Разбор задач

31 Районная олимпиада школьников Красноярского края по информатике, 9-11 классы

4 декабря 2017 г.

ID	Задача	Тема	%
1482	A. Снежинка Коха	Простая математика	18
1480	B. Голосование	Структуры данных	26
1481	C. Дендрохронология	Разбор строк	50
1478	D. Числа Фибоначчи	Целочисленная арифметика	70
1467	E. Крестики-нолики	Теория игр	61

Разбор задач: Беляев Сергей Николаевич

Средняя сложность задач: 45%

Рейтинг олимпиады "Муниципальный этап ВОШ Красноярского края по информатике, 9-11 классы"

4:00 из 4:00
олимпиада завершена

отправлено: 2187, принято: 1174
последняя успешная попытка: Железкин Евгений Вадимович, E, 3:59

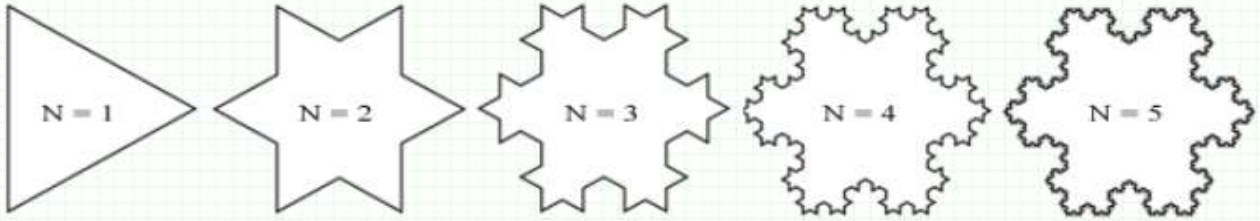
Все муниципалитеты ▾ Все классы ▾

№	Участник	A	B	C	D	E	=	Время	Место
1	Титов Андрей Константинович , г. Красноярск, МАОУ Лицей № 6, 11	90 46:41	100 23:27	92 91:31	100 112:14	80 218:36	462	218	1
2	Цехановская Руслана Юрьевна , г. Красноярск, МБОУ СОШ №10, 11	100 46:09	100 63:35	100 90:42	68 180:37	7 238:22	375	238	2
3	Исаков Антон Сергеевич , г. Красноярск, Лицей №7, 11	100 35:52	100 15:22	56 146:26	24 175:57	92 201:08	372	201	3
4	Марков Игорь Валерьевич , г. Красноярск, МАОУ КУГ №1 Универс, 10	100 64:11	55 18:08	100 101:54	40 112:26	77 238:30	372	238	4
5	Аверин Иван Викторович , Зеленогорск, ЗАТО г. Зеленогорск, МБОУ "ФМЛ№174", 11	100 79:23	100 45:51	60 178:58	40 138:51	70 234:09	370	234	5
6	Шишкин Алексей Владимирович , Шапкино, Енисейский район, МБОУ Шапкинская СОШ №11, 11	100 144:06	100 99:15	40 223:15	40 234:24	85 76:06	365	234	6
7	Прекель Владислав Артёмович , г. Красноярск, Лицей №7, 11	90 176:31	55 21:22	92 57:23	32 237:54	95 212:04	364	237	7
8	Макаренко Илья Сергеевич , Железногорск, ЗАТО г. Железногорск, КГОАУ "Школа Космонавтики", 11	100 203:32	100 43:14	48 211:30	40 124:59	72 109:10	360	211	8
9	Романовский Марк Дмитриевич , г. Красноярск, КГБОУ Красноярский КК, 11	100 34:03	100 80:56	44 103:01	40 139:30	70 195:03	354	195	9
10	Стрекаловская Наталья Андреевна , г. Красноярск, МБОУ СШ№7, 10	10 30:21	100 41:25	100 156:26	100 66:32	35 224:36	345	224	10
11	Чернацкий Евгений Геннадьевич , Зеленогорск, ЗАТО г. Зеленогорск, МБОУ ФМЛ №174, 10	100 33:35	100 47:56	40 222:09	40 99:34	62 193:19	342	222	11
12	Моисеев Никита Игоревич , г. Красноярск, МБОУ СШ №133, 11	80 32:49	100 43:41	60 85:42	20 116:33	77 188:36	337	188	12
13	Железкин Евгений Вадимович , г. Красноярск, Лицей №7, 11	90 203:22	100 93:41	88 142:16	32 192:50	7 239:47	317	239	13
14	Непомнящий Альберт Дмитриевич , г. Красноярск, МАОУ Лицей 7, 11	90 239:53	100 69:48	92 151:00	20 237:52	.	302	237	14
15	Степанов Тимофей Дмитриевич , г. Красноярск, МАОУ Лицей № 6, 11	100 226:31	100 65:13	36 168:25	40 175:03	25 235:25	301	235	15
16	Притуляк Илья Николаевич , г. Красноярск, МБОУ СШ №98, 10	.	100 19:04	68 203:12	100 100:07	25 234:22	293	234	16
17	Федосов Ян Русланович , г. Красноярск, МАОУ Лицей № 6, 9	100 59:27	100 79:06	48 137:39	36 169:47	7 236:46	291	236	17
18	Аносов Алексей Юрьевич , Зеленогорск, ЗАТО г. Зеленогорск, МБОУ "ФМЛ №174", 9	100 116:57	100 185:26	56 234:47	24 94:09	10 190:45	290	234	18
19	Риттер Герман Андреевич , г. Норильск, МБОУ "СШ №1", 11	100 61:12	95 15:35	48 213:15	32 235:57	.	275	235	19
20	Казаков Михаил Вячеславович , Железногорск, ЗАТО г. Железногорск, КГОАУ "Школа Космонавтики", 10	100 237:15	70 149:58	4 165:23	24 229:42	77 212:05	275	237	20

Задача A. Снежинка Коха

(Время: 1 сек. Память: 16 Мб Баллы: 100)

Снежинка Коха – фрактальная кривая, которая строится на основе равностороннего треугольника, который представляет собой снежинку Коха первого порядка (N=1). Снежинка Коха K-го порядка строится из подобной кривой (K-1)-го порядка (K > 1) путем замены каждой стороны данной фигуры четырьмя отрезками, каждый из которых представляет 1/3 от длины исходного отрезка (см. рисунок).



По заданному значению N требуется определить площадь фигуры, ограниченной снежинкой Коха N-го порядка, полагая, что при N=1 площадь равна единице.

Входные данные

Входной файл INPUT.TXT содержит натуральное число N ($N \le 10^{18}$).

Выходные данные

В выходной файл OUTPUT.TXT выведите площадь, ограниченную снежинкой Коха N-го порядка не менее чем с шестью знаками после десятичной точки.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1	1.000000
2	2	1.3333333333

Система оценивания

Решения, работающие только для $N \le 5$, будут оцениваться в 40 баллов.

Решения, работающие только для $N \le 10^6$, будут оцениваться в 90 баллов.

Задача A. Снежинка Коха

Заметим, что изначально мы имеем равносторонний треугольник с площадью 1. При построении снежинки Коха 2-го порядка к нему добавляются еще 3 треугольника, площадь каждого из которых в 9 раз меньше площади исходного (т.к. линейные размеры уменьшены втрое). Так мы получили 12-сторонний многоугольник с равными длинами сторон.

Далее, при переходе от фигуры (i-1)-го порядка к i-му порядку очередное количество добавляемых треугольников увеличивается в 4 раза, а их площади уменьшаются в 9 раз.

Используем следующие обозначения:

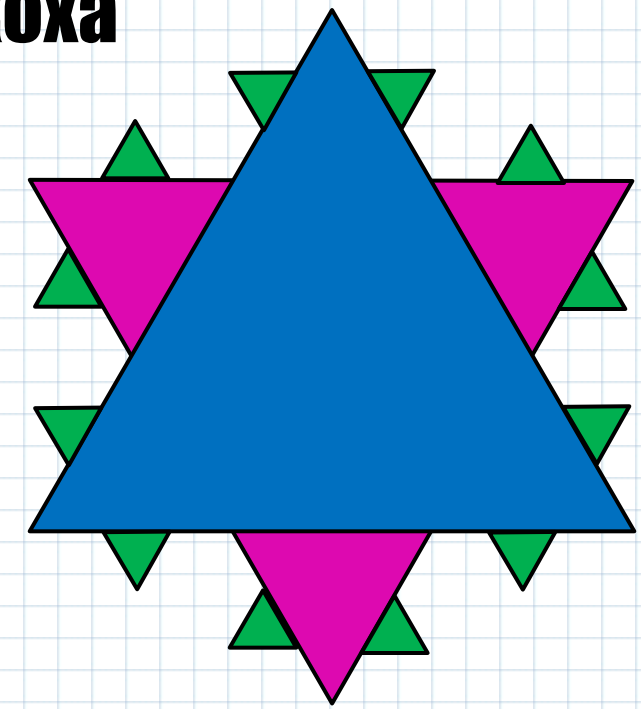
P - площадь добавляемых треугольников

K - количество добавляемых треугольников

S - площадь всех треугольников, включая предыдущие шаги

Таким образом, мы можем на каждом шаге вычислять количество добавляемых треугольников K, их площади P и добавлять к конечному ответу S их общую площадь K·P. В результате за N-1 шаг таких действий мы получим ответ.

