Constants Polisher

Beijing U of Posts and Tel

Unknown

2019 年 10 月 31 日

目录

1 string

1.1 KMP

```
#include <iostream>
#include <string>
using namespace std;
const int MAXN=1000010;

int nex1[MAXN];

void dopattern(string pat){
    nex1[0]=-1;
    int k=-1,j=0;
    int patlen=pat.size();
    while(j<patlen){
        if(k==-1 || pat[j]==pat[k]){
            ++k;++j;
            if(pat[j]!=pat[k])nex1[j]=k;
            else nex1[j]=nex1[k];
        }else{
            k=nex1[k];
        }
    }
}

int search(string str,string pattern){
    dopattern(pattern);
    int i=0,j=0;
    int strlen=str.size();
    int patlen=pattern.size();
    while(i<strlen && j<patlen){
        if(j==-1 || str[i]==pattern[j]){
            i++;j++;
        }else{
            j=nex1[j];
        }
    }
    if(j==pattern.size())return i-j;
    else return -1;
}
```

```cpp
int main(){
    string str;cin>>str;
    int plen;cin>>plen;
    while(plen--){
        string pat;cin>>pat;
        int res=search(str,pat);
        if(res!=-1){
            cout<<str.substr(res)<<endl;
        }else{
            cout<<"Not Found"<<endl;
        }
    }

    return 0;
}
```

1.2 KMP 2

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;
const int MAXN = 1000010;

int next1[MAXN];
void dopattern(string pat){
    next1[0]=-1;
    int k=-1;
    for(int i=1;i<pat.size();i++){
        while(k>-1 && pat[k+1]!=pat[i])k=next1[k];
        if(pat[k+1]==pat[i])k++;
        next1[i]=k;
    }
}

void doval(string pat){
    //fix -1
    for(int i=1;i<pat.size();i++){
        if(next1[i]==-1)next1[i]=0;
    }
    //for(int i=1;i<pat.size();i++){
    //    if(next1[i]!=-1 && pat[i]==pat[next1[i]])next1[i]=next1[next1[i]];
    //}
}
int kmp(string str,string pat){
    int res=0;
    dopattern(pat);
    int k=-1;
    for(int i=0;i<str.size();i++){
        while(k>-1 && pat[k+1]!=str[i])k=next1[k];
        if(pat[k+1]==str[i])k++;
        if(k==pat.size()-1){
            //position at i-pat.size()+1;
            k=-1;
            i=i-pat.size()+1;
            res++;
        }
    }
    return res;
}

int main(){
    for(int k=1;;k++){
        int len;cin>>len;
        if(len==0)break;

        string pat;cin>>pat;
        dopattern(pat);
        //for(int i=0;i<pat.size();i++){
        //    cout<<next1[i]<<" ";
        //}
        //cout<<endl;

        cout<<"Test case #"<<k<<endl;
        for(int i=1;i<pat.size();i++){
            int pei=(i+1)-(next1[i]+1);
            if(pei==0)continue;
            if((i+1)%pei==0 && (i+1)/pei>1){
                cout<<i+1<<" "<<(i+1)/pei<<endl;
            }
        }
        cout<<endl;
    }



    return 0;
}
```

1.3 SA

```
#include <bits/stdc++.h>
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

// SA Template
/*
 * save your string in array x starting from index 1.
 */

int n;
int sa[150], x[150], c[150], y[150];
char a[150];

inline void SA()
{
    int m = 128;
    for (int i = 0; i <= m; i++)
        c[i] = 0;
    for (int i = 1; i <= n; i++)
        c[x[i]]++;
    for (int i = 1; i <= m; i++)
        c[i] += c[i - 1];
    for (int i = n; i; i--)
```

```
        sa[c[x[i]]--] = i;

    for (int k = 1, p; k <= n; k <<= 1)
    {
        p = 0;
        for (int i = n; i > n - k; i--)
            y[++p] = i;
        for (int i = 1; i <= n; i++)
            if (sa[i] > k)
                y[++p] = sa[i] - k;

        for (int i = 0; i <= m; i++)
            c[i] = 0;
        for (int i = 1; i <= n; i++)
            c[x[i]]++;
        for (int i = 1; i <= m; i++)
            c[i] += c[i - 1];
        for (int i = n; i; i--)
            sa[c[x[y[i]]]--] = y[i];

        p = y[sa[1]] = 1;
        for (int i = 2, a, b; i <= n; i++)
        {
            a = sa[i] + k > n ? -1 : x[sa[i] + k];
            b = sa[i - 1] + k > n ? -1 : x[sa[i - 1] + k];
            y[sa[i]] = (x[sa[i]] == x[sa[i - 1]]) && (a == b) ? p : ++p;
        }
        swap(x, y);
        m = p;
    }
}

int main()
{
    scanf("%s", a + 1);

    n = strlen(a + 1);
    for (int i = 1; i <= n; i++)
        x[i] = a[i];
    SA();
    for (int i = 1; i <= n; i++)
        printf("%d", sa[i]);
    exit(0);
}

struct SuffixArray {
    int sa[XN],rank[XN],height[XN],n;

    SuffixArray(const char *s):n(strlen(s+1)) {
        static int temp[2][XN],cnt[XN],*x=temp[0],*y=temp[1];
        int m=256;
        std::fill(cnt+1,cnt+1+m,0);
        for(int i=1;i<=n;++i) cnt[x[i]=s[i]]++;
        std::partial_sum(cnt+1,cnt+1+m,cnt+1);
        for(int i=n;i>=1;--i) sa[cnt[x[i]]--]=i;
        for(int len=1;len<n;len<<=1) {
            int p=0;
            for(int i=n-len+1;i<=n;++i) y[++p]=i;
            for(int i=1;i<=n;++i) if(sa[i]>len) y[++p]=sa[i]-len;
            std::fill(cnt+1,cnt+1+m,0);
            for(int i=1;i<=n;++i) cnt[x[i]]++;
            std::partial_sum(cnt+1,cnt+1+m,cnt+1);
            for(int i=n;i>=1;--i) sa[cnt[x[y[i]]]--]=y[i];
            std::swap(x,y);x[sa[1]]=p=1;
            for(int i=2;i<=n;++i)
                x[sa[i]]=y[sa[i-1]]==y[sa[i]]
                      && (sa[i-1]+len<=n?y[sa[i-1]+len]:0)==
                      (sa[i]+len<=n?y[sa[i]+len]:0)?p:++p;
            if((m=p)==n) break;
        }
        for(int i=1;i<=n;++i) rank[sa[i]]=i;
        for(int i=1,len=0;i<=n;++i)
            if(rank[i]!=1) {
                int j=sa[rank[i]-1];
                while(s[i+len]==s[j+len]) ++len;
                height[rank[i]]=len;
                if(len) len--;
            }
    }
```

```
};
```

1.4 SAM

```
//
/*
查询子串是否出现：这显然跑一次，能在自动机上跑完就是出现过。
统计不同子串的数量：自动机上每条不同的路径对应一个不同的子串。定义 \(d(x)\) 为以 x 为起点的路径数目，递推即可。
计算所有不同子串的长度总和：得到上面的 \(d\)。以 x 为起点，每条路径都会让子串总长度增加路径个。依然是递推。
字典序第 k 小子串：当你有路径数了，只需要按照字典序对节点排序，然后像编码一样找。
最小循环移位：指将原字符串首尾相接移位，找到字典序最小的一个。将字符串 \(S\) 断环成链 \(SS\)，然后建立 SAM，贪心找最小直到长度达到 \(|S|\) 即可。
多组子串出现次数：dfs 预处理每个节点的终点集合大小。在自动机上查找串 \(P\) 对应的节点，存在则答案为该节点的终点集合大小；不存在答案为 \(0\).
所有出现位置：遍历树，一旦发现终点直接输出。
*/
#include <map>

struct state
{
    int len, link;
    std::map<char, int> next;
};

const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;

void sam_init()
{
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}
void sam_extend(char c)
{
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c))
    {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1)
    {
        st[cur].link = 0;
    }
    else
    {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len)
        {
            st[cur].link = q;
        }
        else
        {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q)
            {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

1.5 Manacher

```
#include <vector>
#include <string>
using namespace std;
```

```
void manacher_odd(string str)
{
    int n = str.size();
    vector<int> d1(n);
    for (int i = 0, l = 0, r = -1; i < n; i++)
    {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
        while (0 <= i - k && i + k < n && str[i - k] == s[i + k])
        {
            k++;
        }
        d1[i] = k--;
        if (i + k > r)
        {
            l = i - k;
            r = i + k;
        }
    }
}

// You can also add # to middle and only use `odd` one.
void manacher_even(string str)
{
    int n=str.size();
    vector<int> d2(n);
    for (int i = 0, l = 0, r = -1; i < n; i++)
    {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n && str[i - k - 1] == s[i + k])
        {
            k++;
        }
        d2[i] = k--;
        if (i + k > r)
        {
            l = i - k - 1;
            r = i + k;
        }
    }
}


//Possible Version 2
void Manacher(char *str,int rad[])//str 是原字符串 ma 是加入新字符后的以 i 为中心极大回文子串的半长度
{
    int len=strlen(str+1),l=0;
    for(int i=1;i<=len;++i){
        s[++l]='$';
        s[++l]=str[i];
    }
    s[++l]='$';s[0]='#';//s 是加入新字符后的字符串
    rad[1]=1;
    int R=1,ID=1;//R 是当前极长回文子串的最右的端点 ID 为 R 对应的回文子串的中心
    for(int i=1;i<=l;++i){
        if(i<R)
            rad[i]=min(rad[2*ID-i],R-i+1);//2*ID-i 为 i 在当前这个极长回文子串中在左边相对应的位置
        else
            rad[i]=1;
        for(;s[i+rad[i]]==s[i-rad[i]];++rad[i]);
        if(R<rad[i]+i-1){
            R=rad[i]+i-1;
            ID=i;
        }
    }
    //原字符串的最长回文子串为 max{rad[i]-1}
}
```

1.6 huiwenauto

```
#include <cstring>
using namespace std;

//变量名与上文基本相同，其中 ptr 为转移指针，数组大小应为字符集大小
class PA
{
private:
    static const int N = 100010;
    struct Node
    {
        int len;
        int ptr[26], fail;
        Node(int len = 0) : len(len), fail(0) { memset(ptr, 0, sizeof(ptr)); }
    } nd[N];

    int size, cnt; // size 为字符串长度，cnt 为节点个数
    int cur;       //当前指针停留的位置，即最后插入字符所对应的节点
    char s[N];

    int getfail(int x) //沿着 fail 指针找到第一个回文后缀
    {
        while (s[size - nd[x].len - 1] != s[size])
        {
            x = nd[x].fail;
        }
        return x;
    }

public:
    PA() : size(0), cnt(0), cur(0)
    {
        nd[cnt] = Node(0);
        nd[cnt].fail = 1;
        nd[++cnt] = Node(-1);
        nd[cnt].fail = 0;
        s[0] = '$';
    }

    void extend(char c)
    {
        s[++size] = c;
        int now = getfail(cur);    //找到插入的位置
        if (!nd[now].ptr[c - 'a']) //若没有这个节点，则新建并求出它的 fail 指针
        {
            int tmp = ++cnt;
            nd[tmp] = Node(nd[now].len + 2);
            nd[tmp].fail = nd[getfail(nd[now].fail)].ptr[c - 'a'];
            nd[now].ptr[c - 'a'] = tmp;
        }
        cur = nd[now].ptr[c - 'a'];
    }

    int qlen() { return nd[cur].len; }
} A, B;
```

1.7 lyndon

```
//如果 s 的字典序严格小于 s 的所有后缀的字典序，称 s 是简单串，或者 Lyndon 串 。

//分解:s=w1w2w3w4..... 分解后 w1>=w2>=w3>=..., 该分解唯一
// duval_algorithm
vector<string> duval(string const &s)
{
    int n = s.size(), i = 0;
    vector<string> factorization;
    while (i < n)
    {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j])
        {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k)
        {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}

//基于以上的最小表示法
// smallest_cyclic_string
string min_cyclic_string(string s)
```

```
{
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2)
    {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j])
        {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k)
            i += j - k;
    }
    return s.substr(ans, n / 2);
}
```

1.8 suffixBalancedTree

```
struct SuffixBalancedTree {
    static const double alpha=0.8;

    struct Node {
        Node *son[2];
        double l,r,tag;
        int size,ndct;
        bool exist;
        char ch;
        Node *next;

        Node(double l,double r,char ch,Node
         ↪ *next):l(l),r(r),tag((l+r)/2),size(1),ndct(1),exist(1),ch(ch),next(next)
         ↪ {
            son[0]=son[1]=null;
        }
        Node(void*) {
            size=ndct=exist=0;
            ch=0;
            tag=-1;
            son[0]=son[1]=0;
        }

        void Up() {
            ndct=son[0]->ndct+1+son[1]->ndct;
            size=son[0]->size+exist+son[1]->size;
        }

        bool Unbalanced() {
            return ndct*alpha<std::max(son[0]->ndct,son[1]->ndct);
        }
    }*root;

    std::stack<Node*> nodes;

    static Node *null;

    SuffixBalancedTree():root(null) {
        nodes.push(null);
    }

    Node *&Insert(Node *&pos,double l,double r,char ch,Node *next) {
        if(pos==null) {
            pos=new Node(l,r,ch,next);
            nodes.push(pos);
            return null;
        } else {
            Node *&goat=ch<pos->ch || (ch==pos->ch && next->tag<pos->next->tag)
                        ?Insert(pos->son[0],l,(l+r)/2,ch,next)
                        :Insert(pos->son[1],(l+r)/2,r,ch,next);
            pos->Up();
            return pos->Unbalanced()?pos:goat;
        }
    }
```

```
static Node *Flatten(Node *pos,Node *app) {
    if(pos==null)
        return app;
    else {
        pos->son[1]=Flatten(pos->son[1],app);
        return Flatten(pos->son[0],pos);
    }
}

static std::pair<Node*,Node*> Rebuild(Node *begin,double l,double r,int n) {
    if(n==0) {
        return std::pair<Node*,Node*>(null,begin);
    } else {
        int mid=(1+n)/2;
        std::pair<Node*,Node*> left=Rebuild(begin,l,(l+r)/2,mid-1);
        Node *pos=left.second;
        std::pair<Node*,Node*> right=Rebuild(pos->son[1],(l+r)/2,r,n-mid);
        pos->son[0]=left.first;
        pos->son[1]=right.first;
        pos->l=l,pos->r=r,pos->tag=(l+r)/2;
        pos->Up();
        return std::pair<Node*,Node*>(pos,right.second);
    }
}

static void Rebuild(Node *&root) {
    Node *begin=Flatten(root,null);
    root=Rebuild(begin,root->l,root->r,root->ndct).first;
}

static void Delete(Node *pos,Node *del) {
    if(pos==del) {
        pos->exist=0;
        pos->Up();
    } else {
        Delete(pos->son[pos->tag<del->tag],del);
        pos->Up();
    }
}
```

```
    int LessCount(const char *s) {
        int res=0;
        for(Node *pos=root;pos!=null;) {
            Node *p=pos;
            const char *c=s;
            while(p->ch==*c) {
                p=p->next;
                ++c;
            }
            if(p->ch<*c) {
                res+=pos->son[0]->size+pos->exist;
                pos=pos->son[1];
            } else
                pos=pos->son[0];
        }
        return res;
    }

    void Append(char ch) {
        Node *&goat=Insert(root,0,1,ch,nodes.top());
        if(goat!=null)
            Rebuild(goat);
    }

    void Pop() {
        Delete(root,nodes.top());
        nodes.pop();
    }

    int Count(char *s,int len) {
        s[len+1]=CHAR_MAX;
        int res=LessCount(s+1);
        s[len+1]=CHAR_MIN;
        res-=LessCount(s+1);
        //null's ch must satisfy CHAR_MIN < ch < ALL
        return res;
    }
};
const double SuffixBalancedTree::alpha;
```

```
SuffixBalancedTree::Node *SuffixBalancedTree::null=new SuffixBalancedTree::Node((void*)0);
```

1.9 extendedKMP

```
//nxt 表示 B[i..m] 与 B 的最长公共前缀
//extend 表示 A[i..n] 与 B 的最长公共前缀长度
void exKMP(char *A,char *B,int nxt[],int extend[]) {
    int n=strlen(A+1),m=strlen(B+1),x=1;
    nxt[1]=m;
    for(;x<m&&B[x]==B[x+1];++x);
    nxt[2]=x-1;x=2;

    for(int i=3;i<=m;++i)
        if(i+nxt[i-x+1]-1<nxt[x]+x-1)nxt[i]=nxt[i-x+1];
        else{
            int j=nxt[x]+x-i+1;
            if(j<1)j=1;
            for(;j+i-1<=m&&B[j]==B[j+i-1];++j);
            nxt[i]=j-1;
            x=i;
        }

    x=1;
    for(;A[x]==B[x];++x);
    extend[1]=x-1;
    x=1;
    for(int i=2;i<=n;++i)
        if(i+nxt[i-x+1]-1<extend[x]+x-1)extend[i]=nxt[i-x+1];
        else{
            int j=extend[x]+x-i+1;
            if(j<1)j=1;
            for(;j+i-1<=n&&B[j]==A[j+i-1];++j);
            nxt[i]=j-1;
            if(nxt[x]<=nxt[i])x=i;
        }
}
```

1.10 minimumRepresentation

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

2 utility

2.1 qread 1

```
template<class T>
inline void read(T &x) {
    x = 0; int f = 0; char ch = getchar();
    while (!isdigit(ch)) { if (ch == '-') f = 1; if (ch == EOF) return; ch =
    ↪  getchar(); }
    while (isdigit(ch)) { x = x * 10 + (ch - '0'); ch = getchar(); } if (f) x =
    ↪  -x;
}
```

2.2 qread 2

```
//更快的

#include <cstdlib>
#include <cstdio>
using namespace std;

// #define DEBUG 1  //调试开关
```

```
struct IO
{
#define MAXSIZE (1 << 20)
#define isdigit(x) (x >= '0' && x <= '9')
    char buf[MAXSIZE], *p1, *p2;
    char pbuf[MAXSIZE], *pp;
#if DEBUG
#else
    IO() : p1(buf), p2(buf), pp(pbuf)
    {
    }
    ~IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }
#endif
    inline char gc()
    {
#if DEBUG //调试，可显示字符
        return getchar();
#endif
        if (p1 == p2)
            p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
        return p1 == p2 ? ' ' : *p1++;
    }
    inline bool blank(char ch)
    {
        return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';
    }
    template <class T>
    inline void read(T &x)
    {
        register double tmp = 1;
        register bool sign = 0;
        x = 0;
        register char ch = gc();
        for (; !isdigit(ch); ch = gc())
            if (ch == '-')
                sign = 1;
        for (; isdigit(ch); ch = gc())
            x = x * 10 + (ch - '0');
        if (ch == '.')
            for (ch = gc(); isdigit(ch); ch = gc())
```

```
                tmp /= 10.0, x += tmp * (ch - '0');
        if (sign)
            x = -x;
    }
    inline void read(char *s)
    {
        char ch = gc();
        for (; blank(ch); ch = gc())
            ;
        for (; !blank(ch); ch = gc())
            *s++ = ch;
        *s = 0;
    }
    inline void read(char &c)
    {
        for (c = gc(); blank(c); c = gc())
            ;
    }
    inline void push(const char &c)
    {
#if DEBUG //调试，可显示字符
        putchar(c);
#else
        if (pp - pbuf == MAXSIZE)
            fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
        *pp++ = c;
#endif
    }
    template <class T>
    inline void write(T x)
    {
        if (x < 0)
            x = -x, push('-'); // 负数输出
        static T sta[35];
        T top = 0;
        do
        {
            sta[top++] = x % 10, x /= 10;
        } while (x);
        while (top)
```

```
            push(sta[--top] + '0');
    }
    template <class T>
    inline void write(T x, char lastChar)
    {
        write(x), push(lastChar);
    }
} io;
```

2.3 random num

```
#include <ctime>
#include <iostream>
#include <random>

using namespace std;

int main() {
  mt19937 myrand(time(0));
  cout << myrand() << endl;
  return 0;
}
```

2.4 ArrayPointer

```
template <class T>
struct ArrayPointer {
    int id;

    ArrayPointer(T *x=0) {
        if(!x)
            id=-1;
        else
            a[id=cnt++]=*x;
    }

    T *operator ->() {
        return a+id;
    }

    T &operator *() {
        return a[id];
    }

    static T *a;
    static int cnt;
};

/*
template <> TypeName
 ↪  *ArrayPointer<TypeName>::a=(TypeName*)malloc(SIZE*sizeof(TypeName));
template <> int ArrayPointer<TypeName>::cnt=0;
overload operator_new
*/
```

2.5 SharedPointer

```
template <class T>
struct SharedPointer {
    T *ptr;
    int *cnt;

    void Release() {
        if(ptr && --*cnt==0) {
            delete ptr;
            delete cnt;
        }
    }

    SharedPointer():ptr(0),cnt(0) {}

    SharedPointer(T *p):ptr(0) {
        *this=p;
    }

    SharedPointer(SharedPointer const &other):ptr(0) {
        *this=other;
    }

    ~SharedPointer() {
        Release();
    }

    T *operator ->() {
        return ptr;
    }

    T &operator *() {
        return *ptr;
    }

    bool operator ==(SharedPointer const &other) const {
        return ptr==other.ptr;
    }

    bool operator !=(SharedPointer const &other) const {
        return ptr!=other.ptr;
    }

    SharedPointer &operator =(T *p) {
        Release();
        if(p) {
            ptr=p;
            (*(cnt=new int))=1;
        } else {
            ptr=0;
            cnt=0;
        }
        return *this;
    }

    SharedPointer &operator =(SharedPointer const &other) {
        Release();
        if(other.ptr) {
            ptr=other.ptr;
            (*(cnt=other.cnt))++;
        } else {
            ptr=0;
            cnt=0;
        }
        return *this;
    }
};
```

2.6 moAlgOnTree

```
const int XN=4e4+11,XM=1e5+11;

std::vector<int> G[XN];
int lbd[XN],rbd[XN],dfs[XN*2],dc,block[XN*2],w[XN],par[XN][20],dep[XN],lg2[XN];
void DFS(int pos) {
    dep[pos]=dep[par[pos][0]]+1;
    for(int b=1;b<=lg2[dep[pos]];++b)
        par[pos][b]=par[par[pos][b-1]][b-1];
    dfs[lbd[pos]=++dc]=pos;
    for(int u : G[pos])
        if(!lbd[u]) {
            par[u][0]=pos;
            DFS(u);
        }
    dfs[rbd[pos]=++dc]=pos;
}

int LCA(int x,int y) {
    if(dep[x]<dep[y])
        std::swap(x,y);
    for(int len=dep[x]-dep[y],b=lg2[len];b>=0;--b)
        if(len>>b&1)
            x=par[x][b];
    if(x!=y) {
        for(int b=lg2[dep[x]];b>=0;--b)
            if(par[x][b]!=par[y][b]) {
                x=par[x][b];
                y=par[y][b];
            }
        x=par[x][0];
    }
    return x;
}
```

```
struct Query {
    int l,r,lca,*ans;
};

void Solve(std::vector<Query> &query) {
    std::sort(query.begin(),query.end(),[&](auto const &a,auto const &b)->bool {
        return
         ↪  block[a.l]==block[b.l]?(a.r<b.r)^(block[a.l]&1):block[a.l]<block[b.l];
    });

    static int occ[XN],Ans;

    static auto Modify=[&](int c,int sig) {

    };

    static auto Update=[&](int pos,int sig) {
        Modify(w[pos],(occ[pos]+=sig)==1?1:-1);
    };

    int l=1,r=0;
    for(auto &q : query) {
        for(;l>q.l;Update(dfs[--l],1));
        for(;r<q.r;Update(dfs[++r],1));
        for(;l<q.l;Update(dfs[l++],-1));
        for(;r>q.r;Update(dfs[r--],-1));
        if(q.lca) {
            assert(occ[q.lca]==0);
            Modify(w[q.lca],1);
            *q.ans=Ans;
            Modify(w[q.lca],-1);
        } else
            *q.ans=Ans;
    }
}

void Prep() {
    DFS(1);
    static int Ans[XM];
    std::vector<Query> q;
    for(int i=1;i<=m;++i) {
        int u,v;cin>>u>>v;
        if(lbd[u]>lbd[v])
            std::swap(u,v);
        int lca=LCA(u,v);
        if(lca==u)
            q.push_back({lbd[u],lbd[v],0,&Ans[i]});
        else
            q.push_back({rbd[u],lbd[v],lca,&Ans[i]});
    }
    int size=sqrt(2*n);//can be modified
    for(int i=1;i<=2*n;++i)
        block[i]=(i-1)/size;
    Solve(q);
}
```

