Олимпиада школьников «Шаг в будущее» по общеобразовательному предмету "Информатика".
2015 год. Отборочный этап. 10-11 классы. Билет 1. Условия и решения.

Задача 1: Числа Стирлинга (10)

Число неупорядоченных разбиений n-элементного множества на k непустых подмножеств задается числом Стирлинга 2-го рода S(n, k). По определению полагают: $S(0, 0) = 1$, $S(0, k) = S(n, 0) = 0$. Очевидно, что $S(n, 1) = 1$, $S(n, n) = 1$, $S(n, k) = 0$ при $k > n$. Основное рекуррентное соотношение для чисел Стирлинга 2-го рода имеет вид: $S(n, k) = S(n\text{-}1, k\text{-}1) + k\,S(n\text{-}1, k)$. Для заданных чисел n и k вычислить число Стирлинга 2-го рода $S(n, k)$.
Входные данные. Входной файл содержит одну строку, в которой записаны два целых числа n и k ($1 \le n, k \le 10^3$).
Выходные данные. В выходной файл вывести одно целое число – значение вычисленного числа Стирлинга 2-го рода.

Примеры входного файла	Примеры выходного файла
1 1	1
5 3	25

Решение задачи 1

```
#include "stdafx.h"

FILE *ifs, *ofs;

unsigned long long Stirling(unsigned int n, unsigned int k)
{
	if (n == 0 && k == 0) return 1;
	else if (n == 0 && k > 0) return 0;
	else if (n > 0 && k == 0) return 0;
	else if (n > 0 && k == 1) return 1;
	else if (n == k && n > 0) return 1;
	else if (n > 0 && k > n) return 0;
	else return Stirling(n-1, k-1) + k * Stirling(n-1, k);
}

int _tmain(int argc, _TCHAR* argv[])
{
	unsigned int n, k;
	if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
	if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
	fscanf( ifs, "%u %u", &n, &k );
	fprintf( ofs, "%llu\n", Stirling(n, k));
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```

Задача 2: Шифрование (20)

Пусть алфавит некоторого языка задан в виде строки "ABCDEFGHIJKLMNOPQRSTUVWXYZ_.", которая в дальнейшем будет называться алфавитной строкой. Для алфавитной строки определим операцию циклического сдвига вправо на n позиций. Например, циклический сдвиг алфавитной строки вправо на 1 (одну) позицию приведет к следующей последовательности действий: 'A'→'B', 'B'→'C', ..., 'Z'→'_', '_'→'.' и '.'→'A'. А циклический сдвиг алфавитной строки вправо на 3 (три) позиции приведет к следующей последовательности действий: 'A'→'D', 'B'→'E', ..., '.'→'C'. Предлагается метод шифрования слов (т. е. строк) над заданным алфавитом, состоящий из двух шагов. На первом шаге выполняется реверс слова, а на втором шаге выполняется циклический сдвиг алфавитной строки вправо на n позиций и символы реверсированного слова заменяются сдвинутыми символами алфавитной строки. Например, строка "ABCD" после реверса примет вид "DCBA", а после циклического сдвига алфавитной строки вправо на 1 (одну) позицию примет вид "EDCB". Реализовать описанный метод шифрования.
Входные данные. Входной файл содержит одну строку, в которой записано целое число n ($1 \le n \le 27$) – параметр сдвига и слово над заданным алфавитом. Длина слова не превышает 40 символов.
Выходные данные. В выходной файл вывести зашифрованное слово.

Примеры входного файла	Примеры выходного файла
1 ABCD	EDCB
3 YO_THERE.	CHUHKWBR.
1 .DOT	UPEA
14 ROAD	ROAD
1 SNQZDRQDUDQ	REVERSE_ROT

Решение задачи 2