При этом следует учесть, что ответ нужно выводить с заданной точностью и что при N>50 в качестве ответа можно вывести площадь снежинки Коха 50-го порядка, т.к. последовательность площадей снежинки Коха сходится к числу 1.6 и при больших значениях N в качестве ответа можно выводить значение 1.6 .



```
float s=1, p=1, k=3

read(n)
n = min(50, n)

for i = 2..n{
    p = p/9
    s = s+k*p
    k = k*4
}

write(s)
```

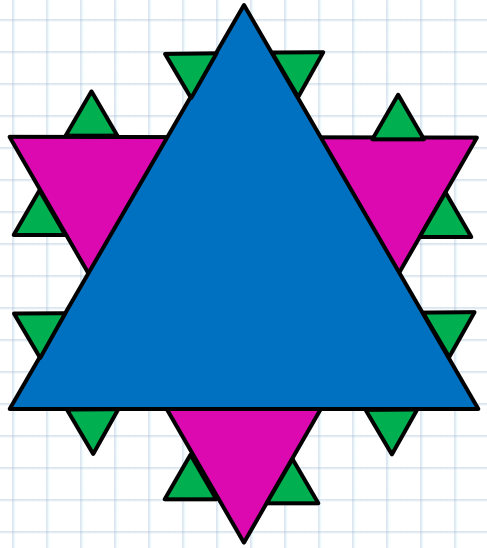
Задача A. Снежинка Коха

```pascal
//Pascal
var n : int64;
    s,p,k : real;
begin
  read(n);
  if n>50 then n := 50;
  s := 1;
  p := 1;
  k := 3;
  while n>1 do begin
    p := p/9;
    s := s + k*p;
    k := k*4;
    dec(n)
  end;
  write(s)
end.
```

```cpp
//GNU C++
#include <bits/stdc++.h>

using namespace std;

int main(){
  long long n;
  double s=1,p=1,k=3;

  cin >> n;
  n = min(50LL, n);

  while(--n)
    p /= 9,
    s += k*p,
    k *= 4;

  cout << fixed << s;
  return 0;
}
```

Задача A. Снежинка Коха

Площадь снежинки Коха можно вычислить математически и представить решение одной формулой:



$$S_1 = 1$$

$$S_2 = S_1 + 3 \cdot \frac{1}{9} \cdot S_1 = 1 + \frac{1}{3}$$

$$S_3 = S_2 + 3 \cdot 4 \cdot \frac{1}{9^2} \cdot S_1 = 1 + \frac{1}{3} + \frac{1}{3} \cdot \frac{4}{9}$$

$$S_4 = S_3 + 3 \cdot 4^2 \cdot \frac{1}{9^3} \cdot S_1 = 1 + \frac{1}{3} + \frac{1}{3} \cdot \frac{4}{9} + \frac{1}{3} \cdot \frac{4^2}{9^2}$$

$$S_N = 1 + \frac{1}{3} + \frac{1}{3} \cdot \frac{4}{9} + \frac{1}{3} \cdot \frac{4^2}{9^2} + \ldots + \frac{1}{3} \cdot \frac{4^{N-2}}{9^{N-2}} = 1 + \frac{1}{3} \cdot \sum_{i=0}^{N-2} \left(\frac{4}{9}\right)^i = 1 + \frac{1}{3} \cdot \left(\frac{1 - \left(\frac{4}{9}\right)^{N-1}}{1 - \left(\frac{4}{9}\right)} \right) = 1 + \frac{3}{5} \cdot \left(1 - \left(\frac{4}{9}\right)^{N-1}\right)$$

Откуда легко видеть, что площадь бесконечной снежинки Коха составляет ровно 1.6, так как верно, что

$$\lim_{n \to \infty} \left(1 + \frac{3}{5} \cdot \left(1 - \left(\frac{4}{9}\right)^{n-1} \right) \right) = 1\frac{3}{5} = 1.6$$

Алгоритмическая реализация:

```
float n
read(n)
write(1+0.6*(1-pow(4/9,n-1)))
```

Задача A. Снежинка Коха

```
//PascalABC.NET 3.2
begin
  write(1+3/5*(1-power(4/9, ReadReal-1)))
end.
```

$$S_N = 1 + \frac{3}{5} \cdot \left(1 - \left(\frac{4}{9}\right)^{N-1}\right)$$

```
//GNU C++
#include <bits/stdc++.h>
using namespace std;
int main(){
  double n;
  cin >> n;
  cout << fixed << 1+0.6*(1-pow(4./9, n-1));
  return 0;
}
```

```
#Python
print(1+0.6*(1-(4/9)**(int(input())-1)))
```

Задача B. Голосование

(Время: 1 сек. Память: 16 Мб Баллы: 100)

В выборах участвовало несколько кандидатов. В результате проведения тайного голосования был получен список из N фамилий. Каждая фамилия – это голос, отданный участником голосования в пользу этого кандидата.

На основании этих данных требуется построить гистограмму результатов проведенного голосования.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – количество отданных голосов ($N \le 1000$). В последующих N строках идут фамилии кандидатов по одному в каждой строке. Каждая фамилия содержит от 1 до 10 букв английского алфавита, при этом первая буква прописная, а остальные – строчные.

Выходные данные

В выходной файл OUTPUT.TXT выведите гистограмму голосования в форме прямоугольника, состоящего из символов «.» (ASCII 46) и «X» (ASCII 88). Число строк должно соответствовать максимальному количеству голосов, отданных за какого-либо кандидата. Число столбцов должно соответствовать количеству кандидатов, участвовавших в выборах. Гарантируется, что за каждого кандидата был отдан как минимум один голос. Высота i-го столбца (число символов «X») таблицы должна соответствовать числу отданных голосов за i-го кандидата. При этом кандидаты перечисляются в алфавитном порядке слева направо. Столбцы обозначаются символами «X» снизу вверх, пустые места – символами «.».

Пример

№	INPUT.TXT	OUTPUT.TXT
1	7 Ivanov Petrov Sidorov Petrov Zubov Petrov Zubov	.X.. .X.X XXXX

Задача B. Голосование

Сначала посчитаем для каждого из кандидатов количество отданных за него голосов. Для этого удобно использовать структуру данных, именуемую ассоциативным массивом или словарем, желательно упорядоченным по ключу. Многие языки поддерживают такую структуру. Например, в C++ это map, в PascalABC - SortedDictionary, а в Python - dict:

```
// пример описания ассоциативного массива на C++
map <string, int> m;
```

Поскольку ограничения в задаче невелики, то можно также обойтись обычным массивом структур из двух полей:

```
//структура "Кандидат", хранящая информацию о голосах кандидата
struct candidate{
  string surname;  // фамилия кандидата
  int votes;       // количество голосов
}

candidate m[1..N]; //массив кандидатов с информацией о голосах
```

Последовательно читая фамилии кандидатов surname, будем увеличивать значение m[surname] на единицу в случае наличия такого элемента, либо создавать элемент m[surname]=1 в противном случае. При этом несложно вычислить максимальное значение mx голосов у победившего кандидата. Зная ширину (m.size - количество уникальных фамилий кандидатов) и высоту (mx) таблицы, несложно реализовать вывод решения.

Задача B. Голосование

```
//PascalABC.NET 3.2
var
  i, mx, n: integer;
  s : string;
  a := new SortedDictionary <string, integer>;
begin
  readln(n);
  for i:=1 to n do begin
    readln(s);
    if s in a then a[s] += 1
              else a[s] := 1;
    mx := max(mx, a[s])
  end;
  for i:=mx downto 1 do begin
    foreach var p in a do
      if p.value<i then write('.')
                   else write('X');
    writeln
  end
end.
```

Входные данные:

```
7
Ivanov
Petrov
Sidorov
Petrov
Zubov
Petrov
Zubov
```

Выходные данные:

```
.X..
.X.X
XXXX
```

Задача B. Голосование

```
//GNU C++ 5.1
#include <bits/stdc++.h>

using namespace std;

int n,i,mx;
string s;
map<string, int> m;

int main(){
  cin >> n;
  while(cin >> s)
    mx = max(mx, ++m[s]);
  for(i=0; i<mx; i++){
    for(auto x : m)
      cout << (mx-i > x.second?'.':'X');
    cout << endl;
  }
  return 0;
}
```

Входные данные:

```
7
Ivanov
Petrov
Sidorov
Petrov
Zubov
Petrov
Zubov
```

Выходные данные:

```
.X..
.X.X
XXXX
```

Задача C. Дендрохронология

(Время: 2 сек. Память: 32 Мб Баллы: 100)

Дендрохронология – метод, в основе которого лежит закон природы, согласно которому каждый год толщина дерева увеличивается на одно кольцо. Толщина каждого кольца зависит от погоды в год образования кольца. У деревьев, растущих в одном и том же месте и климате толщина колец примерно одинакова. По числу колец можно определить возраст дерева, а по толщине колец – погодные условия каждого года, в который это дерево росло.

Используя «перекрестную датировку», можно построить дендрохронологическую шкалу путем увязывания воедино следующих друг за другом поколений деревьев, годы жизни которых перекрываются. Именно так возможно определить погодные условия от некоторого времени до настоящего момента. А на основании построенной шкалы можно датировать те или иные деревянные предметы, сохранившиеся с давних лет.

Однажды, в одном древнем городе, основанном в болотистой местности, археологи обнаружили N слоев мостовых, положенных один над другим и состоящих из отлично сохранившихся бревен. При этом первый слой был положен в момент основания города, а последний – в текущем году. Определив толщину колец бревен для каждого слоя мостовой, они задались вопросом: каков возраст этого города?

Ваша задача – помочь археологам вычислить возраст города по имеющимся данным.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – количество слоев мостовых. Далее следует N строк, описывающих в хронологическом порядке дендрохронологическую шкалу для каждого бревна, взятого из отдельного слоя мостовой. Каждая строка состоит из цифр от 1 до 3, соответствующих толщине годовых колец в миллиметрах описываемого дерева от первого до последнего года его жизни. Суммарная длина всех строк не превышает 10^6. Гарантируется, что дерево, из которого получено бревно следующего уровня, начало расти не раньше, чем дерево, из которого получено бревно предыдущего уровня. Аналогично гарантируется, что дерево, из которого получено бревно следующего уровня, было срублено не раньше, чем дерево, из которого получено бревно предыдущего уровня. Также известно, что годы жизни деревьев соседних уровней перекрываются. При неоднозначном определении общего периода жизни для деревьев смежных слоев следует выбирать максимально возможный диапазон.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное целое число – возраст города.

Пример

№	INPUT.TXT	OUTPUT.TXT	Пояснение
1	4 123123123 3123111321 13212212 212331	13	123123123 .....3123111321 ....13212212 ....212331 (на основании этих данных можно определить дендрохронологическую шкалу для последних 22 лет)

Система оценивания

Решения, работающие только в тех случаях, когда общий период жизни деревьев в смежных слоях составляет не более 10 лет, будут оцениваться в 40 баллов.

