Index

1 Enviroment Settings

1.1 .vimrc

```
syntax on
set enc=utf-8 fencs=utf-8,big5
set bs=2
set smd nu bg=dark hls ls=2 wmnu so=5 ru cul
set ts=4 sw=4 ai sta si
set list lcs=tab:>\  "# a space after '\'
map<F9> :!g++ "%" -o "%:r.out" -Wall -Wshadow -O2
     -Im -std=c++11 && echo "===== done =====" &&
     "./%:r.out"
```

2 Computational Geometry

2.1 Geometry on Plane

```
struct node {
  double x,y;
  node(double _x=0, double _y=0) : x(_x),y(_y) {}
  node operator+(const node& rhs) const
    { return node(x+rhs.x, y+rhs.y); }
  node operator-(const node& rhs) const
    { return node(x-rhs.x, y-rhs.y); }
  node operator*(const double& rhs) const
    { return node(x*rhs, y*rhs); }
  node operator/(const double& rhs) const
    { return node(x/rhs, y/rhs); }
  double operator*(const node& rhs) const
    { return x*rhs.x+y*rhs.y; }
  double operator^(const node& rhs) const
    { return x*rhs.y-y*rhs.x; }
  double len2() const { return x*x+y*y; }
  double len() const { return sqrt(x*x+y*y); }
  node unit() const { return *this/len(); }
  node T() const { return node(-y,x); }
  node rot(double rad) const { // rotate counter-
      clockwise in rad
    return node(cos(rad)*x-sin(rad)*y, sin(rad)*x
        +cos(rad)*y);
  }
};
node __mirror(node normal, double constant, node
    point){ //2D3D
  double scale=(normal*point+constant)/(normal*
      normal);
  return point-normal*(2*scale);
}
node mirror(node p1, node p2, node p3){ // 2D3D
  return __mirror((p2-p1).T(), (p2-p1).T()*p1
      *(-1), p3);
}
double ori(const node& p1, const node& p2, const
    node& p3){ //平行四邊形面積(帶正負)
  return (p2-p1)^(p3-p1);
}
bool intersect(const node& p1, const node& p2,
    const node& p3, const node& p4){
  return (ori(p1,p2,p3)*ori(p1,p2,p4)<0 && ori(p3
      ,p4,p1)*ori(p3,p4,p2)<0);
}
pair<node,node> two_circle_intersect(node p1,
    double r1, node p2, double r2){
  double degree=acos(((p2-p1).len2()+r1*r1-r2*r2)
      /(2*r1*(p2-p1).len()));
  return make_pair(p1+(p2-p1).unit().rot(degree)*
      r1, p1+(p2-p1).unit().rot(-degree)*r1);
}
node intersectionPoint(node p1, node p2, node p3,
     node p4){
  double a123 = (p2-p1)^(p3-p1);
  double a124 = (p2-p1)^(p4-p1);
  return (p4*a123-p3*a124)/(a123-a124);
}
```

2.2 Minimum Covering Circle

```
node center(node p0, node p1, node p2) {
  node a = p1-p0;
  node b = p2-p0;
  double c1 = a.len2()/2;
  double c2 = b.len2()/2;
  double d = a^b;
  double x = p0.x + (c1 * b.y - c2 * a.y) / d;
  double y = p0.y + (a.x * c2 - b.x * c1) / d;
  return node(x, y);
}

pair<node,double> mcc(node p[], int n) {
  random_shuffle(p, p+n);
  node oo;
  double r2 = 0;
  for(int i=0; i<n; i++) {
    if ((oo-p[i]).len2() <= r2) continue;
    oo = p[i];
    r2 = 0;
    for(int j=0; j<i; j++) {
      if ((oo-p[j]).len2() <= r2) continue;
      oo = (p[i]+p[j]) / 2;
      r2 = (oo-p[j]).len2();
      for(int k=0; k<j; k++) {
        if ((oo-p[k]).len2() <= r2) continue;
        oo = center(p[i], p[j], p[k]);
        r2 = (oo-p[k]).len2();
      }
    }
  }
  return make_pair(oo, r2);
}
```

2.3 KDTree

```
struct NODE{
  int x , y;
  int x1 , x2 , y1 , y2;
  NODE *L , *R;
};
bool cmpx( const NODE& a , const NODE& b ){
  return a.x < b.x;
}
bool cmpy( const NODE& a , const NODE& b ){
  return a.y < b.y;
}
NODE* KDTree( int L , int R , int depth ){
  if ( L > R ) return 0;
  int M = ( L + R ) >> 1;
  node[M].f = depth % 2;
  nth_element( node+L , node+M , node+R+1 , node[
      M].f ? cmpy : cmpx );
  node[M].L = KDTree( L , M-1 , depth+1 );
  node[M].R = KDTree( M+1 , R , depth+1 );
  node[M].x1 = node[M].x2 = node[M].x;
  node[M].y1 = node[M].y2 = node[M].y;
  if ( node[M].L ){
    node[M].x1 = min( node[M].x1 , node[M].L->x1
        );
    node[M].y1 = min( node[M].y1 , node[M].L->y1
        );
  }
  if ( node[M].R ){
    node[M].x2 = max( node[M].x2 , node[M].L->x2
        );
    node[M].y2 = max( node[M].y2 , node[M].L->y2
        );
  }
  return node+M;
}
inline int mayTouchRectangle( NODE* r , int x ,
    int y , long long d2 ){
  long long d = ( long long )( sqrt(d2) + 1 );
  return x >= r->x1 - d && x <= r->x2 +d && y >=
      r->y1 - d && y <= r->y2 + d;
}
// find the nearest point near p
// r is tree node
```

```
void nearest( NODE* r , NODE* p , long long &dmin
    ){
  if ( !r || !mayTouchRectangle( r , p->x , p->y
      , dmin ) ) return;
  if ( p->i != r->i ) dmin = min( dmin , dis( *r
      , *p ) );  // dis returns the dis^2
  int whichFirst = r->f ? p->y < r->y: p->x < r->
      x;
  if ( whichFirst ){
    nearest( r->L , p , dmin );
    nearest( r->R , p , dmin );
  }
  else{
    nearest( r->R , p , dmin );
    nearest( r->L , p , dmin );
  }
}
```

3 Data Structure

3.1 PB DS

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

#include <ext/pb_ds/priority_queue.hpp>
typedef __gnu_pbds::priority_queue<T, greater<T>,
    pairing_heap_tag> Heap;
/*
 * method: push, pop, modify(iter, val), erase,
    join
 * point_iterator push(const_reference r_val)
 * pop
 * void modify(point_iterator it,const_reference
    r_new_val)
 * size_type erase_if(Pred prd) - return earsed
    number
 * void join(priority_queue &other)
 * void split(Pred prd, priority_queue &other) -
    move v into other if prd(v)
 *
 * tags: pairing_heap_tag, binary_heap_tag,
    binomial_heap_tag, rc_binomial_heap_tag,
    thin_heap_tag
 */

#include <ext/pb_ds/tree_policy.hpp>
typedef tree<int, null_type, less<int>,
    rb_tree_tag,
    tree_order_statistics_node_update> RBTree;
typedef tree<int, null_type, less<int>,
    splay_tree_tag,
    tree_order_statistics_node_update> Splay;
/*
 * point_iterator find_by_order(size_type order)
    [0, size)
 * size_type order_of_key(const_key_reference
    r_key) - number of elements < r_key
 * void split(const_key_reference r_key, tree &
    other) - move elements > r_key
 */

#include<ext/pb_ds/trie_policy.hpp>
#include<ext/pb_ds/tag_and_trait.hpp>
typedef trie<string, null_type,
    trie_string_access_traits<>, pat_trie_tag,
    trie_prefix_search_node_update> Trie;
typedef trie<string, null_type,
    string_trie_e_access_traits, pat_trie_tag,
    trie_prefix_search_node_update> Trie;
/*
 * pair<Trie::iterator, bool> insert(string s) -
    iterator and is new string
 * pair<Trie::iterator, Trie::iterator>
    prefix_range(string pre)
 */