```
#include "stdafx.h"
#include <stdlib.h>
#include <string.h>

FILE *ifs, *ofs;
char symb[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ_.";

int _tmain(int argc, _TCHAR* argv[])
{
	char str[80], out_s[80];
	int smesh, b, k;
	size_t i, j;
	if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
	if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
	while ( !feof( ifs ) )
	{
		fscanf( ifs, "%d %s", &smesh, str );
		if (feof( ifs )) break;
		k = 0;
		i = strlen(str);
		while ( b = strchr(symb, str[--i]) && ( i >= 0))
		{
			for (j = 0; j < strlen(symb); j++)
				if (str[i] == symb[j])
				{
					out_s[k++] = symb[(j+smesh)%(strlen(symb))];
				}
		}
		out_s[k] = '\0';
		fprintf( ofs, "%s\n", out_s );
	}
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```

Задача 3: Количество множеств (15)

Рассмотрим множество целых положительных чисел меньших или равных некоторому числу *n*. Обозначим через *k* и *s* количество и сумму элементов множества соответственно. Для $n = 9$, $k = 3$ и $s = 23$ будем иметь одно такое множество: {6, 8, 9}. Для $n = 9$, $k = 3$ и $s = 22$ таких множеств будет уже два: {5, 8, 9} и {6, 7, 9}. Подсчитать, сколько существует различных множеств для заданных *n*, *k* и *s*.

Входные данные. Входной файл содержит одну строку, в которой записаны три целых числа: n ($1 \le n \le 20$) – максимальный элемент множества, k ($1 \le k \le 10$) – количество элементов множества и s ($1 \le s \le 155$) – сумма элементов множества.

Выходные данные. В выходной файл вывести одно целое число – количество полученных множеств.

Примеры входного файла	Примеры выходного файла
9 3 23	1
9 3 22	2
10 3 28	0

Решение задачи 3

```
#include "stdafx.h"

FILE *ifs, *ofs;

int find_all( int max_number, int current_depth, int max_depth, int current_sum, int sum_max )
{
	int found = 0, changed = 0;
	for (int i = max_number; i > 0 && changed == 0; i--)
	{
		current_sum += i;
		if (current_sum == sum_max && current_depth == max_depth)
		{
			found = 1;
			changed = 1;
		}
		else
			if (current_sum < sum_max && current_depth < max_depth)
				found += find_all( i-1, current_depth+1, max_depth, current_sum,
```

```
sum_max );
                current_sum -= i;
        }
        return found;
}

int _tmain(int argc, _TCHAR* argv[])
{
        int n = 0, k = 0, s = 0;
        if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
        if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
        fscanf( ifs, "%d %d %d", &n, &k, &s);
        if (k < s)
        {
                int result = find_all( n, 1, k, 0, s);
                fprintf( ofs, "%d\n", result );
        }
        else
                fprintf( ofs, "0\n" );
        fclose( ifs );
        fclose( ofs );
        return 0;
}
```

Задача 4: Треугольник Паскаля (25)

Сформируем таблицу, в которой *n*-я строка содержит биномиальные коэффициенты степени *n*:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
…

В этой таблице каждое число внутри равно сумме двух чисел: стоящего выше и стоящего выше и левее. Такую таблицу называют треугольником Паскаля. Если в приведенном треугольнике Паскаля заменить 3-ю строку новыми значениями: <13, 2, 5, 7>, то 4-я строка примет вид: <13, 15, 7, 12, 7>, а 2-й элемент в 6-й строке будет равен 50 . Заметим, что нумерация строк и элементов в строке начинается с 0. Определить *k*-е значение в *n*-й строке модифицированного треугольника Паскаля, если заменить существующие значения в *m*-й строке на новые, и продолжить построение треугольника Паскаля обычным образом.

Входные данные. Первая строка входного файла содержит целое число m ($1 \le m \le 50000$) – номер заменяемой строки. Вторая строка входного файла содержит $m + 1$ целых чисел – новые значения в *m*-й строке (каждое число не более 1000). В третьей строке входного файла записаны два целых числа: n ($m < n \le 200000$) – номер строки и k ($0 \le k \le n$) – номер элемента в *n*-й строке.

