Московский авиационный институт
(национальный исследовательский университет)

Факультет «Прикладная математика и физика»
Кафедра «Вычислительная математика и программирование»

Курсовой проект по курсу дискретной математики

Выполнил: Забарин Н.И. 80-08Б
Руководитель: к. ф.-м.н.с. доцент Алексеев Н.С.

Москва, 2016

Введение

В качестве курсового проекта предлагается задачи по различным разделам дискретной математики, в частности по теории графов и комбинаторике, решенные автором на официальных соревнованиях-олимпиадах и сборах по программированию. Все приведенные ниже задачи были проверенны на множестве тестов.

Задача 1

Условие:

Посчитать количество инверсий в массиве.

Метод решения:

Основным алгоритмом применяемым в задаче является дерево Фенвика — позволяет быстро(за логарифм от длины) считать заданную обратимую функцию на префиксе массива, с помощью логических функций И и ИЛИ. Структура представляет собой дерево, где у каждой вершины есть единственный предок и не более двух потомков, родитель хранит в себе результат применения заданной функции к значениям хранящимся в потомках.

Построим дерево Фенвика, в котором будем хранить кол-во элементов массива которые меньше текущего. Будем по очереди добавлять в него элементы массива и сразу считать сколько чисел, которые больше текущего мы уже встретили, просуммировав это мы и получим ответ.

Асимптотика:

На каждом шаге мы делаем порядка логарифм N операций, всего N шагов, получаем сложность в O(NlogN).

Листинг:

```
program d02;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  Math;

var
  mas, d, used : array [0..1000001] of integer;
  n, i : integer;

procedure up (var x : integer);
begin
  x := x or (x + 1);
end;

procedure down (var x : integer);
begin
  x := (x and (x + 1)) — 1;
end;
procedure UpDate(i, x : integer);
begin
  while i < 1000000 do begin
    inc(used[i], x);
    up(i);
  end;
end;

function Sum (r : integer) : int64;
begin
  result := 0;
  while r >= 0 do begin
    inc(result, used[r]);
    down(r);
  end;
end;

begin
```

```
  reset(input, 'inverse.in');
  rewrite(output, 'inverse.out');

  fillchar(used, sizeof(used), 0);
  fillchar(d, sizeof(d), 0);
  fillchar(mas, sizeof(mas), 0);

  read(n);

  for i := 0 to n - 1 do
    read(mas[i]);

  for i := 0 to n - 1 do begin
    UpDate(mas[i], 1);
    d[mas[i]] := sum(1000000) - sum(mas[i]);
  end;
  n := 0;
  for i := 0 to 1000001 do inc(n, d[i]);
  write(n);
end.
```

Задача 2

Условие:

Назовем мега-инверсией в перестановке P такую тройку (i, j, k), что Pi > Pj > Pk и i < j < k. Написать алгоритм для быстрого подсчета количества мега-инверсий в данной перестановке.

Метод решения:

Основным алгоритмом применяемым в задаче является дерево отрезков(или RMQ) — позволяет быстро(за логарифм от длины) считать заданную обратимую функцию на префиксе массива, с помощью метода «Разделяй и Властвуй». Структура представляет собой дерево, где у каждой вершины есть единственный предок и не более двух потомков, родитель хранит в себе результат применения заданной функции к значениям хранящимся в потомках. В итоге в каждой вершине хранится результат действия функции для определенного подмассива исходного массива.

Построим три RMQ: первое для указания какие числа уже есть в перестановке, второе для подсчета инверсий, третье для подсчета мега инверсий. Будем последовательно обновлять эти три дерева. Для получаения ответа просуммируем все последнее дерево.

