Index

1 Enviroment Settings

1.1 .vimrc

```
" set encoding
set encoding=utf-8
set fileencodings=utf-8,big5
set showmode
syntax on
set hlsearch
set background=dark
set laststatus=2
set wildmenu
set scrolloff=5 " keep at least 5 lines above/
    below
set ruler
set cursorline
set ic    " ignore case when searching
set bs=2 " enable backspace
set number
set tabstop=4
set shiftwidth=4
set autoindent
set smarttab
set smartindent
""""""" abbr
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 '\'
imap<F9> <ESC>:w<Enter><F9>
map<F9> :!g++ "%:t" -o "%:r.out" -Wall -Wshadow -
    O2 -Im && "./%:r.out"
map<F10> :!g++ "%:t" -o "%:r.out" -Wall -Wshadow
    -O2 -Im
autocmd! BufNewFile * silent! 0r ~/.vim/skel/
    Template.%:e
```

2 Computational Geometry

2.1 Geometry on Plane

```cpp
struct node {
  double x,y;
  node() {}
  node(double _x, double _y) : 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 len2()const{ return x*x+y*y; }
  double len()const{ return sqrt(x*x+y*y); }
  node unit()const{ return *this/len(); }
  double operator^(const node& rhs)const{ return
    x*rhs.y-y*rhs.x; }
  node T()const{ return node(-y,x); }
  node rot(double rad)const{ //逆時針旋轉 弧度
    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 KDTree

```cpp
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 BigInteger

```cpp
#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;
```

```cpp
    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;
```

```cpp
    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;
}
```

3.2   Fenwick Tree Range Modify [1, size]

```cpp
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;
  }
};
```

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

```cpp
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;
}
```

```cpp
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;
}
```

3.4 Splay Tree

```cpp
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);
```

```cpp
    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;
```

3.5 Treap

```cpp
struct Node {
  Node *l,*r;
  int v,delta,rev,size,minx,w;
  void up() {
    minx = v;
    size = 1;
    if(l) size += l->size, minx = min(minx, l->
        minx);
    if(r) size += r->size, minx = min(minx, r->
        minx);
  }
  void down() {
    if(delta) {
      if(l) l->delta += delta, l->v += delta, l->
          minx += delta;
      if(r) r->delta += delta, r->v += delta, r->
          minx += delta;
      delta = 0;
    }
    if(rev) {
      swap(l,r);
```

```cpp
      if(l) l->rev ^= 1;
      if(r) r->rev ^= 1;
      rev = 0;
    }
  }
}*root = NULL, *list = NULL;
inline int sz(Node *o) { return o ? o->size : 0;
    }
int ran() {
  static int ranx = 123456789;
  ranx += (ranx<<2) + 1;
  return ranx;
}
void New_node(Node *&o, int val) {
  if(list == NULL) {
    Node *tt = new Node[100];
    for(int i = 0; i < 100; i ++) {
      tt[i].w = ran();
      tt[i].r = list;
      list = tt + i;
    }
  }
  o = list;
  list = o->r;
  o->l = o->r = NULL;
  o->v = o->minx = val;
  o->size = 1;
  o->delta = o->rev = 0;
}
void Reuse(Node *o) { if(o) { o->r = list; list =
    o; } }
void cut(Node *o, Node *&p, Node *&q, int num) {
  if(num == 0) {
    p = NULL; q = o;
  } else if(num == sz(o)) {
    p = o; q = NULL;
  } else {
    o->down();
    if(num <= sz(o->l)) {
      q = o;
      cut(o->l,p,q->l,num);
      q->up();
    } else {
      p = o;
      cut(o->r,p->r,q,num-sz(o->l)-1);
      p->up();
    }
  }
}
void merge(Node *&o, Node *p, Node *q) {
  if(!p || !q) {
    o = p ? p : q;
  } else {
    if(p->w > q->w) {
      p->down();
      o = p;
      merge(o->r,p->r,q);
    } else {
      q->down();
      o = q;
      merge(o->l,p,q->l);
    }
    o->up();
  }
}
void insert(Node *&o, int pos, int val) {
  if(o == NULL) {
    New_node(o,val);
  } else {
    Node *l , *r , *n;
```

```cpp
    New_node( n , val );
    cut ( o , l , r , pos );
    merge( l , l , n );
    merge( root , l , r );
  }
}
void add(int l, int r, int val) {
  Node *a, *b, *c;
  cut(root,a,b,l-1);
  cut(b,b,c,r-l+1);
  b->v += val;
  b->minx += val;
  b->delta += val;
  merge(a,a,b);
  merge(root,a,c);
}
void remove(int pos) {
  Node *a, *b, *c;
  cut(root,a,b,pos-1);
  cut(b,b,c,1);
  merge(root,a,c);
  Reuse(b);
}
int query(int l, int r) {
  Node *a, *b, *c;
  cut(root,a,b,l-1);
  cut(b,b,c,r-l+1);
  int ret = b->minx;
  merge(a,a,b);
  merge(root,a,c);
  return ret;
}
void reverse(int l, int r) {
  Node *a, *b, *c;
  cut(root,a,b,l-1);
  cut(b,b,c,r-l+1);
  b->rev ^= 1;
  merge(a,a,b);
  merge(root,a,c);
}
void revolve(int l, int m, int r) {
  Node *a, *b, *c, *d;
  cut(root,a,b,l-1);
  cut(b,b,c,m-l+1);
  cut(c,c,d,r-m);
  merge(a,a,c);
  merge(a,a,b);
  merge(root,a,d);
}
```

