Ответы и критерии оценивания заданий заключительного этапа олимпиады школьников "Ломоносов" по информатике 2014/2015 учебного года

5-9 классы

Задача 1. Системы счисления.

Быстрое решение:

Поскольку $3^3=27$, можно сразу переводить из троичной системы в 27-ричную. Для этого разобъём запись числа в троичной системе на тройки цифр, начиная с самой правой цифры: 2 211 201 122 110 201

Теперь каждую триаду переведём из троичной в десятичную систему счисления:

2 22 19 17 12 19

В 27-ричной системе 22 соответствует цифре M, 19 – J, 17 – H, 12 – C, следовательно ответ записывается так: $2MJHCJ_{27}$

Медленное решение:

Решать обычным способом, используя как промежуточную десятичную систему, медленно, неудобно и рискованно из-за возможных арифметических ошибок в расчётах. Всё же рассмотрим этом метод. Для перевода числа в десятичную систему воспользуемся схемой Горнера, позволяющей сэкономить на вычислениях:

$2211201122110201_3 =$

$=(2\times3+2)\times3+1)\times3+1)\times3+2)\times3+0)\times3+1)\times3+1)\times3+2)\times3+2)\times3+1)\times3+1)\times3+0)\times3+2)\times3+0)\times3+1$

$=40776229_{10}$

Переводим число из десятичной системы в 27-ричную. Для этого находим остаток и частное от деления нацело числа 40776229 на 27, далее повторяем деление для полученного частного и т. д. Получаем 19, 12, 17, 19, 22 и 2. Выписываем в обратном порядке цифрами 27-ричной системы: $2MJHCJ_{27}$

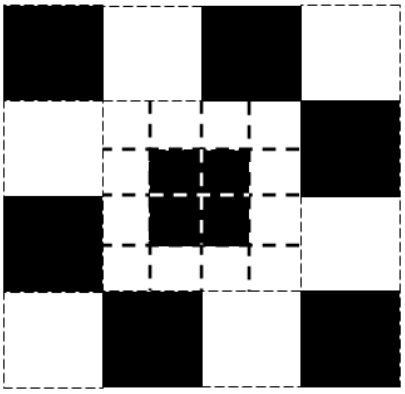
Ответ к задаче 1: $2MJHCJ_{27}$

Критерий к задаче 1: Полный балл ставился за верный ответ с правильным обоснованием.

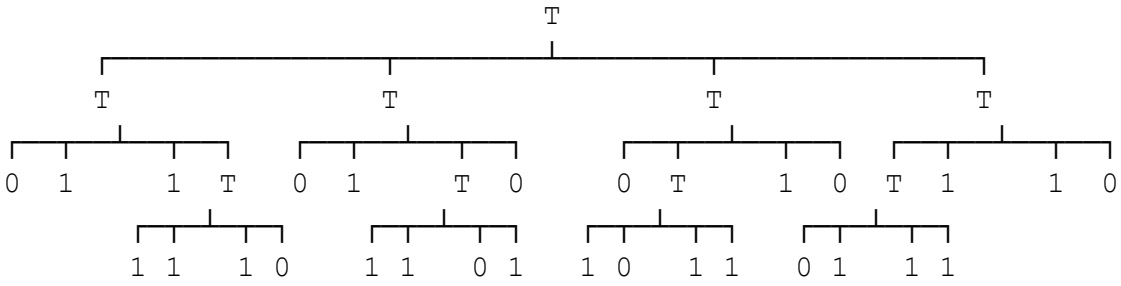
Задача 2. Квадродерево. Решение

Чтобы решить задачу, нужно разбить исходное изображение на квадраты по принципу, на котором основано квадродерево, затем построить квадродерево в виде графа, после чего составить линейную запись построенного дерева.

После разбиения изображение выглядит так, как показано на рисунке:



Граф квадродерева выглядит так:



Линейная запись всего квадродерева состоит из следующих частей:

T<запись 1-го поддерева><запись 2-го п/д><запись 3-го п/д><запись 4-го п/д>

Находим запись первого поддерева: T011T1110

Находим запись второго поддерева: T01T11010

Находим запись третьего поддерева: T0T101110

Находим запись четвертого поддерева: TT0111110

По найденным записям составляем запись ответа:

TT011T1110T01T11010T0T101110TT0111110

Ответ к задаче 2: TT011T1110T01T11010T0T101110TT0111110

Критерий к задаче 2: Полный балл ставился за верный ответ с правильным обоснованием.