#include <ext/pb_ds/hash_policy.hpp>
typedef cc_hash_table<string, int> Hash;
typedef gp_hash_table<string, int> Hash;
```

3.2 BigInteger

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <iomanip>
using namespace std;
```

```
template<class T>
T abs(const T& n) {return n>=T(0)?n:-n;}
class BigInteger {
public:
  BigInteger(const int& num=0) : len(0), sign(1)
      {
    int num2=num;
    memset(arr, 0, sizeof(arr));
    if( num2<0 ) sign=-1, num2*=-1;
    while( num2 ) arr[len++]=num2%step, num2/=
        step;
  }
  BigInteger(const char* num0) : len(0), sign(1)
      {
    *this = num0;
  }
  BigInteger(const BigInteger& b) : len(b.len),
      sign(b.sign) {
    memset(arr, 0, sizeof(arr));
    for(int i=0; i<len; ++i) arr[i]=b.arr[i];
  }
  ~BigInteger() {}
  BigInteger & operator = (const BigInteger& b) {
    len=b.len;
    sign=b.sign;
    memset(arr, 0, sizeof(arr));
    for(int i=0; i<len; ++i) arr[i]=b.arr[i];
    return *this;
  }
  BigInteger & operator = (const int& num) {
    int num2=num;
    memset(arr, 0, sizeof(arr));
    len=0, sign=1;
    if( num2<0 ) sign=-1, num2*=-1;
    while( num2 ) arr[len++]=num2%step, num2/=
        step;
    return *this;
  }
  BigInteger & operator = (const char* num0) {
    char num[strlen(num0)];
    int offset = 0;
    len = 0;
    sign = 1;
    if( num0[0] == '-' ) sign = -1, ++offset;
    else if( num0[0] == '+' ) ++offset;
    while( num0[offset]=='0' ) ++offset;
    strcpy(num, num0+offset);
    int tmp = strlen(num);
    for(int i=tmp-digit; i>=0; i-=digit) {
      arr[len] = 0;
      for(int j=0; j<digit; ++j) arr[len] = arr[
          len]*10 + num[i+j]-'0';
      ++len;
    }
    arr[len] = 0;
    for(int j=0; j<tmp%digit; ++j) arr[len] = arr
        [len]*10 + num[j]-'0';
    if( tmp%digit ) ++len;
    return *this;
  }
  BigInteger operator + (const BigInteger& b)
      const {
    if( *this>0 && b<0 ) return *this-(-b);
    if( *this<0 && b>0 ) return -(-*this-b);
    BigInteger res=*this;
    int len2=max(res.len, b.len);
    for(int i=0; i<len2; ++i) {
      res.arr[i]+=b.arr[i];
      if( res.arr[i]>=step ) res.arr[i]-=step,
          res.arr[i+1]++;
    }
    res.len=len2;
    if(res.arr[len2]) ++res.len;
    return res;
  }
  BigInteger operator - (const BigInteger& b)
      const {
    if( *this<b ) return -(b-*this);
    if( *this<0 && b<0 ) return -(-*this+b);
    if( *this>0 && b<0 ) return *this+(-b);
    BigInteger res=*this;
    int len2=max(res.len, b.len);
    for(int i=0; i<len2; ++i) {
      res.arr[i]-=b.arr[i];
      if( res.arr[i]<0 ) res.arr[i]+=step, res.
          arr[i+1]--;
    }
    while( len2>0 && res.arr[len2-1]==0 ) --len2;
    res.len=len2;
    return res;
  }
  BigInteger operator * (const BigInteger& b)
      const {
    if( *this==0 || b==0 ) return BigInteger(0);
    BigInteger res;
    for(int i=0; i<len; ++i) {
      for(int j=0; j<b.len; ++j) {
        res.arr[i+j]+=arr[i]*b.arr[j];
        res.arr[i+j+1]+=res.arr[i+j]/step;
        res.arr[i+j]%=step;
      }
    }
    res.len=len+b.len-1;
    while( res.arr[res.len] ) ++res.len;
    res.sign=sign*b.sign;
    return res;
  }
  BigInteger operator / (const int& b) const {
    if( b==0 ) return 0;
    BigInteger res;
    long long reduce=0;
    int signb=b>0?1:-1, b2=b*signb;
    for(int i=len-1; i>=0; --i) {
      res.arr[i] = (arr[i]+reduce*step)/b2;
      reduce = (arr[i]+reduce*step)%b2;
    }
    res.len = len;
    while( res.len>0 && res.arr[res.len-1]==0 )
        --res.len;
    if( res.len==0 ) res.sign=1;
    else res.sign=sign*signb;
    return res;
  }
  BigInteger operator / (const BigInteger& b)
      const {
    BigInteger abs_this=abs(*this);
    if( b==0 ) return 0;
    BigInteger st=0, ed, md;
    if( b.arr[0]>0 ) ed=abs_this/b.arr[0];
    else if( b.arr[1]*b.step+b.arr[0]>0 ) ed=
        abs_this/b.arr[1]*b.step+b.arr[0];
    else ed=abs_this;
    while( st<ed ) {
      md = (st+ed)/2+1;
      if( md*b<=abs_this ) st=md;
      else ed=md-1;
    }
    if( st.len==0 ) st.sign=1;
    else st.sign=sign*b.sign;
```

```
  return st;
}
BigInteger operator % (const int& b) const {
  if( b<=0 ) return 0;
  BigInteger res;
  long long reduce=0;
  for(int i=len-1; i>=0; --i)
    reduce = (arr[i]+reduce*step)%b;
  return reduce*sign;
}
BigInteger operator % (const BigInteger& b)
    const {
  if( b.isInt() ) return *this%int(b.toInt());
  if( b<=0 ) return 0;
  return *this-*this/b*b;
}
bool operator <  (const BigInteger& b) const {
  if( sign!=b.sign ) return sign<b.sign;
  if( len!=b.len ) return len*sign<b.len*b.sign
      ;
  for(int i=len-1; i>=0; --i)
    if( arr[i]!=b.arr[i] ) return arr[i]*sign<b
        .arr[i]*b.sign;
  return false;
}
bool operator == (const BigInteger& b) const {
  if( sign!=b.sign ) return false;
  if( len!=b.len ) return false;
  for(int i=len-1; i>=0; --i)
    if( arr[i]!=b.arr[i] ) return false;
  return true;
}
bool operator <= (const BigInteger& b) const {
    return *this<b || *this==b; }
bool operator >  (const BigInteger& b) const {
    return b<*this; }
bool operator >= (const BigInteger& b) const {
    return b<=*this; }
bool operator != (const BigInteger& b) const {
    return !(*this==b); }
BigInteger operator-() const {
  BigInteger res = *this;
  if( res.len>0 ) res.sign*=-1;
  return res;
}
template<class T> BigInteger operator +  (const
     T& b) const {return *this+BigInteger(b);}
template<class T> BigInteger operator -  (const
     T& b) const {return *this-BigInteger(b);}
template<class T> bool   operator == (const T&
    b) const {return *this==BigInteger(b);}
void print(const char *str="") const {
  if( len==0 ) printf("0");
  else {
    printf("%d", arr[len-1]*sign);
    for(int i=len-2; i>=0; --i) printf("%04d",
        arr[i]);
  }
  printf("%s", str);
}
bool isInt() const {
  if( len>2 ) return false;
  if( len<2 ) return true;
  long long res=toInt();
  return res<(1ll<<31) && res>=-(1ll<<31);
}
friend ostream& operator << ( ostream& out,
    const BigInteger &rhs ) {
  if( rhs.len==0 ) out << '0';
  else {
    out << rhs.arr[rhs.len-1]*rhs.sign;
    for(int i=rhs.len-2; i>=0; --i) out <<
        setfill('0') << setw(BigInteger::digit)
         << rhs.arr[i];
  }
  return out;
}
long long toInt() const {return sign*(1ll*arr
    [1]*step+arr[0]);}
private:
  static const int length = 100;
  static const int digit = 4, step = 10000;
  int arr[length];
  int len, sign;
};
istream& operator >> ( istream& in, BigInteger &
    rhs ) {
  char s[1000];
  in >> s;
  rhs = s;
  return in;
}
```

3.3 Fenwick Tree Range Modify [1, size]

```
inline int lowbit(int x) { return x&-x; }
template<class T>
class fenwick {
public:
  fenwick(int __size=SIZE) {
    size = __size+10;
    a = new T[size], b=new T[size];
    memset(a, 0, sizeof(T)*size);
    memset(b, 0, sizeof(T)*size);
  }
  ~fenwick() { delete[] a, delete[] b;}
  inline void add(int l, int r, long long n) {
    __add(a, r, r*n), __add(a, l-1, (l-1)*-n);
    __add(b, r, n), __add(b, l-1, -n);
  }
  inline long long sum(int l, int r) { return
      __sum(r)-__sum(l-1); }
private:
  int size;
  T *a, *b;
  inline void __add(T *arr, int x, T n) { for(; x
      &&n&&x<size; x+=lowbit(x)) arr[x]+=n; }
  inline T __sum(T x) { return __sum(a, x)+(__sum
      (b, size)-__sum(b, x))*x; }
  inline T __sum(T *arr, int x) {
    T res=0;
    for(; x; x-=lowbit(x)) res+=arr[x];
    return res;
  }
};
```

3.4 Fenwick Tree 2D - [1, size][1, size]

```
int tree[size+1][size+1]={{0}};
inline int lowbit(const int &x) {return x&(-x);}
inline void add(int x, int y, int z) {
  for(int i; x<=n; x+=lowbit(x))
    for(i=y; i<=n; i+=lowbit(i)) tree[x][i]+=z;
}
inline int query(short x, short y) {
  int res=0;
  for(int i; x; x-=lowbit(x))
    for(i=y; i; i-=lowbit(i))
      res+=tree[x][i];
  return res;
}
```

3.5 Skew Heap

```
/*
 * merge : root = merge(x, y)
 * pop   : root = merge(root.lc, root.rc)
*/
const int MAXSIZE = 10000;

class Node {
public:
  int num, lc, rc;
  Node(int _v=0) : num(_v), lc(-1), rc(-1) {}
} tree[MAXSIZE];

int merge(int x, int y){
    if( x==-1 ) return y;
    if( y==-1 ) return x;
    if( tree[x].num<tree[y].num ) // key
        swap(x, y);
    tree[x].rc = merge(tree[x].rc, y);
    swap(tree[x].lc, tree[x].rc);
    return x;
}
```