Асимптотика:

Сложность алгоритма O(NlogN)

Листинг:

```
program a01;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  Math;

type
  TElem = record
    value : int64;
    l, r : integer;
  end;
  TTree = array [1..300000] of TElem;

var
  used, inv, minv : TTree;
  mas : array [1..200000] of integer;
  n, i : integer;

function func (a, b : int64) : int64;
begin
  result := a + b;
end;

procedure Build(var tree : TTree; l, r, id : integer);
begin
  tree[id].l := l;
  tree[id].r := r;
  if r - l = 1 then
    tree[id].value := mas[l]
  else begin
    Build(tree, l, (r + l) div 2, id * 2);
    Build(tree, (r + l) div 2, r, id * 2 + 1);
    tree[id].value := func(tree[id * 2].value, tree[id * 2].value);
  end;
end;
```

```
procedure UpDate(var tree : TTree; id, n : integer; v : int64);
begin
  if (n >= tree[id].r) or (n < tree[id].l) then exit;
  if tree[id].r - tree[id].l = 1 then
    tree[id].value := v
  else begin
    Update(tree, id * 2, n, v);
    Update(tree, id * 2 + 1, n, v);
    tree[id].value := func(tree[id * 2].value, tree[id * 2 + 1].value);
  end;
end;
function Proc(var tree : TTree; l, r, id : integer) : int64;
begin
  if (tree[id].l >= r) or (tree[id].r <= l) then begin
    result := 0;
    exit;
  end;
  if (l <= tree[id].l) and (tree[id].r <= r) then begin
    result := tree[id].value;
    exit;
  end;
  result := func(Proc(tree, l, r, id * 2), Proc(tree, l, r, id * 2 + 1));
end;

begin
  reset(input, 'mega.in');
  rewrite(output, 'mega.out');

  fillchar(used, sizeof(used), 0);
  fillchar(inv, sizeof(inv), 0);
  fillchar(minv, sizeof(minv), 0);
  fillchar(mas, sizeof(mas), 0);

  readln(n);

  Build(used, 0, n, 1);
  Build(inv, 0, n, 1);
  Build(minv, 0, n, 1);
  for i := 0 to n - 1 do
    readln(mas[i]);
  for i := 0 to n - 1 do begin
    UpDate(used, 1, mas[i] - 1, 1);
    UpDate(inv, 1, mas[i] - 1, Proc(used, mas[i], n, 1));
    UpDate(minv, 1, mas[i] - 1, Proc(inv, mas[i], n, 1));
  end;

  write(Proc(minv, 0, n, 1));
end.
```

Задача 3

Условие:

Реализовать сбалансированное дерево поиска, поддерживающее следующие команды: добавление элемента, удаление элемента, проверка на существование элемента, следующий/предыдущий по возрастанию элемент после данного, k-тый по возрастанию элемент.

Метод решения:

Основным алгоритмом применяемым в задаче является Декартово дерево со случайным ключом. Все необходимые операции производятся с помощью функций разрезания дерева на два и сливания двух деревьев. Случайный же ключ гарантирует, что все операции будут производиться с логарифмической сложностью.

Асимптотика:

Сложность алгоритма O(NlogN)