Задача 3. Крестики-нолики. Решение

Рассмотрим различные способы записи партий, чтобы выбрать из них самый короткий, как этого требуется в условии задачи.

Первый очевидный способ строится по тому как партия представлена в условии. Там дана последовательность снимков игрового поля, сделанных после очередного хода. Поле состоит из девяти клеток. Каждая клетка может быт либо пуста, либо заполнена X, либо 0. Значит, для кодирования клетки можно использовать два бита: 00 – пустая клетка; 01 – клетка с X, 10 – клетка с 0. Снимок поля составляется из записей девяти клеток (например, следующих по горизонталям от верхней к нижней, а внутри горизонтали слева направо). Запись снимка поля после первого хода:010000000000000000 занимает 18 бит Запись партии из условия по первому способу:

010000000000000000010000001000000000010000001000000000101000000100010000101000100010001000010110010010001000010110010010001001010110010010101001010

Она занимает 8 ×18 = 144 бит.

Попробуем найти более короткий способ. Заметим, что в записи по первому способу много места отводится под пустые клетки, и многократно дублируются сведения о ранее сделанных ходах. Что если записывать партию сделанными ходами? Каждый ход будем записывать, указывая клетку, в которую поставлен символ. Для этого занумеруем клетки:

```
0|1|2
-+-+-
3|4|5
-+-+-
6|7|8
```

Указывать, какой именно ставится символ необязательно, так как порядок, по которому делают ходы игроки, единый для всех партий. На первом ходу всегда ставится X, на втором – 0, на третьем – X и т. д. Ход будем кодировать номером клетки, записанным в двоичной системе. На каждый ход нам потребуется 4 бита, что много меньше, чем 18 бит для снимка поля. Запись партии по второму способу:

00000100100001100010000101110101 занимает 8 × 4 = 32 бита, что много меньше, чем в первом способе.

Можно ли улучшить второй способ? Если бы нам удалось не использовать восьмёрку, то каждый ход занимал бы 3 бита, что дало бы существенную экономию. Поступим так. Будем записывать ход в верхнюю левую клетку и ход в правую нижнюю клетку одним и тем же кодом: 000. Чтобы различать какой из этих двух ходов сделан в партии раньше, добавим дополнительный бит в конец записи. Ноль будет означать, что когда 000

встретится в записи в первый раз, это следует прочесть как ход в верхнюю левую клетку, а во второй раз – в нижнюю правую. Единица будет указывать на обратное. Запись партии по улучшенному второму способу: 000100000110010001111010 занимает 25 бит.

Можно ли записывать ещё короче? Заметим, что последнее усовершенствование использовало тот факт, что повторно сделать ход в уже занятую клетку нельзя. Значит, записывая ходы в конце партии, можно сэкономить за счёт указания номера клетки, но не одной из девяти клеток поля, а одной из оставшихся свободными клеток, которых меньше, чем 9. Чтобы нумеровать свободные клетки будем рассматривать клетки также как при предыдущей нумерации, но занятые клетки будем пропускать, не давая им номера. Например, после пятого хода в заданной партии останется четыре свободные клетки, занумерованные так (ходы второго игрока обозначены символом O, чтобы отличать от свободной клетки с номером *0*):

```
X|0|X
-+-+-
1|O|2
-+-+-
O|3|X
```

Таким образом, для первого хода будет затрачено четыре бита, для ходов со второго по пятый – три бита, для шестого и седьмого ходов – два бита, для восьмого хода – один бит. На девятом ходу остаётся единственная клетка, но ход либо будет сделан и партия закончится (0), либо не будет сделан (1). Значит, тоже нужен один бит. Запись заданной партии займёт 4 + 4 × 3 + 2 × 2 + 2 = 22 бита. Для удобства составления этой записи приведём нумерацию свободных клеток после первого, второго и других ходов (ходы второго игрока по-прежнему обозначены символом O, чтобы отличать от свободной клетки с номером *0*)

```
X|0|1   X|0|1   X|0|1   X|0|1   X|0|X   X|O|X   X|O|X   X|O|X
-+-+-   -+-+-   -+-+-   -+-+-   -+-+-   -+-+-   -+-+-   -+-+-
2|3|4   2|O|3   2|O|3   2|O|3   1|O|2   0|O|1   0|O|1   0|O|O
-+-+-   -+-+-   -+-+-   -+-+-   -+-+-   -+-+-   -+-+-   -+-+-
5|6|7   4|5|6   4|5|X   O|4|X   O|3|X   O|2|X   O|X|X   O|X|X
```