Задача C. Дендрохронология

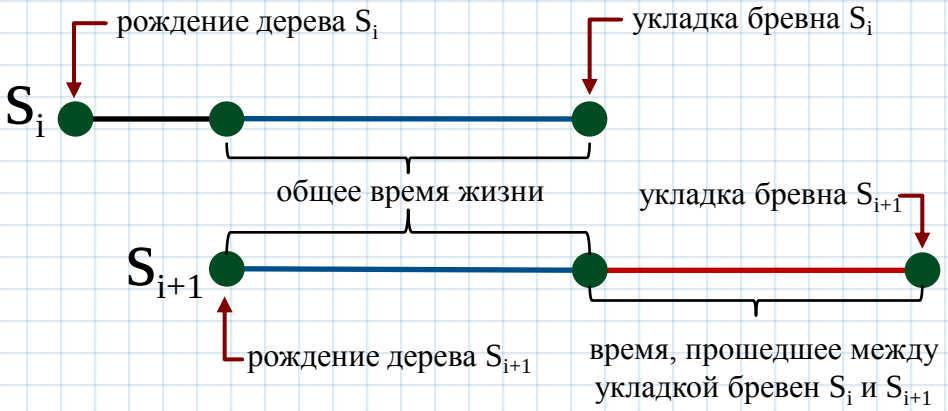
Для определения возраста города достаточно вычислить время между укладкой каждой пары смежных бревен, т.е. то время, в течении которого последующее бревно S_{i+1} еще было деревом в то время, как предыдущее S_i уже было срублено и использовано в постройке. Искомое время равно разнице между временем жизни дерева S_{i+1} и общим временем жизни деревьев S_i и S_{i+1}. Ответом на задачу будет сумма всех таких (N-1) значений.

Задача сводится к поиску максимального общего префикса строки S_{i+1} и суффикса строки S_i. Иными словами, нам нужно найти такое максимальное значение M, что подстрока $S_{i+1}[1..M]$ равна подстроке $S_i[L_i-M+1 .. L_i]$, где L_i - длина строки S_i. Тогда искомая разница будет равна $L_{i+1} - M$.

Ключевым моментом в программе будет служить эффективная реализация поиска значения M. Здесь предусматривается возможность реализации как с асимптотикой $O((L_i + L_{i+1})^2)$, так и $O(L_i + L_{i+1})$. В первом случае можно рассчитывать не более чем на 60-бальное решение. Для реализации эффективного линейного решения потребуется использование префикс-функции или Z-функции. Также эффективная реализация возможна с использованием хешей. В последнем случае можно потерять несколько баллов, если использовать хеши в кольце вычетов по модулю 2^{32} или 2^{64}, т.к. есть тесты со строками Туэ-Морса.

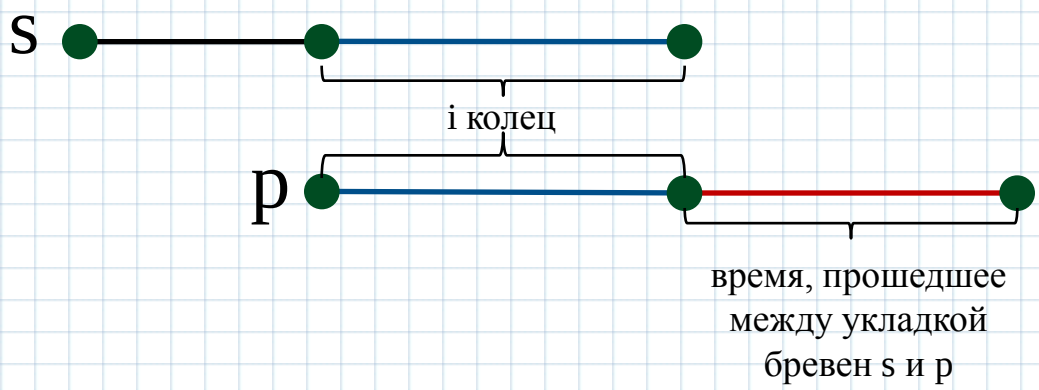
Приведем псевдокод простейшего решения на 40 баллов, которое использует факт, описанный в разделе "Система оценивания", благодаря чему получаем линейный алгоритм для некоторой группы тестов:

```
res = 0
readln(n,s)
for i=2..n{
  readln(p)
  m = min(10,len(s),len(p))
  while(p[1..m]!=s[len(s)-m+1..len(s)])
    m = m-1
  res = res + len(p)-m
  s = p
}
write(res)
```



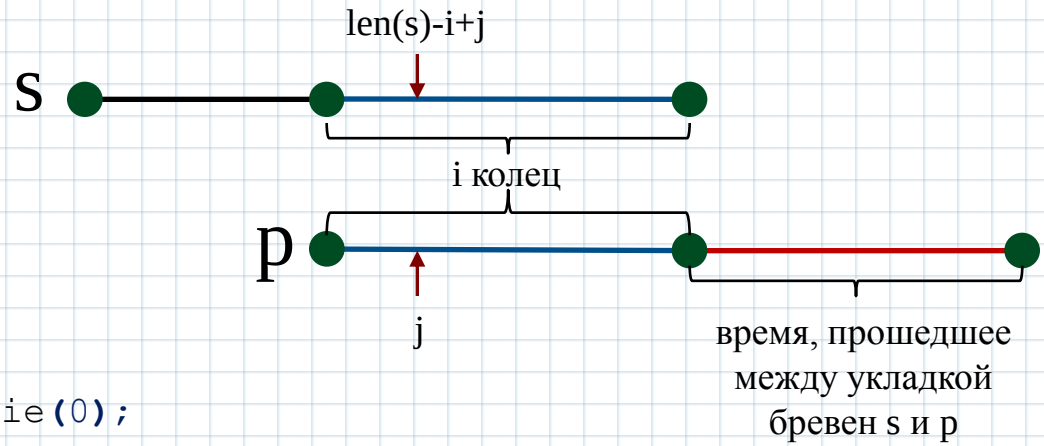
Задача C. Дендрохронология

```
//GNU C++ - 40 баллов
#include <bits/stdc++.h>

using namespace std;

int n,i,res;
string s,p;

int main(){
  ios_base::sync_with_stdio(0); cin.tie(0);
  cin >> n >> s;
  while(cin >> p){
    i = min(10u, min(s.length(), p.length()));
    while(p.substr(0,i) != s.substr(s.length()-i)) i--;
    res += p.length()-i;
    s = p;
  }
  cout << res;
  return 0;
}
```



Задача C. Дендрохронология

```
//GNU C++ - 60 баллов
#include <bits/stdc++.h>

using namespace std;

int n,i,j,ok,res;
string s,p;

int main(){
  ios_base::sync_with_stdio(0); cin.tie(0);
  cin >> n >> s;
  while(cin >> p){
    i = min(s.length(), p.length());
    while(1){
      ok = 1;
      for(j=0; j<i; j++)
        if(p[j]!=s[s.length()-i+j]){ok = 0; break;}
      if(ok) break;
      i--;
    }
    res += p.length()-i;
    s = p;
  }
  cout << res;
  return 0;
}
```

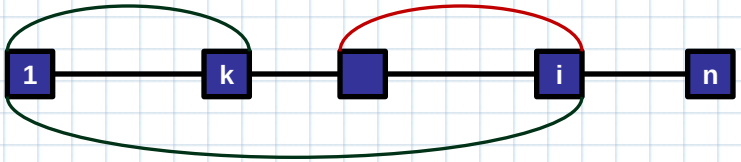


Вычисление префикс-функции

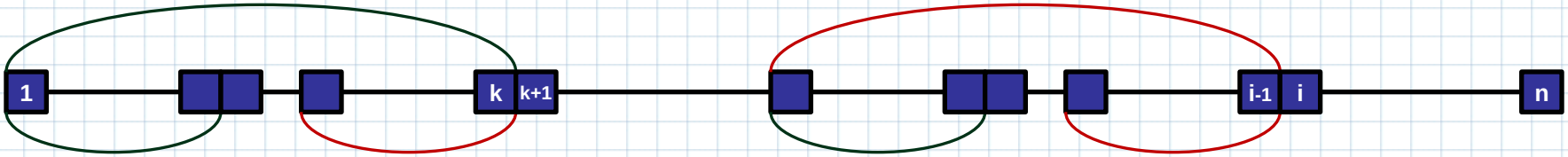
Дана строка s[1..n]. Требуется вычислить для нее префикс-функцию, т.е. массив чисел p[1..n], где p[i] – наибольшая длина наибольшего собственного суффикса подстроки s[1..i], совпадающего с ее префиксом.

Тривиальный алгоритм – $O(N^3)$:

```
p[1..n]={0…0}
for i=2..n
  for k=1..i-1
    if(s[1..k]=s[i-k+1..i]) p[i]=k
```



Можно заметить, что p[i+1] не более, чем на единицу превышает p[i], что позволяет оптимизировать вышеописанный алгоритм до $O(N^2)$. Используя предыдущие вычисления можно избавиться от прямого сравнения строк за O(N), что приведет к эффективному алгоритму за O(N).



Эффективный алгоритм – O(N):

```
p[1..n]={0…0}
for i=2..n{
  k=p[i-1]
  while(k>0 and s[i]<>s[k+1]) k=p[k]
  if(s[i]=s[k+1]) p[i]=k+1
}
```

```
i: 1234567
S: aabaaab
P: 0101223
```

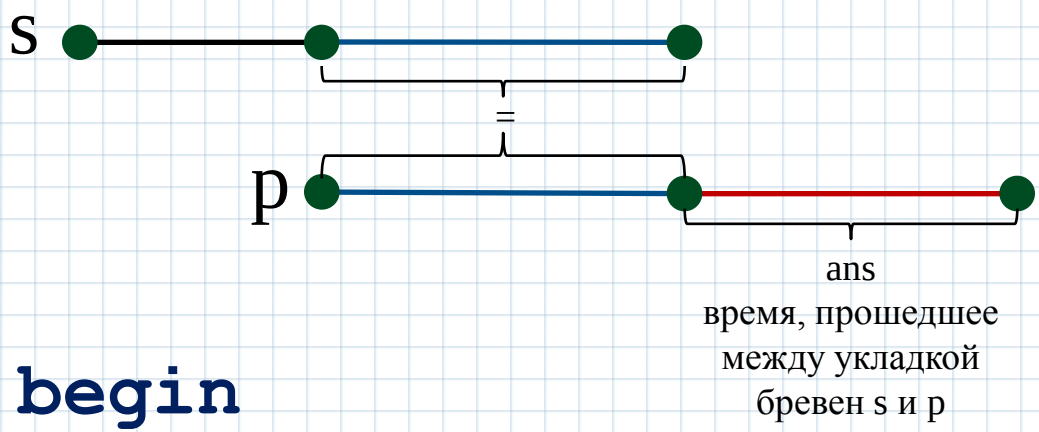
```
i: 123456789…
S: abacabadabacaba
P: 001012301234567
```