Листинг:

```
program b01;

{$APPTYPE CONSOLE}

uses
  SysUtils;

const
  num = 100000;

type
  TElem = record
    x, y : integer;
    l, r : integer;
  end;
  TPair = record
    a, b : integer;
  end;

var
  stack, sz : array [0..num] of integer;
  r, t, i, n, q : integer;
  a : array [1..num] of TElem;
  s : string;
  tmp : TPair;

function cnt(t : integer): integer;
begin
 result := 0;
 if t = -1 then exit;
 result := sz[t];
end;

procedure update(t : integer);
begin
 if t = -1 then exit;
 sz[t] := cnt(a[t].l) + cnt(a[t].r) + 1;
end;

procedure push(a : integer);
begin
  inc(r);
  stack[r] := a;
end;

function new : integer;
begin
  result := stack[r];
  dec(r);
```

```
end;

function merge (t1, t2 : integer) : integer;
begin
  result := t2;
  if t1 = -1 then exit;
  result := t1;
  if t2 = -1 then exit;
  if (a[t1].y > a[t2].y) then begin
    a[t1].r := merge(a[t1].r, t2);
    update(t1);
    result := t1;
  end
  else begin
    a[t2].l := merge(t1, a[t2].l);
    update(t2);
    result := t2;
  end
end;

function split (t, k : integer): TPair;
var
  tmp : TPair;
begin
  result.a := -1;
  result.b := -1;
  if t = -1 then exit;
  if k > a[t].x then begin
    tmp := split(a[t].r, k);
    a[t].r := tmp.a;
    update(t);
    update(tmp.b);
    result.a := t;
    result.b := tmp.b;
  end
  else begin
    tmp := split(a[t].l, k);
    a[t].l := tmp.b;
    update(tmp.a);
    update(t);
    result.a := tmp.a;
    result.b := t;
  end;
end;

function exist (t, k: integer) : boolean;
begin
  result := false;
  if t = -1 then exit;
  result := true;
  if a[t].x = k then exit;
  if a[t].x > k then begin
    result := exist(a[t].l, k);
    exit;
  end;
  if a[t].x < k then begin
    result := exist(a[t].r, k);
    exit;
  end;
end;

function insert (t, k : integer) : integer;
var
  p : integer;
  tmp : TPair;
begin
  result := t;
  if exist(t, k) then exit;
  p := new;
  a[p].x := k;
  a[p].y := random(100000000);
  a[p].l := -1;
  a[p].r := -1;
  tmp := split(t, k);
  tmp.a := merge(tmp.a, p);
  result := merge(tmp.a, tmp.b);
end;

function delete (t, k : integer): integer;
```

```
begin
  result := -1;
  if t = -1 then exit;
  if a[t].x < k then begin
    a[t].r := Delete(a[t].r, k);
    result := t;
    exit;
  end;
  if a[t].x > k then begin
    a[t].l := delete(a[t].l, k);
    result := t;
    exit;
  end;
  result := merge(a[t].l, a[t].r);
end;



procedure exists (t, k: integer);
begin
 if exist(t, k) then
   writeln('true')
 else
   writeln('false');
end;

function next (t, key : integer): TPair;
var
  tmp : TPair;
  v : integer;
begin
  tmp := split(t, key + 1);
  v := tmp.b;
  result.b := v;
  result.a := tmp.a;
  if v = -1 then exit;
  while a[v].l <> -1 do
    v := a[v].l;
  tmp.a := merge(tmp.a, tmp.b);
  tmp.b := v;
  result :=  tmp;
end;

function prev (t, key : integer) : TPair;
var
  tmp : TPair;
  v : integer;
begin
  tmp := split(t, key);
  v := tmp.a;
  result.a := t;
  result.b := v;
  if v = -1 then exit;
  while a[v].r <> -1 do
    v := a[v].r;
  result.a := merge(tmp.a, tmp.b);
  result.b := v;
end;

function kth (t, key : integer): integer;
begin
  result := -1;
  if t = -1 then exit;

  result := -1;
  if cnt(t) < key then exit;

  if key <= cnt(a[t].l) then begin
    result := kth(a[t].l, key);
    exit;
  end;

  if key = cnt(a[t].l) + 1 then begin
    result := a[t].x;
    exit;
  end;

  result := kth(a[t].r, key - cnt(a[t].l) - 1);
end;
```

```
begin
  reset(input, 'bst2.in');
  rewrite(output, 'bst2.out');
  Randomize;
  fillchar(stack, sizeof(stack), 0);
  fillchar(a, sizeof(a), -1);
  r := 0;
  for i:= num downto 1 do push(i);
  t := -1;
  while not(eof(input)) do begin
    readln(s);
    if s[1] = 'i' then begin
      n := strtoint(copy(s, 8, 10));
      t := insert(t, n);
    end;
    if s[1] = 'e' then begin
      n := strtoint(copy(s, 8, 10));
      exists(t, n);
    end;
    if s[1] = 'd' then begin
      n := strtoint(copy(s, 8, 10));
      t := delete(t, n);
    end;
    if s[1] = 'n' then begin
      n := strtoint(copy(s, 6, 10));
      tmp := next(t, n);
      t := tmp.a;
      if  tmp.b = -1 then
        writeln('none')
      else
        writeln(a[tmp.b].x);
    end;
    if s[1] = 'p' then begin
      n := strtoint(copy(s, 6, 10));
      tmp := prev(t, n);
      t := tmp.a;
      if  tmp.b = -1 then
        writeln('none')
      else
        writeln(a[tmp.b].x);
    end;
    if s[1] = 'k' then begin
      n := strtoint(copy(s, 5, 10));
      q := kth(t, n);
      if  q = -1 then
        writeln('none')
      else
        writeln(q);
    end;
  end;
  close(input);
  close(output);
end.
```