Полученная запись: 000001111010000100 1011

Ещё один способ заключается в том, чтобы перенумеровать все возможные партии и сделать записью партии её номер. Всего может быть 1 партия, в которой не сделано ни одного хода, 9 партий длиной в 1 ход, 9×8 партий длиной в 2 хода, 9×8×7 партий длиной в 3 хода, 9×8×7×6 партий длиной в 4 хода, 9×8×7×6×5 партий длиной в 5 ходов. Партий длиной в 6 ходов не более 9×8×7×6×5×4 (не более, потому, что в некоторых

партиях после 5 хода наступает победа первого игрока и они не могут быть продолжены). Партий длиной в 7 ходов не более 9×8×7×6×5×4×3 (по тем же обстоятельствам, к которым добавляются партии выигранные вторым игроком на 6-м ходу). Партий длиной в 8 ходов не более 9! (часть партий не может быть продолжена после 5-го, 6-го или 7-го ходов). Партий длиной в 9 ходов не более 9! (часть партий не может быть продолжена после 5-го, 6-го, 7-го или 8-го ходов). Оценить сверху количество всевозможных партий можно, сложив все эти числа. Сумма равна 1172850 < 2^{21} = 2097152. Значит для записи номера партии достаточно 21 бита. Выберем такой способ нумерации, чтобы партия из условия была первой, тогда её запись будет такой: 00000000000000000001. Неважно какие номера у остальных партий, ведь в задаче требуется найти запись лишь для одной заданной.

Реальное количество законченных и незаконченных партий меньше найденной нами грубой оценки сверху 1172850. 1440 партий завершаются на пятом ходу, 5328 – на шестом, 47952 – на седьмом, 72576 – на восьмом, 81792 – на девятом. То есть количество партий длиной 6 ходов (завершённых и незавершенных) составляет 54720, длиной 7 ходов (завершённых и незавершенных) – 148176, длиной 8 ходов (завершённых и незавершенных) – 154368. Общее количество завершённых и незавершённых партий составляет 457786. Получается, что номер партии занимает 19 бит. В качестве записи заданной в условии партии можно указать любое число от 0 до 457785, записанное в двоичной системе в 19 битах.

При составлении задачи предполагалось, что грубой оценки сверху для решения задачи достаточно.

Критерий к задаче 3: Полный балл ставился за способ записи, обеспечивающий однозначность расшифровки, и использующий не более 32 бит для записи партии из условия задачи.

Задача 4. Счетные палочки. Решение

Идея решения заключается в том, чтобы сначала посимвольно считать набор палочек первого игрока, а затем набор палочек второго. При чтении в массиве N из пяти элементов-чисел нужно подсчитывать для первого игрока количество встретившихся палочек каждого типа (в N[i] должно оказаться количество палочек i-го типа, i = 1,5, массив N проинициализирован нулями). Аналогично в массиве M нужно подсчитать палочки второго игрока. Заметим что хранить целиком данные наборы палочек не следует, так только понапрасну расходуется память. Если язык позволяет, можно сразу подсчитать суммы очков. Иначе, следующим шагом будет “нормализация” массивов N и

M. Мы как-бы обмениваем палочки младших типов на палочки старших типов, учитывая их номиналы – количества обозначаемых палочками очков. То есть, обмениваем по пять палочек первого типа на одну палочку второго типа до тех пор пока в N[1] и в M[1] не останется меньше пяти палочек. Аналогично палочки второго типа обмениваются на палочки третьего (по курсу пять к одной), палочки третьего типа обмениваются на палочки четвертого (по курсу десять к одной), палочки четвёртого типа обмениваются на палочки пятого (по курсу десять к одной). После “нормализации” мы можем сравнивать массивы N и M поэлементно, не вычисляя сумму очков. Сравнение начинается с пятых элементов массива, если они не равны, то победил тот игрок, у которого палочек пятого типа больше. Иначе сравниваем четвёртые элементы по тому же принципу и т. д.. Когда сравнение доходит до первых элементов, в случае их равенства выводится результат 0, иначе номер игрока, у которого больше палочек первого типа (следовательно, и очков).