Задача C. Дендрохронология

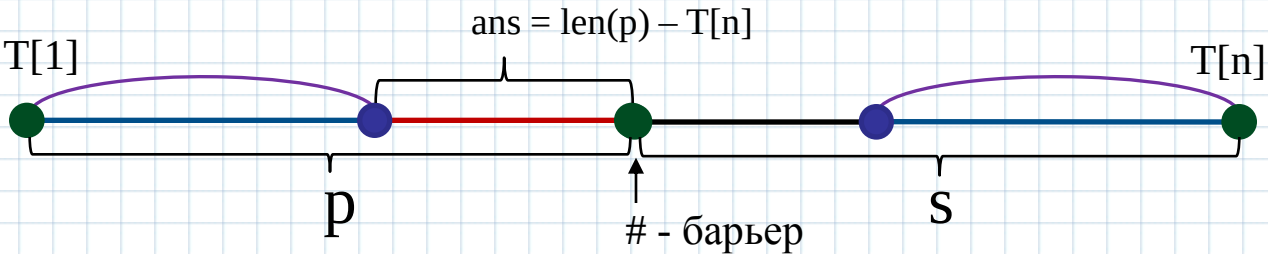
```
//Free Pascal
var n,i,res : integer;
    s,p : string;

function p_last(s : string) : integer;
var i,k,n : integer;
    p : array of integer;
begin
  n := length(s);
  SetLength(p,n+1);
  for i:=2 to n do begin
    k := p[i-1];
    while (k>0) and (s[i]<>s[k+1]) do k := p[k];
    if s[i]=s[k+1] then p[i] := k+1
  end;
  result := p[n]
end;
```

Задача C. Дендрохронология

```
begin
  readln(n);
  readln(s);
  res := 0;
  for i:=2 to n do begin
    readln(p);
    inc(res, length(p) - p_last(p+'#'+s));
    s := p;
  end;
  write(res)
end.
```

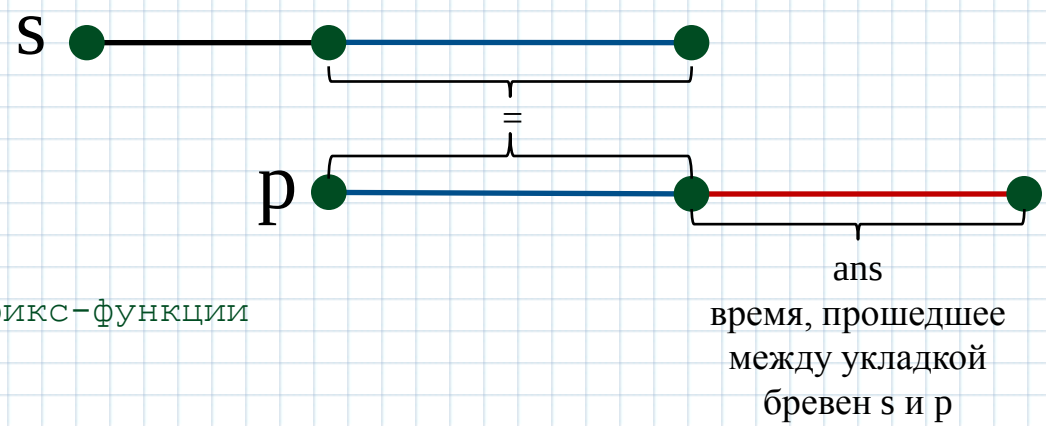


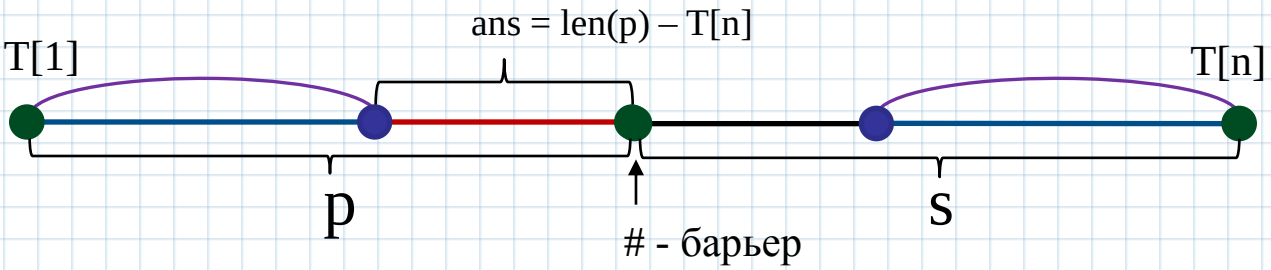
Префикс-функция T:



Задача C. Дендрохронология

```
//GNU C++ (префикс-функция)
#include <bits/stdc++.h>

using namespace std;

int n,i,res;
string s,p;

//Вычисление последнего элемента префикс-функции
int p_last(string s){
  int i, k, n=s.length();
  vector<int> p(n);
  for(i=1; i<n; i++){
    k = p[i-1];
    while(k && s[i]-s[k]) k = p[k-1];
    if(s[i]==s[k]) p[i] = k+1;
  }
  return p[n-1];
}

int main(){
  cin >> n >> s;
  while(cin >> p)
    res += p.length() - p_last(p+'#'+s),
    s = p;
  cout << res;
  return 0;
}
```





Полиномиальный хеш

Полиномиальным хешем строки (или массива чисел) **s[1..n]** в кольце вычетов по модулю **m** называют следующее значение:

$$h = (s_1 + s_2 \cdot p + s_3 \cdot p^2 + s_4 \cdot p^3 + \ldots + s_n \cdot p^{n-1}) \bmod m$$

Здесь $\mathbf{s_i}$ – числовой эквивалент алфавита, **p** – некоторое число, превышающее любое s_i и взаимнопростое с m, т.е. НОД(p,m)=1. Обычно p – нечетное число, а $m = 2^{64}$. В последнем случае при использовании 8-байтового целого типа нет необходимости вычисления остатка от деления.

Пример реализации вычисления хеша строки s[1..n]:

```
//псевдокод с определенным m
h=0; pp=1; p=257; m=10^9+9
for i=1..n{
  h  = (h+s[i]*pp) mod m
  pp = (pp*p) mod m
}
```

```
//код C++ с переполнением long long
long long h=0, pp=1, p=257;
for(int i=0; i<s.length(); i++){
  h += s[i]*pp;
  pp *= p;
}
```

Очевидно, что у равных строк хеши совпадают и сравнение хешей быстрее сравнения длинных строк. Использование данного факта увеличивает производительность программ. Поскольку у разных строк хеши могут совпадать, то неизбежна вероятность коллизий, которую на практике можно свести к нулю при решении олимпиадных задач, если в качестве m выбрать случайное большое число, число операций при этом возрастет из-за присутствия элемента взятия остатка от деления при вычислении хеша. Иногда организаторы олимпиад добавляют тесты к задачам против хеширования по основанию 2^{64}, используя строку Туэ-Морса:

ABBABAABBAABABBABAABABBAABBABAABBAABABBAABBABAABABBABAABBAABABBA...

Данная строка обладает тем свойством, что значение хеша для всех ее подстрок длины более 2048 при любом значении **p** равно нулю. Следует напомнить, что такой контрпример возможен только благодаря знанию основания хеша.

Сравнение строк с помощью хешей

Для сравнения двух строк непосредственное вычисление их хешей не позволит увеличить производительность. В задачах часто требуется работать с элементами одной большой строки. Поэтому далее будем говорить о сравнении подстрок некоторой строки s[1..n].

```
//определение массива степеней pp[1..n]
//pp[i] = p^(i-1)
pp[1]=1
for i=2..n
  pp[i] = pp[i-1]*p
```

```
//вычисление хешей всех префиксов s[1..n]
//h[i] = hash(s[1..i])
h[0]=0
for i=1..n
  h[i] = h[i-1]+s[i]*pp[i]
```

Нас интересует быстрое вычисление за O(1) значения хеша для подстроки s[l..r]:

$$h[l..r] = s[l]*pp[1] + s[l+1]*pp[2] + \ldots + s[r]*pp[r-l+1]$$

Откуда:

$$h[l..r]*pp[l] = s[l]*pp[l] + s[l+1]*pp[l+1] + \ldots + s[r]*pp[r] = h[r]-h[l-1]$$
$$h[l..r] = (h[r]-h[l-1])/pp[l] = (h[r]-h[l-1])*Inv(pp[l])$$

Пусть у нас есть две подстроки $s[l_1,r_1]$ и $s[l_2,r_2]$, тогда справедливы следующие сравнения:

$$h[l_1..r_1] = h[l_2..r_2]$$
$$(h[r_1]-h[l_1-1])/pp[l_1] = (h[r_2]-h[l_2-1])/pp[l_2]$$
$$(h[r_1]-h[l_1-1])*pp[l_2] = (h[r_2]-h[l_2-1])*pp[l_1]$$

Мы избавились от вычисления обратного элемента. Задачу можно еще упростить, если при вычислении хеша приводить все степени к единой максимальной, домножая разницу хеш-префиксов на некоторый элемент массива pp так, чтобы у элемента s[l] получить множитель p^{n-1} (=pp[n]):

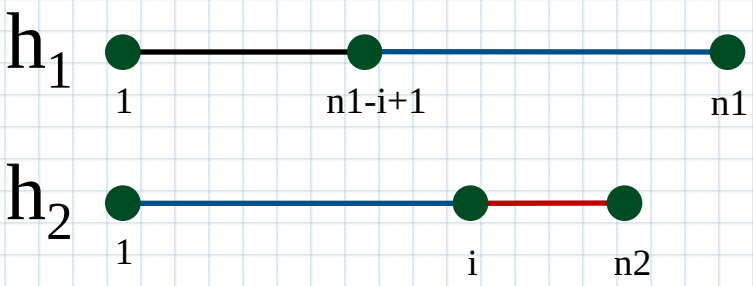
$$hash(l,r) = s[l]*p^{n-1} + \ldots + s[r]*p^{n+r-l-1} = (h[r]-h[l-1])*pp[n-l+1]$$