2.7 moAlgOnSequence

```
int block[XM],m;
struct Query {
    int l,r;
    unsigned int *ans;
}q[XM];

void Solve() {
    std::sort(q+1,q+1+m,[&](auto const &a,auto const &b)->bool {
        return
         ↪  block[a.l]==block[b.l]?(a.r<b.r)^(block[a.l]&1):block[a.l]<block[b.l];
    });

    unsigned int Ans=0;
    static auto Update=[&](int pos,int sig) {

    };
    int l=1,r=0;
    for(int i=1;i<=m;++i) {
        for(;l>q[i].l;Update(--l,1));
        for(;r<q[i].r;Update(++r,1));
        for(;l<q[i].l;Update(l++,-1));
        for(;r>q[i].r;Update(r--,-1));
        *q[i].ans=Ans;
    }
    while(l<=r)
        Update(r--,-1);
}
```

2.8 qread 3

```
namespace IO {
    const int IN=1e6;
    char in[IN],*ip=in,*ie=in;
    #define getchar() (ip==ie &&
     ↪  (ie=(ip=in)+fread(in,1,IN,stdin),ip==ie)?EOF:*ip++)
    struct Istream {
        template <class T>
        Istream &operator >>(T &x) {
            static char ch;static bool neg;
            for(ch=neg=0;ch<'0' || '9'<ch;neg|=ch=='-',ch=getchar());
            for(x=0;'0'<=ch && ch<='9';(x*=10)+=ch-'0',ch=getchar());
            x=neg?-x:x;
            return *this;
        }
    }fin;

    const int OUT=1e6;
    char out[OUT],*op=out,*oe=out+OUT;
    #define flush() fwrite(out,1,op-out,stdout)
    #define putchar(x) ((op==oe?(flush(),op=out,*op++):*op++)=(x))
    struct Ostream {
        ~Ostream() {
            flush();
        }
        template <class T>
        Ostream &operator <<(T x) {
            x<0 && (putchar('-'),x=-x);
            static char stack[233];static int top;
            for(top=0;x;stack[++top]=x%10+'0',x/=10);
            for(top==0 && (stack[top=1]='0');top;putchar(stack[top--]));
            return *this;
        }
        Ostream &operator <<(char ch) {
            putchar(ch);
            return *this;
        }
    }fout;
}

using IO::fin;
using IO::fout;
```

3 math

3.1 cipolla algorithm

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
using ll=long long;
const int P=998244353;

//二次剩余算法
//solve a x to `x^2=a (mod p)` s.t. p is a prime and doesn't divide a
//this alogirthm relys on the posibility.
//expected complexity O(lg^2 n)
inline int Pow(ll x, int y=P-2){
    int ans=1;
    for(; y; y>>=1, x=x*x%P) if(y&1) ans=ans*x%P;
    return ans;
}
inline pair<int,int> pMul(pair<int,int> x, pair<int,int> y, int f){
    return make_pair(
        (int)(((ll)x.first*y.first+(ll)x.second*y.second%P*f)%P),
        (int)(((ll)x.second*y.first+(ll)x.first*y.second)%P)
    );
}
inline int Quadratic_residue(int a){
    if(Pow(a, (P-1)/2)!=1) return -1;
    int x, f;
    do x=(((ll)rand()<<15)^rand())%(a-1)+1; while(Pow(f=((ll)x*x-a+P)%P,
     ↪  (P-1)/2)==1);
    pair<int,int> ans=make_pair(1, 0), t=make_pair(x, 1);
    for(int i=(P+1)/2; i; i>>=1, t=pMul(t, t, f)) if(i&1) ans=pMul(ans, t, f);
    return min(ans.first, P-ans.first);
}

int main(){
    cout<<Quadratic_residue(17)<<endl;

    return 0;
}
```

3.2 fft

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <complex>
#include <cmath>
#include <cstdlib>
using namespace std;
const double PI=acos(-1);

inline complex<double> gomega(int n,int k,bool rev=false){
    complex<double> res(cos(2*PI/n*k),sin(2*PI/n*k));
    if(rev)return conj(res);
    else return res;
}
const int MAXN=10;
struct FFT{
    complex<double> omega[MAXN],omegaI[MAXN];

    FFT(int n){
        for(int i=0;i<n;i++){
            omega[i]=gomega(n,i);
            omegaI[i]=gomega(n,i,1);
        }
    }
    void transform(complex<double> *a,int n,const complex<double> *omega){
        int k=0;
        while((1<<k)<n)k++;
        for(int i=0;i<n;i++){
            int t=0;
            for(int j=0;j<k;j++)if(i&(1<<j))t|=(1<<(k-j-1));
            if(i<t)swap(a[i],a[t]);
        }
        for(int l=2;l<=n;l*=2){
            int m=l/2;
            for(complex<double> *p=a;p!=a+n;p+=l){
                for(int i=0;i<m;i++){
                    complex<double> t=omega[n/l*i]*p[m+i];
                    p[m+i]=p[i]-t;
                    p[i]+=t;
                }
            }
        }
    }
    void dft(complex<double> *a,int n){
        transform(a,n,omega);
    }
    void idft(complex<double> *a,int n){
        transform(a,n,omegaI);
        for(int i=0;i<n;i++)a[i]/=n;
    }
};



int main(){

    return 0;
}
```

3.3 fft mul

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <complex>
```

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

const int MAXN=300000;
const double PI=acos(-1);

inline complex<double> gomega(int n,int k,bool rev=false){
    complex<double> res(cos(2*PI/n*k),sin(2*PI/n*k));
    if(rev)return conj(res);
    else return res;
}
struct FFT{
    complex<double> omega[MAXN],omegaI[MAXN];

    FFT(){
    }
    void init(int n){
        for(int i=0;i<n;i++){
            omega[i]=gomega(n,i);
            omegaI[i]=gomega(n,i,1);
        }
    }

    void transform(complex<double> *a,int n,const complex<double> *omega){
        for(int i=0,j=0;i<n;i++){
            if(i>j)swap(a[i],a[j]);
            //二进制换位
            for(int l=n/2;(j^=l)<l;l>>=1);
        }
        for(int l=2;l<=n;l<<=1){
            int m=l/2;
            for(complex<double> *p=a;p!=a+n;p+=l){
                for(int i=0;i<m;i++){
                    //蝴蝶操作
                    complex<double> t=omega[n/l*i]*p[m+i];
                    p[m+i]=p[i]-t;
                    p[i]+=t;
                }
            }
        }
    }
    void dft(complex<double> *a,int n){
        transform(a,n,omega);
    }
    void idft(complex<double> *a,int n){
        transform(a,n,omegaI);
        for(int i=0;i<n;i++)a[i]/=n;
    }
};

complex<double> a[2][MAXN];
int ans[MAXN];
FFT fft;
int main(){

    int nlen;cin>>nlen;
    int n=1;
    while(n<2*nlen)n*=2;
    fft.init(n);


    for(int i=0;i<2;i++){
        string inp;cin>>inp;
        for(int j=0,k=inp.size()-1;j<inp.size();j++,k--){
            a[i][j]=complex<double>(inp[j]-'0',0);
        }
        fft.dft(a[i],n);
    }
    for(int i=0;i<n;i++)a[0][i]=a[0][i]*a[1][i];
    fft.idft(a[0],n);
    int reslen=nlen+nlen-1;
    for(int i=reslen-1,k=0;i>=0;i--,k++)
        ans[k]=(int)floor(a[0][i].real()+0.5);
    /*
    for(int i=0;i<reslen;i++)cout<<ans[i]<<" ";
    cout<<endl;
    */
    for(int i=0;i<MAXN;i++){
        ans[i+1]+=ans[i]/10;
        ans[i]%=10;
    }
    int ptr=MAXN-1;
    while(ans[ptr]==0)ptr--;
    for(;ptr>=0;ptr--)cout<<ans[ptr];
    cout<<endl;

    return 0;
}
```

3.4 linear-basis

```
#include <iostream>
using namespace std;
using ll=long long;

constexpr int MAXN=60;
constexpr int MAXBASE=60;

ll num[MAXN];
ll basis[MAXBASE];

int nlen;
int cal(){
    for(int i=0;i<nlen;i++){
        for(int j=MAXBASE-1;j>=0;j--){
            if(num[i]>>j&1){
                if(basis[j])num[i]^=basis[j];
                else{
                    basis[j]=num[i];
                    for(int k=j-1;k>=0;k--)if((basis[j]>>k&1) &&
                    ↪  basis[k])basis[j]^=basis[k];
                    for(int
                    ↪  k=j+1;k<MAXBASE;k++)if(basis[k]>>j&1)basis[k]^=basis[j];
                    break;
                }
            }
        }
    }
    return 0;
}
```

```
int main(){
    cin>>nlen;
    for(int i=0;i<nlen;i++)cin>>num[i];
    cal();
    ll ans=0;
    for(int i=0;i<MAXBASE;i++)ans^=basis[i];
    cout<<ans<<endl;

    return 0;
}
```

3.5 simpson

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <algorithm>
#include <vector>
using namespace std;
double a,b;
double f(double x){
    return 2*sqrt(b*b*(1-x*x/a/a));
}
inline double fittingf(double l,double r){
    return (f(l)+f(r)+4*f((l+r)/2))*(r-l)/6;
}
double adaptive_simpson(double l,double r,double eps){
    double mid=(l+r)/2;
    double res=fittingf(l,r);
    double lr=fittingf(l,mid),rr=fittingf(mid,r);
    if(abs(lr+rr-res)<15*eps)
        return lr+rr+(lr+rr-res)/15;
    else
        return adaptive_simpson(l,mid,eps/2)+adaptive_simpson(mid,r,eps/2);
}
```

```
int main(){
    int kase;cin>>kase;
    while(kase--){
        double l,r;
        cin>>a>>b>>l>>r;
        cout<<fixed<<setprecision(3)<<adaptive_simpson(l,r,1e-7)<<endl;
    }

    return 0;
}
```

3.6 fraction

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

template<typename T>
struct Frac{
    T a,b;
    Frac(T a=0,T b=1):a(a),b(b){}
    static T gcd(T a,T b){
        return !b?a:gcd(b,a%b);
    }
    void simplify(){
        T g=gcd(abs(a),abs(b));
        if(g==0)return;
        a/=g;
        b/=g;
    }
    Frac operator*(const Frac &other)const{
        Frac res=*this;
        res*=other;
        return res;
    }
    Frac operator/(const Frac &other)const{
        Frac res=*this;
        res/=other;
        return res;
    }
    Frac operator+(const Frac &other)const{
        Frac res=*this;
        res+=other;
        return res;
    }
    Frac operator-(const Frac &other)const{
        Frac res=*this;
        res-=other;
        return res;
    }

    Frac& operator+=(const Frac &other){
        a=a*other.b+b*other.a;
        b*=other.b;
        simplify();
        return *this;
    }
    Frac& operator-=(const Frac &other){
        a=a*other.b-b*other.a;
        b*=other.b;
        simplify();
        return *this;
    }
    Frac& operator*=(const Frac &other){
        a*=other.a;
        b*=other.b;
        simplify();
        return *this;
    }
    Frac& operator/=(const Frac &other){
        a*=other.b;
        b*=other.a;
        simplify();
        return *this;
    }
};
```

3.7 millerRabinAndPollardRho

```
namespace MillerRabin {
    long long Mul(long long a,long long b,long long mo){
        long long tmp=a*b-(long long)((long double)a/mo*b+1e-8)*mo;
        return (tmp%mo+mo)%mo;
    }

    long long Pow(long long a,long long b,long long mo){
        long long res=1;
        for(;b;b>>=1,a=Mul(a,a,mo))
            if(b&1)
                res=Mul(res,a,mo);
        return res;
    }

    bool IsPrime(long long n){
        if(n==2)return 1;
        if(n<2||!(n&1))return 0;
        static const auto tester={2,3,5,7,11,13,17,19,23};
        long long x=n-1;int t=0;
        for(;!(x&1);x>>=1)++t;
        for(int p : tester) {
            long long a=p%(n-1)+1,res=Pow(a%n,x,n),last=res;
            for(int j=1;j<=t;++j){
                res=Mul(res,res,n);
                if(res==1&&last!=1&&last!=n-1)return 0;
                last=res;
            }
            if(res!=1)return 0;
        }
        return 1;
    }
}

namespace PollardRho {
    using namespace MillerRabin;
    unsigned long long seed;

    long long Rand(long long mo){
        return (seed+=4179340454199820289ll)%mo;
    }

    long long F(long long x,long long c,long long mo){
        return (Mul(x,x,mo)+c)%mo;
    }

    long long gcd(long long a,long long b){
        return b?gcd(b,a%b):a;
    }

    long long Get(long long c,long long n){
        long long x=Rand(n),y=F(x,c,n),p=n;
        for(;x!=y&&(p==n||p==1);x=F(x,c,n),y=F(F(y,c,n),c,n))
            p=x>y?gcd(n,x-y):gcd(n,y-x);
        return p;
    }

    void Divide(long long n,long long p[]){
        if(n<2)return;
        if(IsPrime(n)){p[++*p]=n;return;}
        for(;;){
            long long tmp=Get(Rand(n-1)+1,n);
            if(tmp!=1&&tmp!=n){
                Divide(tmp,p);
                Divide(n/tmp,p);
                return;
            }
        }
    }
}
```

3.8 simplex

```
//单纯型
namespace Simplex {//(<=)+(Maximize)
    const int XN=0,XM=0;
    const double eps=1e-5,inf=1e100;

    int sgn(double const &x) {
        return (x>-eps)-(x<eps);
    }

    int n,m;
    double a[XM][XN],b[XM],c[XN],v;

    void Pivot(int l,int e) {
        b[l]/=a[l][e];
        for(int i=1;i<=n;++i)
            if(i!=e) a[l][i]/=a[l][e];
        a[l][e]=1/a[l][e];
        for(int i=1;i<=m;++i)
            if(i!=l && sgn(a[i][e])) {
                b[i]-=a[i][e]*b[l];
                for(int j=1;j<=n;++j)
                    if(j!=e)
                        a[i][j]-=a[i][e]*a[l][j];
                a[i][e]*=-a[l][e];
            }
        v+=c[e]*b[l];
        for(int i=1;i<=n;++i)
            if(i!=e)
                c[i]-=c[e]*a[l][i];
        c[e]*=-a[l][e];
    }

    double Run() {
        for(int l,e;(e=std::find_if(c+1,c+1+n,[&](double const &x)->bool {
                        return sgn(x)>0;} )-c)!=n+1;) {
            double lim=inf;
            for(int i=1;i<=m;++i)
                if(IsPositive(a[i][e]) && Reduce(lim,b[i]/a[i][e]))
                    l=i;
            if(lim==inf)
                return inf;
            else
                Pivot(l,e);
        }
        return v;
    }
}
```

3.9 Gauss

```
typedef double Square[XN][XN];
void Gauss(Square A,int n) {
    for(int i=1;i<=n;++i) {
        int id=i;
        for(int j=i+1;j<=n;++j)
            if(abs(A[j][i])>abs(A[id][i]))
                id=j;
        std::swap_ranges(A[i]+1,A[i]+n+2,A[id]+1);
        for(int k=i+1;k<=n+1;++k)
            A[i][k]/=A[i][i];
        A[i][i]=1;
        for(int j=i+1;j<=n;++j) {
            for(int k=i+1;k<=n+1;++k)
                A[j][k]-=A[j][i]*A[i][k];
            A[j][i]=0;
        }
    }
    for(int i=n;i>=1;--i) {
        for(int j=i+1;j<=n;++j) {
            A[i][n+1]-=A[j][n+1]*A[i][j];
            A[i][j]=0;
        }
    }
}
```

3.10 determinant

```
//行列式
typedef int Square[XN][XN];
//Matrix-Tree 度数-邻接
int Determinant(Square a,int n) {
    for(int i=1;i<=n;++i)
        for(int j=1;j<=n;++j)
            ((a[i][j]%=P)+=P)%=P;
    int f=1;
    for(int i=1;i<=n;++i) {
        int &A=a[i][i];
        for(int j=i+1;j<=n;++j) {
            for(int &B=a[j][i];B;f=P-f) {
                int t=A/B;
                for(int k=1;k<=n;++k)
                    a[i][k]=Minus(a[i][k],Mul(a[j][k],t));
                std::swap_ranges(a[i]+1,a[i]+1+n,a[j]+1);
            }
        }
    }
    int res=f;
    for(int i=1;i<=n;++i)
        res=Mul(a[i][i],res);
    return res;
}
```

3.11 FWT

```
template<class T>
void FWT(int a[], int n, T F) {
    for(int len=2;len<=n;len*=2)
        for(int i=0;i<n;i+=len)
            for(int j=i;j<i+len/2;++j)
                F(a[j],a[j+len/2]);
}

void Tand(int &x,int &y) {
    x+=y;
}

void Iand(int &x,int &y) {
    x-=y;
}

void Tor(int &x,int &y) {
    y+=x;
}

void Ior(int &x,int &y) {
    y-=x;
}

void Ixor(int &x,int &y) {
    std::tie(x,y)=std::make_tuple(x+y,x-y);
}

void Txor(int &x,int &y) {
    std::tie(x,y)=std::make_tuple((x+y)/2,(x-y)/2);
}
```

3.12 discreteLogarithmBSGS

```
int BSGS(int y,int z,int P) {
    if(y%P) {
        std::unordered_map<int,int> S;
        int B=sqrt(P)+0.5;
        long long zyi=z;
        for(int i=0;i<=B;i++,(zyi*=y)%=P)
            if(!S.count(zyi))
                S[zyi]=i;
        int yb=Pow(y,B,P);
        long long ybi=yb;
        for(int i=1;i<=B;i++) {
            if(S.count(ybi))
                return B*i-S[ybi];
            (ybi*=yb)%=P;
        }
    }
    return -1;
}
```

3.13 extendedLucas

```
int Exgcd(int a,int b,long long &x,long long &y) {
    if(!b) {
        x=1,y=0;
        return a;
    } else {
        int d=Exgcd(b,a%b,x,y);
        long long t=y;y=x-(a/b)*y,x=t;
        return d;
    }
}
```

```
int Inverse(int a,int n) {
    long long x,y;
    int d=Exgcd(a,n,x,y);
    assert(d==1);
    return (x%n+n)%n;
}

int Pow(long long base,long long v,int P) {
    long long res=1;
    for(;v;v>>=1,(base*=base)%=P)
        if(v&1)
            (res*=base)%=P;
    return res;
}

struct Lucas {
    struct Divisor {
        int p,t,pt,tM;
        std::vector<int> table;

        Divisor(int p,int t,int pt,int tM):p(p),t(t),pt(pt),tM(tM),table(pt) {
            table[0]=1;
            for(int i=1;i<pt;++i)//0?
                table[i]=i%p==0?table[i-1]:(long long)table[i-1]*i%pt;
        }

        int Calc(long long n) {
            if(n<p)//pt..
                return table[n];
            else
                return (long
                 ↪  long)Calc(n/p)*Pow(table[pt-1],n/pt,pt)%pt*table[n%pt]%pt;
        }

        long long CalcTimes(long long x) {
            long long res=0;
            for(;x;x/=p)
                res+=x/p;
            return res;
        }

        long long Solve(long long n,long long m) {
            long long times=CalcTimes(n)-CalcTimes(m)-CalcTimes(n-m);
            if(times>=t)
                return 0;
            else
                return (long long)Pow(p,times,pt) *Calc(n)%pt *Inverse((long
                ↪   long)Calc(m)*Calc(n-m)%pt,pt)%pt *tM;
        }
    };

    int P;
    std::vector<Divisor> ps;

    Lucas(int P):P(P) {
        for(int d=2,x=P;x!=1;d=(long long)d*d<=P?d+1:x)
            if(x%d==0) {
                int t=0,pt=1;
                do {
                    ++t;pt*=d;
                    x/=d;
                } while(x%d==0);
                ps.push_back(Divisor(d,t,pt,(long
                ↪   long)Inverse(P/pt,pt)*(P/pt)%P));
            }
    }

    int operator ()(long long n,long long m) {
        long long res=0;
        for(Divisor &d : ps)
            (res+=d.Solve(n,m))%=P;
        return res;
    }
};
```

3.14 lucas

```
int Lucas(int n,int m) {
    int res=1;
    while(n && m) {
        (res*=C(n%P,m%P))%=P;
        n/=P,m/=P;
    }
    return res;
}
```

3.15 min25

```
const int N=1e5,XN=N+11;

int prime[XN*2],pcnt;
void Prep() {
    static bool notPrime[XN*2];
    for(int i=2;i<=N*2;++i) {
        if(!notPrime[i])
            prime[++pcnt]=i;
        for(int j=1;j<=pcnt && i*prime[j]<=N*2;++j) {
            notPrime[i*prime[j]]=1;
            if(i%prime[j]==0)
                break;
        }
    }
}

namespace Min25 {
    typedef unsigned long long ans_t;

    std::function<ans_t(int,int)> F;

    long long n;
    int lim,psz;

    struct Identifier {
        int id[2][XN],cnt;
        int &operator [](long long x) {
            int &res=x<=lim?id[0][x]:id[1][n/x];
            if(res==0)
                res=++cnt;
            return res;
        }
    }id;

    ans_t g[XN*2],fps[XN];

    ans_t H(long long n,int m) {
        if(n<=1 || m>psz)
            return 0;
        ans_t res=g[id[n]]-fps[m-1];
        for(int i=m;i<=psz && (long long)prime[i]*prime[i]<=n;++i) {
            long long pt=prime[i],pt1=pt*prime[i];
            for(int t=1;pt1<=n;++t,pt=pt1,pt1*=prime[i])
                res+=F(prime[i],t)*H(n/pt,i+1)+F(prime[i],t+1);
        }
        return res;
    }

    ans_t Solve(long long n,std::function<ans_t(int,int)>
     ↪  F,std::function<ans_t(long long)> gInit) {
        static long long kp[XN*2];
        int kpc=0;
        lim=sqrt(n)+0.5,psz=std::upper_bound(prime+1,prime+1+pcnt,lim)-prime;
        for(int i=id.cnt=0;i<=lim;++i)
            id.id[0][i]=id.id[1][i]=0;
        Min25::F=F;
        Min25::n=n;
        for(long long l=1,r;l<=n;l=r+1) {
            r=n/(n/l);
            g[id[kp[++kpc]=n/l]]=gInit(n/l);
        }
        for(int i=1;i<=psz;++i)
            fps[i]=fps[i-1]+F(prime[i],1);
        for(int j=1;j<=psz;++j)
            for(int i=1;i<=kpc && (long long)prime[j]*prime[j]<=kp[i];++i)
                g[id[kp[i]]]-=F(prime[j],1)*(g[id[kp[i]/prime[j]]]-fps[j-1]);
        return H(n,1);
    }
}
```