Выходные данные. В выходной файл вывести одно целое число – *k*-е значение в *n*-й строке модифицированного треугольника Паскаля.

Примеры входного файла	Примеры выходного файла
3 13 2 5 7 6 2	50
0 2 15 6	10010

Решение задачи 4

```
#include "stdafx.h"
#include <stdlib.h>

FILE *ifs, *ofs;
typedef unsigned long long int ULLI;

void pascal(ULLI *arr, int length, int N, int K)
{
        int i;
        ULLI *newarr;
        newarr=(ULLI *)calloc( length+1, sizeof(ULLI) );
        newarr[0]=arr[0];
```

```
	for(i = 1; i < length+1; i++)
		newarr[i]=arr[i]+arr[i-1];
	newarr[i]=arr[i-1];
	if (i == N)
	{
		fprintf( ofs, "%llu\n", newarr[K] );
		return;
	}
	pascal( newarr, i, N, K);
}

int _tmain(int argc, _TCHAR* argv[])
{
	int R, N, K, i;
	ULLI *arr;
	if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
	if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
	fscanf( ifs, "%d", &R );
	arr=(ULLI *)calloc( R+1, sizeof(ULLI) );
	for (i = 0; i < R+1; i++)
		fscanf( ifs, "%llu", &arr[i]);
	fscanf( ifs, "%d %d", &N, &K);
	pascal( arr, R, N, K );
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```

Задача 5: Стражники (30)

Неориентированный граф G определяется как пара (V, E), где V – множество вершин или узлов, а E – множество рёбер. В графе G каждое ребро определяет непрерывную линию, соединяющую две вершины. Дерево T – это связный ациклический граф. Связность означает наличие путей между любой парой вершин, ацикличность – отсутствие циклов и то, что между парами вершин имеется только по одному пути. Карта дорог средневекового города образуют дерево. Требуется расставить минимальное количество стражников в вершины дерева дорог так, чтобы они могли обозревать все дороги.

Входные данные. В первой строке входного файла записано одно целое число N ($3 \le N \le 1500$) – число вершин дерева. В следующих N строках даны описания вершин в следующем формате: v:(m) u_1 u_2 … u_m, где v – номер вершины (номера вершин – целые числа от 0 до N – 1); m – число ребер, не учтенных ранее; u_1 u_2 … u_m – номера вершин, соединенных с вершиной v ребром (учитываются только те ребра, которые не были описаны ранее). Заметим, что каждое ребро появляется во входных данных ровно один раз. Список u_1 u_2 … u_m может быть пустым, если все ребра для вершины v были описаны ранее.

Выходные данные. В выходной файл вывести одно целое число – минимальное количество стражников.

Примеры входного файла	Примеры выходного файла
4 0:(1) 1 1:(2) 2 3 2:(0) 3:(0)	1
5 3:(3) 1 4 2 1:(1) 0 2:(0) 0:(0) 4:(0)	2

Решение задачи 5

```
#include "stdafx.h"
#include <stdlib.h>
#include <string.h>

FILE *ifs, *ofs;

int _tmain(int argc, _TCHAR* argv[])
{
	int **a = NULL; // Матрица смежности
	int n = 0; // Число вершин дерева
	int result = 0; // Минимальное число стражников
	// Открытие файлов
```