3.6 Splay Tree

```
template<class T>
struct TNode {
  TNode<T> *c[2], *fa;
  T val, inc, sum;
  int sz;
  void down() {
    val += inc;
    if( lc->fa ) lc->inc += inc;
    if( rc->fa ) rc->inc += inc;
    inc = 0;
  }
  void up() {
    sz = lc->sz + rc->sz + 1;
    sum = val;
    if( lc->fa ) sum += lc->sum + lc->inc*lc->sz;
    if( rc->fa ) sum += rc->sum + rc->inc*rc->sz;
  }
};
template<class T>
class SplayTree {
public:
  void init(const int& n) {
    null = &node[0];
    null->fa = NULL;
    null->val = null->inc = null->sum = null->sz
        = 0;
    ncnt = 0;
    root = newnode(-1, null);
    root->rc = newnode(-1, root);
    root->rc->lc = build(1, n, root->rc);
    root->rc->up(), root->up();
  }
  void update(int l, int r, T val) {
    RotateTo(l-1, null);
    RotateTo(r+1, root);
    root->rc->lc->inc += val;
    root->rc->lc->up();
  }
  ll query(int l, int r) {
    if( l>r ) swap(l, r);
    RotateTo(l-1, null);
    RotateTo(r+1, root);
    TNode<T> *now = root->rc->lc;
    now->up();
    return now->sum + now->inc*now->sz;
  }
private:
  TNode<T> *root, *null;
  TNode<T> node[MAXN];
  int ncnt;
  TNode<T>* newnode(T val, TNode<T> *fa) {
    TNode<T> *x = &node[++ncnt];
    x->lc = x->rc = null;
    x->fa = fa;
    x->val = x->sum = val, x->inc = 0, x->sz = 1;
    return x;
  }
  TNode<T>* build(int l, int r, TNode<T> *fa) {
    if( l>r ) return null;
    int md = (l+r)>>1;
    TNode<T> *now = newnode(all[md], fa);
    now->lc = build(l, md-1, now);
    now->rc = build(md+1, r, now);
    now->up();
    return now;
  }
  void RotateTo(int x, TNode<T> *aim) {
    // find k-th element
    TNode<T> *now = root;
    while( now->lc->sz != x ) {
      if( now->lc->sz > x ) now = now->lc;
      else x -= now->lc->sz+1, now = now->rc;
    }
    splay(now, aim);
  }
  void splay(TNode<T> *now, TNode<T> *aim) {
    // make now become aim's child
    TNode<T> *fa, *fafa;
    while( now->fa != aim ) {
      if( now->fa->fa == aim ) Rotate(now, now->
          fa->lc==now);
      else {
        fa = now->fa, fafa = fa->fa;
        int pos = ( fafa->c[1] == fa );
        if( fa->c[pos] == now ) Rotate(fa, !pos);
        else Rotate(now, pos);
        Rotate(now, !pos);
      }
    }
    now->up();
    if( aim == null ) root = now;
  }
  void Rotate(TNode<T> *now, int fl) {
    // fl : 0 - L-Rotate
    //      1 - R-Rotate
    TNode<T> *fa = now->fa;
    now->down();
    fa->c[!fl] = now->c[fl];
    if( now->c[fl] != null ) now->c[fl]->fa = fa;
    now->fa = fa->fa;
    if( fa->fa != null ) fa->fa->c[ fa->fa->c
        [1]==fa ] = now;
    now->c[fl] = fa, fa->fa = now;
    now->inc = fa->inc, fa->inc = 0;
    fa->up();
  }
};
SplayTree<ll> tree;
```

3.7 Treap

```
class Treap{
  private:
  class Node{
    public:
      Node(  ){ }
```

```
    Node( int val ){ initVal( val ); }
    void initVal( int val ){
      pri = rand();
      this -> val = val;
      this -> maxSum = val;
      this -> maxSumLeft = val;
      this -> maxSumRight = val;
      this -> size = 1;
      this -> l = r = NULL;
      this -> reverse = false;
      this -> setValTag = false;
      this -> sum = val;
    }
    int pri, val, size, maxSum , sum;
    int maxSumLeft, maxSumRight;
    bool reverse, setValTag;
    Node *l, *r;
};

Node* root;
Node* pool;
Node** stk;
int top;
int getSize( Node *x ){
  return x ? x->size : 0;
}
int getMaxSumLeft( Node *x ){
  return x ? x->maxSumLeft : -INF;
}
int getMaxSumRight( Node *x ){
  return x ? x->maxSumRight : -INF;
}
int getMaxSum( Node *x ){
  return x ? x->maxSum : -INF;
}
int getSum( Node *x ){
  return x ? x->sum : 0;
}

void setVal( Node *x , int val ){
  x->val = val;
  x->maxSumLeft = x->maxSumRight = x->maxSum =
      val > 0 ? getSize( x ) * val : val;
  x->sum = val * getSize( x );
  x->setValTag = true;
}
void setReverse( Node *x  ){
  swap( x->l, x->r );
  swap( x->maxSumLeft, x->maxSumRight );
  x->reverse = !x->reverse;
}
void pull( Node *x ){
  x->size = getSize( x->l ) + getSize( x->r ) +
       1;
  int maxSum = max( 0 , getMaxSumRight( x->l )
      ) + x->val + max( 0 , getMaxSumLeft( x->r
       ) );
  maxSum = max( maxSum , max( getMaxSum( x->l )
       , getMaxSum( x->r ) ) );
  x->maxSum = maxSum;

  int maxSumLeft = max( getMaxSumLeft( x->l )
      , getSum( x->l ) + x->val + max( 0 ,
      getMaxSumLeft( x->r ) ) );
  x->maxSumLeft = maxSumLeft;

  int maxSumRight = max( getMaxSumRight( x->r )
      , getSum( x->r ) + x->val + max( 0 ,
      getMaxSumRight( x->l ) ) );
  x->maxSumRight = maxSumRight;
  x->sum = getSum( x->l ) + x->val + getSum( x
      ->r );
}
void push( Node *x ){
  if ( x->reverse ){
    if ( x->r ) setReverse( x->r );
    if ( x->l ) setReverse( x->l );
    x->reverse = false;
  }
  if ( x->setValTag ){
    if ( x->r ) setVal( x->r , x->val );
    if ( x->l ) setVal( x->l , x->val );
    x->setValTag = false;
  }
}
void split( Node *root, Node* &x , Node* &y ,
    int k ){
  if ( !root ){
    x = y = NULL;
    return;
  }
  if ( getSize( root->l ) >= k ){
    y = root;
    push( y );
    split( root->l, x, y->l , k );
    pull( y );
  }
  else{
    x = root;
    push( x );
    split( root->r , x->r , y , k-getSize( root
        ->l )-1 );
    pull( x );
  }
}
Node* Merge( Node *x, Node *y ){
  if ( !x || !y )
    return x ? x : y;

  if ( x->pri > y->pri ){
    push( x );
    x->r = Merge( x->r  , y );
    pull( x );
    return x;
  }
  else{
    push( y );
    y->l = Merge( x , y->l );
    pull( y );
    return y;
  }
}
public:
Treap(){
  root = NULL;
  pool = new Node[MAX_SIZE];
  stk = new Node*[MAX_SIZE];
  for ( top = 0 ; top < MAX_SIZE ; top++ )
    stk[top] = &pool[top];
}
~Treap(){
  delete[] pool;
  delete[] stk;
}
Node* newNode( int val ){
  Node *ret = stk[--top];
  ret -> initVal(val);
  return ret;
}
```

```
  void release( Node *x ){
    if ( x->l )
      release( x->l );
    if ( x->r )
      release( x->r );
    stk[top++] = x;

  }
  void insert( int val ){
    root = Merge( root , newNode( val ) );
  }
  void insert(  int k , queue<int>& q  ){
    Node *l, *r, *newTree = NULL;
    split( root , l , r , k );
    while ( !q.empty() ){
      int a = q.front(); q.pop();
      newTree = Merge( newTree , newNode( a ) );
    }
    root = Merge( l , Merge( newTree , r ) );
  }
  void remove( int k , int sz ){
    Node *l, *m , *r;
    split( root , l , m , k-1 );
    split( m , m , r , sz );
    release( m );
    root = Merge( l , r );
  }
  void setVal( int k , int sz , int val ){
    Node *l, *m , *r;
    split( root , l , m , k-1 );
    split( m , m , r , sz );
    setVal( m , val );
    root = Merge( l , Merge( m , r ) );
  }
  void reverse( int k , int sz ){
    Node *l, *m , *r;
    split( root , l , m , k-1 );
    split( m , m , r , sz );
    setReverse( m );
    root = Merge( l , Merge( m , r ) );
  }
  int getSum( int k , int sz ){
    int ret = 0;
    Node *l, *m , *r;
    split( root , l , m , k-1 );
    split( m , m , r , sz );
    ret = getSum( m );
    root = Merge( l , Merge( m , r ) );
    return ret;
  }
  int getMaxSum(  ){
    return getMaxSum( root );
  }
};
```

3.8 劃分樹

```
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
#define N 100005
int a[N], as[N];//原數組，排序後數組
int n, m;
int sum[20][N];//紀錄第i層的1~j劃分到左子樹的元素個數(包括j)
int tree[20][N];//紀錄第i層元素序列
void build(int c, int l, int r) {
  int i, mid=(l+r)>>1, lm=mid-l+1, lp=l, rp=mid+1;
  for (i=l; i<=mid; i++)
    if (as[i] < as[mid]) lm--;
    //先假設左邊的(mid-l+1)個數都等于as[mid],然后把實際上小于as[mid]的減去
  for (i = l; i <= r; i++){
    if (i == l) sum[c][i] = 0;
    //sum[i]表示[l, i]內有多少個數分到左邊，用DP來維護
    else sum[c][i] = sum[c][i-1];
    if (tree[c][i] == as[mid]){
      if (lm){
        lm--;
        sum[c][i]++;
        tree[c+1][lp++] = tree[c][i];
      }else
        tree[c+1][rp++] = tree[c][i];
    } else if (tree[c][i] < as[mid]){
      sum[c][i]++;
      tree[c+1][lp++] = tree[c][i];
    } else
      tree[c+1][rp++] = tree[c][i];
  }
  if (l == r)return;
  build(c+1, l, mid);
  build(c+1, mid+1, r);
}
int query(int c, int l, int r, int ql, int qr,
    int k){
  int s;//[l, ql)內將被劃分到左子樹的元素數目
  int ss;//[ql, qr]內將被劃分到左子數的元素數目
  int mid=(l+r)>>1;
  if (l == r)
    return tree[c][l];
  if (l == ql){//這裡要特殊處理！
    s = 0;
    ss = sum[c][qr];
  }else{
    s = sum[c][ql 1];
    ss = sum[c][qr]- ;
  } //假設要在區間[l,r]中查找第k大元素，t為當前節點，lch，rch為左右孩子，left，mid為節點t左邊界界和中間點。
  if (k <= ss)//sum[r]-sum[l-1]>=k，查找lch[t],區間對應為[ left+sum[l-1], left+sum[r]-1 ]
    return query(c+1, l, mid, l+s, l+s+ss-1, k);
  else
    //sum[r]-sum[l-1]<k,查找rch[t]，區間對應為[mid+1+l-left-sum[l-1], mid+1+r-left-sum[r]]
    return query(c+1, mid+1, r, mid-l+1+ql-s, mid
        -l+1+qr-s-ss, k-ss);
}
int main(){
  int i, j, k;
  while(~scanf("%d%d", &n, &m)){
    for(i=1; i<=n; i++) {
      scanf("%d", &a[i]);
      tree[0][i] = as[i] = a[i];
    }
    sort(as+1, as+1+n);
    build(0, 1, n);
    while(m--){
      scanf("%d%d%d", &i, &j, &k);
      // i,j分別為區間起始點，k為該區間第k大的數。
      printf("%d\n", query(0, 1, n, i, j, k));
    }
  }
  return 0;
}
```