3.16 polynomial

```
const int XN=1<<19,//MAKE2(XN)*2
          P=998244353;

int inv[XN];
/*
inv[1]=1;
for(int i=2;i<XN;++i)
    inv[i]=1LL*(P-P/i)*inv[P%i]%P;
 */
void M(int &x) {
    ((x>=P) && (x-=P)) || ((x<0) && (x+=P));
}

int Pow(long long base,int v) {
    long long res;
    for(res=1;v;v>>=1,(base*=base)%=P)
        if(v&1)
            (res*=base)%=P;
    return res;
}

int Make2(int x) {
    return 1<<((32-__builtin_clz(x))+((x&(-x))!=x));
}

void NTT(int a[],int n,int op) {
    for(int i=0,j=0;i<n;i++) {
        if(i>j)
            std::swap(a[i],a[j]);
        for(int k=n>>1;(j^=k)<k;k>>=1);
    }
    for(int len=2;len<=n;len<<=1) {
        int rt=Pow(3,(P-1)/len);
        for(int i=0;i<n;i+=len) {
            int w=1;
            for(int j=i;j<i+len/2;++j) {
                int u=a[j],t=1LL*a[j+len/2]*w%P;
                M(a[j]=u+t);M(a[j+len/2]=u-t);
                w=1LL*w*rt%P;
            }
        }
    }
    if(op==-1) {
        std::reverse(a+1,a+n);
        for(int i=0;i<n;++i)
            a[i]=1LL*a[i]*inv[n]%P;
    }
}

using Polynomial=std::vector<int>;

Polynomial operator *(Polynomial const &A,Polynomial const &B) {
    static int a[XN],b[XN];
    int n=Make2((int)A.size()+(int)B.size()-1);
    for(int i=0;i<n;++i) {
        a[i]=i<(int)A.size()?A[i]:0;
        b[i]=i<(int)B.size()?B[i]:0;
    }
    NTT(a,n,1);NTT(b,n,1);
    for(int i=0;i<n;++i)
        a[i]=1LL*a[i]*b[i]%P;
    NTT(a,n,-1);
    return Polynomial(a,a+(int)A.size()+(int)B.size()-1);
}

Polynomial Inverse(Polynomial const &A){
    static int a[XN],b[XN];
    int n=Make2((int)A.size());
    b[0]=Pow(A[0],P-2);
    for(int len=2;len<=n;len*=2) {
        for(int i=0;i<len*2;++i) {
            a[i]=i<std::min((int)A.size(),len)?A[i]:0;
            b[i]=i<len/2?b[i]:0;
        }
        NTT(a,len*2,1);NTT(b,len*2,1);
        for(int i=0;i<len*2;++i)
            M(b[i]=1LL*b[i]*(2-1LL*a[i]*b[i]%P)%P);
        NTT(b,len*2,-1);
    }
    return Polynomial(b,b+(int)A.size());
}

Polynomial Diff(Polynomial const &A) {
    Polynomial B((int)A.size()-1);
    for(int i=0;i<(int)A.size()-1;++i)
        B[i]=1LL*A[i+1]*(i+1)%P;
    return B;
}

Polynomial Int(Polynomial const &A) {
    Polynomial B((int)A.size()+1);
    for(int i=(int)B.size()-1;i>=1;--i)
        B[i]=1LL*A[i-1]*inv[i]%P;
    return B;
}

Polynomial SquareRoot(Polynomial const &A) {
    static int a[XN],b[XN],c[XN];
    int n=Make2((int)A.size());
    assert(A[0]==1);b[0]=1;
    for(int len=2;len<=n;len*=2) {
        std::fill(b+len/2,b+len,0);
        auto t=Inverse(Polynomial(b,b+len));
        for(int i=0;i<len;++i) {
            a[i]=i<std::min((int)A.size(),len)?A[i]:0;
            b[i]=i<len/2?b[i]:0;
            c[i]=i<len/2?t[i]:0;
        }
        NTT(c,len,1);NTT(b,len,1);NTT(a,len,1);
        for(int i=0;i<len;++i)
            b[i]=1LL*inv[2]*(1LL*a[i]*c[i]%P+b[i])%P;
        NTT(b,len,-1);
    }
    return Polynomial(b,b+(int)A.size());
}

Polynomial Log(Polynomial const &A) {
    auto B=Int(Diff(A)*Inverse(A));
    B.resize(A.size());
    return B;
}

Polynomial Exp(Polynomial const &A) {
    static int a[XN],b[XN],c[XN];
    int n=Make2((int)A.size());
    b[0]=1;
    for(int len=2;len<=n;len*=2) {
        std::fill(b+len/2,b+len,0);
        auto logb=Log(Polynomial(b,b+len));
        for(int i=0;i<len*2;++i) {
            a[i]=i<std::min((int)A.size(),len)?A[i]:0;
            b[i]=i<len/2?b[i]:0;
            c[i]=i<len?logb[i]:0;
        }
        NTT(a,len*2,1);NTT(b,len*2,1);NTT(c,len*2,1);
        for(int i=0;i<len*2;++i)
            M(b[i]=1LL*b[i]*(1-c[i]+a[i])%P);
        NTT(b,len*2,-1);
    }
    return Polynomial(b,b+(int)A.size());
}

Polynomial Pow(Polynomial const &A,int v) {
    static int a[XN];
    int A0=A[0],inv0=Pow(A[0],P-2);
    for(int i=0;i<(int)A.size();++i)
        a[i]=1LL*A[i]*inv0%P;
    auto loga=Log(Polynomial(a,a+(int)A.size()));
    for(int i=0;i<(int)A.size();++i)
        loga[i]=1LL*loga[i]*v%P;
    auto exp=Exp(loga);
    int k=Pow(A0,v);
    for(int i=0;i<(int)A.size();++i)
        exp[i]=1LL*exp[i]*k%P;
    return exp;
}
```

3.17 polynomialFreeMod

```
typedef long long LL;

const int XN = (1 << 18) + 31, P = 1000000007;
const double PI2 = 2 * 3.14159265358979323846264338327 9;
int N = 1 << 18, L = 15, K = (1 << L) - 1;

struct X {
    double x, y;

    X() {}
    X(double _x, double _y) : x(_x), y(_y) {}

    X operator+(const X &z) const { return X(x + z.x, y + z.y); }

    X operator-(const X &z) const { return X(x - z.x, y - z.y); }

    X operator*(const X &z) const { return X(x * z.x - y * z.y, x * z.y + y *
     ↪  z.x); }

    X conj() const { return X(x, -y); }
} w[XN];

void init() {
    for (int i = 0; i < N; i++) w[i] = X(cos(PI2 / N * i), sin(PI2 / N * i));
}

void trans(int n, X x[], bool f) {
    for (int i = 0, j = 0; i < n; i++) {
        if (i < j)
            std::swap(x[i], x[j]);
        for (int l = n >> 1; (j ^= l) < l; l >>= 1)
            ;
    }
    for (int i = 2; i <= n; i <<= 1) {
        int l = i >> 1, d = N / i;
        for (int j = 0; j != n; j += i)
            for (int k = 0; k != l; k++) {
                X &a = x[j + k], &b = x[j + k + l], t = w[d * k] * b;
                b = a - t;
                a = a + t;
            }
    }
    if (!f) {
        std::reverse(x + 1, x + n);
        for (int i = 0; i < n; i++) x[i].x /= n, x[i].y /= n;
    }
}

void conv(int na, int a[], int nb, int b[], int nc, int c[]) {
    int n = 1;
    static X x[XN], y[XN], z[XN], w[XN];
    while (n < na + nb - 1) n <<= 1;
    for (int i = 0; i < n; i++) {
        x[i] = i < na ? X(a[i] & K, a[i] >> L) : X(0, 0);
        y[i] = i < nb ? X(b[i] & K, b[i] >> L) : X(0, 0);
    }
    trans(n, x, 1);
    trans(n, y, 1);
    X r0(0.5, 0), r1(0, -0.5), r(0, 1);
    for (int i = 0; i < n; i++) {
        int j = (n - i) & (n - 1);
        X x0 = (x[i] + x[j].conj()) * r0;
        X x1 = (x[i] - x[j].conj()) * r1;
        X y0 = (y[i] + y[j].conj()) * r0;
        X y1 = (y[i] - y[j].conj()) * r1;
        z[i] = x0 * (y0 + y1 * r);
        w[i] = x1 * (y0 + y1 * r);
    }
    trans(n, z, 0);
    trans(n, w, 0);
    for (int i = 0; i < nc; i++) {
        int c00 = (LL)(z[i].x + 0.5) % P, c01 = (LL)(z[i].y + 0.5) % P;
        int c10 = (LL)(w[i].x + 0.5) % P, c11 = (LL)(w[i].y + 0.5) % P;
        c[i] = ((((LL)c11 << L) + c01 + c10 << L) + c00) % P;
    }
}

void inv(int n, int f[], int g[]) {
    if (n == 1)
```

```
        g[0] = 1;
    else {
        int l = n + 1 >> 1;
        static int t[XN];
        inv(l, f, g);
        conv(n, f, l, g, n, t), conv(l, g, n - l, t + l, n - l, g + l);
        for (int i = l; i < n; i++)
            if (g[i])
                g[i] = P - g[i];
    }
}

int qpow(int a, int b) {
    int c = 1;
    for (; b; b >>= 1) {
        if (b & 1)
            c = (LL)c * a % P;
        a = (LL)a * a % P;
    }
    return c;
}

int inv(int x) { return qpow(x, P - 2); }

int z[XN];

inline void ln(int n, int f[], int g[]) {
    static int t[XN];
    inv(n, f, t);
    for (int i = 1; i < n; i++) g[i - 1] = (LL)i * f[i] % P;
    g[n - 1] = 0;
    conv(n, t, n, g, n, t);
    for (int i = n - 1; i; i--) g[i] = (LL)t[i - 1] * z[i] % P;
    g[0] = 0;
}

inline void exp(int n, int f[], int g[]) {
    if (n == 1) {
        g[0] = 1;
        return;
    }
    static int t[XN];
    int l = n + 1 >> 1;
    exp(l, f, g);
    ln(n, g, t);
    for (int i = 0; i < n; i++) t[i] = (f[i] + P - t[i]) % P;
    t[0]++;
    conv(n, g, n, t, n, g);
}

int f[XN], g[XN];
int n = 0, k = 0;

int main() {
    init();
    scanf("%d%d", &n, &k);
    for (int i = 1; i <= k; i++) z[i] = inv(i);
    for (int i = 1; i <= k; i++) f[i] = (LL)z[i] * (n - 1) % P;
    for (int i = 2; i <= n; i++)
        for (int j = 1; i * j <= k; j++) f[i * j] = (f[i * j] + P - z[j]) % P;
    exp(k + 1, f, g);
    printf("%d\n", g[k]);
    return 0;
}
```

4 graph

4.1 Dijkstra

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

const int MAXV=50,MAXE=50;
struct Edge{
    int v,n,w;
}edges[MAXE];
int head[MAXV],idx=0;
```

```
void adde(int u,int v,int w){
    edges[++idx].v=v;
    edges[idx].n=head[u];
    edges[idx].w=w;
    head[u]=idx;
}
void addee(int u,int v,int w){
    adde(u,v,w);
    adde(v,u,w);
}


struct v4q{
    int u,dist;
    v4q(){}
    v4q(int u,int dist):u(u),dist(dist){}
    bool operator<(const v4q &b)const{
        return dist>b.dist;
    }
};
priority_queue<v4q> pq;
bool vis[MAXV];
int dist[MAXV];
int dijkstra(int S){
    memset(dist,0x7f,sizeof(dist));
    dist[S]=0;
    pq.push(v4q(S,0));
    while(!pq.empty()){
        int cur=pq.top().u;pq.pop();
        if(vis[cur])continue;
        vis[cur]=1;

        for(int ei=head[cur];ei;ei=edges[ei].n){
            Edge &e=edges[ei];
            if(dist[e.v]>dist[cur]+e.w){
                dist[e.v]=dist[cur]+e.w;
                pq.push(v4q(e.v,dist[e.v]));
            }
        }
    }

    return 0;
}

int main(){
    int elen;cin>>elen;
    for(int i=0;i<elen;i++){
        int u,v,w;cin>>u>>v>>w;
        addee(u,v,w);
    }
    int S;cin>>S;
    dijkstra(S);
    while(cin>>S){
        cout<<dist[S]<<endl;
    }

    return 0;
}
```

4.2 Dinic

```
#include <iostream>
#include <cstring>
#include <vector>
#include <algorithm>
#include <queue>
using namespace std;
const int MAXV=100,MAXE=100;

struct Edge{
    int u,v,cap;
    Edge(){}
    Edge(int u,int v,int cap):u(u),v(v),cap(cap){}
} edges[MAXE];
int idx=0;
vector<int> graph[MAXV];
void adde(int u,int v,int cap){
    graph[u].push_back(idx);
    edges[idx++]=Edge(u,v,cap);
    graph[v].push_back(idx);
    edges[idx++]=Edge(v,u,0);
}

int S,T;
int dis[MAXV];
int current[MAXV];
bool BFS(){
    queue<int> q;
    q.push(S);
    memset(dis,0x3f,sizeof(dis));
    dis[S]=0;
    while(!q.empty()){
        int h=q.front();q.pop();
        for(int ei=0;ei<graph[h].size();ei++){
            Edge &e=edges[graph[h][ei]];
            if(e.cap>0 && dis[e.v]==0x3f3f3f3f){
                dis[e.v]=dis[h]+1;
                q.push(e.v);
            }
        }
    }
    return dis[T]<0x3f3f3f3f;
}
int dinic(int u,int maxflow){
    if(u==T)return maxflow;
    for(int ei=current[u];ei<graph[u].size();ei++){
        current[u]=ei;
        Edge &e=edges[graph[u][ei]];
        if(dis[e.v]==dis[u]+1 && e.cap>0){
            int flow=dinic(e.v,min(maxflow,e.cap));
            if(flow){
                e.cap-=flow;
                edges[graph[u][ei]^1].cap+=flow;
                return flow;
            }
        }
    }
    return 0;
}
int DINIC(){
    int ans=0;
    while(BFS()){
        memset(current,0,sizeof(current));
        int flow;
        while(flow=dinic(S,0x3f3f3f3f))ans+=flow;
    }
    return ans;
}
/*      1    2   3   4
 *  n+1 0    0   0   0
 *  n+2 0    1   1   0
 *  n+3 1    0   1   0
 *  n+4 0    1   0   0
 */

char game[5][5];
int horizon[5][5];
int vertical[5][5];
int tick=10;
int main(){
    int len;
    while(cin>>len){
        if(len==0)break;
        for(int i=0;i<MAXV;i++)graph[i].clear();
        idx=0;
        for(int i=0;i<=4;i++)for(int j=0;j<=4;j++)horizon[i][j]=vertical[i][j]=0;
        tick=10;

        for(int i=1;i<=len;i++){
            for(int j=1;j<=len;j++){
                cin>>game[i][j];
            }
        }
        for(int i=0;i<=len;i++)game[i][0]='X';
        for(int j=0;j<=len;j++)game[0][j]='X';

        for(int i=0;i<=len;i++){
            for(int j=0;j<=len;j++){
                if(game[i][j]!='X')continue;
                while(j<=len && game[i][j]=='X')j++;
                if(j>len)continue;//± X
                j--;
                horizon[i][j]=++tick;
            }
        }

        //debug
        /*
           for(int i=0;i<=len;i++){
           for(int j=0;j<=len;j++){
           cout<<horizon[i][j]<<" ";
           }
           cout<<endl;
           }
           */

        for(int j=0;j<=len;j++){
            for(int i=0;i<=len;i++){
                if(game[i][j]!='X')continue;
                while(i<=len && game[i][j]=='X')i++;
                if(i>len)continue;
                i--;
                vertical[i][j]=++tick;
            }
        }
        /*
           for(int i=0;i<=len;i++){
           for(int j=0;j<=len;j++){
           cout<<vertical[i][j]<<" ";
           }
           cout<<endl;
           }
           */
        for(int i=0;i<=len;i++){
            for(int j=0;j<=len;j++){
                if(vertical[i][j]){
                    for(int ii=i+1;ii<=len;ii++){
                        if(vertical[ii][j])break;
                        for(int jj=j-1;jj>=0;jj--){
                            if(horizon[ii][jj] && game[ii][j]!='X'){
                                adde(vertical[i][j],horizon[ii][jj],1);
                                //cout<<"add
                                 ↪   "<<vertical[i][j]<<"->"<<horizon[ii][jj]<<endl;
                                break;
                            }
                        }
                    }
                }
            }
        }


        S=1,T=2;
        for(int i=0;i<=len;i++){
            for(int j=0;j<=len;j++){
                if(vertical[i][j])adde(S,vertical[i][j],1);
            }
        }
        for(int i=0;i<=len;i++)for(int
         ↪  j=0;j<=len;j++)if(horizon[i][j])adde(horizon[i][j],T,1);

        cout<<DINIC()<<endl;
    }

    return 0;
}
```

4.3 Double LCA

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

const int MAXV=50,MAXE=50;
const int MAXP=50;
struct Edge{
    int v,n,w;
}edges[MAXE];
int head[MAXV],idx=0;
int vlen;
void adde(int u,int v,int w){
    edges[++idx].v=v;
    edges[idx].n=head[u];
    edges[idx].w=w;
    head[u]=idx;
}
void addee(int u,int v,int w){
    adde(u,v,w);
    adde(v,u,w);
}

int lca[MAXV][MAXP],lcaw[MAXV][MAXP];
int depth[MAXV];
int P=0;
void dfs_fa(int u,int fa){
    for(int ei=head[u];ei;ei=edges[ei].n){
        Edge &e=edges[ei];
        if(e.v==fa)continue;
        lca[e.v][0]=u;
        lcaw[e.v][0]=e.w;
        depth[e.v]=depth[u]+1;
        dfs_fa(e.v,u);
    }
}
void init_lca(int root=1){
    for(P=0;(2<<P)<vlen;P++);
    depth[root]=0;
    dfs_fa(root,-1);

    for(int p=1;p<=P;p++){
        for(int u=1;u<=vlen;u++){
            lca[u][p]=lca[lca[u][p-1]][p-1];
        }
    }
    for(int p=1;p<=P;p++){
        for(int u=1;u<=vlen;u++){
            lcaw[u][p]=lcaw[u][p-1]+lcaw[lca[u][p-1]][p-1];
        }
    }
```

```
}

struct Query{
    int u,v;
    int res,resw;
    Query(){}
};
vector<Query> queries;
void doquery(int qid){
    int u=queries[qid].u;
    int v=queries[qid].v;
    int &res=queries[qid].res,&resw=queries[qid].resw;

    //approve u is deepper than v
    if(depth[u]<depth[v])swap(u,v);
    //pull u to the same depth of v
    for(int p=P;p>=0;p--){
        if(depth[lca[u][p]]>=depth[v]){
            resw+=lcaw[u][p];
            u=lca[u][p];
        }
    }

    //judge if u is v's child.
    if(u==v){
        res=u;
        return;
    }

    //pull in same time.
    for(int p=P;p>=0;p--){
        if(lca[u][p]!=lca[v][p]){
            resw+=lcaw[u][p]+lcaw[v][p];
            u=lca[u][p];
            v=lca[v][p];
        }
    }

    //add last one.
    resw+=lcaw[u][0]+lcaw[v][0];
    res=lca[u][0];
}

void doqueries(){
    for(int i=0;i<queries.size();i++){
        doquery(i);
    }
}

int main(){
    int elen;cin>>elen;
    for(int i=0;i<elen;i++){
        int u,v,w;
        cin>>u>>v>>w;
        addee(u,v,w);
        vlen=max(vlen,max(u,v));
    }
    int qlen;cin>>qlen;
    for(int i=0;i<qlen;i++){
        Query q;
        cin>>q.u>>q.v;
        queries.push_back(q);
    }
    init_lca(1);
    doqueries();

    for(int i=0;i<qlen;i++){
        cout<<queries[i].res<<" "<<queries[i].resw<<endl;
    }

    return 0;

}
```

4.4 MCMF

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
```

```
#include <queue>
using namespace std;
constexpr int MAXV=5050,MAXE=5500000;

struct Edge{
    int u,v,cap,cost;
    Edge(){}
    Edge(int u,int v,int cap,int cost):u(u),v(v),cap(cap),cost(cost){}
}edges[MAXE];
int idx=0;
vector<int> g[MAXV];
void adde(int u,int v,int cap,int cost){
    //cout<<u<<"->"<<v<<"="<<cap<<","<<cost<<endl;
    g[u].push_back(idx);
    edges[idx++]=Edge(u,v,cap,cost);
    g[v].push_back(idx);
    edges[idx++]=Edge(v,u,0,-cost);
}

int S,T;
int dis[MAXV];
int inq[MAXV],from[MAXV];
int cost=0;
int BFS(){
    fill(from,from+MAXV,-1);
    queue<int> q;
    q.push(S);
    memset(dis,0x3f,sizeof(dis));
    dis[S]=0;

    while(!q.empty()){
        int h=q.front();q.pop();
//        cout<<"arrive "<<h<<endl;

        inq[h]=0;
        for(int ei=0;ei<g[h].size();ei++){
            Edge &e=edges[g[h][ei]];
            if(e.cap>0 && dis[e.v]>dis[h]+e.cost){
                dis[e.v]=dis[h]+e.cost;
                from[e.v]=g[h][ei];
                if(!inq[e.v]){
                    q.push(e.v);
                    inq[e.v]=1;
                }
            }
        }
    }
    if(from[T]==-1)return false;
    int flow=0x3f3f3f3f;

    for(int i=from[T];i!=-1;i=from[edges[i^1].v]){
        flow=min(flow,edges[i].cap);
    }
    for(int i=from[T];i!=-1;i=from[edges[i^1].v]){
        edges[i].cap-=flow;
        edges[i^1].cap+=flow;
    }
    cost+=dis[T]*flow;
    return flow;