```
	if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
	if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
	// Чтение числа вершин дерева (размера матрицы смежности)
	fscanf( ifs, "%d", &n );
	// Выделение памяти для матрицы смежности с помощью функции malloc
	a = (int **)malloc( n * sizeof(int *) );
	for(int i = 0; i < n; i++)
		a[i] = (int *)malloc(n * sizeof(int));
	// Заполнение матрицы смежности нулями
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
			a[i][j] = 0;
	// Чтение исходных данных и формирование матрицы смежности
	for (int i = 0; i < n; i++)
	{
		int v_num = 0; // Номер вершины, для которой считываются данные
		char ch1, ch2, ch3;
		int num_connections = 0; // Количество связанных с вершиной ребер
		fscanf( ifs, "%d %c %c %d %c", &v_num, &ch1, &ch2, &num_connections, &ch3 );
		for (int i = 0; i < num_connections; i++)
		{
			int j = 0; // Номер связанной вершины
			fscanf( ifs, "%d", &j);
			a[v_num][j] = 1;
			a[j][v_num] = 1;
		}
	}
	// Подсчет минимального числа стражников
	int done = 1;
	int imax, max_sum, sum;
	while (1)
	{
		imax = -1;
		max_sum = 0;
		for (int i = 0; i < n; i++)
		{
			sum = 0;
			for (int j = 0; j < n; j++)
				sum +=a[i][j];
			if (sum > max_sum)
			{
				max_sum = sum;
				imax = i;
			}
		}
		if (max_sum == 0)
			break;
		else
		{
			for (int j = 0; j < n; j++)
				if (a[imax][j] == 1)
					a[j][imax] = 0;
			for (int j = 0; j < n; j++)
				a[imax][j] = 0;
			result++;
		}
	}
	// Печать ответа
	fprintf( ofs, "%d\n", result );
	// Освобождение памяти, занятой матрицей смежности
	for (int i = 0; i < n; i++)
		free( a[i] );
	free( a );
	//Закрытие файлов
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```

Олимпиада школьников «Шаг в будущее» по общеобразовательному предмету "Информатика".
2015 год. Отборочный этап. 10-11 классы. Билет 2. Условия и решения.

Задача 1: Числа Белла (10)

Количество всех неупорядоченных разбиений n-элементного множества задается числом Белла B_n. По определению полагают: $B_0 = 1$. Основное рекуррентное соотношение для чисел Белла имеет вид:

$$B_n = \sum_{i=0}^{n-1} C_{n-1}^{i} B_i$$

Для заданного числа n вычислить число Белла.
Входные данные. Во входном файле записано одно целое число n ($1 \le n \le 25$).
Выходные данные. В выходной файл вывести одно число – значение вычисленного числа Белла.

Примеры входного файла	Примеры выходного файла
1	1
5	52

Решение задачи 1

```
#include "stdafx.h"

FILE *ifs, *ofs;

unsigned long long factorial(unsigned int n)
{
	unsigned long long p = 1;
	for (unsigned int i = 1; i <= n; i++) p *= i;
	return p;
}
unsigned long long Cnk(unsigned int n, unsigned int k)
{
	unsigned long long p = 1;
	for (unsigned int i = k+1; i <= n; i++) p *= i;
	return p / factorial(n-k);
}
unsigned long long Bell(unsigned int n)
{
	if (n == 0) return 1;
	else
	{
		unsigned long long s = 0;
		for (unsigned int i = 0;  i < n; i++)
			s = s + Cnk(n-1, i) * Bell(i);
		return s;
	}
}
int _tmain(int argc, _TCHAR* argv[])
{
	unsigned int n;
	if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
	if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
	fscanf( ifs, "%u", &n );
	fprintf( ofs, "%llu\n", Bell(n) );
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```

Задача 2: Допустимый пароль (20)

Рассматривается пароль, состоящий не более чем из 20 (двадцати) строчных букв английского алфавита. Пароль считается «допустимым», если одновременно выполняются следующие условия: а) содержит по крайней мере одну гласную букву; б) не содержит последовательность из 3 (трех) идущих подряд гласных или согласных букв; в) не содержит последовательность из 2 (двух) одинаковых букв, кроме "ee" и "oo". Гласными буквами следует считать следующие 5 (пять) букв: a, e, i, o, u. Все остальные буквы следует считать согласными. Определить допустимость коллекции паролей.
Входные данные. Первая строка входного файла содержит целое число N ($1 \le N \le 1000$) – количество паролей. В последующих N строках записаны пароли.
Выходные данные. В выходной файл для каждого пароля во входном файле вывести "YES", если пароль допустимый, или "NO", если пароль не допустимый.

Пример входного файла	Пример выходного файла
5 ptoui bontres wiinq a eep	NO NO NO YES YES

Решение задачи 2