4 Graph

4.1 Bron-Kerbosch

```
typedef long long ll;
int n;
vector<ll> v, ne;
// ne[u] is the neighbours of u
// v is the result, P = (1<<n) - 1
void BronKerbosch(ll R, ll P, ll X){
  if ((P == 0LL) && (X == 0LL)) {v.push_back(R);
      return ;}
  int u = 0;
  for (; u < n; u ++) if ( (P|X) & (1LL << u) )
      break;
  for (int i = 0; i < n; i ++)
    if ( (P&~ne[u]) & (1LL << i) ){
      BronKerbosch(R | (1LL << i), P & ne[i], X &
          ne[i]);
      P -= (1LL << i); X |= (1LL << i);
    }
}
```

4.2 Dinic

```
class Flow {
public:
  int ncnt;
  void reset() {
    for(int i=0; i<3605; ++i)
      edge[i].clear();
  }
  void AddEdge(int s, int t){
    edge[s].emplace_back(t, 1, edge[t].size());
    edge[t].emplace_back(s, 0, edge[s].size()-1);
        //
  }
  int MaxFlow(int s, int t){ // Dinic
    int f = 0, df;
    while (BFS(s, t)!=-1){
      while (true){
        memset(vst, 0, sizeof(vst));
        df = DFS(s, 1<<30, t);
        if (!df) break;
        f += df;
      }
    }
    return f;
  }

private:
  int d[3605]; // Dicic distance
  bool vst[3605];

  struct Edge {
    Edge(const int& t, const int& c, const int& r
        ) : tar(t), cap(c), rev(r) {}
    int tar;
    int cap;
    int rev;
    operator tuple<int&,int&,int&>() { return
        tuple<int&,int&,int&>{tar, cap, rev}; }
  };
  vector<Edge> edge[3605];

  int DFS(int now, int df, int t){
    if (now==t) return df;
    if (vst[now]) return 0;
    vst[now] = true;
    int nxt, re;
    int cap;
    for(auto& edg: edge[now]) {
      tie(nxt, cap, re) = edg;
      if (cap>0 && d[nxt]==d[now]+1){
        int f = DFS(nxt, min(df, cap), t);
        if (f){
          edg.cap -= f;
          edge[nxt][re].cap += f;
          return f;
        }
      }
    }
    return 0;
  }

  int BFS(int s, int t){
    memset(d, -1, sizeof(d));
    memset(vst, false, sizeof(vst));
    d[s] = 0;
    vst[s] = true;
    queue<int> qq;
    qq.push(s);
    int now, nxt, re;
    int cap;
    while (!qq.empty()) {
      now = qq.front();
      qq.pop();
      for(auto& edg: edge[now]) {
        tie(nxt, cap, re) = edg;
        if (!vst[nxt] && cap) {
          vst[nxt] = true;
          d[nxt] = d[now] + 1;
          if (nxt==t) return d[nxt];
          qq.push(nxt);
        }
      }
    }
    return d[t];
  }
};

Flow flow;
```

4.3 Heavy-Light Decomposition

```
#include <cstdio>
#include <cstring>
#include <list>
#include <algorithm>
#define clear(x,y) memset(x, y, sizeof(x))
using namespace std;
const int N=100005;
class zkw_seg_tree {
public:
  struct node {
    node() {add=sum=0, len=1;}
    int len, add, sum;
  };
  zkw_seg_tree(int size) { // [1,size]
    dep=lg2(size-1)+1;
    delta=(1<<dep)-1;
    arr=new node[1<<(dep+1)];
    for(int i=delta; i>0; --i) arr[i].len=arr[i+i
        ].len<<1;
  }
  ~zkw_seg_tree() {
    delete[] arr;
  }
  inline void update(int l, int r, int num=1) {
    l+=delta-1, r+=delta+1;
    int l0=l, r0=r;
    while( r-l>1 ) {
```

```
      if( (l&1)^1 ) ++l, arr[l].add+=num, arr[l].
          sum+=arr[l].len*num;
      if( (r&1)^0 ) --r, arr[r].add+=num, arr[r].
          sum+=arr[r].len*num;
      l>>=1, r>>=1;
    }
    __update(l0), __update(r0);
  }
  inline int query(int l, int r) {
    __down(l+delta), __down(r+delta);
    l+=delta-1, r+=delta+1;
    int res=0;
    while( r-l>1 ) {
      if( (l&1)^1 ) res+=arr[l+1].sum;
      if( (r&1)^0 ) res+=arr[r-1].sum;
      l>>=1, r>>=1;
    }
    return res;
  }
private:
  node *arr;
  int dep;
  int delta;
  inline int lg2(int x) {int r;for(r=-1; x; x
      >>=1, ++r);return r;}
  inline void __update(int x) {
    while( x>1 ) x>>=1, arr[x].sum=arr[x+x].sum+
        arr[x+x+1].sum+arr[x].len+arr[x].add;
  }
  inline void __down(int x) {
    for(int i=dep, tmp; i>0; --i) {
      tmp=x>>i;
      arr[tmp<<1].add+=arr[tmp].add, arr[(tmp<<1)
          +1].add+=arr[tmp].add;
      arr[tmp<<1].sum+=arr[tmp].add*arr[tmp<<1].
          len, arr[(tmp<<1)+1].sum+=arr[tmp].add*
          arr[tmp<<1].len;
      arr[tmp].add=0;
    }
  }
} segtree(N);
list<int> all[N];
int dep[N];
int chn[N], son[N], fat[N], anc[N];
int flag;
int seg_pos[N];
void dfs1(int now) {
  chn[now]=0;
  son[now]=-1;
  for(list<int>::iterator ee=all[now].begin(); ee
      !=all[now].end(); ++ee) {
    if( !~dep[*ee] ) {
      dep[*ee]=dep[now]+1, fat[*ee]=now;
      dfs1(*ee);
      chn[now]+=chn[*ee];
      if( !~son[now] || chn[*ee]>chn[son[now]] )
          son[now]=*ee;
    }
  chn[now]++;
  }
}
void dfs2(int now, int now_anc) {
  anc[now]=now_anc;
  seg_pos[now]=flag++;
  if( ~son[now] ) dfs2(son[now], now_anc);
  for(list<int>::iterator ee=all[now].begin(); ee
      !=all[now].end(); ++ee)
    if( fat[*ee]==now && son[now]!=*ee )
      dfs2(*ee, *ee);
}
inline int cmd(int ch, int v1, int v2) {
  int res=0;
  while( anc[v1]!=anc[v2] ) {
    if( dep[anc[v1]]>dep[anc[v2]] ) swap(v1, v2);
        // anc[v2] is deeper
    if( ch==1 ) {
      if( anc[v2]^v2 ) segtree.update(seg_pos[son
          [anc[v2]]], seg_pos[v2]);
      segtree.update(seg_pos[anc[v2]], seg_pos[
          anc[v2]]);
    }
    else {
      if( anc[v2]^v2 ) res+=segtree.query(seg_pos
          [son[anc[v2]]], seg_pos[v2]);
      res+=segtree.query(seg_pos[anc[v2]],
          seg_pos[anc[v2]]);
    }
    v2=fat[anc[v2]];
  }
  if( v1!=v2 ) {
    if( dep[v1]>dep[v2] ) swap(v1, v2); // v2 is
        deeper
    if( ch==1 ) segtree.update(seg_pos[son[v1]],
        seg_pos[v2]);
    else res+=segtree.query(seg_pos[son[v1]],
        seg_pos[v2]);
  }
  return res;
}
int main() {
  int n, q;
  scanf("%d%d", &n, &q);
  for(int i=0, v1, v2; i<n-1; ++i){
    scanf("%d%d", &v1, &v2);
    all[v1].push_back(v2), all[v2].push_back(v1);
  }
  clear(dep, -1);
  dep[1]=0, fat[1]=0, flag=0;
  dfs1(1);
  dfs2(1, 1);
  char ch[5];
  int v1, v2;
  for(int i=0; i<q; ++i) {
    scanf("%s %d %d", ch, &v1, &v2);
    if( ch[0]=='P' ) cmd(1, v1, v2);
    else printf("%d\n", cmd(2, v1, v2));
  }
}
```

4.4 Hungarian

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;
#define N 550              //max number of
    vertices in one part
#define INF 100000000    //just infinity

int cost[N][N];          //cost matrix
int n, max_match;        //n workers and n jobs
int lx[N], ly[N];        //labels of X and Y
    parts
int xy[N];               //xy[x] - vertex that is
     matched with x,
int yx[N];               //yx[y] - vertex that is
     matched with y
bool S[N], T[N];         //sets S and T in
    algorithm
int slack[N];            //as in the algorithm
    description
```

```
int slackx[N];           //slackx[y] such a
    vertex, that l(slackx[y]) + l(y) - w(slackx[y
    ],y) = slack[y]
int pre[N];              //array for memorizing
    alternating paths


void init_labels() {
  memset(lx, 0, sizeof(lx));
  memset(ly, 0, sizeof(ly));
  for (int x = 0; x < n; x++)
    for (int y = 0; y < n; y++)
      lx[x] = max(lx[x], cost[x][y]);
}
void update_labels() {
  int x, y, delta = INF;             //init delta
       as infinity
  for (y = 0; y < n; y++)            //calculate
      delta using slack
    if (!T[y])
      delta = min(delta, slack[y]);
  for (x = 0; x < n; x++)            //update X
      labels
    if (S[x]) lx[x] -= delta;
  for (y = 0; y < n; y++)            //update Y
      labels
    if (T[y]) ly[y] += delta;
  for (y = 0; y < n; y++)            //update
      slack array
    if (!T[y])
      slack[y] -= delta;
}

void add_to_tree(int x, int prex) {
  S[x] = true;
  pre[x] = prex;
  for (int y = 0; y < n; y++)
    if (lx[x] + ly[y] - cost[x][y] < slack[y]){
      slack[y] = lx[x] + ly[y] - cost[x][y];
      slackx[y] = x;
    }
}

void augment() {
  if (max_match == n) return;
  int x, y, root;
  int q[N], wr = 0, rd = 0;

  memset(S, false, sizeof(S));
  memset(T, false, sizeof(T));
  memset(pre, -1, sizeof(pre));
  for (x = 0; x < n; x++)
    if (xy[x] == -1){
      q[wr++] = root = x;
      pre[x] = -2;
      S[x] = true;
      break;
    }

  for (y = 0; y < n; y++){
    slack[y] = lx[root] + ly[y] - cost[root][y];
    slackx[y] = root;
  }

  while (true){
    while (rd < wr){
      x = q[rd++];
      for (y = 0; y < n; y++)
        if (cost[x][y] == lx[x] + ly[y] &&  !T[y
            ]){
          if (yx[y] == -1) break;
          T[y] = true;
          q[wr++] = yx[y];