}

int DINIC(){
    int ans=0;
    cost=0;
    int temp=0;
    while(temp=BFS())ans+=temp;

    return ans;
}
constexpr int MAXN=2010;
int num[MAXN];
int main(){
    ios::sync_with_stdio(false);
    int kase;cin>>kase;
    while(kase--){
        int nlen,sel;cin>>nlen>>sel;
        for(int i=1;i<=nlen*2;i++)g[i].clear();
        g[5000].clear();g[5001].clear();g[5002].clear();
        idx=0;

        for(int i=1;i<=nlen;i++){
            cin>>num[i];
            adde(i,nlen+i,1,-num[i]);
        }
        for(int i=1;i<=nlen;i++){
            for(int j=1;j<i;j++){
                if(num[j]<=num[i])adde(nlen+j,i,1,0);
            }
        }
        S=5000;
        adde(5000,5001,sel,0);
        for(int i=1;i<=nlen;i++)adde(5001,i,1,0);
        for(int i=1;i<=nlen;i++)adde(i+nlen,5002,1,0);
        T=5002;
        DINIC();
        cout<<-cost<<endl;
    }

    return 0;
}
```

4.5 tarjan LCA

```
#include <iostream>
#include <vector>
using namespace std;
const int MAXN=50,MAXV=50,MAXE=50;
struct UT{
    int fa[MAXN];
    UT(){
        for(int i=0;i<MAXN;i++)fa[i]=i;
    }
    int find(int u){
        return fa[u]==u?u:fa[u]=find(fa[u]);
    }
    bool isc(int u,int v){
        return find(u)==find(v);
    }
    void con(int u,int v){
        fa[find(u)]=v;
    }
};

struct Edge{
    int v,n;
}edges[MAXE];
int head[MAXV],idx=0;
inline void adde(int u,int v){
    edges[++idx].v=v;
    edges[idx].n=head[u];
    head[u]=idx;
}
inline void addee(int u,int v){
    adde(u,v);
    adde(v,u);
}
int vis[MAXV];
vector<int> query[MAXV];
UT ut;
int a,b;
void tarjan(int now){
    vis[now]=1;
    for(int ei=head[now];ei;ei=edges[ei].n){
        Edge &e=edges[ei];
        if(!vis[e.v]){
        tarjan(e.v);
        ut.con(e.v,now);
        }
    }
    for(int i=0;i<query[now].size();i++){
        if(vis[query[now][i]]){
            cout<<now<<" "<<query[now][i]<<"->"<<ut.find(query[now][i])<<endl;
        }
    }
}

int main(){
    int e;cin>>e;
    for(int i=0;i<e;i++){
        int u,v;cin>>u>>v;
        addee(u,v);
    }
    int a,b;cin>>a>>b;
    query[a].push_back(b);
    query[b].push_back(a);
    tarjan(1);

    return 0;
}
```

4.6 tarjan scc

```
#include <iostream>
#include <algorithm>
#include <stack>
using namespace std;
const int MAXV = 110, MAXE = 110*110;
struct Edge {
    int v, n;
} edges[MAXE];
int head[MAXV], idx = 0;
void adde(int u, int v) {
    edges[++idx].v = v;
    edges[idx].n = head[u];
    head[u] = idx;
}

int tick = 0, cpidx = 0;
int dfn[MAXV], low[MAXV];
int instack[MAXV];
int incp[MAXV];
stack<int> cp;
void tarjan(int u) {
    instack[u] = 2;
    dfn[u] = low[u] = ++tick;
    cp.push(u);
    for (int ei = head[u]; ei; ei = edges[ei].n) {
        Edge &e = edges[ei];
        if (dfn[e.v] == 0) {
            tarjan(e.v);
            low[u] = min(low[u], low[e.v]);
        }
        else if (instack[e.v] == 2) {
            low[u] = min(low[u], dfn[e.v]);
        }
    }
    if (low[u] == dfn[u]) {
        cpidx++;
        while (!cp.empty()) {
            int cur = cp.top(); cp.pop();
            instack[cur] = 1;
            incp[cur] = cpidx;
            if (cur == u)break;
        }
    }
}
int ind[MAXV],outd[MAXV];
int main() {
    int vlen; cin >> vlen;
    for (int i = 1; i <= vlen; i++) {
        int v;
        while (cin >> v) {
            if (v == 0)break;
            adde(i, v);
        }
    }
    int res = 0;
    for (int i = 1; i <= vlen; i++) {
        if (!dfn[i])tarjan(i);
    }
    for (int u = 1; u <= vlen; u++) {
        for (int ei = head[u]; ei; ei = edges[ei].n) {
            Edge &e = edges[ei];
            if (incp[u] != incp[e.v]) {
                //TODO: 重复计算
                ind[incp[e.v]]++;
                outd[incp[u]]++;
            }
        }
    }
    if (cpidx == 1) {
        //特殊情况
        //以及 vlen==1 的时候
        cout << 1 << endl;
        cout << 0 << endl;
    }
    else {
        int sub1 = 0, sub2 = 0;
        for (int u = 1; u <= cpidx; u++)if (ind[u] == 0)sub1++;
        for (int u = 1; u <= cpidx; u++)if (outd[u] == 0)sub2++;


        cout << sub1 << endl;
        cout << max(sub1, sub2) << endl;
    }

    //while (1);
    return 0;
}
```

4.7 割点

```
#include <iostream>
#include <queue>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
using namespace std;
const int MAXV=600,MAXE=30000;

struct Edge{
    int v,n;
}edges[MAXE];
int head[MAXV],idx=0;

void adde(int u,int v){
    edges[++idx].v=v;
    edges[idx].n=head[u];
    head[u]=idx;
}
void adee(int u,int v){
    adde(u,v);
    adde(v,u);
}
int dfn[MAXV],low[MAXV],tick=0;
int root=0;
set<int> res;
void tarjan(int u,int fa,int fe){
    dfn[u]=low[u]=++tick;
    int sum=0;
    for(int ei=head[u];ei;ei=edges[ei].n){
        Edge &e=edges[ei];
        if(!dfn[e.v]){
            tarjan(e.v,u,ei);
            if(low[e.v]<low[u])low[u]=low[e.v];
            if(low[e.v]>=dfn[u])sum++;
        }else if(dfn[e.v]<low[u])low[u]=dfn[e.v];
    }
    if(sum>=2 || (sum==1 && u!=root)){
        res.insert(u);
    }
}


int main(){
    int vlen,elen;
    cin>>vlen>>elen;
    for(int i=0;i<elen;i++){
        int u,v;cin>>u>>v;
        adee(u,v);
    }
    tarjan(0,-1,0);
    int qlen;cin>>qlen;
    while(qlen--){
        int u;cin>>u;
        if(res.count(u)){
            cout<<"Red Alert: City "<<u<<" is lost!"<<endl;
        }else{
            cout<<"City "<<u<<" is lost."<<endl;
        }
    }
    cout<<"Game Over."<<endl;

    return 0;
}
```

4.8 circle in directed

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

const int N = 5000 + 10;
vector<int> g[N];
int st[N], instack[N], mark[N], top;
void dfs(int u)
{
    instack[u] = true;
    mark[u] = true;
    st[++top] = u;
    for (int i = 0; i < g[u].size(); ++i)
    {
        int v = g[u][i];
        if (!instack[v])
            dfs(v);
        else
        {
            int t;
            for (t = top; st[t] != v; --t)
                ;                      //在栈中找到环的起始点
            printf("%d:", top - t + 1); //这这里就能判断是奇数环还是偶数环
            for (int j = t; j <= top; ++j)
                printf("%d ", st[j]);
            puts("");
        }
    }
    instack[u] = false;
    top--;
}
void init(int n)
{
    for (int i = 1; i <= n; ++i)
    {
        g[i].clear();
        instack[i] = mark[i] = 0;
    }
    top = 0;
}
int main()
{
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF)
    {
        init(n);
        int u, v;
        for (int i = 1; i <= m; ++i)
        {
            scanf("%d%d", &u, &v);
            g[u].push_back(v);
        }
        for (int i = 1; i <= n; ++i)
            if (!mark[i])
                dfs(i);
    }
    return 0;
}
```

4.9 steinerTree

```
//假设原来已经给定了个点，库朗等指出需要引进的点数至多为，此种点称为斯坦纳点。过每一斯坦纳点，至多有三条边通过。若为三条边，则它们两两交成 120° 角；若为两条边，则此斯坦纳点必为某一已给定的点，且此两条边交成的角必大于或等于 120°。其中最小的网络称为已给定点的集合的最小斯坦纳树，记作 SMT。若此 SMT 的斯坦纳点中有等于给定点的点，则称此 SMT 为退化的，此给定点称为退化点。
// 斯坦纳树问题是组合优化学科中的一个问题。将指定点集合中的所有点连通，且边权总和最小的生成树称为最小斯坦纳树（Minimal Steiner Tree），其实最小生成树是最小斯坦纳树的一种特殊情况。而斯坦纳树可以理解为使得指定集合中的点连通的树，但不一定最小。
```

```
/*
 * Steiner Tree: 求，使得指定 K 个点连通的生成树的最小总权值
 * st[i] 表示顶点 i 的标记值，如果 i 是指定集合内第 m(0<=m<K) 个点，则 st[i]=1<<m
 * endSt=1<<K
 * dptree[i][state] 表示以 i 为根，连通状态为 state 的生成树值
 */
#define CLR(x,a) memset(x,a,sizeof(x))

int dptree[N][1<<K],st[N],endSt;
bool vis[N][1<<K];
queue<int> que;

int input()
{
   /*
    *   输入，并且返回指定集合元素个数 K
    *   因为有时候元素个数需要通过输入数据处理出来，所以单独开个输入函数。
    */
}

void initSteinerTree()
{
    CLR(dptree,-1);
    CLR(st,0);
    for(int i=1;i<=n;i++) CLR(vis[i],0);
    endSt=1<<input();
    for(int i=1;i<=n;i++)
        dptree[i][st[i]]=0;
}

void update(int &a,int x)
{
    a=(a>x || a==-1)? x : a;
}

void SPFA(int state)
{
    while(!que.empty()){
        int u=que.front();
        que.pop();
        vis[u][state]=false;
        for(int i=p[u];i!=-1;i=e[i].next){
            int v=e[i].vid;
            if(dptree[v][st[v]|state]==-1 ||
                dptree[v][st[v]|state]>dptree[u][state]+e[i].w){

                dptree[v][st[v]|state]=dptree[u][state]+e[i].w;
                if(st[v]|state!=state || vis[v][state])
                    continue; //只更新当前连通状态
                vis[v][state]=true;
                que.push(v);
            }
        }
    }
}

void steinerTree()
{
    for(int j=1;j<endSt;j++){
        for(int i=1;i<=n;i++){
            if(st[i] && (st[i]&j)==0) continue;
            for(int sub=(j-1)&j;sub;sub=(sub-1)&j){
                int x=st[i]|sub,y=st[i]|(j-sub);
                if(dptree[i][x]!=-1 && dptree[i][y]!=-1)
                    update(dptree[i][j],dptree[i][x]+dptree[i][y]);
            }
            if(dptree[i][j]!=-1)
                que.push(i),vis[i][j]=true;
        }
        SPFA(j);
    }
}
```

4.10 edgeBiconnectedComponent

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt,flag;}e[N*2];
int head[N],n,ecnt=1;
int low[N],dfn[N],bccno[N],dfs_clock,bcc_cnt;
vector<pair<int,int> >bridge;
vector<int>bcc[N];
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u],0};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v],0};head[v]=ecnt;
}
void dfs(int u,int fa){
    dfn[u]=low[u]=++dfs_clock;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]){
                bridge.push_back(make_pair(u,v));
                e[i].flag=e[i^1].flag=1;
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            low[u]=min(low[u],dfn[v]);
        }
}
void dfs_(int u){
    bccno[u]=bcc_cnt;
    bcc[bcc_cnt].push_back(u);
    for(int i=head[u];i;i=e[i].nxt)
        if(!e[i].flag){
            dfs_(e[i].to);
        }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
    for(int i=1;i<=n;++i)
        if(!bccno[i]){
            ++bcc_cnt;
            dfs_(i);
        }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

4.11 vertexBiconnectedComponent

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N*2];
int head[N],n,ecnt;
int dfn[N],low[N],bccno[N],dfs_clock,bcc_cnt;
bool iscut[N];
vector<int>bcc[N];
stack<pair<int,int> >stk;
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v]};head[v]=ecnt;
}
void dfs(int u,int fa){
    low[u]=dfn[u]=++dfs_clock;
    int child=0;
    for(int i=head[u],v;i;i=e[i].nxt){
        if(!dfn[v=e[i].to]){
            stk.push(make_pair(u,v));
            ++child;
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]){
                iscut[u]=1;
                bcc[++bcc_cnt].clear();
                for(;;){
                    pair<int,int>x=stk.top();stk.pop();
                    if(bccno[x.first]!=bcc_cnt){
                        bcc[bcc_cnt].push_back(x.first);
                        bccno[x.first]=bcc_cnt;
                    }
                    if(bccno[x.second]!=bcc_cnt){
                        bcc[bcc_cnt].push_back(x.second);
                        bccno[x.second]=bcc_cnt;
                    }
                    if(x.first==u&&x.second==v)break;
                }
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            stk.push(make_pair(u,v));
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(fa<0&&child==1)iscut[u]=0;
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

4.12 stronglyConnectedComponent

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N];
int head[N],n,ecnt;
int low[N],dfn[N],stk[N],sccno[N],size[N],top,dfs_clock,scc_cnt;
bool instk[N];
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
}
void dfs(int u){
    dfn[u]=low[u]=++dfs_clock;
    stk[++top]=u;instk[u]=1;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v);
            low[u]=min(low[u],low[v]);
        }else if(instk[v]){
            low[u]=min(low[u],dfn[v]);
        }
    if(dfn[u]==low[u])
        for(++scc_cnt;;){
            int x=stk[top--];
            instk[x]=0;
            sccno[x]=scc_cnt;
            ++size[scc_cnt];
            if(x==u)break;
        }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

4.13 cutEdge

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N*2];
int head[N],n,ecnt;
int low[N],dfn[N],dfs_clock;
vector<pair<int,int> >bridge;
```

```
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v]};head[v]=ecnt;
}
void dfs(int u,int fa){
    dfn[u]=low[u]=++dfs_clock;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]){
                bridge.push_back(make_pair(u,v));
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            low[u]=min(low[u],dfn[v]);
        }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

4.14 cutVertex

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N*2];
int head[N],n,ecnt;
int low[N],dfn[N],dfs_clock;
bool iscut[N];
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v]};head[v]=ecnt;
}
void dfs(int u,int fa){
    dfn[u]=low[u]=++dfs_clock;
    int child=0;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v,u);
            ++child;
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]){
                iscut[u]=1;
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            low[u]=min(low[u],dfn[v]);
        }
    if(fa<0&&child==1){
        iscut[u]=0;
    }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

4.15 dijkstraBasedPBDS

```
#include <ext/pb_ds/priority_queue.hpp>
```

```
typedef __gnu_pbds::priority_queue<std::pair<long
 ↪  long,int>,std::greater<std::pair<long long,int>
 ↪  >,__gnu_pbds::pairing_heap_tag> Heap;
long long Dijkstra(int s,int t) {
    static long long sp[XN];
    static Heap::point_iterator ref[XN];
    Heap Q;
    memset(sp,31,sizeof(sp));
    sp[s]=0;
    Q.push(std::make_pair(0,s));
    while(!Q.empty()) {
        int pos=Q.top().second;Q.pop();
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(Reduce(sp[u],sp[pos]+e->v)) {
                if(ref[u]!=0)
                    Q.modify(ref[u],std::make_pair(sp[u],u));
                else
                    ref[u]=Q.push(std::make_pair(sp[u],u));
            }
        }
    }
    return sp[t];
}
```

4.16 hungary

```
//匈牙利算法
//你应该优先去看 KM 部分

namespace Hungary{
    const int N=500+10;
    int nx,ny;
    bool vis[N],w[N][N];
    int boy[N],girl[N];
    int dfs(int x){
        for(int y=1;y<=ny;++y)
            if(w[x][y]&&!vis[y]){
                vis[y]=1;
                if(!boy[y]||dfs(boy[y])){
                    girl[x]=y,boy[y]=x;
                    return 1;
                }
            }
        return 0;
    }
    int run(){
        int res=0;
        for(int x=1;x<=nx;++x)
            if(!girl[x]){
                memset(vis,0,sizeof (bool)*(ny+1));
                res+=dfs(x);
            }
        return res;
    }
}
```

4.17 ISAP

```
//最大流 ISAP 算法

const int INF=1e9,XN=200+11;

struct Edge {
    int to,cap,v;
    Edge *rev,*pre;

    Edge(int to,int cap,Edge *pre):to(to),cap(cap),v(0),rev(0),pre(pre) {}

    void *operator new(size_t flag) {
        static Edge *Pool=(Edge*)malloc((XN<<1)*sizeof(Edge)),*Me;
        return flag?Me++:(Me=Pool);
    }
}*G[XN],*preArc[XN];

int Aug(int t) {
    int d=INF;
    for(int pos=t;preArc[pos];pos=preArc[pos]->rev->to)
        Reduce(d,preArc[pos]->cap-preArc[pos]->v);
    for(int pos=t;preArc[pos];pos=preArc[pos]->rev->to) {
        preArc[pos]->v+=d;
        preArc[pos]->rev->v-=d;
    }
    return d;
}

int ISAP(int s,int t,int n) {
    static int num[XN],d[XN];
    static Edge *cArc[XN];
    std::fill(num+1,num+n,0);
    std::fill(d+1,d+1+n,0);
    std::copy(G+1,G+1+n,cArc+1);
    num[0]=n;preArc[s]=0;
    int flow=0;
    for(int pos=s;d[s]<n;) {
        if(pos==t) {
            flow+=Aug(t);
            pos=s;
        }
        bool adv=0;
        for(Edge *&e=cArc[pos];e;e=e->pre) {
            int u=e->to;
            if(e->cap>e->v && d[u]+1==d[pos]) {
                adv=1;
                preArc[pos=u]=e;
                break;
            }
        }
        if(!adv) {
            if(--num[d[pos]]==0)
                break;
            d[pos]=n;
            for(Edge *e=cArc[pos]=G[pos];e;e=e->pre)
                if(e->cap>e->v)
                    Reduce(d[pos],d[e->to]+1);
            num[d[pos]]++;
            if(pos!=s)
                pos=preArc[pos]->rev->to;//cArc
        }
    }
    return flow;
}
```

4.18 kuhnMunkres

```
//匈牙利算法

namespace KM {
    using namespace std;
    const int N=400+10;
    const int oo=2e9+10;
    int n,boy[N],girl[N],slack[N],pre[N],q[N],lx[N],ly[N],w[N][N];
    bool visx[N],visy[N];
    void aug(int y){//翻转匹配边和非匹配边，使匹配点对 +1
        for(int x,z;y;y=z){
            x=pre[y],z=girl[x];//pre 为增广路径的上一个点
            girl[x]=y,boy[y]=x;
        }//girl[x] 为男生 x 的伴侣，boy[y] 为女生 y 的伴侣
    }
    void bfs(int s){
        memset(visx,0,sizeof (bool)*(n+1));
        memset(visy,0,sizeof (bool)*(n+1));
        for(int i=1;i<=n;++i)slack[i]=oo;
        int h=0,t=1;q[0]=s;
        for(;;){
            for(;h!=t;){
                int x=q[h++];
                visx[x]=1;
                for(int y=1;y<=n;++y)
                    if(!visy[y]){
                        if(lx[x]+ly[y]==w[x][y]){
                            pre[y]=x;
                            if(!boy[y]){
                                aug(y);
                                return;//找到完备匹配
                            }else{
                                visy[y]=1;
                                q[t++]=boy[y];
                            }
                        }else if(lx[x]+ly[y]-w[x][y]<slack[y]){
                            pre[y]=x;
                            slack[y]=lx[x]+ly[y]-w[x][y];//更新 slack
                        }
                    }
            }
            int d=oo;
            for(int y=1;y<=n;++y)
                if(!visy[y])d=min(d,slack[y]);
            for(int i=1;i<=n;++i){
                if(visx[i])lx[i]-=d;
                if(visy[i])ly[i]+=d;else slack[i]-=d;//松弛操作
            }
            for(int y=1;y<=n;++y){
                if(!visy[y]&&!slack[y]){
                    if(!boy[y]){
                        aug(y);
                        return;
                    }else{
                        visy[y]=1;
                        q[t++]=boy[y];//松弛之后加入新的点
                    }
                }
            }
        }
    }
    long long run(int nx,int ny){//nx 为男生数量,ny 为女生数量
        n=max(nx,ny);//补足人数
        for(int i=1;i<=n;++i)
            for(int j=1;j<=n;++j)
                lx[i]=max(lx[i],w[i][j]);//lx,ly 为点标,w 为边权
        for(int i=1;i<=n;++i)bfs(i);
        long long res=0;
        for(int i=1;i<=n;++i)res+=lx[i]+ly[i];
        return res;
        //w[i][girl[i]]?girl[i]:0
    }
}

int main() {
}
```

4.19 kthShortestPathAStar

```
#include <queue>
#include <cstdio>
#include <cstring>
using namespace std;
#define N 100200
int n,m,xx[N],yy[N],zz[N],tot,first[1005],next[N],v[N],w[N],s,e,k,h[1005],vis[1005];
void add(int x,int y,int z){w[tot]=z,v[tot]=y,next[tot]=first[x],first[x]=tot++;}
struct Node{int now,h,g;}jy;
priority_queue<Node>pq;
bool operator < (Node a,Node b){return a.g+a.h>b.g+b.h;}
void Dijkstra(){
    memset(h,0x3f,sizeof(h));
    h[e]=0,jy.now=e;
    pq.push(jy);
    while(!pq.empty()){
        Node t=pq.top();pq.pop();
        if(!vis[t.now])vis[t.now]=1;
        else continue;
        for(int i=first[t.now];~i;i=next[i])
            if(!vis[v[i]]&&h[v[i]]>h[t.now]+w[i]){
                h[v[i]]=h[t.now]+w[i];
                jy.now=v[i];jy.g=h[v[i]];
                pq.push(jy);
            }
    }
}
int A_star(){
    memset(vis,0,sizeof(vis));
    jy.now=s;jy.g=0;jy.h=h[s];
    pq.push(jy);
    while(!pq.empty()){
        Node t=pq.top();pq.pop();
        vis[t.now]++;
        if(vis[t.now]>k)continue;
        if(vis[e]==k)return t.g;
        for(int i=first[t.now];~i;i=next[i]){
            jy.now=v[i],jy.g=t.g+w[i],jy.h=h[jy.now];
            pq.push(jy);
        }
    }
    return -1;
}

int main(){
    memset(first,-1,sizeof(first));
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
        scanf("%d%d%d",&xx[i],&yy[i],&zz[i]),add(yy[i],xx[i],zz[i]);
    scanf("%d%d%d",&s,&e,&k);
    if(s==e)k++;
    Dijkstra();
    tot=0,memset(first,-1,sizeof(first));
    for(int i=1;i<=m;i++)add(xx[i],yy[i],zz[i]);
    printf("%d\n",A_star());
}
```

4.20 SAP

```
//SAP 最大流算法

struct Edge {
    int to,cap,v;
    Edge *rev;
};
std::list<Edge> G[XN];
std::list<Edge>::iterator curr[XN];

void AddEdge(int x,int y,int cap) {
    G[x].push_back({y,cap,0,0});
    G[y].push_back({x,0,0,0});
    G[x].back().rev=&G[y].back();
    G[y].back().rev=&G[x].back();
}

long long Sap(int s,int t,int n) {
    static int dep[XN],gap[XN];
    std::function<int(int,int)> DFS=[&](int pos,int mx)->int {
        if(pos==t)
            return mx;
        else {
            int tot=0;
            for(auto &e=curr[pos];e!=G[pos].end();++e)
                if(dep[pos]==dep[e->to]+1 && e->cap>e->v) {
                    int f=DFS(e->to,std::min(mx-tot,e->cap-e->v));
                    e->v+=f;e->rev->v-=f;
                    if((tot+=f)==mx)
                        return tot;
                }
            if(!--gap[dep[pos]++])
                dep[s]=n+1;
            ++gap[dep[pos]];
            curr[pos]=G[pos].begin();
            return tot;
        }
    };
    for(int i=1;i<=n;++i)
        curr[i]=G[i].begin();
    long long flow=0;
    for(gap[0]=n;dep[s]<=n;flow+=DFS(s,std::numeric_limits<int>::max()));
    return flow;
}
```