```
#include "stdafx.h"
#include <string.h>

FILE *ifs, *ofs;

int isVowel(char c)
{
    if(c == 'a' || c == 'e' || c == 'i' || c == 'u' || c == 'o') return 1;
    else return 0;
}

int isAccept(char *str)
{
       int b = 0;
       for(size_t i = 0; i < strlen(str); i++)
              if(isVowel(str[i]))
              {
                     b = 1;
                     break;
              }
       if(b)
       {
              for(size_t i = 1; i < strlen(str); i++)
                     if(str[i]==str[i-1] && str[i]!='e' && str[i]!='o')
                     {
                            b = 0;
                            break;
                     }
              if(b)
              {
                     for(size_t i = 2; i < strlen(str); i++)
                     {
                            if(isVowel(str[i-2]) && isVowel(str[i-1]) && isVowel(str[i]))
                            {
                                   b = 0;
                                   break;
                            }
                            if(!isVowel(str[i-2]) && !isVowel(str[i-1]) && !isVowel(str[i]))
                            {
                                   b = 0;
                                   break;
                            }
                     }
                 if(b) return 1;
                 else return 0;
                  }
          else return 0;
       }
    else return 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
       int n;
       char a[32];
       if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
       if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
       fscanf( ifs, "%d", &n );
       for (int i = 0; i < n; i++)
       {
              fscanf( ifs, "%s", &a );
              if(isAccept( a )) fprintf( ofs, "YES\n" );
              else fprintf( ofs, "NO\n" );
```

```
	}
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```

Задача 3: Последовательность (15)

Последовательность чисел G_1, G_2, …G_i, … задается следующими условиями: а) G_1 и G_2 – произвольные целые числа ($0 < G_1 \le G_2$); б) $G_i = G_{i-1} + G_{i-2}$ для $i > 2$. Например, для $G_1 = 1$ и $G_2 = 3$ будем иметь такую последовательность чисел: 1, 3, 4, 7, 11, 18, 29, … . Подбирая соответствующим образом G_1 и G_2 можно получить последовательность чисел, содержащую любое наперед заданное целое положительное число n. Для заданного целого положительного числа n найти такие G_1 и G_2, которые будут наименьшими возможными целыми числами, удовлетворяющими условию (а).

Входные данные. Во входном файле записано одно целое число n ($2 \le n \le 10^9$) – число, которое должно появиться в последовательности.

Выходные данные. В выходной файл вывести одну строку, в которой записаны два целых числа a и b ($0 < a \le b$), такие, что $G_1 = a$, $G_2 = b$ и $G_k = n$ для некоторого k. Числа a и b должны быть как можно меньшими.