Задача C. Дендрохронология

```
//GNU C++ (полиномиальный хеш)
#include <bits/stdc++.h>
#define Int long long
#define mn 1000005
#define mod 1000000009

using namespace std;

Int i, n, n1, n2, res, P=53;
string s1, s2;

Int p[mn], h1[mn], h2[mn];

void GetHash(string s, Int *h){
  for(int i=0; i<s.size(); i++)
    h[i+1] = (h[i] + s[i]*p[i]) % mod;
}
```

Задача C. Дендрохронология

```
int main(){
  ios_base::sync_with_stdio(0); cin.tie(0);
  p[0]=1;
  for(i=1; i<mn; i++)
    p[i] = (p[i-1]*P) % mod;
  cin >> n >> s1;
  n2 = s1.length(); GetHash(s1, h2);
  while(cin >> s2){
    for(i=0; i<=n2; i++) h1[i] = h2[i];
    n1 = n2; n2 = s2.length();
    GetHash(s2, h2);
    i = min(n1, n2);
    while((h1[n1]-h1[n1-i]+mod)%mod != h2[i]*p[n1-i]%mod) i--;
    res += n2-i;
    s1 = s2;
  }
  cout << res;
  return 0;
}
```



Задача D. Числа Фибоначчи

(Время: 1 сек. Память: 16 Мб Баллы: 100)

Последовательностью Фибоначчи называется бесконечная числовая последовательность целых чисел F_k, где $F_0 = 0$, $F_1 = 1$, $F_k = F_{k-1} + F_{k-2}$ (для любого целого k). Пример части данной последовательности:

k	-2	-1	0	1	2	3	4	5	6
F_k	-1	1	0	1	1	2	3	5	8

Требуется найти остаток от деления n-го число Фибоначчи на 10^9.

Напомним, что остатком от деления целого числа a на целое число b называют такое неотрицательное целое число r ($0 \le r < |b|$), что выполняется следующее равенство: $a = b \cdot q + r$, где q – некоторое целое число.

Входные данные

Входной файл INPUT.TXT содержит целое число n ($-10^{18} \le n \le 10^{18}$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – остаток от деления F_n на 10^9.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1	1
2	10	55
3	-2	999999999

Система оценивания

Решения, работающие для $0 \le n \le 30$, будут оцениваться в 20 баллов.

Решения, работающие для $-10^6 \le n \le 10^6$, будут оцениваться в 40 баллов.

Решения, работающие для $-10^9 \le n \le 10^9$, будут оцениваться в 80 баллов.

Задача D. Числа Фибоначчи

Решение №1 – 20 баллов

Для вычисления значения F_n при $0 \le n \le 30$ используем рекуррентную формулу, описанную в условии задачи:

$$\begin{cases} F_i = i, \ 0 \le i < 2 \\ F_i = F_{i-1} + F_{i-2}, \ i > 1 \end{cases}$$

На основе данной формулы несложно записать следующую рекурсивную реализацию:

```
int F(n){
  if(n<2) return n
     else return F(n-1)+F(n-2)
}

read(n)
write(F(n))
```

Поскольку асимптотика данного решения равна O(F(n)), то для больших n такое решение не пройдет по времени. Вычисление остатка ответа на 10^9 также не принесет дополнительных бонусов. Однако, если учесть отрицательные значения n в данной реализации, то можно получить еще 4 балла.

Решение №2 – 40 баллов

Вычислить F_i можно как через предыдущие значения F_{i-1} и F_{i-2}, так и через последующие F_{i+1} и F_{i+2}, что выражается следующими формулами:

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_i = F_{i-1} + F_{i-2}, \ i > 1 \\ F_i = F_{i+2} - F_{i+1}, \ i < 0 \end{cases}$$

Это позволяет вычислить все значения F_i в диапазоне от -10^6 до 10^6, что можно представить в виде следующего алгоритма:

```
int F[-10^6..10^6]
F[0]=0; F[1]=1

for i=2..10^6
  F[i] = (F[i-1]+F[i-2]) mod 10^9
for i=-1..-10^6
  F[i] = (F[i+2]-F[i+1]+10^9) mod 10^9

read(n)
write(F[n])
```

Задача D. Числа Фибоначчи

```cpp
//C++  (20 баллов)
#include <iostream>

using namespace std;

int F(int n){
  if(n<2) return n;
     else return F(n-1)+F(n-2);
}

int main(){
  int n;
  cin >> n;
  cout << F(n);
  return 0;
}
```

```cpp
//C++ (40 баллов)
#include <iostream>
#define mod 1000000000

using namespace std;

int n,a,b=1;

int main(){
  cin >> n;

  if(n>0)
    while(n--)
      b = (a+b) % mod,
      a = (b-a+mod) % mod;
  else
    while(n++)
      a = (b-a+mod) % mod,
      b = (b-a+mod) % mod;

  cout << a;

  return 0;
}
```

Задача D. Числа Фибоначчи

Решение №3 – 80 баллов

Заметим, что достаточно научиться вычислять F_i для $i \geq 0$, т.к. для отрицательных значений i мы получим, что $F_i = F_{-i}$, если i нечетно и $F_i = -F_{-i}$, если i четно. Это несложно доказать, используя метод математической индукции и формулу $F_i = F_{i+2} - F_{i+1}$.

Очевидно, что при больших значениях n прямое вычисление значений для $n > 10^8$ может не проходить по времени. Однако, мы можем написать программу предподсчета (precalc), которая будет вычислять все значения для $n=0..10^9$. На современном компьютере это возможно сделать быстрее, чем за 10 секунд. Более того, в условиях олимпиады возможно вычислить все значения для $n=0..10^{11}$, что может потребовать порядка 30 минут. Вычисляя все значения таким образом, мы можем выводить информацию о парах чисел Фибоначчи, порядковые номера которых равны 10^6*k и 10^6*k+1 для $k = 0..1000$. В итоге мы получим список из 2002 чисел Фибоначчи, взятых по модулю 10^9, которые добавим в нашу программу с решением в виде массива следующего вида:

$f[] = \{F_0, F_1, F_{1000000}, F_{1000001}, F_{2000000}, F_{2000001}, F_{3000000}, F_{3000001}, \ldots, F_{1000000000}, F_{1000000001}\}$

Теперь для вычисления n-го числа Фибоначчи достаточно определить наименьшую пару чисел в нашем массиве и выполнить процесс вычислений, начав с данной пары. Очевидно, что для этого нам потребуется не более, чем миллион операций. Работу программы можно ускорить, если находить не наименьшую, а ближайшую пару чисел и перебирать индекс не только в большую сторону, но и в меньшую.

При использовании компилятора GNU C++ и эффективном написании кода тестирующая система сайта «Школа программиста» позволяет вычислять более 10^8 чисел Фибоначчи в секунду. Это позволяет увеличить шаг с 10^6 до $2 \cdot 10^8$. Таким образом, описанный выше метод позволяет решить задачу для $|n| \leq 2 \cdot 10^{11}$, однако подобная реализация не позволяет пройти больше тестов в данной задаче.

Задача D. Числа Фибоначчи

```
//GNU C++ (precalc)
#include <bits/stdc++.h>
#define div 100000000
#define mod 1000000000

using namespace std;

int n=mod, a=0, b=1;

ofstream out("output.txt");

int main(){
  for(int i=0; i<=n; ++i){
    if(i%div==0){
      cout << i << "\r";
      out << a << "," << b << ",";
    }
    b = (a+b) % mod,
    a = (b-a+mod) % mod;
  }

  return 0;
}
```

```
//GNU C++ (80 баллов)
#include <bits/stdc++.h>
#define div 100000000
#define mod 1000000000

using namespace std;

int n,a,b,c,p,k,f[] = {
  0,1,760546875,60937501,839453125,100390626,
  600000000,800000001,360546875,860937501,
  439453125,900390626,200000000,600000001,
  960546875,660937501,39453125,700390626,
  800000000,400000001,560546875,460937501};

int main(){
  cin >> n;
  if(n<0){p = n%2==0; n = -n;}

  k = n/div*2;
  n %= div;
  a = f[k]; b = f[k+1];

  while(n--){
    c = a+b; a = b;
    b = (c>=mod)?c-mod:c;
  }

  cout << ((p&a)?mod-a:a);

  return 0;
}
```

Задача D. Числа Фибоначчи

Решение №4 – 100 баллов

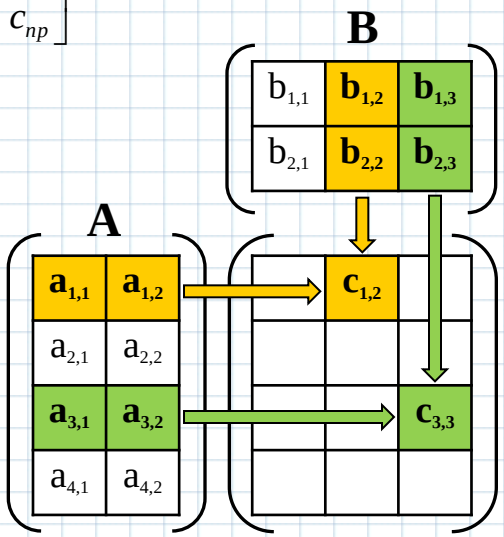
Полное решение этой задачи заключается в использовании матриц. *Матрица* – это прямоугольная таблица, состоящая из строк и столбцов. Когда число количество строк совпадает с количеством столбцов, то матрицу называют *квадратной*. В качестве элементов матрицы могут выступать объекты любой природы. Над матрицами определены операции сложения и умножения.

В данной задаче мы будем использовать только матрицы размером 1×2 и 2×2, элементы которых – целые числа. Также нам потребуется уметь умножать матрицы. Ниже описан принцип умножения матриц.