5 geometry

5.1 minimum ball coverage

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <cmath>
using namespace std;
const int MAXN=110;

struct Point{
    double x,y,z;
    Point(double x=0,double y=0,double z=0):x(x),y(y),z(z){}
    Point operator-(const Point &b)const{
        return Point(x-b.x,y-b.y,z-b.z);
    }
    double dist(const Point &b)const{
        double dx=x-b.x,dy=y-b.y,dz=z-b.z;
        return sqrt(dx*dx+dy*dy+dz*dz);
    }
};

int n;
Point p[MAXN];
double SA(Point start){
    const double DELTA=1000;
    const double EPS=1e-8;
    double delta=DELTA;
    double ans=1e20;
    while(delta>EPS){
        int d=0;
        for(int i=0;i<n;i++){
            if(p[i].dist(start)>p[d].dist(start))d=i;
        }
        double r=p[d].dist(start);
        ans=min(ans,r);
        start.x+=(p[d].x-start.x)*delta/DELTA;
        start.y+=(p[d].y-start.y)*delta/DELTA;
        start.z+=(p[d].z-start.z)*delta/DELTA;
        delta*=0.98;
    }
    return ans;
}

int main(){
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>p[i].x>>p[i].y>>p[i].z;
    }

    cout<<fixed<<setprecision(8)<<SA(Point(0,0,0))<<endl;
    return 0;
}
```

5.2 computionalgeometry

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
typedef double num;

const double EPS=1e-7;
const double PI=acos(-1);
int sgn(double x){
    return (x>-EPS)-(x<EPS);
}
struct Vec{
    double x,y;//never change it yourself unless you dont need polar angle.
    double _polar;// make cache to accumulate speed as atan is too slow.

    Vec(){
        x=y=0;
    }
    Vec(double x,double y):x(x),y(y){
        _polar=atan2(y,x);
    }
    double dot(const Vec &b)const{
        return x*b.x+y*b.y;
    }
    double cross(const Vec &b)const{
        return x*b.y-b.x*y;
    }
    double len(){
        return sqrt(sqlen());
    }
    double sqlen(){
        return x*x+y*y;
    }
    Vec normalize(){
        double l=len();
        return Vec(x/l,y/l);
    }
    Vec rotate(double angle){
        return Vec(x*cos(angle)-y*sin(angle),x*sin(angle)+y*cos(angle));
    }
    Vec operator * (double factor)const{
        return Vec(x*factor,y*factor);
    }
    double operator * (const Vec &b)const{
        return cross(b);
    }
    Vec operator - (const Vec &b)const{
        return Vec(x-b.x,y-b.y);
    }
    Vec operator +(const Vec &b)const{
        return Vec(x+b.x,y+b.y);
    }
    double polar()const{
        return _polar;
    }
    bool leftby(const Vec &b)const{
        return sgn(b.cross(*this))>0;
    }
    //该函数认为端点也和线段同向
    bool samed(const Vec &b)const{
        return sgn(this->cross(b))==0 && sgn(this->dot(b))>=0;
    }
    bool operator<(const Vec &b)const{
        return this->polar()<b.polar();
    }
};
ostream& operator<<(ostream& out,const Vec &b){
    out<<"("<<b.x<<","<<b.y<<")";
    return out;
}
typedef Vec Point;
```

```
struct Line{
    Point pos;
    Vec dirc;
    Line(Point pos=Point(0,0),Vec dirc=Vec(0,0)):pos(pos),dirc(dirc){}
    static Line fromPoints(Point a,Point b){
        return Line(a,b-a);
    }
    double getarea(const Line &b)const{
        return abs(dirc.cross(b.dirc));
    }
    // 获得垂线
    Line getppd(){
        return Line(pos+dirc*0.5,dirc.rotate(PI/2));
    }

    //TODO: what will happen if they have no intersection,-nan
    Point getintersection(const Line &b)const{
        Vec down=this->pos-b.pos;
        double aa=b.dirc.cross(down);
        double bb=this->dirc.cross(b.dirc);
        return this->pos+this->dirc*(aa/bb);
    }
    bool point_on_line(Point point){
        if(!dirc.samed(point-pos))
        return false;
        if(sgn((point-pos).sqlen()-dirc.sqlen())>=0)
        return false;
        return true;
    }

    double get_distance(Point point){
        Line ppd=getppd();
        ppd.pos=point;

        Point intersection=getintersection(ppd);

        ppd.dirc=intersection-point;
        Vec v=intersection-pos;

        return abs(v.cross(point-pos)/v.len());
    }

    double get_distance(Line line){
        return get_distance(line.pos);
    }
};

struct Circle{
    Point center;
    Vec radius;
    Circle(Point center,Vec radius):center(center),radius(radius){}
    Circle(Point center,double r){
        this->center=center;
        radius=Vec(r,0);
    }
    Circle(const Line &diameter){
        Point center=diameter.pos;
        center=center+diameter.dirc*0.5;
        this->center=center;
        radius=diameter.dirc*0.5;
    }
    //-1: inner
    //0: on
    //1: outer
    int cover(Point point){
        double t=(point-center).sqlen();
        double r=radius.sqlen();
        return sgn(t-r);
    }
};
// passed in minimum_covering_circle
Circle circumcircle(Point a,Point b,Point c){
    Line l1=Line::fromPoints(a,b);
    Line l2=Line::fromPoints(a,c);
    Line l1p=l1.getppd(),l2p=l2.getppd();
    Point center=l1p.getintersection(l2p);
    return Circle(center,(a-center).len());

}
// passed ensured by
```

```
Circle minimum_covering_circle(vector<Point> &points){
    random_shuffle(points.begin(),points.end());
    Circle C(points[0],0);
    for(int i=1;i<points.size();i++){
        if(C.cover(points[i])<=0)continue;
        C=Circle(points[i],0);
        for(int j=0;j<=i-1;j++){
            if(C.cover(points[j])<=0)continue;
            C=Circle(Line(points[i],points[j]-points[i]));
            for(int k=0;k<=j-1;k++){
                if(C.cover(points[k])<=0)continue;
                C=circumcircle(points[i],points[j],points[k]);
            }
        }
    }
    return C;
}
//index of result starts at 1
vector<Point> convexHull(vector<Point> &points){
    vector<Point> stack;
    stack.push_back(Point(0,0));

    int top=0;//length of stack
    sort(points.begin(),points.end(),[](const Point &a,const Point &b){
            return a.x<b.x || (a.x==b.x && a.y<b.y);
            });

    for(int i=0;i<points.size();i++){
        //cout<<points[i]<<endl;
        //it won't remove points in same line.
        while(top>=2 && (stack[top]-stack[top-1]).leftby(points[i]-stack[top])){
            top--;
            stack.pop_back();
        }
        stack.push_back(points[i]);
        top++;
    }
    int downcaselen=top;
    for(int i=points.size()-1-1;i>=0;i--){
        while(top>=downcaselen+1 &&
         ↪  (stack[top]-stack[top-1]).leftby(points[i]-stack[top])){
            top--;
            stack.pop_back();
        }
        stack.push_back(points[i]);
        top++;
    }
    return stack;
}

double get_area(vector<Point> &points){
    if(points.size()<3){
        return -1;
    }
    sort(points.begin(),points.end(),[](Point &a,Point &b){
        return a.polar()<b.polar();
    });
    Point base(0,0);
    Point last=points[0];
    double res=0;
    for(int i=1;i<points.size();i++){
        Vec a=last-base,b=points[i]-base;
        res+=a.cross(b)/2;

        last=points[i];
    }
    //add the last point(also the first point)
    Vec a=last-base,b=points[0]-base;
    res+=a.cross(b)/2;

    return res;
}

bool point_in_polygen(Point point, vector<Point> &points){

}



vector<Point> points;
```

```
bool cmp(const Point &a,const Point &b){
    return a.polar()<b.polar();
}
int main(){
    Line a=Line::fromPoints(Point(0,0),Point(3,0));
    Line b=Line::fromPoints(Point(2,0),Point(2,2));

    Point inter=a.getintersection(b);
    cout<<inter<<a.point_on_line(inter)<<endl;
    /* convexHull
    int n;
    while(cin>>n){
        vector<Point> p;
        for(int i=0;i<n;i++){
            int x,y;cin>>x>>y;
            p.push_back(Point(x,y));
        }
        vector<Point> hull=convexHull(p);
        for(int i=1;i<hull.size();i++){
            cout<<hull[i]<<endl;
        }
    }
    */
    /*
    Line a=Line::fromPoints(Point(0,0),Point(10,10));
    Point point(0,1);

    cout<<a.get_distance(point)<<endl;
    */
   /*
    vector<Point> points;
    points.push_back(Point(2,0));
    points.push_back(Point(3,2));
    points.push_back(Point(1,4));
    points.push_back(Point(-1,3));

    cout<<get_area(points)<<endl;
    */
    //dasd

    return 0;
}
```

5.3 diameterOfPoints

```
double MaxDist(Point p[],int n) {
    //输入必须有序
    if(n==2) {
        return Dist(p[1],p[2]);
    } else {
        double res=0;
        for(int i=1,cp=2;i<=n;++i) {
            Line cl(p[i],p[i%n+1]-p[i]);
            while(Dist(p[cp],cl)<Dist(p[cp%n+1],cl))
                cp=cp%n+1;
            Enlarge(res,std::max(Dist(p[cp],p[i]),Dist(p[cp],p[i%n+1])));
        }
        return res;
    }
}
```

5.4 shortestDistanceOfPoints

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
```

```
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
         ↪  &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

6 data structure

6.1 AVL

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Node{
    int data;
    Node *ch[2];
    Node *fa;
    int height;
    Node(){
        height=1;
        data=0;
        ch[0]=ch[1]=NULL;
    }
    Node(int data):data(data){
        height=1;
        ch[0]=ch[1]=NULL;
    }
    int update(){
        int a=0,b=0;
        if(ch[0])a=ch[0]->height;
        if(ch[1])b=ch[1]->height;
        height=max(a,b)+1;
        return height;
    }
    bool isbalance(){
        int a=0,b=0;
        if(ch[0])a=ch[0]->height;
        if(ch[1])b=ch[1]->height;
        return abs(a-b)<=1;
    }
    int initheight(){
        if(ch[0]==NULL && ch[1]==NULL)return 0;
        if(height)return height;
        int a=0,b=0;
        if(ch[0])a=ch[0]->initheight();
        if(ch[1])b=ch[1]->initheight();
        height=max(a,b)+1;
        return height;
    }
    bool pos(Node *node){
        return ch[1]==node;
    }
};
Node *root;
void rotate(Node *node){
    Node *fa=node->fa;
    if(fa==NULL)return;
    int wai=fa->pos(node);
    fa->ch[wai]=node->ch[!wai];
    if(fa->ch[wai]!=NULL)fa->ch[wai]->fa=fa;
    node->ch[!wai]=fa;

    node->fa=fa->fa;
    if(node->fa!=NULL)node->fa->ch[node->fa->pos(fa)]=node;
    else root=node;
    fa->fa=node;
}
bool autorotate(Node *fafa,Node *fa,Node *node){
    if(fafa==NULL)return true;
    node->update();
    fa->update();
    fafa->update();
    if(!fafa->isbalance()){
        int fapos=fafa->pos(fa);
        int pos=fa->pos(node);
        if((fapos^pos)==0){
            rotate(fa);
            fafa->update();
            fa->update();
            if(fa->fa!=NULL)
                return autorotate(fa->fa->fa,fa->fa,fa);
        }else{
            rotate(node);
            fa->update();
            node->update();
            rotate(node);
            fafa->update();
            node->update();
            if(node->fa!=NULL)
                return autorotate(node->fa->fa,node->fa,node);
        }
    }else return autorotate(fafa->fa,fafa,fa);

    return true;
}
void insert(int x,Node *fafa,Node *fa,Node **node){
    if(*node==NULL){
        *node=new Node(x);
        (*node)->fa=fa;
        if(fa==NULL)return;
        fa->update();

        if(fafa==NULL)return;
        autorotate(fafa,fa,*node);
        /*
        if(!fafa->isbalance()){
            int fapos=fafa->pos(fa);
            int pos=fa->pos(*node);
            if((fapos^pos)==0){
                rotate(fa);
                fafa->update();
                fa->update();
            }else{
                rotate(nn);
                fa->update();
                nn->update();
                rotate(nn);
                fafa->update();
                nn->update();
            }
        }
        */
        return;
    }

    if(x<=(*node)->data){
        insert(x,fa,*node,&((*node)->ch[0]));
    }else{
        insert(x,fa,*node,&((*node)->ch[1]));
    }
    if(fa==NULL)return;
    fa->update();
    if(fafa==NULL)return;
    fafa->update();
}
void INSERT(int x){
    if(root==NULL)insert(x,NULL,NULL,&root);
    else{
        if(x<=root->data){
            insert(x,NULL,root,&(root->ch[0]));
        }else{
            insert(x,NULL,root,&(root->ch[1]));
        }
    }
}
void travel(Node *node){
    if(node->ch[0]){
        cout<<node->data<<"->"<<node->ch[0]->data<<endl;
        travel(node->ch[0]);
    }
    if(node->ch[1]){
        cout<<node->data<<"->"<<node->ch[1]->data<<endl;
        travel(node->ch[1]);
    }
}

int main(){
    int len;cin>>len;
    for(int i=0;i<len;i++){
        int x;cin>>x;
        INSERT(x);
    }
    cout<<root->data<<endl;

    return 0;
}
```

6.2 FT and qread

```
//树状数组
//需要知道输入信息的范围
/*
using ll=long long;
const int MAXN=10;
*/

const int MAXFT=500000;
int ft[MAXFT];
inline int lowbit(int x) {
    return x&-x;
}
inline void ftadd(int pos, int x) {
    while (pos <= MAXN) {
        ft[pos] += x;
        pos += lowbit(pos);
    }
}
inline ll ftget(int pos) {
    ll res = 0;
    while (pos != 0) {
        res += ft[pos];
        pos -= lowbit(pos);
    }
    return res;
}
```

6.3 PBDS hashtable

6.4 PBDS priorityqueue

```
#include <iostream>
#include <ext/pb_ds/priority_queue.hpp>
#include <functional>
using namespace std;
using namespace __gnu_pbds;
//Time analysis
//pairing_heap_tag: push,join(1),other(logn)
//binary_heap_tag:just push and pop(logn)
//binomial_heap_tag: push(1),other(logn)
//rc_binomial_heap_tag:push(1),other(logn)
//thin_heap_tag:push(1),no join,other(lgn); modify(1) if just increasing key.
//not support means TOO SLOW using it.

//usage
//join-> pairing
__gnu_pbds::priority_queue<int,less<int>,pairing_heap_tag> pq;
int main(){
    pq.clear();
    priority_queue<int>::point_iterator it1=pq.push(1);
    p.modify(it1,9);
    p.erase(it1);
    pq.push(2);
    pq.top();pq.pop();
    pq.empty();
    pq.size();

    //
    auto it=pq.begin();

    pq.join(/*other pq*/);
}
```

6.5 PBDS tree

```
#include <iostream>
#include <string>
#include <algorithm>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/tag_and_trait.hpp>
using namespace std;
using namespace __gnu_pbds;
tree<double,null_type,less<double>,rb_tree_tag,tree_order_statistics_node_update>
↪  bbt;
vector<int> a;
double pri=0;
void push(int x){
    bbt.insert((double)x+0.0000001*(pri++));
    a.push_back(x);
}
void pop(){
    bbt.erase(bbt.lower_bound((double)a[a.size()-1]));
    cout<<a[a.size()-1]<<endl;
    a.pop_back();
}
int main(){
    int n;cin>>n;
    while(n--){
        string inp;
        cin>>inp;
        if(inp=="Pop"){
            if(!bbt.empty()){
                pop();
            }else cout<<"Invalid"<<endl;
        }else if(inp=="PeekMedian"){
            if(!bbt.empty()){
                if(bbt.size()%2==0)
                    cout<<(int)*bbt.find_by_order(bbt.size()/2-1)<<endl;
                else cout<<(int)*bbt.find_by_order((bbt.size()+1)/2-1)<<endl;
            }else cout<<"Invalid"<<endl;
        }else if(inp=="Push"){
            int x;cin>>x;
            push(x);
            ↪  //cout<<"size="<<bbt.size()<<",smallest"<<bbt.find_by_order(0)->num<<endl
        }
    }
    return 0;
}
```

6.6 PBDS tree2

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <iostream>
#include <string>
using namespace std;
using namespace __gnu_pbds;

tree<int,string> t;
//use set? tree<int,null_type> t;
//iterator will also become Key.
//need null_mapped_type if version is lower than 4.4.0
//TAG:
//rb_tree_tag
//splay_tree_tag
//ov_tree_tag
int main(){
    t.begin();
    t.end();
    t.size();
    t.empty();
    t.clear();
    t[1]="orz";
    t[2]="123";
    t.find(1);
    t.lower_bound(1);
    t.upper_bound(1);
    t.erase(2);

    //t.join(other);
    //t.split(key,other);//split all node bigger than key into other
}
```

6.7 Trie

```
#include <iostream>
#include <vector>
#include <cstring>
#include <algorithm>
using namespace std;
const int MAXN=1010;
struct Node{
    Node *ch[30];
    int flag;
    int rc;
    Node(){
        for(int i=0;i<30;i++)ch[i]=NULL;
        flag=rc=0;
    }
} *root;

vector<string> inps;
string ans[MAXN];
void build(Node *node,string str,int ptr,int idx){
    node->rc++;
    if(ptr>=str.size()){
        node->flag=idx;
        return;
    }

    if(node->ch[str[ptr]-'a']==NULL)node->ch[str[ptr]-'a']=new Node();
    build(node->ch[str[ptr]-'a'],str,ptr+1,idx);
}

void travel(Node *node,string t){
    //cout<<node->rc<<endl;
    if(node->flag){
        //cout<<inps[node->flag]<<" "<<t<<endl;
        ans[node->flag]=t;
    }
    for(int i=0;i<26;i++){
        if(node->ch[i]!=NULL){
            //cout<<"->"<<char('a'+i)<<"->"<<endl;
            if(node->ch[i]->rc>1 || node->rc>1)travel(node->ch[i],t+char('a'+i));
            else travel(node->ch[i],t);
        }
    }
}
int main(){
    root=new Node();
    string inp;
    inps.push_back("#");
    while(cin>>inp){
        inps.push_back(inp);
        build(root,inp,0,inps.size()-1);
    }
    travel(root,"");
    for(int i=1;i<inps.size();i++){
        cout<<inps[i]<<" "<<ans[i]<<endl;
    }


    return 0;
}
```

6.8 dominator tree

```
#include <iostream>
#include <cstring>
#include <queue>
#include <algorithm>
#include <vector>
using namespace std;
constexpr int MAXV=100010;
constexpr int MAXP=20;

vector<int> g[MAXV];
vector<int> rev_g[MAXV];
int ind[MAXV];
void adde(int u,int v){
    g[u].push_back(v);
    rev_g[v].push_back(u);
    ind[v]++;
}

vector<int> topo_res;
int vlen,elen;
constexpr int SUPERS=100005;
void topo_sort(int u){
    queue<int> q;
    for(int u=1;u<=vlen;u++){
        if(!ind[u]){
            q.push(u);
            //mogician
            adde(SUPERS,u);
        }
    }
    while(!q.empty()){
        int u=q.front();q.pop();
        topo_res.push_back(u);
        for(auto v: g[u]){
            ind[v]--;
            if(!ind[v])q.push(v);
        }
    }
}

int lca[MAXV][MAXP];
int depth[MAXV];
void lca_increse(int u,int fa){
    lca[u][0]=fa;
    depth[u]=depth[fa]+1;
    for(int p=1;p<MAXP;p++){
        lca[u][p]=lca[lca[u][p-1]][p-1];
    }
}

int get_lca(int u,int v){
    if(depth[u]<depth[v])swap(u,v);
    for(int p=MAXP-1;p>=0;p--)if(depth[lca[u][p]]>=depth[v])u=lca[u][p];
    if(u==v)return u;
    for(int p=MAXP-1;p>=0;p--)if(lca[u][p]!=lca[v][p])u=lca[u][p],v=lca[v][p];
    return lca[u][0];
}



bool added[MAXV];
int main(){
    ios::sync_with_stdio(false);
    int kase;cin>>kase;
    while(kase--){
        cin>>vlen>>elen;
        for(int i=1;i<=vlen;i++)g[i].clear(),rev_g[i].clear();
        for(int i=0;i<=vlen;i++)for(int p=0;p<MAXP;p++)lca[i][p]=0;
        memset(added,0,sizeof(added));
        topo_res.clear();
        memset(ind,0,sizeof(ind));

        for(int i=0;i<elen;i++){
            int u,v;cin>>u>>v;
            adde(v,u);
        }
        topo_sort(SUPERS);
        //for(auto u:topo_res)cout<<u<<" ";
        //cout<<endl;

        added[SUPERS]=1;
        for(int i=0;i<topo_res.size();i++){
            //find common lca
            int clca=0;
            int u=topo_res[i];
            for(auto v:rev_g[u]){
                if(!added[v])continue;
                if(clca==0)clca=v;
                else clca=get_lca(clca,v);
                //cout<<"merge "<<v<<" = "<<clca<<endl;
            }
            lca_increse(u,clca);
            //cout<<clca<<"->"<<u<<endl;
            added[u]=1;
        }
        int qlen;cin>>qlen;
        while(qlen--){
            int u1,u2;cin>>u1>>u2;
            int l=get_lca(u1,u2);
            cout<<depth[u1]+depth[u2]-depth[l]<<endl;
        }
    }

    return 0;
}
```

6.9 linesegtree

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAX_IDX=20;
int a[MAX_IDX];//´  ¾
int id[MAX_IDX],idx=0;  ½//
int root;
int lc[MAX_IDX],rc[MAX_IDX];
int data[MAX_IDX]; ¾ ¤«  ½//
  ¼ //
int build(int &n,int l, int r){
    if(!n)n=++idx;
    data[n]=a[l];
    if(l>=r)return a[l];
    int mid=(l+r)/2;
    //max
    data[n]=max(build(lc[n],l,mid),build(rc[n],mid+1,r));
    return data[n];
}
//²        ² • ¶X  µ±ǰ½  儦 X µ±ǰ½
int query(int l,int r,int L,int R,int node){
    if(l<=L && R<=r)return data[node];//²  • ¶X
    int mid=(L+R)/2;
    int res=0;
    //   l
    if(l<=mid)res=max(res,query(l,r,L,mid,lc[node]));
    //  v
    if(r>mid)res=max(res,query(l,r,mid+1,R,rc[node]));
    return res;
}
//¼¼² µ á
int ql,qr;
int query(int l,int r,int node){
    if(l<=ql && qr<r)return data[node];
    int mid=(l+r)/2;
    int res=0;
    if(ql<=mid)res=max(res,query(l,mid,lc[node]));
    if(r>mid)res=max(res,query(mid+1,r,rc[node]));

    return res;
}
int main(){
    int n;cin>>n;
    for(int i=0;i<n;i++)cin>>a[i];
    build(root,0,n-1);
    for(int i=0;i<n;i++){
        for(int j=i;j<n;j++)debug(query(i,j,0,n-1,root));
    }
    return 0;
}
```

6.10 persistent seg

```
//别名主席树
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int maxn = 1e5; //数据范围
int tot, n, m;
int sum[(maxn << 5) + 10], rt[maxn + 10], ls[(maxn << 5) + 10],
    rs[(maxn << 5) + 10];
int a[maxn + 10], ind[maxn + 10], len;
inline int getid(const int &val) //离散化
{
    return lower_bound(ind + 1, ind + len + 1, val) - ind;
}
```

```
int build(int l, int r) //建树
{
    int root = ++tot;
    if (l == r)
        return root;
    int mid = l + r >> 1;
    ls[root] = build(l, mid);
    rs[root] = build(mid + 1, r);
    return root; //返回该子树的根节点
}
int update(int k, int l, int r, int root) //插入操作
{
    int dir = ++tot;
    ls[dir] = ls[root], rs[dir] = rs[root], sum[dir] = sum[root] + 1;
    if (l == r)
        return dir;
    int mid = l + r >> 1;
    if (k <= mid)
        ls[dir] = update(k, l, mid, ls[dir]);
    else
        rs[dir] = update(k, mid + 1, r, rs[dir]);
    return dir;
}
//left root, right root, querying l,r, the k-th
int query(int u, int v, int l, int r, int k) //查询操作
{
    int mid = l + r >> 1,
        x = sum[ls[v]] - sum[ls[u]]; //通过区间减法得到左儿子的信息
    if (l == r)
        return l;
    if (k <= x) //说明在左儿子中
        return query(ls[u], ls[v], l, mid, k);
    else //说明在右儿子中
        return query(rs[u], rs[v], mid + 1, r, k - x);
}
inline void init()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        scanf("%d", a + i);
    memcpy(ind, a, sizeof ind);
    sort(ind + 1, ind + n + 1);
    len = unique(ind + 1, ind + n + 1) - ind - 1;
    rt[0] = build(1, len);
    for (int i = 1; i <= n; ++i)
        rt[i] = update(getid(a[i]), 1, len, rt[i - 1]);
}
int l, r, k;
inline void work()
{
    while (m--)
    {
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", ind[query(rt[l - 1], rt[r], 1, len, k)]); //回答询问
    }
}
int main()
{
    init();
    work();
    return 0;
}
```

6.11 segmenttree flag

```
/*
 * Segment Tree
 * ver 1.0.0
 * node interval: [] (]
 * save data in array.
 *
 * support function:
 * 1. query interval
 * 2. modify interval (using flag)
 *
 * potential bugs:
 * flag doesn't replace data at leaves, which may lead to problems.
 * potential bugs in updating data.
 */
#include <iostream>
```

```
#include <algorithm>
using namespace std;
const int MAXN=10;

int a[MAXN];
int lc[MAXN],rc[MAXN],idx=0;
int root;
int data[MAXN];

int build(int &n,int l,int r){
    if(!n)n=++idx;
    data[n]=a[l];
    if(l>=r)return data[n];
    int mid=(l+r)/2;
    data[n]=max(build(lc[n],l,mid),build(rc[n],mid+1,r));
    return data[n];
}
void collectchild_n(int node){
    data[node]=max(data[lc[node]],data[rc[node]]);
}
int query_n(int l,int r,int L,int R,int node){
    if(l<=L && R<=r)return data[node];
    int mid=(L+R)/2;
    int res=0;
    if(l<=mid)res=max(res,query_n(l,r,L,mid,lc[node]));
    if(mid<r)res=max(res,query_n(l,r,mid+1,R,rc[node]));
    return res;
}
void modify_noflag(int l,int r,int x,int L,int R,int node){
    if(L>=R){
        data[node]=x;
        return;
    }
    int mid=(L+R)/2;
    if(l<=mid)modify_noflag(l,r,x,L,mid,lc[node]);
    if(mid<r)modify_noflag(l,r,x,mid+1,R,rc[node]);
    collectchild_n(node);
}

int flag[MAXN];
```

```
void collectchild(int node){
    int res=-0x7f7f7f7f;