Примеры входного файла	Примеры выходного файла
89	1 1
123	1 3
1000	2 10

Решение задачи 3

```
#include "stdafx.h"

FILE *ifs, *ofs;

int _tmain(int argc, _TCHAR* argv[])
{
	int Num, Pred, Pred2, O1, O2, Min, Dub, i=0;
	if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
	if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
	fscanf( ifs, "%d", &Num );
	Min = Num;
	Pred = Num/2;
	O1 = 1;
	O2 = Num-1;
	while (Pred < Num)
	{
		Pred = Num/2+i;
		Pred2 = Num;
		while (1)
		{
			Dub = Pred;
			if (Dub < Pred2 - Pred)
				break;
			if (Pred2 - Pred > 0)
			{
				Pred = Pred2 - Pred;
				Pred2 = Dub;
			}
			else
				break;
		}
		if (Pred + Pred2 < Min)
		{
			O1 = Pred;
			O2 = Pred2;
			Min = Pred + Pred2;
		}
		i++;
	}
	fprintf( ofs, "%d %d\n", O1, O2);
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```

Задача 4: Карточная игра (25)

В игре используется колода из N карт. Каждая карта помечена целым положительным числом. В каждом туре игры удаляется ровно одна карта (кроме первой и последней). За каждую удаленную карту начисляются очки - произведение метки удаляемой карты на сумму меток двух соседних карт. Заканчивается игра, когда остаются две карты - первая и последняя. Суть игры состоит в том, чтобы набрать максимальное число очков.
Входные данные. В первой строке входного файла записано одно целое число N ($3 \le N \le 700$) – количество карт в колоде. Во второй строке входного файла записаны N целых положительных чисел, которыми помечены карты (метка ≤ 1000).
Выходные данные. В выходной файл вывести одно целое число – максимальное число набранных очков.

Пример входного файла	Пример выходного файла
4 4 5 6 2	86

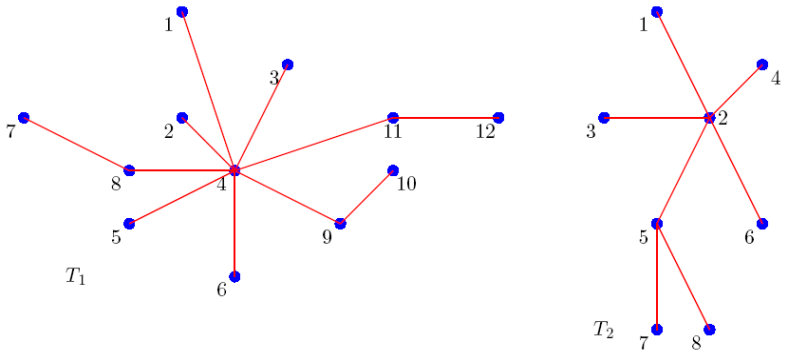
Решение задачи 4

```
program Project_4;

{$APPTYPE CONSOLE}

uses
        SysUtils;

type
        arr=array[1..1000] of integer;

procedure find(mas: arr; var max, max_i: integer; n: integer);
var
        s, i: integer;
begin
        max:=0;
        for i:=2 to n-1 do
        begin
                s:=(mas[i-1]+mas[i+1])*mas[i];
                if max<s then
                begin
                        max:=s;
                        max_i:=i;
                end;
        end;
end;

procedure shift(var mas: arr; max_i, n: integer);
var
        i: integer;
begin
        for i:=max_i to n-1 do
                mas[i]:=mas[i+1];
end;

var
        ifs, ofs: textfile;
        mas: arr;
        n, max, max_i, sum, i: integer;

begin
        assign(ifs, 'in.txt');
        reset(ifs);
        readln(ifs, n);
        i:=0;
        while not(eoln(ifs)) do
        begin
                inc(i);
                read(ifs, mas[i]);
        end;
        close(ifs);
        sum:=0;
        while n<>2 do
        begin
                find(mas, max, max_i, n);
                sum:=sum+max;
                shift(mas, max_i, n);
                dec(n);
        end;
        assign(ofs, 'out.txt');
```