              //add vertex yx[y], which is
              matched with y, to the queue
          add_to_tree(yx[y], x);
                                          //add
              edges (x,y) and (y,yx[y]) to the
              tree
        }
      if (y < n) break;

       //augmenting path found!
    }
    if (y < n) break;

       //augmenting path found!
    update_labels();

       //augmenting path not found, so improve
       labeling
    wr = rd = 0;
    for (y = 0; y < n; y++)
      if (!T[y] && slack[y] == 0){
        if (yx[y] == -1){

          //exposed vertex in Y found -
          augmenting path exists!
          x = slackx[y];
          break;
        }
        else{
          T[y] = true;

             //else just add y to T,
          if (!S[yx[y]]){
            q[wr++] = yx[y];
                                              //
                add vertex yx[y], which is
                matched with y, to the queue
            add_to_tree(yx[y], slackx[y]);
                                //and add edges
                (x,y) and (y, yx[y]) to the tree
          }
        }
      }
    if (y < n) break;

       //augmenting path found!
  }
  if (y < n){

    //we found augmenting path!
    max_match++;

       //increment matching
    //in this cycle we inverse edges along
       augmenting path
    for (int cx = x, cy = y, ty; cx != -2; cx =
       pre[cx], cy = ty){
      ty = xy[cx];
      yx[cy] = cx;
      xy[cx] = cy;
    }
    augment();

       //recall function, go to step 1 of the
       algorithm
  }
```

```
}
int hungarian(){
  int ret = 0;                        //weight of
      the optimal matching
  max_match = 0;                      //number of
      vertices in current matching
  memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx));
  init_labels();                      //step 0
  augment();                          //steps 1-3
  for (int x = 0; x < n; x++)         //forming
      answer there
    ret += cost[x][xy[x]];
  return ret;
}
int main(){
  while ( scanf("%d",&n) == 1 ){
    for ( int i = 0 ; i < n ;i++ )
      for ( int j = 0 ; j < n ; j++ )
        scanf("%d",&cost[i][j]);
    int ret = hungarian();
    for ( int i = 0 ; i < n ;i++ )
      printf("%d%c",lx[i],i==n-1?'\n':' ');
    for ( int i = 0 ; i < n ; i++ )
      printf("%d%c",ly[i],i==n-1?'\n':' ');
    printf("%d\n",ret);
  }
  return 0;
}
```

4.5 Kuhn Munkres (bcw)

```
struct KM{
// Maximum Bipartite Weighted Matching (Perfect
    Match)
  static const int MXN = 650;
  static const int INF = 2147483647; // long long
  int n,match[MXN],vx[MXN],vy[MXN];
  int edge[MXN][MXN],lx[MXN],ly[MXN],slack[MXN];
  // ^^^^ long long
  void init(int _n){
    n = _n;
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
  }
  void add_edge(int x, int y, int w){ // long
      long
    edge[x][y] = w;
  }
  bool DFS(int x){
    vx[x] = 1;
    for (int y=0; y<n; y++){
      if (vy[y]) continue;
      if (lx[x]+ly[y] > edge[x][y]){
        slack[y] = min(slack[y], lx[x]+ly[y]-edge
            [x][y]);
      } else {
        vy[y] = 1;
        if (match[y] == -1 || DFS(match[y])){
          match[y] = x;
          return true;
        }
      }
    }
    return false;
  }
  int solve(){
    fill(match,match+n,-1);
    fill(lx,lx+n,-INF);
    fill(ly,ly+n,0);
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        lx[i] = max(lx[i], edge[i][j]);
    for (int i=0; i<n; i++){
      fill(slack,slack+n,INF);
      while (true){
        fill(vx,vx+n,0);
        fill(vy,vy+n,0);
        if ( DFS(i) ) break;
        int d = INF; // long long
        for (int j=0; j<n; j++)
          if (!vy[j]) d = min(d, slack[j]);
        for (int j=0; j<n; j++){
          if (vx[j]) lx[j] -= d;
          if (vy[j]) ly[j] += d;
          else slack[j] -= d;
        }
      }
    }
    int res=0;
    for (int i=0; i<n; i++)
      res += edge[match[i]][i];
    return res;
  }
}graph;
```

4.6 Manhattan MST (bcw)

```
#include<bits/stdc++.h>
#define REP(i,n) for(int i=0;i<n;i++)
using namespace std;
typedef long long LL;
const int N=200100;
int n,m;
struct PT {int x,y,z,w,id;}p[N];
inline int dis(const PT &a,const PT &b){return
    abs(a.x-b.x)+abs(a.y-b.y);}
inline bool cpx(const PT &a,const PT &b){return a
    .x!=b.x? a.x>b.x:a.y>b.y;}
inline bool cpz(const PT &a,const PT &b){return a
    .z<b.z;}
struct E{int a,b,c;}e[8*N];
bool operator<(const E&a,const E&b){return a.c<b.
    c;}
struct Node{
  int L,R,key;
}node[4*N];
int s[N];
int F(int x){return s[x]==x?x:s[x]=F(s[x]);}
void U(int a,int b){s[F(b)]=F(a);}
void init(int id,int L,int R) {
  node[id]=(Node){L,R,-1};
  if(L==R)return;
  init(id*2,L,(L+R)/2);
  init(id*2+1,(L+R)/2+1,R);
}
void ins(int id,int x) {
  if(node[id].key==-1 || p[node[id].key].w>p[x].w
      )node[id].key=x;
  if(node[id].L==node[id].R)return;
  if(p[x].z<=(node[id].L+node[id].R)/2)ins(id*2,x
      );
  else ins(id*2+1,x);
}
int Q(int id,int L,int R){
  if(R<node[id].L || L>node[id].R)return -1;
  if(L<=node[id].L && node[id].R<=R)return node[
      id].key;
  int a=Q(id*2,L,R),b=Q(id*2+1,L,R);
  if(b==-1 || (a!=-1 && p[a].w<p[b].w)) return a;
  else return b;
```

```
}
void calc() {
  REP(i,n) {
    p[i].z=p[i].y-p[i].x;
    p[i].w=p[i].x+p[i].y;
  }
  sort(p,p+n,cpz);
  int cnt=0,j,k;
  for(int i=0;i<n;i=j){
    for(j=i+1;p[j].z==p[i].z && j<n;j++);
    for(k=i,cnt++;k<j;k++)p[k].z=cnt;
  }
  init(1,1,cnt);
  sort(p,p+n,cpx);
  REP(i,n) {
    j=Q(1,p[i].z,cnt);
    if(j!=-1)e[m++]=(E){p[i].id,p[j].id,dis(p[i],
        p[j])};
    ins(1,i);
  }
}
LL MST() {
  LL r=0;
  sort(e,e+m);
  REP(i,m) {
    if(F(e[i].a)==F(e[i].b))continue;
    U(e[i].a,e[i].b);
    r+=e[i].c;
  }
  return r;
}
int main(){
  int ts;
  scanf("%d", &ts);
  while (ts--) {
    m = 0;
    scanf("%d",&n);
    REP(i,n) {
      scanf("%d%d",&p[i].x,&p[i].y);
      p[i].id=s[i]=i;
    }
    calc();
    REP(i,n)p[i].y= -p[i].y;
    calc();
    REP(i,n)swap(p[i].x,p[i].y);
    calc();
    REP(i,n)p[i].x=-p[i].x;
    calc();
    printf("%lld\n",MST()*2);
  }
  return 0;
}
```

4.7 maximum matching in general graph

```
//Problem:http://acm.timus.ru/problem.aspx?space
    =1&num=1099
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int N=250;
int n;
int head;
int tail;
int Start;
int Finish;
int link[N];    //表示哪個點匹配了哪個點
int Father[N];    //這個就是增廣路的Father……但
    是用起來太精髓了
int Base[N];    //該點屬於哪朵花
int Q[N];
bool mark[N];
bool map[N][N];
bool InBlossom[N];
bool in_Queue[N];

void CreateGraph(){
  int x,y;
  scanf("%d",&n);
  while (scanf("%d%d",&x,&y)!=EOF)
    map[x][y]=map[y][x]=1;
}
void BlossomContract(int x,int y){
  fill(mark,mark+n+1,false);
  fill(InBlossom,InBlossom+n+1,false);
  #define pre Father[link[i]]
  int lca,i;
  for (i=x;i;i=pre) {i=Base[i]; mark[i]=true; }
  for (i=y;i;i=pre) {i=Base[i]; if (mark[i]) {lca
      =i; break;} }  //尋找lca之旅……一定要注意i
      =Base[i]
  for (i=x;Base[i]!=lca;i=pre){
    if (Base[pre]!=lca) Father[pre]=link[i]; //對
        於BFS樹中的父邊是匹配邊的點，Father向後跳
    InBlossom[Base[i]]=true;
    InBlossom[Base[link[i]]]=true;
  }
  for (i=y;Base[i]!=lca;i=pre){
    if (Base[pre]!=lca) Father[pre]=link[i]; //同
        理
    InBlossom[Base[i]]=true;
    InBlossom[Base[link[i]]]=true;
  }
  #undef pre
  if (Base[x]!=lca) Father[x]=y;   //注意不能從
      lca這個奇環的關鍵點跳回來
  if (Base[y]!=lca) Father[y]=x;
  for (i=1;i<=n;i++)
    if (InBlossom[Base[i]]){
      Base[i]=lca;
      if (!in_Queue[i]){
        Q[++tail]=i;
        in_Queue[i]=true;  //要注意如果本來連向
            BFS樹中父結點的邊是非匹配邊的點，可能
            是沒有入隊的
      }
    }
}
void Change(){
  int x,y,z;
  z=Finish;
  while (z){
    y=Father[z];
    x=link[y];
    link[y]=z;
    link[z]=y;
    z=x;
  }
}
void FindAugmentPath(){
  fill(Father,Father+n+1,0);
  fill(in_Queue,in_Queue+n+1,false);
  for (int i=1;i<=n;i++) Base[i]=i;
  head=0; tail=1;
  Q[1]=Start;
  in_Queue[Start]=1;
  while (head!=tail){
```