    //we suppose 0 is no flag.
    if(flag[lc[node]])res=max(res,flag[lc[node]]);
    else res=max(res,data[lc[node]]);
    if(flag[rc[node]])res=max(res,flag[rc[node]]);
    else res=max(res,data[rc[node]]);
    data[node]=res;
}
void pushdown(int node){
    if(!flag[node])return;
    flag[lc[node]]=flag[node];
    flag[rc[node]]=flag[node];
    flag[node]=0;
}
int query(int l,int r,int L,int R,int node){
    if(l<=L && R<=r){
        if(flag[node])return flag[node];
        return data[node];
    }
    pushdown(node);
    int res=0;
    int mid=(L+R)/2;
    if(l<=mid)res=max(res,query(l,r,L,mid,lc[node]));
    if(mid<r)res=max(res,query(l,r,mid+1,R,rc[node]));
    return res;
}

void modify(int l,int r,int x,int L,int R,int node){
    if(l<=L && R<=r){
        flag[node]=x;
        return;
    }
    pushdown(node);
    int mid=(L+R)/2;
    if(l<=mid)modify(l,r,x,L,mid,lc[node]);
    if(mid<r)modify(l,r,x,mid+1,R,rc[node]);
    collectchild(node);
}
```

```
/*
 * a problem
 * when flag being pushed down to leaves, it doesn't replace data.
 */




int main(){
    int n;cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }
    build(root,1,n);
    int op;
    while(cin>>op){
        if(op==1){
            int l,r;cin>>l>>r;
            cout<<query(l,r,1,n,root)<<endl;
        }else if(op==2){
            int l,r,x;cin>>l>>r>>x;
            modify(l,r,x,1,n,root);
        }else if(op==0){
            for(int i=1;i<=n;i++){
                cout<<query(i,i,1,n,root)<<" ";
            }
            cout<<endl;
        }
    }

    return 0;
}
```

6.12 splay pool data

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

const int MAXN=10;

```

```
struct Node{
    Node *ch[2];
    int key;
    Node *fa;
    int data;//this is just a example.in this, we count the size of subtree.
    //return the position of node, left or right child.
    int pos(Node *node){
        return ch[1]==node;
    }
    Node(int val):key(val){}
    Node(){}
    Node(int val,Node *fa):key(val),fa(fa){}

    static int size(Node *node){
        return node?node->data:0;
    }
    static void update(Node *node){
        node->data=Node::size(node->ch[0])+Node::size(node->ch[1])+1;
        cout<<"node "<<node->key<<":"<<node->data<<endl;
    }

};

struct NodePool{
    Node nodes[MAXN];
    queue<Node*> q;
    NodePool(){
        for(int i=0;i<MAXN;i++)q.push(&nodes[i]);
    }
    Node* newnode(){
        if(q.empty()){
            cout<<"ERROR:memory pool is empty!"<<endl;
            return 0;
        }
        Node *res=q.front();q.pop();
        res->ch[0]=res->ch[1]=0;
        res->fa=0;
        res->key=0;
        return res;
    }
    void freenode(Node *node){
        q.push(node);
    }
};
struct Splay{
    NodePool pool;
    Node *root;
    Splay(){
        root=NULL;
    }
    void rotate(Node *node){
        Node *fa=node->fa;
        if(fa==NULL)return;

        int wai=fa->pos(node);
        fa->ch[wai]=node->ch[!wai];
        if(fa->ch[wai]!=NULL) fa->ch[wai]->fa=fa;

        node->ch[!wai]=fa;

        node->fa=fa->fa;
        if(node->fa!=NULL)node->fa->ch[node->fa->pos(fa)]=node;
        else root=node;
        fa->fa=node;

        //update the data and push tag down.
        Node::update(fa);Node::update(node);
    }
    //splay will rotate node until it becomes target's children.
    void splay(Node *node,Node *target){
        //if node and node'father are all at same position,we rotate p and x in order.
        //if node and node'father are at different position, we rotate x twice.
        while(node->fa!=target){
            Node *fa=node->fa;
            Node *gr=fa->fa;
            if(gr==target)rotate(node);
            else{
                if(gr->pos(fa) ^ fa->pos(node)){
                    rotate(node);rotate(node); //zig-zig
```

```
             }else{
                 rotate(fa);rotate(node); //zig-zag
             }
         }
     }
 }
//insert a new value at a empty leaf, then splay it.
 void insert(int val){
     if(root==NULL)root=new Node(val);
     //to make sure left child is smaller strictly than its father
     for(Node *node=root;node;node=node->ch[val>=node->key]){
         if(node->key==val){
             //TODO: keys in this tree are unique. maybe it should be
             ↪   improved.
             splay(node,NULL);return;
         }
         if(node->ch[val>=node->key]==NULL)
             node->ch[val>=node->key]=new Node(val,node);
     }
 }
 //take the one to root.
 //if it has no left child, just delete it and push its right child.
 //if not, find and splay the biggest child in its left subtree to root's. at
 ↪   this time, this node has no right child absolutely(just calculate it).
 ↪   attach root's right tree to it. well done.
 void erase(int key){
     Node *node=root;
     //find node with this key.
     while(node){
         if(node->key==key)break;
         node=node->ch[key>node->key];
     }
     if(node!=NULL){
         //splay it to root.
         splay(node,NULL);
         if(node->ch[0]==NULL){
             //if it has no left child, take it right one to root.
             root=node->ch[1];
             if(root!=NULL)root->fa=NULL;
         }else{
                Node *lc=node->ch[0];
                while(lc->ch[1]!=NULL)lc=lc->ch[1];
                splay(lc,node);root=lc;
                root->fa=NULL;
                lc->ch[1]=node->ch[1];
                if(lc->ch[1]!=NULL)lc->ch[1]->fa=lc;
            }
        }
    }
    void print(Node *node){
        if(node==NULL)return;
        cout<<node->key<<":";
        if(node->ch[0]!=NULL)cout<<node->ch[0]->key;
        cout<<",";
        if(node->ch[1]!=NULL)cout<<node->ch[1]->key;
        cout<<endl;
        print(node->ch[0]);print(node->ch[1]);
    }
};

int main(){
    Splay splay;
    int op,inp;
    while(cin>>op>>inp){
        if(op==1)splay.insert(inp);
        else if(op==2)splay.erase(inp);
        else if(op==3)splay.print(splay.root);
        cout<<"============="<<endl;
    }
    return 0;
}
```

6.13 splay seq

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

const int MAXN=100;
```

```
struct Node{
    Node *ch[2];
    double key;
    int val;
    int isum=0;
    Node *fa;
    int data;//this is just a example.in this, we count the size of subtree.
    bool sign;
    //return the position of node in self's children, left or right child.
    int pos(Node *node){
        return ch[1]==node;
    }
    Node(double key):key(key){
        ch[0]=ch[1]=NULL;
        fa=NULL;
    }
    //Node(){}
    Node(double key,Node *fa):Node(key){
        this->fa=fa;
    }

    void setval(int val){
        this->val=val;
    }
    static int getval(Node *node){
        return node?node->val:0;
    }
    static int getsum(Node *node){
        return node?node->isum:0;
    }

    static int size(Node *node){
        return node?node->data:0;
    }
    static void update(Node *node){
        node->data=Node::size(node->ch[0])+Node::size(node->ch[1])+1;

         ↪  node->isum=Node::getsum(node->ch[0])+Node::getsum(node->ch[1])+Node::getval(n
        cout<<"node "<<node->key<<":"<<node->data<<endl;
        cout<<"node.s "<<node->key<<":"<<node->isum<<endl;
    }
    static bool tag(Node *node){
        return node?node->sign:0;
    }
    static bool tag(Node *node,bool val){
        if(node)node->sign=val;
        return tag(node);
    }

};

struct Splay{
    Node *root;
    Splay(){
        root=NULL;
    }
    void rotate(Node *node){
        Node *fa=node->fa;
        if(fa==NULL)return;

        //tag should be pushed down before rotating.
        pushdown(fa);
        pushdown(node);

        int wai=fa->pos(node);
        fa->ch[wai]=node->ch[!wai];
        if(fa->ch[wai]!=NULL) fa->ch[wai]->fa=fa;

        node->ch[!wai]=fa;

        node->fa=fa->fa;
        if(node->fa!=NULL)node->fa->ch[node->fa->pos(fa)]=node;
        else root=node;
        fa->fa=node;

        //update the data.
        Node::update(fa);Node::update(node);
    }
    //splay will rotate node until it becomes target's children.
```

```
void splay(Node *node,Node *target){
    //if node and node'father are all at same position,we rotate p and x in
     ↪  order.
    //if node and node'father are at different position, we rotate x twice.
    while(node->fa!=target){
        Node *fa=node->fa;
        Node *gr=fa->fa;
        if(gr==target)rotate(node);
        else{
            if(gr->pos(fa) ^ fa->pos(node)){
                rotate(node);rotate(node); //zig-zig
            }else{
                rotate(fa);rotate(node); //zig-zag
            }
        }
    }
    node->update(node);
}
void pushdown(Node *node){
    if(!Node::tag(node))return;
    Node::tag(node->ch[0],Node::tag(node));
    Node::tag(node->ch[1],Node::tag(node));
}
//[l,r]
void buildseq(int l,int r){
    Node *nl=findpre(l);
    Node *rl=findsuf(r);
    splay(nl,NULL);
    splay(rl,nl);
}



//insert a new value at a empty leaf, then splay it.
void insert(double key,int val){
    if(root==NULL){
        root=new Node(key);
        root->setval(val);
    }
    //to make sure left child is smaller strictly than its father
```

```
        for(Node *node=root;node;node=node->ch[key>=node->key]){
            if(node->key==key){
                //TODO: keys in this tree are unique. maybe it should be
                 ↪  improved.
                splay(node,NULL);return;
            }
            if(node->ch[key>=node->key]==NULL){
                node->ch[key>=node->key]=new Node(key,node);
                node->ch[key>=node->key]->setval(val);
            }
        }
    }
    //take the one to root.
    //if it has no left child, just delete it and push its right child.
    //if not, find and splay the biggest child in its left subtree to root's. at
     ↪  this time, this node has no right child absolutely(just calculate it).
     ↪  attach root's right tree to it. well done.
    void erase(int key){
        Node *node=root;
        //find node with this key.
        while(node){
            if(node->key==key)break;
            node=node->ch[key>node->key];
        }
        if(node!=NULL){
            //splay it to root.
            splay(node,NULL);
            if(node->ch[0]==NULL){
                //if it has no left child, take it right one to root.
                root=node->ch[1];
                if(root!=NULL)root->fa=NULL;
            }else{

                Node *lc=node->ch[0];
                while(lc->ch[1]!=NULL)lc=lc->ch[1];
                splay(lc,node);root=lc;
                root->fa=NULL;
                lc->ch[1]=node->ch[1];
                if(lc->ch[1]!=NULL)lc->ch[1]->fa=lc;
            }
        }
    }
    int getdata(int key){
        Node *t=root;
        while(t){
            if(t->key==key)break;
            t=t->ch[key>t->key];
        }
        int res=-1;
        if(t!=NULL){
            res=t->isum+t->val;
            splay(t,NULL);
        }
        return res;
    }
    Node* find(int key){
        Node *t=root;
        while(t){
            if(t->key==key)break;
            t=t->ch[key>t->key];
        }
        if(t!=NULL){
            splay(t,NULL);
            return t;
        }
        return NULL;
    }
    Node* findpre(int key){
        Node *t=find(key);
        if(t==NULL)return NULL;
        Node *p=t->ch[0];
        if(p==NULL)return NULL;
        while(p->ch[1]!=NULL)p=p->ch[1];
        splay(p,NULL);
        return p;
    }
    Node* findsuf(int key){
        Node *t=find(key);
        if(t==NULL)return NULL;
        Node *p=t->ch[1];
        if(p==NULL)return NULL;
        while(p->ch[0]!=NULL)p=p->ch[0];
        splay(p,NULL);
        return p;
    }
    Node* findbyrank(int rank){
        Node *t=root;
        while(t!=NULL){
            if(Node::size(t->ch[0])+1==rank){
                splay(t,NULL);
                return t;
            }
            else if(Node::size(t->ch[0])>=rank)t=t->ch[0];
            else{
                rank-=Node::size(t->ch[0])+1;
                t=t->ch[1];
            }
        }
        return NULL;
    }
    void print(Node *node){
        if(node==NULL)return;
        cout<<node->key<<":";
        if(node->ch[0]!=NULL)cout<<node->ch[0]->key;
        cout<<",";
        if(node->ch[1]!=NULL)cout<<node->ch[1]->key;
        cout<<endl;
        print(node->ch[0]);print(node->ch[1]);
    }
};
int tick=1;
int main(){
    Splay splay;
    /*
    int nlen,mlen;cin>>nlen>>mlen;
    splay.insert(tick,0);
    for(int i=0;i<nlen;i++){
        int t;cin>>t;
        splay.insert(tick*100000,t);
        t++;
    */
    string op;
    int inp;
    splay.insert(0,0);
    splay.insert(0x3f3f3f3f,0);
    while(cin>>op>>inp){
        if(op=="i"){
            int val;cin>>val;
            splay.insert(inp,val);
        }
        else if(op=="e")splay.erase(inp);
        else if(op=="rk"){
            Node *res=splay.findbyrank(inp);
            if(res)cout<<res->key<<endl;
            else cout<<"NO"<<endl;
        }
        else if(op=="d")splay.print(splay.root);
        else if(op=="fp")splay.findpre(inp);
        else if(op=="fs")splay.findsuf(inp);
        else if(op=="size")cout<<splay.getdata(inp)<<endl;
        else if(op=="ms"){
            int r;cin>>r;
            splay.buildseq(inp,r);
        }
        cout<<"============="<<endl;
    }
    return 0;
}
```

6.14 splay simple

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

const int MAXN=10;

struct Node{
    Node *ch[2];
    int key;
    Node *fa;
    //return the position of node, left or right child.
    int pos(Node *node){
        return ch[1]==node;
    }
    Node(int val):key(val){}
    Node(){}
    Node(int val,Node *fa):key(val),fa(fa){}
}nodes[MAXN];
struct Splay{
    Node *root;
    Splay(){
        root=NULL;
    }
    void rotate(Node *node){
        Node *fa=node->fa;
        if(fa==NULL)return;

        int wai=fa->pos(node);
        fa->ch[wai]=node->ch[!wai];
        if(fa->ch[wai]!=NULL) fa->ch[wai]->fa=fa;

        node->ch[!wai]=fa;

        node->fa=fa->fa;
        if(node->fa!=NULL)node->fa->ch[node->fa->pos(fa)]=node;
        else root=node;
        fa->fa=node;
    }
    //splay will rotate node until it becomes target's children.
    void splay(Node *node,Node *target){
        //if node and node'father are all at same position,we rotate p and x in
         ↪  order.
        //if node and node'father are at different position, we rotate x twice.
        while(node->fa!=target){
            Node *fa=node->fa;
            Node *gr=fa->fa;
            if(gr==target)rotate(node);
            else{
                if(gr->pos(fa) ^ fa->pos(node)){
                    rotate(node);rotate(node); //zig-zig
```

```
             }else{
                 rotate(fa);rotate(node); //zig-zag
             }
         }
     }
 }
//insert a new value at a empty leaf, then splay it.
 void insert(int val){
     if(root==NULL)root=new Node(val);
     //to make sure left child is smaller strictly than its father
     for(Node *node=root;node;node=node->ch[val>=node->key]){
         if(node->key==val){
             //TODO: keys in this tree are unique. maybe it should be
             ↪  improved.
             splay(node,NULL);return;
         }
         if(node->ch[val>=node->key]==NULL)
             node->ch[val>=node->key]=new Node(val,node);
     }
 }
 //take the one to root.
 //if it has no left child, just delete it and push its right child.
 //if not, find and splay the biggest child in its left subtree to root's. at
 ↪  this time, this node has no right child absolutely(just calculate it).
 ↪  attach root's right tree to it. well done.
 void erase(int key){
     Node *node=root;
     //find node with this key.
     while(node){
         if(node->key==key)break;
         node=node->ch[key>node->key];
     }
     if(node!=NULL){
         //splay it to root.
         splay(node,NULL);
         if(node->ch[0]==NULL){
             //if it has no left child, take it right one to root.
             root=node->ch[1];
             if(root!=NULL)root->fa=NULL;
         }else{
                Node *lc=node->ch[0];
                while(lc->ch[1]!=NULL)lc=lc->ch[1];
                splay(lc,node);root=lc;
                root->fa=NULL;
                lc->ch[1]=node->ch[1];
                if(lc->ch[1]!=NULL)lc->ch[1]->fa=lc;
            }
        }
    }
    void print(Node *node){
        if(node==NULL)return;
        cout<<node->key<<":";
        if(node->ch[0]!=NULL)cout<<node->ch[0]->key;
        cout<<",";
        if(node->ch[1]!=NULL)cout<<node->ch[1]->key;
        cout<<endl;
        print(node->ch[0]);print(node->ch[1]);
    }
};

int main(){
    Splay splay;
    int op,inp;
    while(cin>>op>>inp){
        if(op==1)splay.insert(inp);
        else if(op==2)splay.erase(inp);
        else if(op==3)splay.print(splay.root);
        cout<<"============="<<endl;
    }
    return 0;
}
```

6.15 splay ununique

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

const int MAXN=100010;
```

```
struct Node{
    Node *ch[2];
    int key;
    int val;
    int isum=0;
    Node *fa;
    int data;//this is just a example.in this, we count the size of subtree.
    bool sign;
    int cnt;
    //return the position of node in self's children, left or right child.
    int pos(Node *node){
        return ch[1]==node;
    }
    Node(int key):key(key){
        ch[0]=ch[1]=NULL;
        fa=NULL;
        cnt=0;
    }
    //Node(){}
    Node(int key,Node *fa):Node(key){
        this->fa=fa;
    }

    void setval(int val){
        this->val=val;
    }
    static int getval(Node *node){
        return node?node->val:0;
    }
    static int getsum(Node *node){
        return node?node->isum:0;
    }

    static int size(Node *node){
        return node?node->data:0;
    }
    static void update(Node *node){
        node->data=Node::size(node->ch[0])+Node::size(node->ch[1])+node->cnt;
        ↪ node->isum=Node::getsum(node->ch[0])+Node::getsum(node->ch[1])+Node::getval(n
        //cout<<"node "<<node->key<<":"<<node->data<<endl;
        //cout<<"node.s "<<node->key<<":"<<node->isum<<endl;
    }
    static bool tag(Node *node){
        return node?node->sign:0;
    }
    static bool tag(Node *node,bool val){
        if(node)node->sign=val;
        return tag(node);
    }

};

struct Splay{
    Node *root;
    Splay(){
        root=NULL;
    }
    void rotate(Node *node){
        Node *fa=node->fa;
        if(fa==NULL)return;

        //tag should be pushed down before rotating.
        pushdown(fa);
        pushdown(node);

        int wai=fa->pos(node);
        fa->ch[wai]=node->ch[!wai];
        if(fa->ch[wai]!=NULL) fa->ch[wai]->fa=fa;

        node->ch[!wai]=fa;

        node->fa=fa->fa;
        if(node->fa!=NULL)node->fa->ch[node->fa->pos(fa)]=node;
        else root=node;
        fa->fa=node;

        //update the data.
        Node::update(fa);Node::update(node);
    }
    //splay will rotate node until it becomes target's children.
    void splay(Node *node,Node *target){
        //if node and node'father are all at same position,we rotate p and x in
         ↪  order.
        //if node and node'father are at different position, we rotate x twice.
        while(node->fa!=target){
            Node *fa=node->fa;
            Node *gr=fa->fa;
            if(gr==target)rotate(node);
            else{
                if(gr->pos(fa) ^ fa->pos(node)){
                    rotate(node);rotate(node); //zig-zig
                }else{
                    rotate(fa);rotate(node); //zig-zag
                }
            }
        }
        node->update(node);
    }
    void pushdown(Node *node){
        if(!Node::tag(node))return;
        Node::tag(node->ch[0],Node::tag(node));
        Node::tag(node->ch[1],Node::tag(node));
    }
    //[l,r]
    void buildseq(int l,int r){
        Node *nl=findpre(l);
        Node *rl=findsuf(r);
        splay(nl,NULL);
        splay(rl,nl);
    }