Произведение матриц A и B размером n×m и m×p соответственно:

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{im} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \quad B=\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1j} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2j} & \dots & b_{2p} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mj} & \dots & b_{mp} \end{bmatrix} \quad C=A*B=\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \dots & \dots & c_{ij} & \dots \\ c_{n1} & c_{n2} & \dots & c_{np} \end{bmatrix}$$

$$c_{ij}=\sum_{k=1}^{m} a_{ik}\cdot b_{kj}=a_{i1}\cdot b_{1j}+a_{i2}\cdot b_{2j}+\dots+a_{ik}\cdot b_{kj}+\dots+a_{im}\cdot b_{mj} \quad (i=1..n,\ j=1..p)$$



Пример:

$$\begin{bmatrix} 2 & 3 \\ -1 & 4 \end{bmatrix} * \begin{bmatrix} 2 & -3 & 4 \\ 3 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2\cdot 2+3\cdot 3 & 2\cdot(-3)+3\cdot 1 & 2\cdot 4+3\cdot 0 \\ -1\cdot 2+4\cdot 3 & -1\cdot(-3)+4\cdot 1 & -1\cdot 4+4\cdot 0 \end{bmatrix} = \begin{bmatrix} 13 & -3 & 8 \\ 10 & 7 & -4 \end{bmatrix}$$

Задача D. Числа Фибоначчи

Решение №4 – 100 баллов

Рассмотрим следующее равенство:

$$\begin{pmatrix} F_i & F_{i+1} \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} F_{i+1} & F_{i+2} \end{pmatrix}$$

Откуда следует, что

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

Таким образом получаем, что решение задачи сводится к возведению матрицы в n-ю степень. Ответом будет служить левый нижний элемент матрицы.

Для вычисления значения A^n будем использовать метод бинарного возведения в степень с асимптотикой O(log n), который использует следующую рекуррентную формулу:

$$A^n = \begin{cases} \left(A^{\frac{n}{2}}\right)^2, & если\ n-четно \\ A \cdot \left(A^{\frac{n-1}{2}}\right)^2, & если\ n-нечетно \end{cases}$$

Здесь удобно использовать рекурсивную реализацию для возведения матрицы A в n-ю степень в кольце вычетов по модулю 10^9:

```
Matrix Power(A, n){
  if(n==0) return E
  B = Power(A, n div 2)
  if(odd(n)) return B*B*A
        else return B*B
}
```

Здесь приняты следующие обозначения:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Также для возведения произвольной матрицы размером 2×2 в квадрат используем, что

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^2 = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} a^2+bc & ab+bd \\ ac+cd & bc+d^2 \end{pmatrix}$$

Реализация основного кода:

```
read(n)
C = Power(A, |n|)
res = C[2][1]
if(n<0 and n mod 2==0 and res>0)
  res = 10^9 - res
write(res)
```

Задача D. Числа Фибоначчи

```
//PascalABC.NET 3.2
const ost = 1000000000;

var n,a,b,c,d : int64;

procedure power(n: int64;
                var a, b, c, d : int64);
var aa,bb,cc,dd : int64;
begin
  if n=0 then begin
   a:=1; d:=1; b:=0; c:=0;
   exit
  end;
  power(n div 2, aa, bb, cc, dd);
  a := (aa*aa + bb*cc) mod ost;
  b := (aa*bb + bb*dd) mod ost;
  c := (aa*cc + cc*dd) mod ost;
  d := (bb*cc + dd*dd) mod ost;
  if odd(n) then begin
    aa:=a; bb:=b; cc:=c; dd:=d;
    a := bb;
    b := (aa + bb) mod ost;
    c := dd;
    d := (cc + dd) mod ost;
  end
end;
```

```
begin
  read(n);
  power(abs(n), a, b, c, d);
  if (n<0) and not odd(n) then
    c := (ost - c) mod ost;
  write(c)
end.
```

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^2 = \begin{pmatrix} a^2+bc & ab+bd \\ ac+cd & bc+d^2 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} b & a+b \\ d & c+d \end{pmatrix}$$

Задача D. Числа Фибоначчи

```
//GNU C++
#include <bits/stdc++.h>
#define Int long long
#define mod 1000000000

using namespace std;

Int n,a,b,c,d;

void power(Int n,  Int &a,
           Int &b, Int &c, Int &d){
  Int aa,bb,cc,dd;
  if(n==0){a=d=1; b=c=0; return;}
  power(n/2, aa, bb, cc, dd);
  a = (aa*aa + bb*cc) % mod;
  b = (aa*bb + bb*dd) % mod;
  c = (aa*cc + cc*dd) % mod;
  d = (bb*cc + dd*dd) % mod;
  if(n%2){
    aa = a; bb = b; cc = c; dd = d;
    a = bb;
    b = (aa + bb) % mod;
    c = dd;
    d = (cc + dd) % mod;
  }
}
```

```
int main(){
  cin >> n;

  power(abs(n), a, b, c, d);

  if(n<0 && n%2==0 && c) c = mod - c;
  cout << c;

  return 0;
}
```

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^2 = \begin{pmatrix} a^2+bc & ab+bd \\ ac+cd & bc+d^2 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} b & a+b \\ d & c+d \end{pmatrix}$$

Задача D. Числа Фибоначчи

```
#include <bits/stdc++.h>

using Int = long long;
using namespace std;

Int n,res,mod=1e9;
map<Int, int> m;

Int f(Int n){
  if(m.count(n)) return m[n];
  Int a,b,c,d = n/2;
  a = f(d-1);
  b = f(d);
  c = f(d+1);
  if(n%2) return m[n]=(b*b+c*c)%mod;
     else return m[n]=((c*c-a*a)%mod+mod)%mod;
}

int main(){
  cin >> n;

  m[0]=0; m[1]=m[2]=1;
  res = f(abs(n));
  if(n<0 && n%2==0 && res) res = mod - res;
  cout << res;

  return 0;
}
```

Решение №5 – 100 баллов

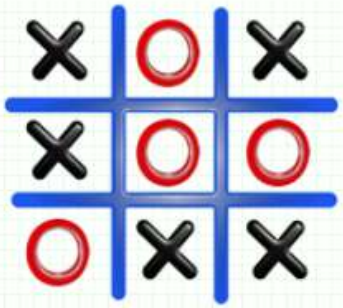
Задача имеет более простое решение при использовании следующих формул:

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

Задача E. Крестики-нолики

(Время: 1 сек. Память: 16 Мб Баллы: 100)



Это интерактивная задача.

«Крестики-нолики» – популярная логическая игра на квадратном поле 3×3 для двух игроков, которые по очереди ставят на свободные клетки знаки (первый ставит «крестики», второй – «нолики»). Выигрывает тот, кто первым поставит свои фигуры в ряд по вертикали, горизонтали или диагонали. Известно, что при выборе оптимальной стратегии обоими игроками игра сводится к ничьей.

Требуется написать программу, которая играет и не проигрывает программе жюри.

Протокол взаимодействия

Сначала на вход программе подается целое число P ($1 \le P \le 2$) – номер вашего хода: если P=1, то вы играете «крестиками», в противном случае – «ноликами». Далее идет последовательность обменов ходами вашей программы и программы жюри. Каждый ход описывается парой целых чисел X и Y ($1 \le X, Y \le 3$) – номер столбца и строки, куда следует разместить очередной «крестик» или «нолик». При P=2 после чтения значения P следует считывать первый ход программы жюри. Всего за игру первый игрок совершает 5 ходов, а второй – 4. Таким образом, первый игрок выполняет последний ход. В конце игры (когда вы сделаете последний свой ход) ваша программа должна немедленно завершиться.

Гарантируется, что программа жюри не проигрывает и придерживается оптимальной стратегии. Также гарантируется, что в каждом отдельном тесте при одинаковой последовательности ваших действий, программа жюри будет также повторять свою последовательность ходов.

Примеры

№	стандартный ввод	стандартный вывод	пояснение
1	1 3 1 3 2 1 1 2 3	2 2 1 2 3 3 2 1 1 3	OXO XXO XOX
2	2 2 2 2 3 3 1 1 2 3 3	1 1 2 1 1 3 3 2	OOX XXO OXX

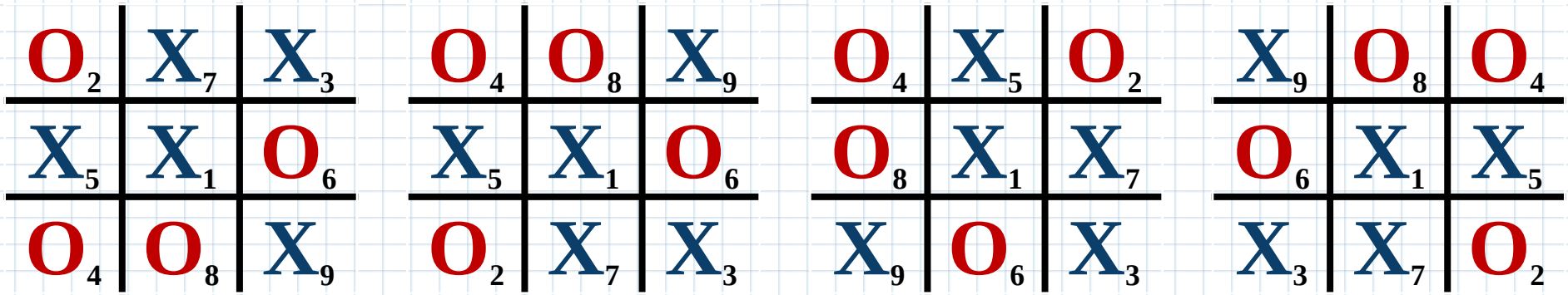
Система оценки

Решения, работающие только для P = 1, будут оцениваться в 25 баллов.

Задача E. Крестики-нолики

Решение №1 – 25 баллов

Рассмотрим случай, когда P=1, т.е. когда мы играем крестиками и наш ход первый. Можно понять, что в данном случае может быть немного вариантов развития игры, если первый ход мы сделаем в центр. Действительно, таких вариантов всего четыре и они симметричны. Из возможных 8 ходов, которые может сделать второй игрок, программа жюри может выбрать только 4 – это все ходы в угол, т.к. в противном случае это проигрышный ход. Все варианты развития игры представлены ниже:



Так несложно видеть, что все последующие ходы (кроме первого) программы жюри будут однозначны.