```
        rewrite(ofs);
        writeln(ofs, sum);
        close(ofs);
end.
```

Задача 5: Диаметр дерева (30)

Неориентированный граф G определяется как пара (V, E), где V – множество вершин или узлов, а E – множество рёбер. В графе G каждое ребро определяет непрерывную линию, соединяющую две вершины. Дерево T – это связный ациклический граф. Связность означает наличие путей между любой парой вершин, ацикличность – отсутствие циклов и то, что между парами вершин имеется только по одному пути. Диаметр дерева – максимальная длина (в рёбрах) кратчайшего пути в дереве между любыми двумя вершинами. На приведенном ниже рисунке диаметр дерева T_1 равен 4, а диаметр дерева T_2 равен 3.



Для заданного дерева с N вершинами, пронумерованными целыми числами от 1 до N, вычислить его диаметр.
Входные данные. В первой строке входного файла записано одно целое число N ($3 \le N \le 50$) – число вершин дерева. В следующих (N – 1) строках записаны номера вершин дерева, соединенных ребром.
Выходные данные. В выходной файл вывести одно целое число – диаметр дерева.

Примеры входного файла	Примеры выходного файла
12 1 4 2 4 3 4 7 8 8 4 4 9 9 10 4 11 11 12 4 5 4 6	4
8 1 2 3 2 4 2 5 2 6 2 5 7 5 8	3

Решение задачи 5

```
#include "stdafx.h"
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

FILE *ifs, *ofs;

int _tmain(int argc, _TCHAR* argv[])
{
       size_t graphs_count = 0;
       size_t n;
       if ( (ifs = fopen( "in1.txt", "r" )) == NULL ) return 1;
       if ( (ofs = fopen( "out1.txt", "w" )) == NULL ) return 1;
       fscanf( stdin, "%d", &n);
```

```
	++graphs_count;
	size_t **graph = (size_t **)malloc(n * sizeof(size_t *));
	for(int i = 0; i < n; i++)
		graph[i] = (size_t *)malloc(n * sizeof(size_t));
	size_t *graph_sizes = (size_t *)malloc(n * sizeof(size_t));
	for (size_t i = 0; i < n-1; ++i)
	{
		size_t from, to;
		fscanf( stdin, "%d %d", &from, &to);
		--from;
		--to;
		graph[from][graph_sizes[from]++] = to;
		graph[to  ][graph_sizes[to  ]++] = from;
	}
	size_t root = 0;
	size_t *distances = (size_t *)malloc(n * sizeof(size_t));
	size_t edge_vertex = 0, max_distance = 0;
	int *visited = (int *)malloc(n * sizeof(int));
	size_t *queue = (size_t *)malloc(n * sizeof(size_t));
	size_t queue_size = 1;
	queue[0] = 0;
	while (queue_size != 0)
	{
		size_t current_vertex = queue[0];

		for (size_t i = 1; i < queue_size; ++i)
			queue[i-1] = queue[i];
		--queue_size;
		visited[current_vertex] = 1;
		size_t distance = distances[current_vertex];
		if (max_distance < distance)
		{
			max_distance = distance;
			edge_vertex  = current_vertex;
		}
		for (size_t i = 0; i < graph_sizes[current_vertex]; ++i)
		{
			size_t followed_vertex = graph[current_vertex][i];
			if (!visited[followed_vertex])
			{
				queue[queue_size++]        = followed_vertex;
				distances[followed_vertex] = distance+1;
			}
		}
	}

	visited = (int *)malloc(n * sizeof(int));
	distances = (size_t *)malloc(n * sizeof(size_t));
	queue[0] = edge_vertex;
	queue_size = 1;
	while (queue_size != 0)
	{
		size_t current_vertex = queue[0];
		for (size_t i = 1; i < queue_size; ++i)
				queue[i-1] = queue[i];
		--queue_size;
		visited[current_vertex] = 1;
		size_t distance = distances[current_vertex];
		max_distance = max_distance > distance ? max_distance : distance;
		for (size_t i = 0; i < graph_sizes[current_vertex]; i++)
		{
			size_t followed_vertex = graph[current_vertex][i];
			if (!visited[followed_vertex])
			{
				queue[queue_size++]        = followed_vertex;
				distances[followed_vertex] = distance+1;
			}
		}
	}
	fprintf( stdout, "%d %d\n", graphs_count, max_distance);
	fclose( ifs );
	fclose( ofs );
	return 0;
}
```