾次增、刪或換字變成 $S_2$

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

- Longest Palindromic Subsequence

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

## 7.8 TreeDP 有幾個 path 長度為 k

```
#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];
    //統計橫跨u但還是在u的子樹中合計長度為k的:
    //考慮u有一子節點v，在v子樹中距離v長度為x的
    //以及不在v子樹但在u子樹中(這樣才會是橫跨u)且距離u長度為k
        - x - 1的
    //共有0.5 * (dp[v][x] * (dp[u][k - x - 1] -
        dp[v][k - x - 2]))
    //以上算式是重點，可使複雜度下降，否則枚舉一定超時
    //其中 dp[u][k - x - 1]是所有u子樹中距離u為k - x
        - 1的節點
    // - dp[v][k - x -
        2]是因為我們不要v子樹的節點且距離u為k - x -
        1的(要v子樹以外的)，
```

```
    //那些點有dp[v][k - x - 2]，最後0.5是由於計算中i -> j以及j -> i(i、j是不同節點)
    //都會被算一遍，所以要 * 0.5
    long long cnt = 0;
    for (int v: G[u]) {
        if (v == p)
            continue;
        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() {
    scanf("%d %d", &n, &k);
    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);
    }
    dfs(1, -1);
    printf("%lld\n", res);
    return 0;
}
```

## 7.9 TreeDP reroot

```
/*Re-root經典題
1. 選0作為root
2. 以0為root去求出所有節點的subtreeSize
3. 觀察到re-root後的關係式
配合思考圖片
f(0)與f(2)的關係
f(2) = f(0) + a - b
a = n - b, (subtree(2)以外的節點)
b = subtreeSize(2), (subtree(2))
所以f(n)是n為root到所有點的距離
f(2) = f(0) + n - 2 * subtreeSize(2)
這就是快速得到答案的轉移式
f(child) = f(parent) + n - 2 * subtreeSize(child)
流程
    1. root = 0去求各項subtreeSize
    2. 求f(root)
    3. 以f(0)去求出re-root後的所有f(v), v != 0
整體來說
暴力解 O(n ^ 2)
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);
            }
        }
    }
};
```

## 7.10 WeightedLIS

```
/*概念基本上與LIS相同，但不能用greedy的LIS，所以只能用dp版LIS
  但有個問題是dp版要O(n^2)
  n最大200000一定超時，所以這題要改一下dp的LIS
  在DP版中有一層迴圈是要往前搜height[j] < height[i](j in 1 ~ i - 1)的然後挑B[j]最大的
  這for loop造成O(n ^ 2)
  注意到子問題是在1 ~ i - 1中挑出B[j]最大的
  這一步可以用線段樹優化
  所以最後可以在O(nlogn)完成*/
#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;
}
```