Решение №2 – X баллов

Еще большее количество баллов в этой задаче возможно набрать, если выполнять ходы в случайные пустые клетки. В таком случае можно рассчитывать примерно на 25% успешных партий. Здесь важно не использовать генерацию псевдослучайных чисел с подключением таких команд как randomize (паскаль) и srand (C++), которые позволяют при запуске одной и той же программы в разное время выдавать разный результат (это может привести к тому, что при перетестировании ваша программа не пройдет первый тест и вы получите 0 баллов). В случае WA1 можно в начало программы добавить несколько вызовов функции, возвращающей случайное число. Такое решение в теории позволит набрать от 2 до 100 баллов, но наиболее вероятный исход – 25 баллов. Если это решение объединить с предыдущим (т.е. добавить правильную обработку при P=1), то наиболее вероятный результат будет характеризоваться 35 баллами. И такое решение можно улучшить, если дополнительно проверять все тройки и в случае наличия в одной из них двух ноликов и пустой клетки ставить туда свой крестик (в эту пустую клетку).

Задача E. Крестики-нолики

```
//C++ (25 баллов)
#include <iostream>

using namespace std;

int x,y,p,k,a[3][3];

int main(){
  cin >> p;
  cout << "2 2\n";
  cout.flush();
  cin >> x >> y;
  if(x==1 && y==1) cout << "3 1\n1 2\n2 1\n3 3\n";
  if(x==1 && y==3) cout << "3 3\n1 2\n2 3\n3 1\n";
  if(x==3 && y==1) cout << "3 3\n2 1\n3 2\n1 3\n";
  if(x==3 && y==3) cout << "1 3\n3 2\n2 3\n1 1\n";
  cout.flush();
  return 0;
}
```

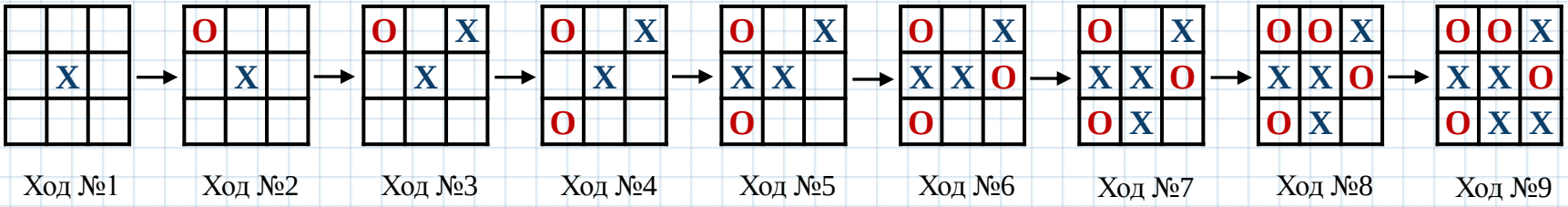
Задача E. Крестики-нолики

```
//C++ (27 баллов)
#include <iostream>

using namespace std;

int x,y,p,k,a[3][3];

int main(){
  rand();rand();
  cin >> p;
  for(k=0; k<5; k++){
    if(k>0 || p==1){
      do{
        x = rand()%3;
        y = rand()%3;
      }while(a[y][x]);
      a[y][x] = p;
      cout << x+1 << " " << y+1 << endl;
      cout.flush();
      if(k==4 && p==1) break;
    }
    cin >> x >> y;
    a[y-1][x-1] = 3-p;
  }
  return 0;
}
```

Задача E. Крестики-нолики

Решение №3 – 100 баллов

Можно ошибочно предположить, что количество различных партий равно 9!. Не всегда на i-м ходу может быть 10-i пустых клеток, в которые можно поставить крестик или нолик так, чтобы оказаться в беспроигрышной ситуации. На самом деле существует 127872 возможных исхода игры, заканчивающейся ничьей с учетом оптимальной игры, включая симметричные партии. Можно исключить все симметричные варианты путем вращения матрицы. Случай при P=1, когда мы играем крестиками был уже рассмотрен ранее и там можно обеспечить всего один вариант развития игры (с учетом симметрии):

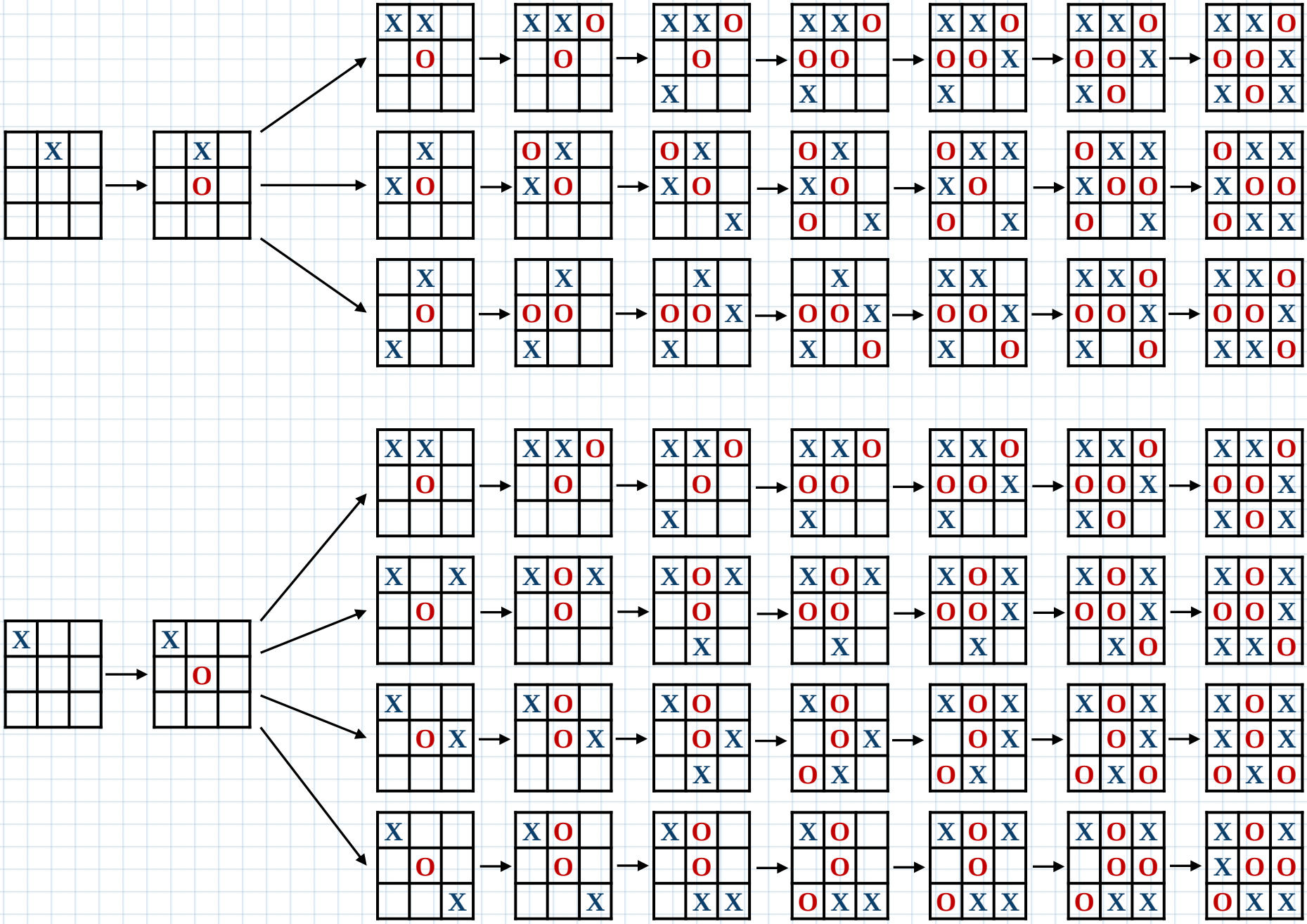


При P=2 мы получим 22 варианта развития игры. Когда наш ход, мы не будем рассматривать все возможные варианты, мы выберем один из возможных, который содержит наименьшее число разветвлений. На первом ходу программа жюри может сделать 3 варианта хода: поставить крестик в угол (4 варианта развития игры), поставить крестик в центр (15 вариантов развития игры) и поставить крестик сбоку (3 варианта развития игры).

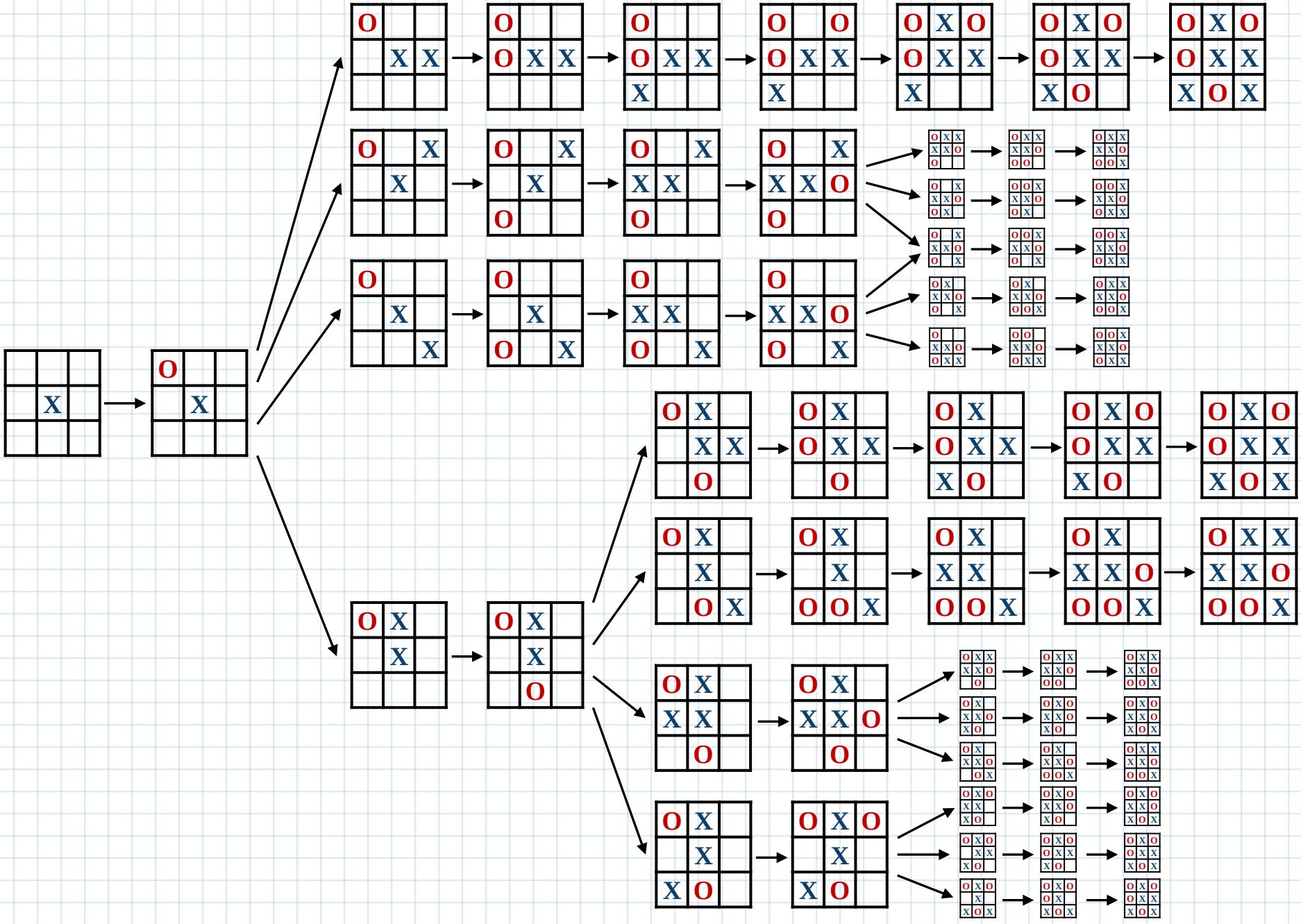
Несмотря на казалось бы небольшое число различных комбинаций развития игры, все же рассмотрение всех этих вариантов представляет собой достаточно сложную реализацию и такое решение не является оптимальным с точки зрения трудозатрат и возможных ошибок в процессе реализации 23 вариантов развития игры.