3.6 劃分樹

```cpp
#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 Dinic

```
class Flow{
public:
  Flow(int _ncnt) :ncnt(_ncnt), ecnt(1), path(new
      int[_ncnt + 2]), d(new int[_ncnt + 2]),
      visited(new bool[_ncnt + 2]){
    memset(path, 0, sizeof(int)*(_ncnt + 1));
  }
  ~Flow(){
    delete[](path);
    delete[](d);
    delete[](visited);
  }

  void Reset(){
    memset(path, 0, sizeof(int)*(ncnt + 1));
    ecnt = 1;
  }
  void AddEdge(int s, int t, int cap){
    edge[++ecnt].tar = t, edge[ecnt].cap = cap,
      edge[ecnt].next = path[s], path[s] = ecnt
      ;
    edge[++ecnt].tar = s, edge[ecnt].cap = 0,
      edge[ecnt].next = path[t], path[t] = ecnt
      ;

  }

  int MaxFlow(int s, int t){ // Dinic

    int f = 0, df;
    while (BFS(s, t) < ncnt){
      while (true){
        memset(visited, 0, sizeof(bool)*(ncnt +
            1));
        df = DFS(s, INF, t);
        if (!df) break;
        f += df;
      }
    }
    return f;
  }

private:
  static const int eMaxSize = 40002, INF = (int)
      1e9;
  int ecnt, ncnt;
  int *path, *d; // d for Dicic distance
  bool *visited;

  struct Edge{
    int tar, cap, next;
  }edge[eMaxSize];

  int DFS(int a, int df, int t){
    if (a == t) return df;
    if (visited[a]) return 0;
    visited[a] = true;
    for (int i = path[a]; i; i = edge[i].next){
      int b = edge[i].tar;
      if (edge[i].cap > 0 && d[b] == d[a] + 1){
        int f = DFS(b, std::min(df, edge[i].cap),
            t);
        if (f){
          edge[i].cap -= f, edge[i ^ 1].cap += f;
          return f;
        }
      }
```

```
        }
        return 0;
    }

    int BFS(int s, int t){
        memset(d, 0x7f, sizeof(int)*(ncnt + 1));
        memset(visited, 0, sizeof(bool)*(ncnt + 1));
        d[s] = 0; visited[s] = true;
        std::queue<int> Q;
        Q.push(s);
        while (!Q.empty()){
            int a = Q.front(); Q.pop();
            for (int i = path[a]; i; i = edge[i].next){
                int b = edge[i].tar;
                if (visited[b] || edge[i].cap == 0)
                    continue;
                visited[b] = true;
                d[b] = d[a] + 1;
                if (b == t) return d[b];
                Q.push(b);
            }
        }
        return d[t];
    }
};
Flow flow( 1001 );
```

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

5.5 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 a, ll n) {
  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])==false )
      return false;
  return true;
```

```
}
```

5.6 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 KMP

```cpp
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;
}
```

6.2 Longest Palindromic Substring

```cpp
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)
```

```cpp
    if (i & 1) cout << s[i];
}
```

6.3 Suffix Array

```cpp
int rank[LEN], sa[LEN];
int height[LEN];
int y[LEN], cnt[LEN], rr[2][LEN];
inline bool same(int *rank, int a, int b, int l)
  { return rank[a]==rank[b]&&rank[a+1]==rank[b+
  1]; }
void sa2(char str[], int n, int m) {
  printf("%s!! %d %d\n", str, n, m);
  int *rank1=rr[0], *rank2=rr[1];
  MSET(rr[1], 0);
  int i, p;
  for(i=0; i<m; ++i) cnt[i]=0;
  for(i=0; i<n; ++i) rank2[i]=str[i], cnt[rank2[i
    ]]++;
  for(i=1; i<m; ++i) cnt[i]+=cnt[i-1];
  for(i=n-1; i>=0; --i) sa[--cnt[rank2[i]]]=i;
  for(int j=1; p<n; j<<=1, m=p) {
    // 表示用第二個key(rank2)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;
    for(i=0; i<m; ++i) cnt[i]=0;
    for(i=0; i<n; ++i) cnt[ rank2[y[i]] ] ++;
    for(i=1; i<m; ++i) cnt[i]+=cnt[i-1];
    for(i=n-1; i>=0; --i) sa[ --cnt[ rank2[y[i]]
      ] ]=y[i];
    for(p=i=1, rank1[sa[0]]=0; i<n; ++i)
      rank1[sa[i]]=same(rank2, sa[i], sa[i-1], j)
        ?p-1:p++;
    std::swap(rank1, rank2);
  }
  for(int i=0; i<n; ++i) rank[i]=rank2[i];
}
void make_height(char str[]) {
  int len=strlen(str);
  height[0]=0;
  for(int i=0, j=0; i<len; ++i, j=height[rank[i
    -1]]-1) {
    if( rank[i]==1 ) continue;
    if( j<0 ) j=0;
    while( i+j<len && sa[rank[i]-1]+j<len &&
      str[i+j]==str[sa[rank[i]-1]+j] ) ++j;
    height[rank[i]]=j;
  }
}
int main() {
  char str[LEN];
  scanf("%s", str);
  int len = strlen(str);
  sa2(str, len+1, 256);
  make_height(str);
  for(int i=1; i<=len; ++i) printf("%d %d %s\n",
    i, height[i], str+sa[i]);
}
```

6.4 Z Algorithm

```cpp
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"};
```

```cpp
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

```cpp
struct data {
```

```cpp
   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