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
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
```

2 Data Structure

2.1 Fenwick Tree [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;
  }
};
```

2.2 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;
}
```

2.3 Heap

```
// max heap tree
#define ParentIndex(i) i==0 ? 0 : ((i-1) >> 1)
#define LeftChildIndex(i) ((i)<<1)+1
#define RightChildIndex(i) ((i)<<1)+2
void BuildMaxHeap(int*, const int&);
void MaxHeapBalance(int*, const int&, const int&)
    ;
void MaxHeapDelete(int*, int&);
inline bool comp(int &a, int &b) {return a>b;}
void BuildMaxHeap(int all[], const int &size) {
  for(int i=(size-1) >> 1; i>=0; i--)
    MaxHeapBalance(all, size, i);
}
void MaxHeapBalance(int all[], const int &size,
    const int &root) {
  int aim = root, aim2;
  while(1) {
    aim2 = aim;
    int L = LeftChildIndex(aim2);
    int R = RightChildIndex(aim2);
    if( L < size && comp( all[aim], all[L] ) )
       aim = L;
    if( R < size && comp( all[aim], all[R] ) )
       aim = R;
    if( aim != aim2 ) swap(all[aim], all[aim2]);
    else return;
  }
}
void MaxHeapAdd(int all[], int &size, const int &
    AddNum) {
  all[size] = AddNum;
  ++size;
  int P, index = size-1;
  while( index != 0 ) {
    P = ParentIndex(index);
    if( comp(all[P], all[index]) ) {
      swap(all[P], all[index]);
      index = P;
    }
    else return;
  }
}
void MaxHeapDelete(int all[], int &size) {
  all[0] = all[size-1], --size;
  MaxHeapBalance(all, size, 0);
}
```

2.4 Deap

```
class deap {
public:
  deap() {size=1;}
  ~deap() {}
  inline void insert(int n) {
    arr[++size]=n;
    int now=size;
    if( (now&1) && arr[now-1]>arr[now] )
      swap(arr[now-1], arr[now]), now--;
    while( now>3 ) {
      if( arr[now>>2<<1]>arr[now] )
        swap(arr[now>>2<<1], arr[now]),
        now=now>>2<<1;
      else if( arr[(now>>2<<1)+1]<arr[now] )
        swap(arr[(now>>2<<1)+1], arr[now]),
        now=(now>>2<<1)+1;
      else break;
    }
  }
  inline int min() {
    int res=arr[2];
    swap(arr[2], arr[size--]), down(2);
    return res;
  }
  inline int max() {
    int res=arr[3];
    swap(arr[3], arr[size--]), down(3);
    return res;
  }
private:
  int arr[1000005], size;
  inline void down(int now) {
    while( (now<<1)<=size ) {
      int tmp;
      if( (now&1)==0 ) {
        if( arr[now]>arr[now+1] )
          {swap(arr[now], arr[now+1]);
          now++;continue;}
        tmp=now;
        if( arr[tmp]>arr[now<<1] )
          tmp=now<<1;
        if( (now<<1)+2<=size && arr[tmp]>arr[(now
            <<1)+2] ) tmp=(now<<1)+2;
        if( tmp==now ) break;
        else swap(arr[now], arr[tmp]),
          now=tmp;
      }
      else if( (now&1)==1 ) {
        if( arr[now]<arr[now-1] )
          {swap(arr[now], arr[now-1]);
          now--;continue;}
        tmp=now;
        if( arr[tmp]<arr[(now<<1)-1] )
          tmp=(now<<1)-1;
        if( (now<<1)+1<=size && arr[tmp]<arr[(now
            <<1)+1] ) tmp=(now<<1)+1;
        if( tmp==now ) break;
        else swap(arr[now], arr[tmp]), now=tmp;
      }
    }
    if( (now&1)==0 && now+1<=size && arr[now]>arr
        [now+1] )
      swap(arr[now], arr[now+1]);
    if( (now&1)==1 && arr[now]<arr[now-1] )
      swap(arr[now], arr[now-1]);
  }
};
```

2.5 zkw Segment Tree (range modify and query)

```
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, 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);
```

2.6 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;
}
```

2.7 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 Graph

3.1 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 );
```

3.2 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;
}
```

4 String

4.1 KMP

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

4.2 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];
  }
}
```

4.3 Suffix Array

```
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+l]==rank[b+
    l]; }
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]);
}
```

4.4 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];
}
```

5 Math

5.1 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]];
  }
}
```

5.2 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;
}
```

5.3 Möbius function

```
memset(mobius, 0, sizeof(mobius));
mobius[1]=1;
for(int i=0; i<flag; ++i) mobius[prime[i]]=-1;
for(int i=2, tmp, cntprime; i<MAXN; ++i)
{
  if( !~mobius[i] ) continue;
  tmp=i, cntprime=0;
  for(int j=0; !mobius[tmp]&&prime[j]<=tmp; ++j){
    if( tmp%prime[j]==0 )
      ++cntprime, tmp/=prime[j];
    if( tmp%prime[j]==0 ) {cntprime=0;break;}
  }
  if( cntprime && mobius[tmp] )
    mobius[i]=mobius[tmp]*(cntprime&1?-1:1);
}
```

5.4 China remainder theorem

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

```
int ans, 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 ) {
    ans=-1;
    break;
  }
  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;
}
```

6 Others

6.1 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;
}
```

The End