   //insert a new value at a empty leaf, then splay it.
    void insert(double key,int val){
        if(root==NULL){
            root=new Node(key);
            root->setval(val);
        }
        //to make sure left child is smaller strictly than its father
        for(Node *node=root;node;node=node->ch[key>=node->key]){
            if(node->key==key){
                //TODO: keys in this tree are unique. maybe it should be
                ↪  improved.
                node->cnt++;
                splay(node,NULL);return;
            }
            if(node->ch[key>=node->key]==NULL){
                node->ch[key>=node->key]=new Node(key,node);
                node->ch[key>=node->key]->setval(val);
            }
        }
    }
    //take the one to root.
    //if it has no left child, just delete it and push its right child.
    //if not, find and splay the biggest child in its left subtree to root's. at
    ↪  this time, this node has no right child absolutely(just calculate it).
    ↪  attach root's right tree to it. well done.
    void erase(int key){
        Node *node=root;
        //find node with this key.
        while(node){
            if(node->key==key)break;
            node=node->ch[key>node->key];
        }
        if(node!=NULL){
            //splay it to root.
            splay(node,NULL);
            if(node->cnt>=2){
                node->cnt--;
                return;
            }
            if(node->ch[0]==NULL){
                //if it has no left child, take it right one to root.
                root=node->ch[1];
                if(root!=NULL)root->fa=NULL;
            }else{
                Node *lc=node->ch[0];
                while(lc->ch[1]!=NULL)lc=lc->ch[1];
                splay(lc,node);root=lc;
                root->fa=NULL;
                lc->ch[1]=node->ch[1];
                if(lc->ch[1]!=NULL)lc->ch[1]->fa=lc;
            }
        }
    }
    int getdata(int key){
        Node *t=root;
        while(t){
            if(t->key==key)break;
            t=t->ch[key>t->key];
        }
        int res=-1;
        if(t!=NULL){
            res=t->isum+t->val;
            splay(t,NULL);
        }
        return res;
    }
    Node* find(int key){
        Node *t=root;
        while(t){
            if(t->key==key)break;
            t=t->ch[key>t->key];
        }
        if(t!=NULL){
            splay(t,NULL);
            return t;
        }
        return NULL;
    }
    Node* findpre(int key){
        Node *t=find(key);
        if(t==NULL)return NULL;
        Node *p=t->ch[0];
        if(p==NULL)return NULL;
        while(p->ch[1]!=NULL)p=p->ch[1];
        splay(p,NULL);
        return p;
    }
    Node* findsuf(int key){
        Node *t=find(key);
        if(t==NULL)return NULL;
        Node *p=t->ch[1];
        if(p==NULL)return NULL;
        while(p->ch[0]!=NULL)p=p->ch[0];
        splay(p,NULL);
        return p;
    }
    Node* findbyrank(int rank){
        Node *t=root;
        while(t!=NULL){
            if(Node::size(t->ch[0])+1<=rank &&
            ↪   rank<=Node::size(t->ch[0])+t->cnt){
                splay(t,NULL);
                return t;
            }
            else if(Node::size(t->ch[0])>=rank)t=t->ch[0];
            else{
                rank-=Node::size(t->ch[0])+t->cnt;
                t=t->ch[1];
            }
        }
        return NULL;
    }
    void print(Node *node){
        if(node==NULL)return;
        cout<<node->key<<":";
        if(node->ch[0]!=NULL)cout<<node->ch[0]->key;
        cout<<",";
        if(node->ch[1]!=NULL)cout<<node->ch[1]->key;
        cout<<endl;
        print(node->ch[0]);print(node->ch[1]);
    }
};
int tick=1;
int main(){
    Splay splay;
    /*
    int nlen,mlen;cin>>nlen>>mlen;
    splay.insert(tick,0);
    for(int i=0;i<nlen;i++){
        int t;cin>>t;
        splay.insert(tick*100000,t);
        t++;
    */
    int op;
    int inp;
    cin>>op;
    //splay.insert(0,0);
    //splay.insert(0x3f3f3f3f,0);
    while(cin>>op>>inp){
        if(op==1){
            int val;
            splay.insert(inp,0);
        }
        else if(op==2)splay.erase(inp);
        else if(op==4){
            Node *res=splay.findbyrank(inp);
            if(res)cout<<res->key<<endl;
            else while(1);
        }
        else if(op==3){
            //to find the bucunzai
            splay.insert(inp,0);
            Node *res=splay.find(inp);
            if(res)cout<<Node::size(res->ch[0])+1<<endl;
            else while(1);
            splay.erase(inp);
        }
        else if(op==5){
            //to find the bucunzai
            splay.insert(inp,0);
            cout<<splay.findpre(inp)->key<<endl;
            splay.erase(inp);
        }
        else if(op==6){
            splay.insert(inp,0);
            cout<<splay.findsuf(inp)->key<<endl;
            splay.erase(inp);
        }
        //cout<<"============="<<endl;
    }
    return 0;
}
```

6.16 splaytest

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

const int MAXN=100;

struct Node{
    Node *ch[2];
    int key,val;
    int isum=0;
    Node *fa;
    int data;//this is just a example.in this, we count the size of subtree.
    bool sign;
    //return the position of node in self's children, left or right child.
    int pos(Node *node){
        return ch[1]==node;
    }
    Node(int key):key(key){}
    Node(){}
    Node(int key,Node *fa):key(key),fa(fa){}

    void setval(int val){
        this->val=val;
    }
    static int getval(Node *node){
        return node?node->val:0;
    }
    static int getsum(Node *node){
        return node?node->isum:0;
    }

    static int size(Node *node){
        return node?node->data:0;
    }
    static void update(Node *node){
        node->data=Node::size(node->ch[0])+Node::size(node->ch[1])+1;

        ↪  node->isum=Node::getsum(node->ch[0])+Node::getsum(node->ch[1])+Node::getval(n
        cout<<"node "<<node->key<<":"<<node->data<<endl;
        cout<<"node.s "<<node->key<<":"<<node->isum<<endl;
    }
    static bool tag(Node *node){
        return node?node->sign:0;
    }
    static bool tag(Node *node,bool val){
        if(node)node->sign=val;
        return tag(node);
    }

};

struct NodePool{
    Node nodes[MAXN];
    queue<Node*> q;
    NodePool(){
        for(int i=0;i<MAXN;i++)q.push(&nodes[i]);
    }
    Node* newnode(){
        if(q.empty()){
            cout<<"ERROR:memory pool is empty!"<<endl;
            return 0;
        }
        Node *res=q.front();q.pop();
        res->ch[0]=res->ch[1]=0;
        res->fa=0;
        res->key=0;
        return res;
    }
    void freenode(Node *node){
        q.push(node);
    }
};
struct Splay{
    NodePool pool;
    Node *root;
    Splay(){
        root=NULL;
    }
    void rotate(Node *node){
        Node *fa=node->fa;
        if(fa==NULL)return;

        //tag should be pushed down before rotating.
        pushdown(fa);
        pushdown(node);

        int wai=fa->pos(node);
        fa->ch[wai]=node->ch[!wai];
        if(fa->ch[wai]!=NULL) fa->ch[wai]->fa=fa;

        node->ch[!wai]=fa;

        node->fa=fa->fa;
        if(node->fa!=NULL)node->fa->ch[node->fa->pos(fa)]=node;
        else root=node;
        fa->fa=node;

        //update the data.
        Node::update(fa);Node::update(node);
    }
    //splay will rotate node until it becomes target's children.
    void splay(Node *node,Node *target){
        //if node and node'father are all at same position,we rotate p and x in
         ↪  order.
        //if node and node'father are at different position, we rotate x twice.
        while(node->fa!=target){
            Node *fa=node->fa;
            Node *gr=fa->fa;
            if(gr==target)rotate(node);
            else{
                if(gr->pos(fa) ^ fa->pos(node)){
                    rotate(node);rotate(node); //zig-zig
                }else{
                    rotate(fa);rotate(node); //zig-zag
                }
            }
        }
        node->update(node);
    }
    void pushdown(Node *node){
        if(!Node::tag(node))return;
        Node::tag(node->ch[0],Node::tag(node));
        Node::tag(node->ch[1],Node::tag(node));
    }
    //[l,r]
    void buildseq(int l,int r){
        Node *nl=findpre(l);
        Node *rl=findsuf(r);
        splay(nl,NULL);
        splay(rl,nl);
    }



   //insert a new value at a empty leaf, then splay it.
    void insert(int key){
        if(root==NULL){
            root=new Node(key);
        }
        //to make sure left child is smaller strictly than its father
        for(Node *node=root;node;node=node->ch[key>=node->key]){
            if(node->key==key){
                //TODO: keys in this tree are unique. maybe it should be
                ↪  improved.
                splay(node,NULL);return;
            }
            if(node->ch[key>=node->key]==NULL)
                node->ch[key>=node->key]=new Node(key,node);
        }
```

```
}
//take the one to root.
//if it has no left child, just delete it and push its right child.
//if not, find and splay the biggest child in its left subtree to root's. at
↪  this time, this node has no right child absolutely(just calculate it).
↪  attach root's right tree to it. well done.
void erase(int key){
    Node *node=root;
    //find node with this key.
    while(node){
        if(node->key==key)break;
        node=node->ch[key>node->key];
    }
    if(node!=NULL){
        //splay it to root.
        splay(node,NULL);
        if(node->ch[0]==NULL){
            //if it has no left child, take it right one to root.
            root=node->ch[1];
            if(root!=NULL)root->fa=NULL;
        }else{

            Node *lc=node->ch[0];
            while(lc->ch[1]!=NULL)lc=lc->ch[1];
            splay(lc,node);root=lc;
            root->fa=NULL;
            lc->ch[1]=node->ch[1];
            if(lc->ch[1]!=NULL)lc->ch[1]->fa=lc;
        }
    }
}
int getdata(int key){
    Node *t=root;
    while(t){
        if(t->key==key)break;
        t=t->ch[key>t->key];
    }
    int res=-1;
    if(t!=NULL){
        res=t->data;
```

```
            splay(t,NULL);
        }
        return res;
    }
    Node* find(int key){
        Node *t=root;
        while(t){
            if(t->key==key)break;
            t=t->ch[key>t->key];
        }
        if(t!=NULL){
            splay(t,NULL);
            return t;
        }
        return NULL;
    }
    Node* findpre(int key){
        Node *t=find(key);
        if(t==NULL)return NULL;
        Node *p=t->ch[0];
        if(p==NULL)return NULL;
        while(p->ch[1]!=NULL)p=p->ch[1];
        splay(p,NULL);
        return p;
    }
    Node* findsuf(int key){
        Node *t=find(key);
        if(t==NULL)return NULL;
        Node *p=t->ch[1];
        if(p==NULL)return NULL;
        while(p->ch[0]!=NULL)p=p->ch[0];
        splay(p,NULL);
        return p;
    }
    void print(Node *node){
        if(node==NULL)return;
        cout<<node->key<<":";
        if(node->ch[0]!=NULL)cout<<node->ch[0]->key;
        cout<<",";
        if(node->ch[1]!=NULL)cout<<node->ch[1]->key;
        cout<<endl;
        print(node->ch[0]);print(node->ch[1]);
    }
};

int main(){
    Splay splay;
    string op;
    int inp;
    splay.insert(0);
    splay.insert(0x3f3f3f3f);
    while(cin>>op>>inp){
        if(op=="i")splay.insert(inp);
        else if(op=="e")splay.erase(inp);
        else if(op=="d")splay.print(splay.root);
        else if(op=="fp")splay.findpre(inp);
        else if(op=="fs")splay.findsuf(inp);
        else if(op=="size")cout<<splay.getdata(inp)<<endl;
        else if(op=="ms"){
            int r;cin>>r;
            splay.buildseq(inp,r);
        }
        cout<<"============="<<endl;
    }
    return 0;
}
```

6.17 treap

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

struct Node{
    int key,rk;
    int data;
    int cnt;
    Node *ch[2];
    Node *fa;
    Node(int key,Node *fa){
        this->key=key;
        this->fa=fa;
        cnt=data=0;
        ch[0]=ch[1]=NULL;
        rk=rand();
    }
    int pos(Node *node){
        return ch[1]==node;
    }
    int getsize(){
        return cnt+data;
    }
    void update(){
        data=0;
        if(ch[0])data+=ch[0]->getsize();
        if(ch[1])data+=ch[1]->getsize();
    }
};

struct Treap{
    Node *root;
    Treap(){
        root=NULL;
    }

    void rotate(Node *node){
        Node *fa=node->fa;
        if(fa==NULL)return;

        int wai=fa->pos(node);
        fa->ch[wai]=node->ch[!wai];
        if(fa->ch[wai]!=NULL) fa->ch[wai]->fa=fa;

        node->ch[!wai]=fa;

        node->fa=fa->fa;
        if(node->fa!=NULL)node->fa->ch[node->fa->pos(fa)]=node;
        else root=node;
        fa->fa=node;

        if(fa)fa->update();
        node->update();
    }

    //this node must be in tree
    void treap(Node *node){
        if(node==NULL)return;
        while(node->fa!=NULL && node->fa->rk<node->rk){
            rotate(node);
        }
    }
    Node* find(int key){
        Node *ptr=root;
        while(ptr!=NULL){
            if(ptr->key==key)return ptr;
            if(ptr->key>key)ptr=ptr->ch[0];
            else ptr=ptr->ch[1];
        }
        return NULL;
    }
    void INSERT(int key){
        if(root==NULL)root=new Node(key,NULL);
        insert(root,key);

        Treap::dprint(root);
    }
    void insert(Node *ptr,int key){
        if(ptr->key==key){
            ptr->cnt++;
            treap(ptr);
        }else if(ptr->key>key){
            if(ptr->ch[0]==NULL)ptr->ch[0]=new Node(key,ptr);
            insert(ptr->ch[0],key);
        }else{
            if(ptr->ch[1]==NULL)ptr->ch[1]=new Node(key,ptr);
            insert(ptr->ch[1],key);
        }
        ptr->update();
    }
    Node* findn(int key,bool pre){
        Node *node=find(key);
        Node *fa=node->fa;
        cout<<node->key<<endl;
        Node *left=node->ch[pre^1];
        while(left!=NULL && left->ch[pre]!=NULL)left=left->ch[pre];

        if(fa!=NULL && fa->pos(node)==pre){
            if(left==NULL)return fa;
            else return (left->key>fa->key)==pre?left:fa;
        } else return left;
    }
    static void dprint(Node *node){
        if(!node)return;
        cout<<node->key<<"("<<node->getsize()<<"):";
        if(node->ch[0]){
            cout<<node->ch[0]->key;
        }
        if(node->ch[1]){
            cout<<","<<node->ch[1]->key;
        }
        cout<<endl;
        if(node->ch[0])dprint(node->ch[0]);
        if(node->ch[1])dprint(node->ch[1]);
    }
    int getrank(int key){
        Node *ptr=root;
        //res is the number of node that less than key.
        int res=0;
        while(ptr!=NULL){
            if(ptr->key==key)return res;
            if(ptr->key>key)ptr=ptr->ch[0];
            else{
                if(ptr->ch[1])res+=ptr->ch[1]->getsize();
                res+=ptr->cnt;
                ptr=ptr->ch[1];
            }
        }
        return res+1;
    }
    Node* getbyorder(int order){
        Node *ptr=root;
        while(ptr!=NULL && order>0){
            int temp=0;
            if(ptr->ch[0])temp+=ptr->getsize();
            if(temp+1<=order && order<=temp+ptr->cnt)return ptr;
            if(temp>order){
                order-=temp+ptr->cnt;
                ptr=ptr->ch[1];
            }else{
                ptr=ptr->ch[0];
            }
        }
        return ptr;
    }

};



Treap t;
int main(){
    int kase;cin>>kase;
    while(kase--){
        int op,n;
        cin>>op>>n;
        if(op==1)t.INSERT(n);
        else if(op==2);
        else if(op==3)cout<<t.getrank(n);
        else if(op==4)cout<<t.getbyorder(n)->key<<endl;
        else if(op==5)cout<<t.findn(n,1)->key<<endl;
        else if(op==6)cout<<t.findn(n,0)->key<<endl;
        else if(op==7)Treap::dprint(t.root);
        cout<<"="<<endl;
    }
    return 0;
}
```

6.18 ST table

```
#include <bits/stdc++.h>
using namespace std;
const int logn = 21;
const int maxn = 2000001;
int f[maxn][logn], Logn[maxn];
inline int read()
{
    char c = getchar();
    int x = 0, f = 1;
    while (c < '0' || c > '9')
    {
        if (c == '-')
            f = -1;
        c = getchar();
    }
    while (c >= '0' && c <= '9')
    {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x * f;
}
void pre()
{
    Logn[1] = 0;
    Logn[2] = 1;
    for (int i = 3; i < maxn; i++)
    {
        Logn[i] = Logn[i / 2] + 1;
    }
}
int main()
{
    int n = read(), m = read();
    for (int i = 1; i <= n; i++)
        f[i][0] = read();
    pre();
    for (int j = 1; j <= logn; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
```

```
            f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
    for (int i = 1; i <= m; i++)
    {
        int x = read(), y = read();
        int s = Logn[y - x + 1];
        printf("%d\n", max(f[x][s], f[y - (1 << s) + 1][s]));
    }
    return 0;
}
```

6.19 odt tree

```
//珂朵莉树，用于暴力维护区间信息，在随机数据下表现优秀，且好写（

struct Node_t {
  int l, r;
  mutable int v;
  Node_t(const int &il, const int &ir, const int &iv) : l(il), r(ir), v(iv) {}
  inline bool operator<(const Node_t &o) const { return l < o.l; }
};

set<Node_t> odt;
//区间划分
auto split(int x) {
  if (x > n) return odt.end();
  auto it = --odt.upper_bound((Node_t){x, 0, 0});
  if (it->l == x) return it;
  int l = it->l, r = it->r, v = it->v;
  odt.erase(it);
  odt.insert(Node_t(l, x - 1, v));
  return odt.insert(Node_t(x, r, v)).first;
}
//区间修改
void assign(int l, int r, int v) {
  auto itr = split(r + 1), itl = split(l);
  odt.erase(itl, itr);
  odt.insert(Node_t(l, r, v));
}
//对区间进行某种操作
void performance(int l, int r) {
  auto itr = split(r + 1), itl = split(l);
  for (; itl != itr; ++itl) {
    // Perform Operations here
  }
}
```

6.20 divide combine tree

```
// 析合树
// 对一个 1-n 的排列，称值域连续的区间为一段．询问排列的段的个数．
// {5,3,4,1,2}->[1,1][2,2][3,3][4,4][5,5][2,3][4,5][1,3][2,5][1,5]

#include <bits/stdc++.h>
#define rg register
using namespace std;
const int N = 200010;

int n, m, a[N], st1[N], st2[N], tp1, tp2, rt;
int L[N], R[N], M[N], id[N], cnt, typ[N], bin[20], st[N], tp;
//本篇代码原题应为 CERC2017 Intrinsic Interval
// a 数组即为原题中对应的排列
// st1 和 st2 分别两个单调栈，tp1、tp2 为对应的栈顶，rt 为析合树的根
// L、R 数组表示该析合树节点的左右端点，M 数组的作用在析合树构造时有提到
// id 存储的是排列中某一位置对应的节点编号，typ 用于标记析点还是合点
// st 为存储析合树节点编号的栈，tp 为其栈顶
struct RMQ
{ // 预处理 RMQ（Max & Min）
    int lg[N], mn[N][17], mx[N][17];
    void chkmn(int &x, int y)
    {
        if (x > y)
            x = y;
    }
    void chkmx(int &x, int y)
    {
        if (x < y)
            x = y;
    }
    void build()
    {
        for (int i = bin[0] = 1; i < 20; ++i)
            bin[i] = bin[i - 1] << 1;
        for (int i = 2; i <= n; ++i)
            lg[i] = lg[i >> 1] + 1;
        for (int i = 1; i <= n; ++i)
            mn[i][0] = mx[i][0] = a[i];
        for (int i = 1; i < 17; ++i)
            for (int j = 1; j + bin[i] - 1 <= n; ++j)
                mn[j][i] = min(mn[j][i - 1], mn[j + bin[i - 1]][i - 1]),
                mx[j][i] = max(mx[j][i - 1], mx[j + bin[i - 1]][i - 1]);
    }
    int ask_mn(int l, int r)
    {
        int t = lg[r - l + 1];
        return min(mn[l][t], mn[r - bin[t] + 1][t]);
    }
    int ask_mx(int l, int r)
    {
        int t = lg[r - l + 1];
        return max(mx[l][t], mx[r - bin[t] + 1][t]);
    }
} D;
// 维护 L_i

struct SEG
{ // 线段树
#define ls (k << 1)
#define rs (k << 1 | 1)
    int mn[N << 1], ly[N << 1]; // 区间加; 区间最小值
    void pushup(int k) { mn[k] = min(mn[ls], mn[rs]); }
    void mfy(int k, int v) { mn[k] += v, ly[k] += v; }
    void pushdown(int k)
    {
        if (ly[k])
            mfy(ls, ly[k]), mfy(rs, ly[k]), ly[k] = 0;
    }
    void update(int k, int l, int r, int x, int y, int v)
    {
        if (l == x && r == y)
        {
            mfy(k, v);
            return;
        }
        pushdown(k);
        int mid = (l + r) >> 1;
        if (y <= mid)
            update(ls, l, mid, x, y, v);
        else if (x > mid)
            update(rs, mid + 1, r, x, y, v);
        else
            update(ls, l, mid, x, mid, v), update(rs, mid + 1, r, mid + 1, y, v);
        pushup(k);
    }
    int query(int k, int l, int r)
    { // 询问 0 的位置
        if (l == r)
            return l;
        pushdown(k);
        int mid = (l + r) >> 1;
        if (!mn[ls])
            return query(ls, l, mid);
        else
            return query(rs, mid + 1, r);
        // 如果不存在 0 的位置就会自动返回当前你查询的位置
    }
} T;

int o = 1, hd[N], dep[N], fa[N][18];
struct Edge
{
    int v, nt;
} E[N << 1];
void add(int u, int v)
{ // 树结构加边
    E[o] = (Edge){v, hd[u]};
    hd[u] = o++;
}
void dfs(int u)
{
    for (int i = 1; bin[i] <= dep[u]; ++i)
```

```
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int i = hd[u]; i; i = E[i].nt)
    {
        int v = E[i].v;
        dep[v] = dep[u] + 1;
        fa[v][0] = u;
        dfs(v);
    }
}
int go(int u, int d)
{
    for (int i = 0; i < 18 && d; ++i)
        if (bin[i] & d)
            d ^= bin[i], u = fa[u][i];
    return u;
}
int lca(int u, int v)
{
    if (dep[u] < dep[v])
        swap(u, v);
    u = go(u, dep[u] - dep[v]);
    if (u == v)
        return u;
    for (int i = 17; ~i; --i)
        if (fa[u][i] != fa[v][i])
            u = fa[u][i], v = fa[v][i];
    return fa[u][0];
}

// 判断当前区间是否为连续段
bool judge(int l, int r) { return D.ask_mx(l, r) - D.ask_mn(l, r) == r - l; }

// 建树
void build()
{
    for (int i = 1; i <= n; ++i)
    {
        // 单调栈
        // 在区间 [st1[tp1-1]+1,st1[tp1]] 的最小值就是 a[st1[tp1]]
        // 现在把它出栈，意味着要把多减掉的 Min 加回来。
        // 线段树的叶结点位置 j 维护的是从 j 到当前的 i 的
        // Max{j,i}-Min{j,i}-(i-j)
        // 区间加只是一个 Tag。
        // 维护单调栈的目的是辅助线段树从 i-1 更新到 i。
        // 更新到 i 后，只需要查询全局最小值即可知道是否有解

        while (tp1 && a[i] <= a[st1[tp1]]) // 单调递增的栈，维护 Min
            T.update(1, 1, n, st1[tp1 - 1] + 1, st1[tp1], a[st1[tp1]]), tp1--;
        while (tp2 && a[i] >= a[st2[tp2]])
            T.update(1, 1, n, st2[tp2 - 1] + 1, st2[tp2], -a[st2[tp2]]), tp2--;

        T.update(1, 1, n, st1[tp1] + 1, i, -a[i]);
        st1[++tp1] = i;
        T.update(1, 1, n, st2[tp2] + 1, i, a[i]);
        st2[++tp2] = i;

        id[i] = ++cnt;
        L[cnt] = R[cnt] = i; // 这里的 L,R 是指值域的上下界
        int le = T.query(1, 1, n), now = cnt;
        while (tp && L[st[tp]] >= le)
        {
            if (typ[st[tp]] && judge(M[st[tp]], i))
            {
                // 判断是否能成为儿子，如果能就做
                R[st[tp]] = i, add(st[tp], now), now = st[tp--];
            }
            else if (judge(L[st[tp]], i))
            {
                typ[++cnt] = 1; // 合点一定是被这样建出来的
                L[cnt] = L[st[tp]], R[cnt] = i, M[cnt] = L[now];
                //这里 M 数组的作用是保证合点的儿子排列是单调的
                add(cnt, st[tp--]), add(cnt, now);
                now = cnt;
            }
            else
            {
                add(++cnt, now); // 新建一个结点，把 now 添加为儿子
                // 如果从当前结点开始不能构成连续段，就合并。
                // 直到找到一个结点能构成连续段。而且我们一定能找到这样
                // 一个结点。
                do
                    add(cnt, st[tp--]);
                while (tp && !judge(L[st[tp]], i));
                L[cnt] = L[st[tp]], R[cnt] = i, add(cnt, st[tp--]);
                now = cnt;
            }
        }
        st[++tp] = now; // 增量结束，把当前点压栈

        T.update(1, 1, n, 1, i, -1); // 因为区间右端点向后移动一格，因此整体 -1
    }

    rt = st[1]; // 栈中最后剩下的点是根结点
}
void query(int l, int r)
{
    int x = id[l], y = id[r];
    int z = lca(x, y);
    if (typ[z] & 1)
        l = L[go(x, dep[x] - dep[z] - 1)], r = R[go(y, dep[y] - dep[z] - 1)];
    //合点这里特判的原因是因为这个合点不一定是最小的包含 l，r 的连续段.
    //具体可以在上面的例图上试一下查询 7,10
    else
        l = L[z], r = R[z];
    printf("%d %d\n", l, r);
} // 分 lca 为析或和，这里把叶子看成析的

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);
    D.build();
    build();
    dfs(rt);
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i)
    {
        int x, y;
        scanf("%d%d", &x, &y);
        query(x, y);
    }
    return 0;
}
// 20190612
// 析合树
```

6.21 AhoCorasickAutomation

```
//AC 自动机
//用于进行多模式匹配
// #include <cstring>
// #include <queue>
// using namespace std;
struct AhoCorasickAutomaton {
    struct Node {
        Node *son[26],*fail,*last;
        int cnt;

        Node() {
            memset(son,0,sizeof(son));
            fail=last=0;
            cnt=0;
        }

    }*root;

    void Insert(char *s) {
        Node *pos=root;
        for(;*s;++s) {
            int c=*s-'a';
            if(!pos->son[c])
                pos->son[c]=new Node;
            pos=pos->son[c];
        }
        pos->cnt++;
    }

    void Build() {
        root->fail=root->last=root;
        std::queue<Node*> Q;
        for(int c=0;c<26;++c)
            if(root->son[c]) {
                root->son[c]->fail=root->son[c]->last=root;
                Q.push(root->son[c]);
            } else root->son[c]=root;
        while(!Q.empty()) {
            Node *cur=Q.front();Q.pop();
            for(int c=0;c<26;++c)
                if(cur->son[c]) {
                    Node *u=cur->son[c];
                    u->fail=cur->fail->son[c];
                    u->last=u->fail->cnt?u->fail:u->fail->last;
                    Q.push(u);
                } else cur->son[c]=cur->fail->son[c];
        }
    }

    int Calc(Node *pos) {
        int res=0;
        while(pos->cnt) {
            res+=pos->cnt;
            pos->cnt=0;
            pos=pos->last;
        }
        return res;
    }

    int Match(char *s) {
        Node *pos=root;
        int res=0;
        for(;*s;++s) {
            int c=*s-'a';
            pos=pos->son[c];
            if(pos->cnt)
                res+=Calc(pos);
        }
        return res;
    }
};
```

6.22 simpleBitset

```
struct BitSet {
    unsigned int64 s[(XV>>6)+1];
    int maxI;

    BitSet(int v):maxI(v>>6) {
        memset(s,0,sizeof(s));
    }

    void Set(unsigned int pos,bool val) {
        val==0?(s[pos>>6]&=~(1ull<<(pos&63))):(s[pos>>6]|=1ull<<(pos&63));
    }

    bool Test(unsigned int pos) const {
        return s[pos>>6]>>(pos&63)&1;
    }
};
```

6.23 blockDividedTree