```
    int x=Q[++head];
    for (int y=1;y<=n;y++)
      if (map[x][y] && Base[x]!=Base[y] && link[x
          ]!=y)   //無意義的邊
        if ( Start==y || link[y] && Father[link[y
            ]] ) //精髓地用Father表示該點是否
          BlossomContract(x,y);
        else if (!Father[y]){
          Father[y]=x;
          if (link[y]){
            Q[++tail]=link[y];
            in_Queue[link[y]]=true;
          }
          else{
            Finish=y;
            Change();
            return;
          }
        }
  }
}
void Edmonds(){
  memset(link,0,sizeof(link));
  for (Start=1;Start<=n;Start++)
    if (link[Start]==0)
      FindAugmentPath();
}
void output(){
  fill(mark,mark+n+1,false);
  int cnt=0;
  for (int i=1;i<=n;i++)
    if (link[i]) cnt++;
  printf("%d\n",cnt);
  for (int i=1;i<=n;i++)
    if (!mark[i] && link[i]){
      mark[i]=true;
      mark[link[i]]=true;
      printf("%d %d\n",i,link[i]);
    }
}
int main(){
  CreateGraph();
  Edmonds();
  output();
  return 0;
}
```

5 Math

5.1 China remainder theorem

$ans \equiv a_i \ (mod\ m_i)$

```
int china_remainder_theorem(int n, int ai[], int
    mi[]) {
  int gcdn, x, y, reduce, tmp;
  for(int i=1; i<n; ++i) {
    gcdn=ext_gcd(mi[i-1], mi[i], x, y);
    reduce=ai[i]-ai[i-1];
    if( reduce%gcdn!=0 )
      return -1;
    tmp=mi[i]/gcdn;
    x=(reduce/gcdn*x%tmp+tmp)%tmp;
    ai[i] = ai[i-1] + mi[i-1]*x;
    mi[i] = mi[i-1]*tmp;
  }
  return ai[n-1]%mod;
}
```

5.2 Euler's phi function O(n)

1. $gcd(x,y)=d \Rightarrow \phi(xy)=\frac{\phi(x)\phi(y)}{\phi(d)}$
2. $p\ is\ prime \Rightarrow \phi(p^k)=p^{k-1}\phi(p)$
3. $p\ is\ prime \Rightarrow \phi(p^k)=\phi(p^{k-1})\times p$
4. $n=p_1^{k_1}p_2^{k_2}\cdots p_m^{k_m}$
   $\Rightarrow \phi(n)=p_1^{k_1-1}\phi(p_1)\ p_2^{k_2-1}\phi(p_2)\cdots p_m^{k_m-1}\phi(p_m)$

```
const int MAXN = 100000;
int phi[MAXN], prime[MAXN], pn=0;
memset(phi, 0, sizeof(phi));
for(int i=2; i<MAXN; ++i) {
  if( phi[i]==0 ) prime[pn++]=i, phi[i]=i-1;
  for(int j=0; j<pn; ++j) {
    if( i*prime[j]>MAXN ) break;
    if( i%prime[j]==0 ) {
      phi[i*prime[j]] = phi[i] * prime[j];
      break;
    }
    phi[i*prime[j]] = phi[i] * phi[prime[j]];
  }
}
```

5.3 Extended Euclid's Algorithm

$ax+by=gcd(a,b)$

```
int ext_gcd(int a, int b, int &x, int &y){
  int x2;
  if( b==0 ) {
    x=1, y=0;
    return a;
  }
  int gcdn=ext_gcd(b, a%b, x, y), x2=x;
  x=y, y=x2-a/b*y;
  return gcdn;
}
int ext_gcd(int a, int b, int &x, int &y){
  int t, px=1, py=0, tx,ty;
  x=0, y=1;
  while(a%b!=0) {
    tx=x, ty=y;
    x=x*(-a/b)+px, y=y*(-a/b)+py;
    px=tx, py=ty;
    t=a, a=b, b=t%b;
  }
  return b;
}
```

5.4 FFT

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

typedef long long ll;

int ODD[1<<20], EVEN[1<<20], ODD_B[1<<20], EVEN_B
    [1<<20];

int mul_order(int n) {
  n--;
  int high_bit = n&(-n);
  for(n-=high_bit; n; n-=high_bit)
    high_bit = n&(-n);
  return high_bit<<2;
}

// return a^x mod p
int mod_pow(int a, int x, int p) {
  if(x==0) return 1;
  ll ret = mod_pow(a,x>>1,p);
  ret*=ret;
  ret%=p;
  return (x&1)?(ret*a)%p:ret;
}

// only works for prime p
int mod_inverse(int x, int p) { return mod_pow(x,
    p-2,p); }

// n is a power of 2 and answer is at most
    upper_bound
// 2013265921=1+2^27*3*5 is a prime
inline int suitable_prime(int n, int upper_bound)
     { return 2013265921; }

// not general version
int primitive_root(int p) {
  int ret;
  srand(714091); // THOR
//  srand(time(NULL));
  for(ret=rand()%p;;ret=rand()%p) {
    if(mod_pow(ret,(p-1)/2,p)!=1 &&
      mod_pow(ret,(p-1)/3,p)!=1 &&
      mod_pow(ret,(p-1)/5,p)!=1)
      return ret;
  }
}

// nth principle root of unity (mod p)
int principal_root(int n, int p) {
  int g=primitive_root(p);
  return mod_pow(g,(p-1)/n,p);
}

void cooley_tukey(int *a, int *b, int n, ll r, ll
     p) {
  if(n>1) {
    int half_n=n>>1;
    int *odd=ODD+half_n, *even=EVEN+half_n;
    int *odd_b=ODD_B+half_n, *even_b=EVEN_B+
        half_n;
    for(int i=0; i<n; i+=2) {
      even[i>>1]=a[i];
      odd[i>>1]=a[1|i];
    }
    cooley_tukey(even,even_b,n>>1,r*r%p,p);
    cooley_tukey(odd,odd_b,n>>1,r*r%p,p);
    ll iter=1, p2=p*p;
    for(int i=0; i<half_n; i++) {
      ll odd_part=iter*odd_b[i];
      b[i]=(even_b[i]+odd_part)%p;
      b[half_n|i]=(even_b[i]+p2-odd_part)%p;
      iter*=r;
      iter%=p;
    }
  }
  else b[0]=a[0];
}

void print(int *a, int n) {
  for(int i=0; i<n; i++)
    printf("%d%c",a[i],i==n-1?'\n':' ');
}

// c=a*b where c is initially empty
void multiply(int *a, int *b, int *c,
  int n, ll inv_n, ll r, ll inv_r, ll p) {
  int *f, *g, *h;
  f=new int[n];
  g=new int[n];
  h=new int[n];
  cooley_tukey(a,f,n,r,p);
  cooley_tukey(b,g,n,r,p);
  for(int i=0; i<n; i++) h[i]=(ll)f[i]*g[i]%p;
  cooley_tukey(h,c,n,inv_r,p);
  for(int i=0; i<n; i++) c[i]=c[i]*inv_n%p;
  delete[] f;
  delete[] g;
  delete[] h;
}

int main() {
  int n, N;
  scanf("%d",&n);
  N=mul_order(n);
  printf("N=%d\n",N);
  int *a, *b, *c;
  a=new int[N];
  b=new int[N];
  c=new int[N];
  for(int i = 0; i < n; i++) scanf("%d",&a[i]);
  for(int i = 0; i < n; i++) scanf("%d",&b[i]);
  for(int i = n; i< N; i++)  a[i]=b[i]=0;
  int p=suitable_prime(N,10007);
  ll r=principal_root(N,p);
  ll inv_r=mod_inverse(r,p);
  ll inv_n=mod_inverse(N,p);
  multiply(a,b,c,N,inv_n,r,inv_r,p);
  int last=N-1;
  while(last>=0 && c[last]==0) last--;
  for(int i = 0; i<=last; i++)
    printf("%d%c",c[i],last==i?'\n':' ');
  return 0;
}
```

5.5 Gaussian Elimination

```
// default for module version, comments for
    double version
//double mmap[row][column];
const ll modn = 1000000007;
ll mmap[row][column];
ll inv(ll b)
{
  return (b==1)?1:inv(modn%b)*(modn-modn/b)%modn;
}
void gauss(ll mat[row][column],int n,int m)
{
```

```
  int k=0;
  for(int i=0; i<m; i++)
    for(int j=k; j<n; j++)
      if(mat[j][i]!=0){
        for(int l=i; l<m; l++)
          swap(mat[k][l],mat[j][l]);
        for(j++; j<n; j++){
          if(mat[j][i]==0)
            continue;
          //double scale=mat[j][i]/mat[k][i];
          long long scale=mat[j][i]*inv(mat[k][i
              ])%modn;  //mod version
          for(int p=i+1; p<m; p++)
            //mat[j][p]-=mat[k][p]*scale;
            mat[j][p]=(mat[j][p]-mat[k][p]*scale%
                modn+modn)%modn;
          mat[j][i]=0;
        }
        k++;
        break;
      }
}
```