Задача 4

Условие:

Дано N солдат, пронумерованных от 1 до N. Капрал любит давать приказы вида «Вперед с i по j», солтады стоящие на данном промежутке должны сохраняя порадок переместиться в начало шеренги. Дан список команд, необходимо вычислить конечное положение солдат.

Метод решения:

Основным алгоритмом применяемым в задаче является Декартово дерево со случайным ключом. Команда вперед будет выполняться с помощью разрезания дерева на три части и сливания их в нужном порядке.

Асимптотика:

Сложность алгоритма O(NlogN)

Листинг:

```
program c01;

{$APPTYPE CONSOLE}

uses
  SysUtils;

const
  num = 100000;

type
  TElem = record
    x, y : integer;
    l, r : integer;
  end;
  TPair = record
    a, b : integer;
  end;

var
  used : array [0..num] of boolean;
  stack, sz : array [0..num] of integer;
  r, t, i, n, q, m, aa, b : integer;
  a : array [1..num] of TElem;
  s : string;
  tmp : TPair;

function cnt(t : integer): integer;
begin
 result := 0;
 if t = -1 then exit;
 result := sz[t];
end;

procedure update(t : integer);
begin
 if t = -1 then exit;
 sz[t] := cnt(a[t].l) + cnt(a[t].r) + 1;
end;

procedure push(a : integer);
begin
  inc(r);
  stack[r] := a;
end;

function new : integer;
begin
  result := stack[r];
  dec(r);
end;
```

```
function merge (t1, t2 : integer) : integer;
begin
  result := t2;
  if t1 = -1 then exit;
  result := t1;
  if t2 = -1 then exit;
  if (a[t1].y > a[t2].y) then begin
    a[t1].r := merge(a[t1].r, t2);
    update(t1);
    result := t1;
  end
  else begin
    a[t2].l := merge(t1, a[t2].l);
    update(t2);
    result := t2;
  end
end;

function split (t, k : integer): TPair;
var
  tmp : TPair;
begin
  result.a := -1;
  result.b := -1;
  if t = -1 then exit;
  if k > cnt(a[t].l) then begin
    tmp := split(a[t].r, k - cnt(a[t].l) - 1);
    a[t].r := tmp.a;
    update(t);
    update(tmp.b);
    result.a := t;
    result.b := tmp.b;
  end
  else begin
    tmp := split(a[t].l, k);
    a[t].l := tmp.b;
    update(tmp.a);
    update(t);
    result.a := tmp.a;
    result.b := t;
  end;
end;

function exist (t, k: integer) : boolean;
begin
  result := false;
  if t = -1 then exit;
  result := true;
  if a[t].x = k then exit;
  if a[t].x > k then begin
    result := exist(a[t].l, k);
    exit;
  end;
  if a[t].x < k then begin
    result := exist(a[t].r, k);
    exit;
  end;
end;

function insert (t, k : integer) : integer;
var
  p : integer;
  tmp : TPair;
begin
  result := t;
  if exist(t, k) then exit;
  p := new;
  a[p].x := k;
  a[p].y := random(100000000);
  a[p].l := -1;
  a[p].r := -1;
  tmp := split(t, k);
  tmp.a := merge(tmp.a, p);
  result := merge(tmp.a, tmp.b);
end;

function delete (t, k : integer): integer;
begin
  result := -1;
```

```
  if t = -1 then exit;
  if a[t].x < k then begin
    a[t].r := Delete(a[t].r, k);
    result := t;
    exit;
  end;
  if a[t].x > k then begin
    a[t].l := delete(a[t].l, k);
    result := t;
    exit;
  end;
  result := merge(a[t].l, a[t].r);
end;



procedure exists (t, k: integer);
begin
 if exist(t, k) then
   writeln('true')
 else
   writeln('false');
end;

function next (t, key : integer): TPair;
var
  tmp : TPair;