```
/* 需要链式图存储
const int MAXV=10,MAXE=10;
struct Edge{
    int v,n;
}edges[MAXE];
int head[MAXV];

void adde(int u,int v){
    edges[++idx].v=v;
    edges[idx].n=head[u];
    head[u]=idx;
}
*/
namespace BlockDividedTree{
int cnt=0,B,anc[MAXV],stk[MAXV],top=0,id[MAXV];
void dfs(int u,int fa){
    for(int i=head[u],v,re=top;i;i=edges[i].n)
        if((v=edges[i].v)!=fa){
            dfs(v,u);
            if(top-re>=B){
                for(++cnt;top!=re;--top)id[stk[top]]=cnt;
                anc[cnt]=u;
            }
        }
    stk[++top]=u;
}
void divide(){
    dfs(1,0);
    //cnt 块的数量
    //anc 每块的根
    //id 每个点属于哪个块
    //B  B<= 每个块的的大小 <=3B
    if(top){
        if(!cnt)anc[++cnt]=1;
        for(;top;--top)id[stk[top]]=cnt;
    }
}
}
```

6.24 dynamicChainBasedDC

```
namespace DynamicChainBasedDC {
    struct Part {
        int top;
        Part(int top):top(top) {}
    }*cn[XN];

    int dfs[XN],dc,lbd[XN],rbd[XN],dep[XN],sz[XN],prefer[XN],fa[XN];
    void DFS(int pos) {
        int mxs=0;
        sz[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            fa[u]=pos;dep[u]=dep[pos]+1;
            DFS(u);
            sz[pos]+=sz[u];
            if(Enlarge(mxs,sz[u]))
                prefer[pos]=u;
        }
    }

    void Assign(int pos,int rt) {
        dfs[++dc]=pos;
        lbd[pos]=dc;
        cn[pos]=cn[rt]?cn[rt]:new Part(pos);
        if(prefer[pos]) {
            Assign(prefer[pos],rt);
            for(Edge *e=G[pos];e;e=e->pre)
                if(e->to!=prefer[pos])
                    Assign(e->to,e->to);
        }
        rbd[pos]=dc;
    }

    void Divide() {
        DFS(1);
        cn[1]=new Part(1);
        Assign(1,1);
    }

    void Path(int p1,int p2) {
        while(cn[p1]!=cn[p2]) {
            if(dep[cn[p1]->top]<dep[cn[p2]->top])
                std::swap(p1,p2);
            //p1~cht[p1]
            p1=fa[cn[p1]->top];
        }
        if(lbd[p1]>lbd[p2])
            std::swap(p1,p2);
        //p1~p2
    }
}
```

6.25 FHQTreap2

```
//无旋 Treap 版本 2
struct Treap {
    struct Node {
        Node *son[2];
        int v,add,size;
        long long sum;

        Node(int v):v(v),add(0),size(1),sum(v) {
            son[0]=son[1]=null;
        }

        Node(void*):v(0),add(0),size(0),sum(0) {
            son[0]=son[1]=this;
        }

        void Add(int d) {
            add+=d;
            sum+=(long long)d*size;
            v+=d;
        }

        void Up() {
            sum=son[0]->sum+v+son[1]->sum;
            size=son[0]->size+1+son[1]->size;
        }

        void Down() {
            if(add) {
                if(son[0]!=null)
                    (son[0]=new Node(*son[0]))->Add(add);
                if(son[1]!=null)
                    (son[1]=new Node(*son[1]))->Add(add);
                add=0;
            }
        }
    }*root;

    static Node *null;

    Treap(int a[],int n):root(Build(a,1,n)) {}

    static Node *Build(int a[],int L,int R) {
        if(L>R)
            return null;
        else {
            int M=(L+R)/2;
            Node *pos=new Node(a[M]);
            pos->son[0]=Build(a,L,M-1);
            pos->son[1]=Build(a,M+1,R);
            pos->Up();
            return pos;
        }
    }

    static std::pair<Node*,Node*> Split(Node *pos,int k) {
        if(k==0)
            return std::pair<Node*,Node*>(null,pos);
        else if(k==pos->size)
            return std::pair<Node*,Node*>(pos,null);
        else {
            (pos=new Node(*pos))->Down();
            std::pair<Node*,Node*> res;
            if(k<=pos->son[0]->size) {
                res=Split(pos->son[0],k);
                pos->son[0]=res.second;
                pos->Up();
                res.second=pos;
            } else {
                res=Split(pos->son[1],k-pos->son[0]->size-1);
                pos->son[1]=res.first;
                pos->Up();
                res.first=pos;
            }
            return res;
        }
    }

    static Node *Merge(Node *p1,Node *p2) {
        if(p1==null || p2==null)
            return p1==null?p2:p1;
        else {
            Node *pos;
            if(rand()%(p1->size+p2->size)+1<=p1->size) {
                (pos=new Node(*p1))->Down();
                pos->son[1]=Merge(pos->son[1],p2);
                pos->Up();
            } else {
                (pos=new Node(*p2))->Down();
                pos->son[0]=Merge(p1,pos->son[0]);
                pos->Up();
            }
            return pos;
        }
    }

    struct Triple {
        Node *L,*M,*R;
    };

    static Triple Split(Node *root,int l,int r) {
        std::pair<Node*,Node*> x=Split(root,r),y=Split(x.first,l-1);
        return {y.first,y.second,x.second};
    }

    static Node *Merge(Triple t) {
        return Merge(t.L,Merge(t.M,t.R));
    }
};
Treap::Node *Treap::null=new Treap::Node((void*)0);
```

6.26 FHQTreap1

```
//非旋 Treap
/*****
 * 0. 插入，删除
 * 1. 切分
 * 2. 合并
 * 3. 查询与反查排名
/*
#include <vector>
using namespace std;
const int MAXN=1000;
*/

namespace FHQTree{
```

```
struct Node{
    Node *ch[2];
    int key,val,sz;
    Node(int v){
        sz=1;
        val=v;
        key=rand();
        ch[0]=ch[1]=NULL;
    }
    static int size(Node *node){
        return node?node->sz:0;
    }
    inline void tain(){
        sz=1+(ch[0]?ch[0]->sz:0)+(ch[1]?ch[1]->sz:0);
    }

};
Node *root;
typedef pair<Node*,Node*> D;
//merge b into a
Node* merge(Node *a,Node *b){
    if(!a)return b;
    if(!b)return a;
    if(a->key<b->key){
        a->ch[1]=merge(a->ch[1],b);
        a->tain();
        return a;
    }else{
        b->ch[0]=merge(a,b->ch[0]);
        b->tain();
        return b;
    }
}
//split front k.
//D(splited one,rest one)
D split(Node *node,int k){
    if(!node)return D(NULL,NULL);
    D res;
    if(Node::size(node->ch[0])>=k){
        res=split(node->ch[0],k);
        node->ch[0]=res.second;
        node->tain();
        res.second=node;
    }else{
        res=split(node->ch[1],k-Node::size(node->ch[0])-1);
        node->ch[1]=res.first;
        node->tain();
        res.first=node;
    }
    return res;
}
int getrank(Node *node,int v){
    if(node==NULL)return 0;
    return
     ↪  (node->val>=v)?getrank(node->ch[0],v):Node::size(node->ch[0])+1+getrank(node->ch[
}
inline int getbyrank(int k){
    D x=split(root,k-1);
    D y=split(x.second,1);
    Node *ans=y.first;
    root=merge(merge(x.first,ans),y.second);
    return ans?ans->val:0;
}
inline void insert(int v){
    int k=getrank(root,v);
    D x=split(root,k);
    Node *o=new Node(v);
    root=merge(merge(x.first,o),x.second);
}
void remove(int v){
    int k=getrank(root,v);
    D x=split(root,k);
    D y=split(x.second,1);
    root=merge(x.first,y.second);
}
}

/*
using namespace FHQTree;
int main(){
    int op,x;
    while(cin>>op>>x){
        switch(op){
            case 1:insert(x);break;
            case 2:remove(x);break;
            case 3:cout<<getrank(root,x)+1<<endl;break;
            case 4:cout<<getbyrank(x)<<endl;break;
            case 5:cout<<getbyrank(getrank(root,x))<<endl;break;
            case 6:cout<<getbyrank(getrank(root,x+1)+1)<<endl;break;
        }
    }
    return 0;
}
*/
```

6.27 KDTree

```
//k-d tree 算法 k-d 树（k-dimensional 树的简称）.
//是一种分割 k 维数据空间的数据结构。主要应用于多维空间关键数据的搜索（如：范围搜索和最近邻搜索）
/*****
 * 索引结构中相似性查询有两种基本的方式：一种是范围查询（range searches)，另一种是 K 近邻查询（K-neighbor searches)。范围查询就是给定查询点和查询距离的阈值，从数据集中找出所有与查询点距离小于阈值的数据；K 近邻查询是给定查询点及正整数 K，从数据集中找到距离查询点最近的 K 个数据，当 K=1 时，就是最近邻查询（nearest neighbor searches)。
 *
 * 需要定义两点间的距离
 */
/*
#include <algorithm>
using namespace std;
*/
struct Point {
    int d[XD];
};

long long Dist(Point const &a,Point const &b) {

}
```

```
struct KDTree {
    struct Node {
        Node *son[2];
        Point p,min,max;

        Node(Point p):p(p),min(p),max(p) {
            son[0]=son[1]=null;
        }

        Node(void*):min(),max() {
            son[0]=son[1]=0;
        }

        void Up() {
            for(int i=0;i<k;++i) {
                min.d[i]=std::min(p.d[i],std::min(son[0]->min.d[i],son[1]->min.d[i]));
                max.d[i]=std::max(p.d[i],std::max(son[0]->max.d[i],son[1]->max.d[i]));
            }
        }

        long long Dist(Point const &q) {

        }
    }*root;

    static Node *null;

    KDTree(Point p[],int n):root(Build(p,1,n,0)) {}

    Node *Build(Point p[],int L,int R,int d) {
        if(L>R)
            return null;
        else {
            struct Compare {
                int d;
                Compare(int d):d(d) {}

                bool operator ()(Point const &a,Point const &b) {
                    return a.d[d]<b.d[d];
                }
            };
            int M=(L+R)/2;
            std::nth_element(p+L,p+M,p+R+1,Compare(d));
            Node *pos=new Node(p[M]);
            pos->son[0]=Build(p,L,M-1,(d+1)%k);
            pos->son[1]=Build(p,M+1,R,(d+1)%k);
            pos->Up();
            return pos;
        }
    }

    long long Query(Point p) {
        long long res;
        Query(root,p,res);
        return res;
    }

    void Query(Node *pos,Point p,long long &res) {
        if(pos==null)
            return;
        else {
            Reduce(res,Dist(pos->p,p));
            if(pos->son[0]->Dist(p)<res)
                Query(pos->son[0],p,res);
            if(pos->son[1]->Dist(p)<res)
                Query(pos->son[1],p,res);
        }
    }
};
KDTree::Node *KDTree::null=new KDTree::Node((void*)0);
```

6.28 leftist

```
//左偏树

struct Leftist {
    struct Node {
        Node *son[2];
        int v;
        int dist;

        Node(int const &v):v(v),dist(1) {
            son[0]=son[1]=null;
        }

        Node(void*):v(INF),dist(0) {
            son[0]=son[1]=0;
        }

        void Maintain() {
            if(son[0]->dist<son[1]->dist)
                std::swap(son[0],son[1]);
            dist=son[1]->dist+1;
        }
    }*root;

    static Node *null;

    Leftist():root(null) {}

    static Node *Merge(Node *p1,Node *p2) {
        if(p1==null || p2==null)
            return p1==null?p2:p1;
        else {
            if(p1->v>p2->v)
                std::swap(p1,p2);
            p1->son[1]=Merge(p1->son[1],p2);
            p1->Maintain();
            return p1;
        }
    }

    void Swallow(Leftist &other) {
        root=Merge(root,other.root);
        other.root=null;
    }

    void Push(int v) {
        root=Merge(root,new Node(v));
    }

    int Pop() {
        int res=root->v;
        root=Merge(root->son[0],root->son[1]);
        return res;
    }
};
Leftist::Node *Leftist::null=new Leftist::Node((void*)0);
```

6.29 linkCutTree

```
class LinkCutTrees {
public:
    LinkCutTrees() {}

    void Link(int id1,int id2) {
        Link(node[id1],node[id2]);
    }

    void Cut(int id1,int id2) {
        Cut(node[id1],node[id2]);
    }
private:
    struct Node {

        Node *son[2],*fa;
        bool rev;

        Node(void*):rev(0) {
            son[0]=son[1]=fa=0;
        }

        Node():rev(0) {
            son[0]=son[1]=fa=null;
        }

        int Type() {
            return fa->son[1]==this;
        }
        bool isRoot() {
            return fa->son[0]!=this && fa->son[1]!=this;
        }

        void Adopt(Node *s,int d) {
            son[d]=s;
            if(s!=null)
                s->fa=this;
        }

        void Reverse() {
            rev^=1;
            std::swap(son[0],son[1]);
        }

        void Up() {

        }

        void Down() {
            if(rev) {
                if(son[0]!=null)
                    son[0]->Reverse();
                if(son[1]!=null)
                    son[1]->Reverse();
                rev=0;
            }
        }

    }node*[];

    static Node *null;

    static void Trans(Node *pos) {
        Node *f=pos->fa,*g=f->fa;
        f->Down();pos->Down();
        int d=pos->Type();
        if(!f->isRoot())
            g->son[f->Type()]=pos;
        pos->fa=g;
        f->Adopt(pos->son[!d],d);f->Up();
        pos->Adopt(f,!d);
    }

    static void Splay(Node *pos) {
        pos->Down();
        for(Node *fa;!pos->isRoot();Trans(pos))
            if(!(fa=pos->fa)->isRoot())
                Trans(pos->Type()==fa->Type()?fa:pos);
        pos->Up();
    }

    static void Access(Node *pos) {
        for(Node *pred=null;pos!=null;pred=pos,pos=pos->fa) {
            Splay(pos);
            pos->son[1]=pred;
            pos->Up();
        }
    }

    static void Expose(Node *pos) {
        Access(pos);
        Splay(pos);
    }

    static Node *FindRoot(Node *pos) {
        Expose(pos);
        while(pos->son[0]!=null) {
            pos->Down();
            pos=pos->son[0];
        }
        return pos;
    }

    static void MakeRoot(Node *pos) {
        Expose(pos);
        pos->Reverse();
    }
    static void Cut(Node *p1,Node *p2) {
        MakeRoot(p1);
        Expose(p2);
        p2->son[0]=p1->fa=null;
        p2->Up();
    }

    static void Link(Node *p1,Node *p2) {
        MakeRoot(p1);
        p1->fa=p2;
    }
};
LinkCutTrees::Node *LinkCutTrees::null=new LinkCutTrees::Node((void*)0);
```

6.30 scapeGoat

```
struct Scapegoat {
    static const double alpha=0.8;

    struct Node {
        Node *son[2];
        int v,cnt,size,ndct;

        Node(int v):v(v),cnt(1),size(1),ndct(1) {
            son[0]=son[1]=null;
        }

        Node(void*):size(0),ndct(0) {
            son[0]=son[1]=this;
        }

        void Up() {
            size=son[0]->size+cnt+son[1]->size;
            ndct=son[0]->ndct+1+son[1]->ndct;
        }

        bool Unbalanced() {
            return ndct*alpha<std::max(son[0]->ndct,son[1]->ndct);
        }
    }*root;
    static Node *null;

    Scapegoat():root(null) {}

    static Node *&Insert(Node *&pos,int v) {
        if(pos==null) {
            pos=new Node(v);
            return null;
        } else if(pos->v==v) {
            pos->cnt++;
            pos->Up();
            return null;
        } else {
            Node *&goat=Insert(pos->son[pos->v<v],v);
            pos->Up();
            return pos->Unbalanced()?pos:goat;
        }
    }

    static void Delete(Node *pos,int v) {
        if(pos==null)
            return;
        else if(pos->v==v) {
            pos->cnt--;
            pos->Up();
        } else {
            Delete(pos->son[pos->v<v],v);
            pos->Up();
        }
    }

    static Node *Flatten(Node *pos,Node *app) {
        if(pos==null)
            return app;
        else {
            pos->son[1]=Flatten(pos->son[1],app);
            return Flatten(pos->son[0],pos);
        }
    }
    static std::pair<Node*,Node*> Rebuild(Node *begin,int n) {
        if(n==0) {
            return std::pair<Node*,Node*>(null,begin);
        } else {
            int mid=(1+n)/2;
            std::pair<Node*,Node*> left=Rebuild(begin,mid-1);
            Node *pos=left.second;
            std::pair<Node*,Node*> right=Rebuild(pos->son[1],n-mid);
            pos->son[0]=left.first;
            pos->son[1]=right.first;
            pos->Up();
            return std::pair<Node*,Node*>(pos,right.second);
        }
    }

    static void Rebuild(Node *&root) {
        Node *begin=Flatten(root,null);
        root=Rebuild(begin,root->ndct).first;
    }

    void Insert(int v) {
        Node *&goat=Insert(root,v);
        if(goat!=null)
            Rebuild(goat);
    }

    void Delete(int v) {
        Delete(root,v);
    }

    int Rank(int v) {
        int res=0;
        for(Node *pos=root;pos!=null;) {
            if(pos->v<v) {
                res+=pos->son[0]->size+pos->cnt;
                pos=pos->son[1];
            } else
                pos=pos->son[0];
        }
        return ++res;
    }

    int Kth(int k) {
        for(Node *pos=root;;) {
            int le=pos->son[0]->size+pos->cnt;
            if(k<=le) {
                if(pos->son[0]->size+1<=k && k<=le && pos->cnt)
                    return pos->v;
                else
                    pos=pos->son[0];
            } else {
                k-=le;
                pos=pos->son[1];
            }
        }
        throw;
    }

    int Pred(int v) {
        return Kth(Rank(v)-1);
    }

    int Succ(int v) {
        return Kth(Rank(v+1));
    }
};
const double Scapegoat::alpha;
Scapegoat::Node *Scapegoat::null=new Scapegoat::Node((void*)0);
```

6.31 staticEdgeBasedDC

```
namespace StaticEdgeBasedDC {
    int vtc,n;

    struct Edge {
        int to,v;
        Edge *pre,*rev;
        bool ban;
        Edge(int to,int v,Edge *pre):to(to),v(v),pre(pre),ban(0) {}

    }*G[XN],*oG[XN];

    void AddEdge(Edge *G[],int x,int y,int v) {
        G[x]=new Edge(y,v,G[x]);
        G[y]=new Edge(x,v,G[y]);
        G[x]->rev=G[y];
        G[y]->rev=G[x];
    }

    void Rebuild(int pos,int fa) {
        int cur=pos,cnt=0;
        for(Edge *e=oG[pos];e;e=e->pre)
            if(e->to!=fa) {
                int u=e->to;
                if(++cnt==2) {
                    cnt=0;
                    AddEdge(G,cur,vtc,0);
                    cur=vtc;
                }
                AddEdge(G,cur,u,1);
                Rebuild(u,pos);
            }
    }

    int size[XN];

    int GetSize(int pos,int fa) {
        size[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre)
            if(!e->ban && e->to!=fa) {
                int u=e->to;
                size[pos]+=GetSize(u,pos);
            }
        return size[pos];
    }

    std::pair<int,Edge*> Bridge(int pos,int fa,int tol) {
        std::pair<int,Edge*> res=std::pair<int,Edge*>(INF,0);
```

```
        for(Edge *e=G[pos];e;e=e->pre)
            if(!e->ban && e->to!=fa) {
                int u=e->to;
                Reduce(res,std::min(Bridge(u,pos,tol),
                            std::pair<int,Edge*>(std::max(size[u],tol-size[u]),e)));
            }
        return res;
    }

    long long DC(Edge *brg) {
        if(!brg)
            return 0;
        else {
            brg->ban=brg->rev->ban=1;
            int x=brg->to,y=brg->rev->to;
            long long res=Calc();
            Enlarge(res,std::max(DC(Bridge(x,0,GetSize(x,0)).second),
                        DC(Bridge(y,0,GetSize(y,0)).second)));
            return res;
        }
    }

    long long Run() {
        Rebuild(1,0);
        return DC(Bridge(1,0,GetSize(1,0)).second);
    }
}
```

6.32 staticVertexBasedDC

```
namespace StaticVertexBasedDC {
    bool ud[XN];
    int size[XN];

    int GetSize(int pos,int fa) {
        size[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u] && u!=fa)
                size[pos]+=GetSize(u,pos);
        }
        return size[pos];
    }

    int Centre(int pos,int fa,int const &tol) {
        static int f[XN]={INF};
        int res=0,mxs=0;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u] && u!=fa) {
                int t=Centre(u,pos,tol);
                if(f[t]<f[res])
                    res=t;
                Enlarge(mxs,size[u]);
            }
        }
        f[pos]=std::max(mxs,tol-size[pos]);
        return f[pos]<f[res]?pos:res;
    }


    void DC(int pos) {
        ud[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u]) {

            }
        }

        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u])
                DC(Centre(u,0,GetSize(u,0)));
        }

    }
}
```

6.33 VirtualTree

```
namespace VirtualTree {
    struct Graph {
        struct Edge {
            int to,v;
            Edge *pre;

            Edge(int to,int v,Edge *pre):to(to),v(v),pre(pre) {}

        }*G[XN],*pool,*mem;

        int us[XN],T;

        Graph():pool((Edge*)malloc(XN*2*sizeof(Edge))),mem(pool) {}

        void Check(int x) {
            if(us[x]!=T) {
                us[x]=T;
                G[x]=0;
            }
        }

        Edge *&operator [](int x) {
            Check(x);
            return G[x];
        }

        void operator ()(int x,int y,int c=1) {
            Check(x);
            G[x]=new(mem++) Edge(y,c,G[x]);
        }

        void Reset() {
            mem=pool;
            ++T;
        }
    }R;

    void Build(int h[],int hc) {
        std::sort(h+1,h+1+hc,[](int a,int b)->bool { return dfn[a]<dfn[b]; });
        static int stack[XN],top;
        stack[top=0]=0;R.Reset();
        for(int i=1;i<=hc;++i) {
            for(int lca=LCA(stack[top],h[i]);stack[top]!=lca;) {
                if(dep[lca]>dep[stack[top]])
                    stack[++top]=lca;
                else {
                    R(dep[stack[top-1]]>dep[lca]?stack[top-1]:lca,stack[top]);
                    top--;
                }
            }
            stack[++top]=h[i];
        }
        for(;top;top--)
            R(stack[top-1],stack[top]);
    }
}
```

6.34 liChaoSegTree

```
//李超线段树
namespace LiChaoSegTree{
struct Line {
    int k,b;
    int operator() (int x) { return k*x+b; }
    friend double Cross(Line const &lhs,Line const &rhs) { return ((double)lhs.b-rhs.b)/(rhs.k-lhs.k); }
};

struct SegTree {
    struct Node {
        Node *son[2];
        Line l;
        Node(Line const &l):l(l) { son[0]=son[1]=0; }
    }*root;int L,R;
    SegTree(int L,int R):root(0),L(L),R(R) {}
    void Insert(Node *&pos,int L,int R,Line l) {
        if(!pos) pos=new Node(l);
        else {
            if(L==R) {
                if(pos->l(L)>l(L)) pos->l=l;
            } else {
                Line r=pos->l;if(l(L)>r(L)) std::swap(l,r);
                int M=L+R>=0?(L+R)/2:-((-L-R)/2);
                double x=Cross(l,r);
                if(l.k==r.k || x<L || R<x)//R>x
                    pos->l=l;
                else if(x<=M) {
                    pos->l=r;
                    Insert(pos->son[0],L,M,l);
                } else {
                    pos->l=l;
                    Insert(pos->son[1],M+1,R,r);
                }
            }
        }
    }
    int Query(Node *pos,int L,int R,int x) {
        if(!pos) return INF;
        else {
            int M=L+R>=0?(L+R)/2:-((-L-R)/2);
            return std::min(pos->l(x),x<=M?Query(pos->son[0],L,M,x)
                                          :Query(pos->son[1],M+1,R,x));
        }
    }
    void Insert(Line const &l) { Insert(root,L,R,l); }
    int Query(int x) { return Query(root,L,R,x); }
};

}
```

7 environment

7.1 bashrc

7.2 vimrc