Далее рассмотрим все возможные сценарии развития игры для P=2.

Задача E. Крестики-нолики



Задача E. Крестики-нолики



Задача E. Крестики-нолики

```
//GNU C++11
#include <bits/stdc++.h>

using namespace std;

int i,j,x,y,p;
//матрица вращений
pair<int, int> a[3][3];

//запрос хода к программе жюри
void get(int &x, int &y){
  cin >> x >> y;
  for(int i=0; i<3; i++)
    for(int j=0; j<3; j++)
      if(a[i][j] == make_pair(y-1, x-1))
        {x=j+1; y=i+1; return;}
}

//вывод хода
void put(int x, int y, int skip=1){
  cout << a[y-1][x-1].second+1 << " "
       << a[y-1][x-1].first+1 << endl;
  cout.flush();
  if(!skip) get(::x, ::y);
}
```

```
//разворот по горизонтали
void RotateOX(){
  swap(a[0][0], a[0][2]);
  swap(a[1][0], a[1][2]);
  swap(a[2][0], a[2][2]);
  x = 4-x;
}

//разворот по вертикали
void RotateOY(){
  swap(a[0][0], a[2][0]);
  swap(a[0][1], a[2][1]);
  swap(a[0][2], a[2][2]);
  y = 4-y;
}

//разворот по диагонали
void RotateXY(){
  swap(a[0][1], a[1][0]);
  swap(a[0][2], a[2][0]);
  swap(a[1][2], a[2][1]);
  swap(x,y);
}
```

Задача E. Крестики-нолики

```
int main(){
  //инициализация матрицы вращений
  for(i=0; i<3; i++)
    for(j=0; j<3; j++) a[i][j]={i,j};
  cin >> p;
  //если играем «крестиками»
  if(p==1){
    put(2,2,0);
    if(x==1 && y==3) RotateOY();
    if(x==3 && y==1) RotateOX();
    if(x==3 && y==3)
      RotateOX(), RotateOY();
    put(3,1); put(1,2);
    put(2,1); put(3,3);
    return 0;
  }
  get(x,y);
  //x=2, y=1
  if(x==2 && y==3) RotateOY();
  if(x==3 && y==2) RotateOX();
  if(x==1 && y==2) RotateXY();
  if(x==2 && y==1){
    put(2,2,0);
    if(x==3) RotateOX();
    if(y==1)
      put(3,1), put(1,2), put(2,3);
    if(y==2)
      put(1,1), put(1,3), put(3,2);
    if(y==3)
      put(1,2), put(3,3), put(3,1);
    return 0;
  }
  //x=1, y=1
  if(x==3) RotateOX();
  if(y==3) RotateOY();
  if(x*y==1){
    put(2,2,0);
    if(x==1 || y==3) RotateXY();
    if(x==3 && y==1) put(2,1), put(1,2), put(3,3);
    if(x==2) put(3,1), put(1,2), put(3,3);
    if(y>1) put(2,1), put(1,3), put(3,5-y);
    return 0;
  }
  //x=2, y=2
  put(1,1,0);
  if(x==1 || y==3) RotateXY();
  if(x==3 && y==2) put(1,2), put(3,1), put(2,3);
  if(x==3 && y!=2)
    {put(1,3,0); put(3,2,0); put(2,4-y); return 0;}
  if(x==2 && y==1){
    put(2,3,0);
    if(x==3 && y==2) put(1,2), put(3,1);
    if(x*y==9) put(1,3), put(3,2);
    if(x==1 && y==3)
      {put(3,1,0); put(4-x,2); return 0;}
    if(x==1 && y==2){
      put(3,2,0);
      if(x*y==9) put(3,1); else put(y,x);
    }
  }

  return 0;
}
```

Задача E. Крестики-нолики

Решение №4 – 100 баллов

Наиболее приемлемым вариантом решения данной задачи является использование метода минимакса, который часто применяется в теории игр. Здесь реализуется некоторая рекурсивная функция Best(X, P, N), которая оценивает качество хода X для игрока P (P – номер игрока: 1 или 2) с анализом глубины N, возвращая число: чем число больше, тем ход X для игрока P лучше. Для определения качества хода осуществляется перебор всех возможных ходов Y игрока 3-P (соперника) и вычисляется значение Best(Y, 3-P, N-1). В результате функция возвращает наибольшее из полученных значений со знаком «минус». Действительно, для игрока лучший ход тот, при котором наилучший ход соперника будет наихудшим. Описанную выше идею можно записать следующим образом:

```
float Best(X, P, N){
  if(N==0) return GetValue(P)          //возвращаем тривиальный критерий оценки
  MakeMove(X)                          //выполняем ход X игрока P
  res = -INF
  foreach Y in AllMoves(3-P)           //перебираем все ходы соперника
    res = max(res, Best(Y, 3-P, N-1))  //находим наилучший ход соперника глубины N-1
  RestoreMove(X)                       //восстанавливаем игровое поле, отменяя ход X
  return -res                          //возвращаем оценку
}
```

Для определения лучшего хода достаточно перебрать всевозможные ходы и выбрать тот ход, при котором функция вернула наибольшее значение.

В нашей задаче функция Best(x,y,p) будет возвращать 1, если ход в клетку (x,y) приводит к победе, -1 будет означать поражение, а 0 – ничью. Заметим, что передавать глубину анализа в данной задаче не нужно, т.к. это значение не превысит 9 и перебор всех вариантов возможен. После выполнения хода будем проверять: не образовалась ли тройка в строке y, в столбце x или на диагоналях. В случае положительного результата функция будет возвращать 1. Поле игры будем хранить в массиве 3×3, элементами которого будут целые числа: 0 – клетка пуста, 1 – крестик, 2 – нолик. Для определения возможного хода достаточно среди всех пустых клеток найти такую, от которой функция Best вернет значение 0. Это всегда возможно, т.к. существует беспроигрышная стратегия. По этой же причине функция никогда не будет возвращать значение 1, т.к. выиграть у программы жюри невозможно.

Задача E. Крестики-нолики

```
//PascalABC.NET 3.2
var x,y,bx,by,p,k : integer;
    a : array[1..3, 1..3] of integer;

function Best(x, y, p: integer): integer;
var i, j, res, p3 : integer;
begin
  res := -2;
  p3 := p*p*p;
  a[y,x] := p;
  if (a[1,x]*a[2,x]*a[3,x] = p3) or
     (a[y,1]*a[y,2]*a[y,3] = p3) or
     (a[1,1]*a[2,2]*a[3,3] = p3) or
     (a[1,3]*a[2,2]*a[3,1] = p3) then
       begin
         a[y,x] := 0;
         Best := 1;
         exit
       end;
  for i:=1 to 3 do
    for j:=1 to 3 do
      if a[i,j]=0 then
        res := max(res, Best(j,i,3-p));
  a[y][x] := 0;
  if res>-2 then Best := -res
            else Best := 0
end;
```

```
begin
  readln(p);
  for k:=0 to 4 do begin
    if (k>0) or (p=1) then begin
      for y:=1 to 3 do
        for x:=1 to 3 do
          if (a[y,x]=0) and (Best(x,y,p)=0)
            then (bx,by) := (x,y);
      a[by,bx] := p;
      writeln(bx, ' ', by);
      flush(output);
      if (k=4) and (p=1) then break
    end;
    readln(x,y);
    a[y,x] := 3-p
  end
end.
```

Задача E. Крестики-нолики

```
//C++
#include <iostream>
#include <algorithm>

using namespace std;

int x,y,bx,by,p,k,a[3][3];

int Best(int x, int y, int p){
  int i, j, p3=p*p*p, res = -2;
  a[y][x] = p;
  if(a[0][x]*a[1][x]*a[2][x] == p3 ||
     a[y][0]*a[y][1]*a[y][2] == p3 ||
     a[0][0]*a[1][1]*a[2][2] == p3 ||
     a[0][2]*a[1][1]*a[2][0] == p3)
      {a[y][x] = 0; return 1;}
  for(i=0; i<3; i++)
    for(j=0; j<3; j++)
      if(a[i][j]==0)
        res = max(res, Best(j,i,3-p));
  a[y][x] = 0;
  return ((res > -2)?-res:0);
}
```

```
int main(){
  cin >> p;
  for(k=0; k<5; k++){
    if(k>0 || p==1){
      for(y=0; y<3; y++)
        for(x=0; x<3; x++)
          if(!a[y][x] && !Best(x,y,p))
            bx=x, by=y;
      a[by][bx] = p;
      cout << bx+1 << " " << by+1 << endl;
      cout.flush();
      if(k==4 && p==1) break;
    }
    cin >> x >> y;
    a[y-1][x-1] = 3-p;
  }
  return 0;
}
```