  v : integer;
begin
  tmp := split(t, key + 1);
  v := tmp.b;
  result.b := v;
  result.a := tmp.a;
  if v = -1 then exit;
  while a[v].l <> -1 do
    v := a[v].l;
  tmp.a := merge(tmp.a, tmp.b);
  tmp.b := v;
  result :=  tmp;
end;

function prev (t, key : integer) : TPair;
var
  tmp : TPair;
  v : integer;
begin
  tmp := split(t, key);
  v := tmp.a;
  result.a := t;
  result.b := v;
  if v = -1 then exit;
  while a[v].r <> -1 do
    v := a[v].r;
  result.a := merge(tmp.a, tmp.b);
  result.b := v;
end;

function kth (t, key : integer): integer;
begin
  result := -1;
  if t = -1 then exit;

  result := -1;
  if cnt(t) < key then exit;

  if key <= cnt(a[t].l) then begin
    result := kth(a[t].l, key);
    exit;
  end;

  if key = cnt(a[t].l) + 1 then begin
    result := a[t].x;
    exit;
  end;

  result := kth(a[t].r, key - cnt(a[t].l) - 1);
end;

procedure outputt(t, v: integer);
```

```
begin
 if v = -1 then exit;
 if used[v] then begin
   used[v] := false;
   outputt(t, a[v].l);
 end;
 if not(used[v]) then begin
   write(v, ' ');
   outputt(t, a[v].r);
 end;
end;

function move(var t, l, r : integer): integer;
var
  a, b : TPair;
begin
  a := split(t, r);
  b := split(a.a, l - 1);
  t := merge(b.a, a.b);
  t := merge(b.b, t);
  result := t;
end;

begin
  reset(input, 'movetofront.in');
  rewrite(output, 'movetofront.out');
  Randomize;
  fillchar(stack, sizeof(stack), 0);
  fillchar(a, sizeof(a), -1);
  fillchar(used, sizeof(used), True);
  read(n, m);
  r := 0;
  for i:= num downto 1 do push(i);
  t := -1;
  for i := 1 to n do
    t := insert(t, i);
  for i:=1 to m do begin
    read(aa, b);
    t := move(t, aa, b);
  end;
  outputt(t, t);
  close(input);
  close(output);
end.
```

Задача 5

Условие:

Необходимо посчитать количество чисел сумма цифр на интервале [l, r] которых кратна k.

Метод решения:

Будем считать количество чисел кратных k и меньших чем n, с помощью квадратной динамики по длине числа и текущему остатку от деления на k. Что бы получить ответ посчитаем для левой и правой границы и вычтем одно из другого. Переходы в динамике осуществляются ко комбинаторной формуле.

Асимптотика:

Сложность алгоритма O(LK), где L это длина числа.

Листинг:

```
np = open('coolnumbers.in', 'r').read().split('\n')
incount = 0
out = open('coolnumbers.out', 'w')

def input ():
    global incount
    incount += 1
    return inp[incount - 1]


l, r, k = input().split()
k = int(k)

def sum (n, flag):
    d = [[[0 for i in range(2)] for j in range(k)] for q in range(len(n) + 1)]
    for i in range(int(n[0])):
        d[1][i % k][1] += 1
    d[1][int(n[0]) % k][0] += 1
    for l in range(len(n)):
        for r in range(k):
            for i in range(10):
                d[l + 1][(i + r) % k][1] += d[l][r % k][1]
            for i in range(int(n[l])):
                d[l + 1][(i + r) % k][1] += d[l][r % k][0]
            d[l + 1][(int(n[l]) + r) % k][0] += d[l][r % k][0]
    if flag:
        return d[len(n)][0][1] + d[len(n)][0][0]
    else:
        return d[len(n)][0][1]

out.write(str((sum(r, True) - sum(l, False))))
```

Задача 6

Условие:

Необходимо найти все точки сочленения в неориентированном графе.