5.6 Miller Rabin

```
ll mul(ll a, ll b, ll n) { // a*b%n
  ll r = 0;
  a %= n, b %= n;
  while(b){
    if( b&1 ) r = a+r>=n ? a+r-n : a+r;
    a = a+a>=n ? a+a-n : a+a;
    b >>= 1;
  }
  return r;
}
ll powmod(ll a, ll d, ll n) { // a^d%n
  if(d==0) return 1ll;
  if(d==1) return a%n;
  return mul(powmod(mul(a, a, n), d>>1, n), d%2?a
      :1, n);
}
bool miller_rabin(ll n, ll a) {
  if(__gcd(a,n)==n) return true;
  if(__gcd(a,n)!=1) return false;
  ll d = n-1, r = 0, res;
  while(d%2==0) { ++r; d>>=1; }
  res = powmod(a, d, n);
  if(res==1||res==n-1) return true;
  while(r--) {
    res = mul(res, res, n);
    if(res==n-1) return true;
  }
  return false;
}
bool isPrime(ll n) {
  ll as[7]={2, 325, 9375, 28178, 450775, 9780504,
       1795265022}; //  2, 7, 61
  for(int i=0; i<7; i++)
    if( !miller_rabin(n, as[i]) )
      return false;
  return true;
}
```

5.7 Möbius function

```
int* isp;
char fcnt[N+5];
int mobius[N+5];
void make_mobius(int n) {
  isp = mobius;
  memset(mobius, true, sizeof(mobius));
  memset(fcnt, 0, sizeof(fcnt));
  for(int i=2; i<=n; ++i) {
    if( isp[i] ) {
      fcnt[i] = 1;
      for(int j=i+i; j<=n; j+=i) {
        isp[j] = false;
        if( fcnt[j]!=-1 ) fcnt[j]++;
      }
      if( i<=10000 )
        for(int ii=i*i, j=ii; j<=n; j+=ii) {
          fcnt[j] = -1;
        }
    }
  }
  mobius[0] = 0;
  mobius[1] = 1;
  for(int i=2; i<=n; ++i) {
    if( fcnt[i]==-1 ) mobius[i] = 0;
    else if( fcnt[i]&1 ) mobius[i] = -1;
    else mobius[i] = 1;
  }
}
```

6 String

6.1 AhoCorasick

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

template<int NodeSZ>
class AhoCorasick {
public:
  AhoCorasick() { clear(); }
  void clear() {
    all[0] = Node();
    ncnt = 1;
  }
  void insert(char *s) {
    Node *curr = &all[0], *next;
    for(int i=0; s[i]; ++i) {
      next = curr->next[idx(s[i])];
      if( next == NULL )
        next = &all[ncnt], all[ncnt++] = Node();
      curr = curr->next[idx(s[i])] = next;
    }
    curr->val++;
  }
  void build() {
    queue<Node*> qq;
    qq.push(&all[0]);
    while( !qq.empty() ) {
      Node *curr = qq.front(), *fail;
      qq.pop();
      for(int i=0; i<NodeSZ; ++i) {
        if( !curr->next[i] ) continue;
        qq.push(curr->next[i]);
        fail = curr->fail;
        while( fail && !fail->next[i] )
          fail = fail->fail;
        curr->next[i]->fail = fail ? fail->next[i
            ] : &all[0];
      }
    }
  }
  int count(char *s) {
    build();
    int cnt = 0;
    Node *curr = &all[0], *tmp;
    for(int i=0, ch; s[i]; ++i) {
      ch = idx(s[i]);
      while( curr && !curr->next[ch] )
        curr = curr->fail;
      curr = curr ? curr->next[ch] : all[0].next[
          ch];
      tmp = curr;
      while( tmp && tmp->val ) {
        cnt += tmp->val;
        tmp->val = 0;
        tmp = tmp->fail;
      }
    }
    return cnt;
  }

private:
  struct Node {
    Node() : val(0), fail(NULL) {
      for(int i=0; i<NodeSZ; ++i) next[i] = NULL;
    }
    int val;
    Node *fail, *next[NodeSZ];
  };
  Node all[250005];
  int ncnt;
  inline int idx(char c) { return c-'a'; }
};
AhoCorasick<26> AC;
```

6.2 KMP

```
int KMP(char pat[5005], char str[5005]) {
  if( strlen(pat)>strlen(str) ) return -1;
  int failure[5005];
  int len=strlen(pat);
  for(int i=1, j=failure[0]=-1; i<len; ++i) {
    while( j>=0 && pat[j+1]^pat[i] ) j=failure[j
        ];
    if( pat[j+1]==pat[i] ) ++j;
    failure[i]=j;
  }
  for(int i=0, j=-1; str[i]; ++i) {
    while( j>=0 && str[i]^pat[j+1] ) j=failure[j
        ];
    if( str[i]==pat[j+1] ) ++j;
    if( j==len-1 ) {
      return i-len+1; // rec this!!
      j=failure[j];
    }
  }
  return -1;
}
```

6.3 Longest Palindromic Substring

```
char t[1001];   // 要處理的字串
cahr s[1001 * 2]; // 中間插入特殊字元的t。
int Z[1001 * 2], L, R;  // Gusfield's Algorithm
// 由a往左、由b往右，對稱地作字元比對。
int match(int a, int b) {
  int i = 0;
  while (a-i>=0 && b+i<N && s[a-i] == s[b+i]) i
      ++;
  return i;
}
void longest_palindromic_substring()
{
  int N = strlen(t);
  // 在t中插入特殊字元，存放到s。
  memset(s, '.', N*2+1);
  for (int i=0; i<N; ++i) s[i*2+1] = t[i];
  N = N*2+1;
  // modified Gusfield's lgorithm
  Z[0] = 1;
  L = R = 0;
  for (int i=1; i<N; ++i) {
    int ii = L - (i - L);   // i的映射位置
    int n = R + 1 - i;
    if (i > R) {
      Z[i] = match(i, i);
      L = i;
      R = i + Z[i] - 1;
    }
    else if (Z[ii] == n) {
      Z[i] = n + match(i-n, i+n);
      L = i;
      R = i + Z[i] - 1;
    }
    else Z[i] = min(Z[ii], n);
  }
  // 尋找最長迴文子字串的長度。
  int n = 0, p = 0;
  for (int i=0; i<N; ++i)
    if (Z[i] > n) n = Z[p = i];
  // 記得去掉特殊字元。
```

```
  cout << "最長迴文子字串的長度是" << (n-1) / 2;
  // 印出最長迴文子字串，記得別印特殊字元。
  for (int i=p-Z[p]+1; i<=p+Z[p]-1; ++i)
    if (i & 1) cout << s[i];
}
```

6.4 Suffix Array

```
const int LEN = 1000;
int rk[LEN], sa[LEN];
int height[LEN];
int cnt[LEN], rr[2][LEN];
inline bool same(int *_rk, int a, int b, int l) {
     return _rk[a]==_rk[b]&&_rk[a+l]==_rk[b+l]; }
void make_height(char str[]) {
  int len=strlen(str);
  memset(height, 0, sizeof(height));
  for(int i=0, j=0; i<len; ++i, j=height[rk[i
      -1]]-1) {
    if (rk[i]==1) continue;
    if (j<0) j=0;
    while (i+j<len && sa[rk[i]-1]+j<len &&
        str[i+j]==str[sa[rk[i]-1]+j]) ++j;
    height[rk[i]]=j;
  }
}
void suffix_array(char str[], int n, int MAXRK =
    256) {
  int *rk1=rr[0], *rk2=rr[1]; // rolling array
  int *y = rk; // share memory
  memset(rr[1], 0, sizeof(rr[1]));
  memset(cnt, 0, sizeof(cnt));
  int i, p;
  for(i=0; i<n; ++i) rk2[i]=str[i], cnt[rk2[i
      ]]++;
  for(i=1; i<MAXRK; ++i) cnt[i]+=cnt[i-1];
  for(i=n-1; i>=0; --i) sa[--cnt[rk2[i]]]=i;
  for(int j=1; p<n; j<<=1, MAXRK=p) {
    // 表示用第二個key(rk2)排序後 從 y[i] 開始的
        後綴排第i名
    for(p=0, i=n-j; i<n; ++i) y[p++]=i;
    for(i=0; i<n; ++i) if (sa[i]>=j) y[p++]=sa[i
        ]-j;
    memset(cnt, 0, sizeof(cnt));
    for(i=0; i<n; ++i) cnt[ rk2[y[i]] ] ++;
    for(i=1; i<MAXRK; ++i) cnt[i]+=cnt[i-1];
    for(i=n-1; i>=0; --i) sa[ --cnt[ rk2[y[i]] ]
        ]=y[i];
    for(p=i=1, rk1[sa[0]]=0; i<n; ++i)
      rk1[sa[i]] = same(rk2, sa[i], sa[i-1], j) ?
          p-1 : p++;
    swap(rk1, rk2);
  }
  copy(rk, rk+n, rk2);
  make_height(str);
}
int main() {
  char str[LEN];
  scanf("%s", str);
  int len = strlen(str);
  suffix_array(str, len+1);
  for(int i=1; i<=len; ++i) printf("%d %d %s\n",
      i, height[i], str+sa[i]);
}
```

6.5 Suffix Tree

```
// SWERC 2015 - Text Processor
// Approach: Suffix Tree + Sliding Window. O(|S|
    + Q)
// Author: Miguel Oliveira
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
#include <string>
#include <vector>
using namespace std;

struct SuffixTree {
  struct Node {
    int l, r, par, link = -1, num_children = 0,
        next[26];

    Node(int l=0, int r=0, int par=-1) : l(l), r(
        r), par(par) {
      memset(next, -1, sizeof next);
    }
    int Length()     { return r - l;       }
    int& get(char c) { return next[c-'a']; }

    void SetEdge(char c, int node_index) {
      if (next[c-'a'] != -1 && node_index == -1)
        --num_children;
      else if (next[c-'a'] == -1 && node_index !=
          -1)
        ++num_children;
      next[c-'a'] = node_index;
    }
  };
  struct State {
    int v, pos;
    State(int v, int pos) : v(v), pos(pos)  {}
  };

  string text;
  queue<int> qleaves;
  vector<Node> tree;
  State ptr = State(0, 0);
  long long num_substrings = 0;