Метод решения:

Из случайной вершины(корня) запустим ДФС который будет считать времена входа и выхода для всех вершин, и если для какого то ребра из V1 в V2 время выхода из V2 >= времени входа в V1, то V1 является точкой сочленения.

Если со временем входа все понятно, это номер итерации алгоритма на котором мы первый раз посетили вершину, то время выхода это минимум из времени входа в вершину и ее потомков и времен выхода из потомков.

Листинг:

```
program c01;

{$APPTYPE CONSOLE}

uses
  SysUtils, Math;

const
  inf = 1000000000;

type
  dmas = record
    s, f, len : integer;
  end;
  TElem = record
    key, id, next : integer;
  end;

var
  mas : array [0..500000] of TElem;
  m, num : array[1..100000] of integer;
  n, k, a, b, i, c, T, count, v0 : integer;
  time_in, up, color : array [1..100000] of integer;
  ans : array[-1..500000] of boolean;

procedure add (const x, id : integer; var a : integer);
begin
  //mas[a].next := c;
  mas[c].next := a;
  mas[c].id := id;
  mas[c].key := x;
  a := c;
  inc(c);
end;

procedure dfs (v, q : integer);
var
  w, vto, vfrom : integer;
begin
  time_in[v] := T;
  inc(T);
  up[v] := time_in[v];
  color[v] := 1;



  w := m[v];
  while w <> -1 do begin
    vto := mas[w].key;
    vfrom := mas[w].id;
    if color[vto] = 0 then begin
      if v = v0 then inc(num[v]);
      dfs(vto, vfrom);
      up[v] := min(up[v], up[vto]);
      if (up[vto] >= time_in[v]) and (v <> v0) then ans[v] := true;
    end
```

```
    else if vfrom <> q then
      up[v] := min(up[v], time_in[vto]);
    w := mas[w].next;
  end;
end;


begin
  reset(input, 'points.in');
  rewrite(output, 'points.out');
  readln(n, k);
  c := 1;
  for i:= 1 to n do
    m[i] := -1;
  for i := 1 to k do begin
    readln(a, b);
    add(b, i, m[a]);
    add(a, i, m[b]);
  end;
  fillchar(time_in, sizeof(time_in), 0);
  fillchar(color, sizeof(color), 0);
  fillchar(up, sizeof(up), 0);
  T := 0;
  for i := 1 to n do begin
    if color[i] = 0 then begin
      v0 := i;
      dfs(v0, 0);
    end;
    if num[v0] > 1 then ans[v0] := true;
  end;
  count := 0;
  for i := 1 to k do
    if ans[i] then inc(count);
  writeln(count);
  for i := 1 to k do
    if ans[i] then write(i, ' ');
end.
```

Задача 7

Условие:

Дано множество точек на плоскости, необходимо соединить их, так что бы сумма длин отрезков была минимальной и из любой точки можно было попасть с любую другую.

Метод решения:

Перефразировав задачу получим, что нам необходимо из полного графа найти связный граф минимального веса.

Будем действовать жадно, на каждом шаге будем искать точку наиболее удаленную от всех уже помеченых точек, пересчитываем расстояния до всех непомеченных точек из этой и помечаем ее.

Листинг:

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

const long double inf = 1000000000;

vector<int> used;
vector<long double> way;
vector<pair<int, int> > v;
int n, x, y;

long double sqr(int a) {
	return a * a;
};
long double dist(int a, int b) {
	return sqrt((long double)(sqr(v[a].first - v[b].first) + sqr(v[a].second - v[b].second)));
};

int main()
{
	freopen("unionday.in", "r", stdin);
	freopen("unionday.out", "w", stdout);

	cin >> n;

	used.resize(n, 0);
	way.resize(n, inf);
	way[0] = 0;

	for (int i = 0; i < n; i++){
		cin >> x >> y;
		v.push_back(make_pair(x, y));
	}

	int tmpv = 0;
	for (int i = 0; i < n; i ++){
		tmpv = -1;
		for (int j = 0; j < n; j ++){
			if (((tmpv == -1) || (way[tmpv] > way[j])) && !used[j]) {
				tmpv = j;
			};
		};
		used[tmpv] = 1;
		for (int j = 0; j < n; j ++){
			if ((j != tmpv) && (!used[j])) {
				way[j] = min(way[j], dist(j, tmpv));
			};
		};
	};
	long double ans = 0;
```

```
	for (int i = 0; i < n; i++) {
		ans += way[i];
	};
	cout.precision(20);
	cout << ans;
}
```

Задача 8

Условие:

Дан неориентированный граф, необходимо поддерживать два вида запросов: удаление ребра из графа, и проверка на принадлежность к одному компоненту связности двух вершин.

Метод решения:

Главная идея решения — для каждой вершины выбрать «начальника» другую вершину их этой же компоненты, при этом эти связи не должны образовывать циклы. Что бы определить принадлежат ли две вершины одной компоненте, нужно просто рекурсивно пройтись до самого «главного начальника», если они совпадают, то ответ положительный, иначе нет.

Что бы не искать «начальников» в исходном графе, мы будем идти с конца и добавлять ребра в пустой граф, в пустом графе каждый сам себе «начальник»

При добавлении ребра между двумя вершинами достаточно сделать «главного начальника» одной вершины «начальником» «главного начальника» другой вершины.

Листинг:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
#include <cmath>
#include <stack>
#include <string>
using namespace std;

const int inf = 1000000000;

struct pts{
	int x, y;
};

int n, m, k, a, b;
string s;
vector<int> king, r;
vector<bool> ans;
vector<pair<string, pts> > cmd;

int get (int v){
	if (king[v] == v) {
		return v;
	};
	return get(king[v]);
};

void un (int a, int b){
	int t1 = get(a);
	int t2 = get(b);

	if (r[t1] <= r[t2]){
		swap(t1, t2);
	};
	king[t2] = t1;
	r[t1] = max(king[t1], king[t2] + 1);
};

void ask (int a, int b, int i){
	ans[i] = (get(a) == get(b));
};


int main()
{
```

```
	freopen("cutting.in", "r", stdin);
	freopen("cutting.out", "w", stdout);

	cin >> n >> m >> k;

	king.resize(n);
	ans.resize(k);
	r.resize(n);

	for (int i = 0; i < n; i++){
		king[i] = i;
		r[i] = 1;
	};

	int a, b;
	for (int i = 0; i < m; i ++){
		cin >> a >> b;
	};
	pts t;
	for (int i = 0; i < k; i ++){
		cin >> s >> a >> b;
		t.x = a - 1;
		t.y = b - 1;
		cmd.push_back(make_pair(s, t));
	};
	int j = 0;
	for (int i = k - 1; i > -1; i --){
		if (cmd[i].first == "cut") {
			un(cmd[i].second.x, cmd[i].second.y);
		};
		if (cmd[i].first == "ask") {
			ask(cmd[i].second.x, cmd[i].second.y, j);
			j++;
		};
	};
	for (int i = j - 1; i > -1; i--){
		if (ans[i]){
			cout << "YES" << "\n";
		} else {
			cout << "NO" << "\n";
		};
	};
	return 0;
}
```

Заключение

Решено 8 задач на темы связанные с темами изучаемыми по курсу «Дискретной математики». Семь задач на тему «Графы» и «Деревья», три задача на тему «Комбинаторика».

Все задачи решены на официальных сборах и олимпиадах, по требованию готов предоставить ссылки на тестирующие системы и исходные условия.