  SuffixTree(char* str) {
    text = string(str);
    tree.reserve(2 * text.size() + 1);
    tree.push_back(Node(0));
  }

  State Go(State st, int l, int r) {
    while (l < r) {
      if (st.pos == tree[st.v].Length()) {
        st = State(tree[st.v].get(text[l]), 0);
        if (st.v == -1)
          return st;
      } else {
        if (text[ tree[st.v].l + st.pos ] != text
            [l])
          return State(-1, -1);
        if (r-l < tree[st.v].Length() - st.pos)
          return State(st.v, st.pos + r-l);
        l += tree[st.v].Length() - st.pos;
        st.pos = tree[st.v].Length();
      }
    }
    return st;
  }

  int Split(const State& st) {
    if (st.pos == tree[st.v].Length())
      return st.v;
    if (st.pos == 0)
      return tree[st.v].par;
```

```
  const Node& v = tree[st.v];
  int id = tree.size();
  tree.push_back(Node(v.l, v.l + st.pos, v.par)
      );
  tree[v.par].SetEdge(text[v.l], id);
  tree[id].SetEdge(text[v.l + st.pos], st.v);
  tree[st.v].par = id;
  tree[st.v].l += st.pos;
  return id;
}

int GetLink(int v) {
  if (tree[v].link != -1) return tree[v].link;
  if (tree[v].par == -1)  return 0;
  int to = GetLink(tree[v].par);
  return tree[v].link = Split(Go(State(to,tree[
      to].Length()), tree[v].l + (tree[v].par
      ==0), tree[v].r));
}

void TreeExtend(int pos) {
  int mid;
  num_substrings += qleaves.size();
  do {
    State nptr = Go(ptr, pos, pos+1);
    if (nptr.v != -1) {
      ptr = nptr;
      return;
    }
    mid = Split(ptr);
    int leaf = tree.size();
    num_substrings++;         // new leaf.
    qleaves.push(leaf);
    tree.push_back(Node(pos, text.size(), mid))
        ;
    tree[mid].SetEdge(text[pos], leaf);
    ptr.v = GetLink(mid);
    ptr.pos = tree[ptr.v].Length();
  } while (mid != 0);
}

void TreeDelete(int pos) {
  int leaf = qleaves.front();
  qleaves.pop();
  int par = tree[leaf].par;
  while (tree[leaf].num_children == 0) {
    if (ptr.v != leaf) {
      tree[par].SetEdge(text[tree[leaf].l], -1)
          ;
      num_substrings -= min(tree[leaf].r, pos)
          - tree[leaf].l;
      leaf = par;
      par = tree[leaf].par;
    } else {
      if (ptr.pos == min(tree[leaf].r,pos) -
          tree[leaf].l)
        break;
      int mid = Split(ptr);
      ptr.v = mid;
      num_substrings -= min(tree[leaf].r, pos)
          - tree[leaf].l;
      tree[mid].SetEdge(text[tree[leaf].l], -1)
          ;
      tree[leaf] = tree[mid];
      tree[tree[mid].par].SetEdge(text[tree[mid
          ].l], leaf);
      tree.pop_back();
      break;
    }
  }
  if (leaf != 0 && tree[leaf].num_children ==
      0) {
    qleaves.push(leaf);
    int to = (tree[leaf].par == 0) ? 0 : tree[
        tree[leaf].par].link;
    ptr = Go(State(to, tree[to].Length()), tree
        [leaf].l + (tree[leaf].par==0), tree[
        leaf].r);
    tree[leaf].l = pos - tree[leaf].Length();
    tree[leaf].r = text.size();
  }
 }
};

int main() {
  const int MAXN = 100100;
  long long ans_window[MAXN];
  char text[MAXN];
  int w, n, q;
  scanf("%s %d %d", text, &n, &w);
  SuffixTree suffix_tree(text);
  for (int i = 1; i <= (int)suffix_tree.text.size
      (); ++i) {
    suffix_tree.TreeExtend(i-1);
    if (i >= w) {
      ans_window[i-w] = suffix_tree.
          num_substrings;
      suffix_tree.TreeDelete(i);
    }
  }
  for (int i = 0; i < n; ++i) {
    scanf("%d", &q);
    printf("%lld\n", ans_window[q-1]);
  }
  return 0;
}
```

6.6 Z Algorithm

```
void Z(char G[], int z[]){
  int len = strlen(G);
  z[0] = len;
  int L = 0, R = 1;
  for ( int i = 1 ; i < len ; i++ ) {
    if ( i >= R || z[i-L] >= R-i ) {
      int x = (i>=R) ? i : R;
      while ( x < len && G[x] == G[x-i] )
        x++;
      z[i] = x - i;
      if ( x > i )  L = i , R = x;
    }
    else z[i] = z[i-L];
  }
}
```

7 Others

7.1 8 puzzle - IDA*

```
// 一個盤面。其數值1~8代表方塊號碼，0代表空格。
int board[3][3] = {2, 3, 4, 1, 5, 0, 7, 6, 8};
// 檢查 permutation inversion。檢查不通過，表示盤
    面不合理。
bool check_permutation_inversion(int board[3][3])
{
  int inversion = 0;
  for (int a=0; a<9; ++a)
    for (int b=0; b<a; ++b) {
      int i = a / 3, j = a % 3;
      int ii = b / 3, jj = b % 3;
      if (board[i][j] && board[ii][jj]
        && board[i][j] < board[ii][jj])
        inversion++;
    }
  int row_number_of_0 = 0;
  for (int i=0; i<3 && !row_number_of_0; ++i)
    for (int j=0; j<3 && !row_number_of_0; ++j)
      if (board[i][j] == 0)
        row_number_of_0 = i+1;
  return (inversion + row_number_of_0) % 2 == 0;
}
//////////////////////////
// heuristic function, 採用不在正確位置上的方塊個
    數。
int h(int board[3][3])
{
  int cost = 0;
  for (int i=0; i<3; ++i)
    for (int j=0; j<3; ++j)
      if (board[i][j])
        if (board[i][j] != i*3 + j + 1)
          cost++;
  return cost;
}
//////////////////////////
int taxicab_distance(int x1, int y1, int x2, int
    y2)
{return abs(x1 - x2) + abs(y1 - y2);}

// heuristic function, 採用taxicab distance。
int h(int board[3][3]) {
  // 每塊方塊的正確位置。{0,0}是為了方便編寫程式
      而多加的。
  static const int right_pos[9][2] = {
    {0,0},
    {0,0}, {0,1}, {0,2},
    {1,0}, {1,1}, {1,2},
    {2,0}, {2,1}
  };
  // 計算每個方塊與其正確位置的 taxicab distance
      的總和。
  int cost = 0;
  for (int i=0; i<3; ++i)
    for (int j=0; j<3; ++j)
      if (board[i][j])
        cost += taxicab_distance(
              i, j,
              right_pos[board[i][j]][0],
              right_pos[board[i][j]][1]
            );
  return cost;
}

// 上下左右
const string operator[4] = {"up", "down", "right"
    , "left"};
const int dx[4] = {-1, 1, 0, 0}, dy[4] = {0, 0,
    1, -1};
char solution[30];
  // 正確的推動方式，其數值是方向0~3。
const int reverse_dir[4] = {1, 0, 3, 2};
  // 用表格紀錄每一個方向的反方向。可用於避免來回
      推動的判斷。

int board[3][3] = {2, 3, 4, 1, 5, 0, 7, 6, 8};
  // 起始狀態。其數值1~8代表方塊號碼，0代表空格。

int sx = 1, sy = 2;
  // 空格的位置。可馬上知道推動方塊的目的地。

bool onboard(int x, int y)
{return x>=0 && x<3 && y>=0 && y<3;}

int IDAstar(int x, int y, int gv, int prev_dir,
    int& bound, bool& ans) {
  int hv = h(board);
  if (gv + hv > bound) return gv + hv;
    // 超過，回傳下次的bound
  if (hv == 0) {ans = true; return gv;}
    // 找到最佳解

  int next_bound = 1e9;
  for (int i=0; i<4; ++i) {
    // 四種推動方向
    int nx = x + dx[i], ny = y + dy[i];
      // 空格的新位置
    if (reverse_dir[i] == prev_dir) continue;
      // 避免來回推動
    if (!onboard(nx, ny)) continue;
      // 避免出界
    solution[gv] = oper[i];
      // 紀錄推動方向
    swap(board[x][y], board[nx][ny]);
      // 推動
    int v = IDAstar(nx, ny, gv+1, i, bound, ans);
    if (ans) return v;
    next_bound = min(next_bound, v);
    swap(board[nx][ny], board[x][y]);
      // 回復原狀態
  }
  return next_bound;
}

void eight_puzzle() {
  if (!check_permutation_inversion(board)) {
    cout << "盤面不合理，無法解得答案。" << endl;
    return;
  }
  // IDA*
  bool ans = false;
  int bound = 0;
  while (!ans && bound <= 50)
    bound = IDAstar(sx, sy, 0, -1, bound, ans);
  if (!ans) {
    cout << "50步內無法解得答案。" << endl;
    return;
  }
  // 印出移動方法
  for (int i=0; i<bound; ++i)
    cout << operation[solution[i]] << ' ';
  cout << endl;
}
```

7.2 recursive to stack

replace all variable in data into layer[lay].variable

```
struct data {
```

```
  parameter;
  local variabla;
  direction;    //new
} layer[10000];
int lay=0;  //new
type reval; //new
void go() {
// at the beginning
start:
// call recursive function
  direction = 1;
  lay++, parameter = value;
  goto start;
point1:
  variable = reval;
// return
  reval = value;
  lay--;
  goto trans;
// at the end
trans:
  switch (direction) {
    case 1:
      goto point1;
  }
}
```

The End