Фрагменты программы на FreePascal:

```
program Task4 (input, output);
var     rate : array [1..5] of integer = (5, 5, 10, 10, 1); {массив для “конвертации” палочек}
        N, M : array [1..5] of longint;
        i: integer;
        res : byte;
        ch : char;
begin
        for i:=1 to 5 do begin
                N[i]:= 0;
                M[i]:= 0
        end; {for}
        {посимвольный ввод палочек первого игрока в массив N}
        while (not eoln) do begin
                read(ch);
                case ch of
                ‘1’, ‘2’, ‘3’, ‘4’, ‘5’ : N[ord(ch) - ord(‘0’)]:= N[ord(ch) - ord(‘0’)] + 1
                end; {case}
        end; {while}
        readln;  {переход на следующую строку}
        {посимвольный ввод палочек второго игрока в массив M}
        while (not eoln) do begin
                read(ch);
```

```
            case ch of
            '1', '2', '3', '4', '5' : M[ord(ch) - ord('0')]:= M[ord(ch) - ord('0')] + 1
            end; {case}
        end; {while}
        for i:=1 to 4 do begin {"нормализация" массивов}
            N[i+1]:= N[i+1]+N[i] div rate[i];
            N[i]:= N[i] mod rate[i];
            M[i+1]:= M[i+1]+M[i] div rate[i];
            M[i]:= M[i] mod rate[i];
        end; {for}
        i:= 5;
        while (i > 1) and (N[i] = M[i]) do i:= i - 1; {поэлементное сравнение массивов}
        res:= 0;
        if (N[i] > M[i]) then res:= 1 else if (N[i] < M[i]) then res:= 2;
        write(res)
end.
```

Критерий к задаче 4: Полный балл ставился программу (текстовое описание алгоритма или блок-схему), решающую задачу, не содержащую логических и/или программных ошибок.

10-11 классы

Результаты технической проверки работ по задачам можно найти на странице участника http://ejudge.cs.msu.ru/new-client?contest_id=21 по логину и паролю, выданным на олимпиаде.

Задача A.

Задача в обоих вариантах оценивалась по набору из 13 тестов. Чем больше тестов пройдено программой, тем больший балл она получает за этой задание. Пример решения:

```
#include <iostream>
#include <sstream>
#include <vector>
#include <algorithm>
```

```
using namespace std;

int n;
std::vector<int> cost(10);
std::vector<int> sum;
vector<int> ans;
string s;

int main() {
    cin >> n;
    for (int i = 0; i < 10; ++i) {
        cin >> cost[i];
    }
    getline(cin, s);
    for (int i = 0; i < n; ++i) {
        getline(cin, s);
        stringstream scin(s);
        int tmp;
        int cs = 0;
        while (scin >> tmp) {
            cs += cost[tmp];
        }
        sum.push_back(cs);
    }

    int gmax = *min_element(sum.begin(), sum.end()); // or max_element()

    for (int i = 0; i < n; ++i) {
        if (sum[i] == gmax) {
            ans.push_back(i);
        }
    }

    for (int i = 0; i < (int)ans.size(); ++i) {
        cout << ans[i] + 1 << " ";
    }
    return 0;
}
```

Задача B.

Ответ в варианте 1: 542 2694833998866
Ответ в варианте 2: 639 633238162450

Для получения полного балла требовалось обоснование ответа. В случае его отсутствия за подсчет лишь одного из деревьев ставился неполный балл.

Задача C.

См. решение задачи 3 для 5-9 классов.

При проверке учитывалась корректность функций и длина получающегося кода. В зависимости от длины получается полный или частичный балл. Так же частично оценивались решения с в корректным кодированием и ошибками раскодирования. Решения, которые кодировали только один ход, считались неверными.

Задача D.

Пример одной из правильных программ.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <cstdlib>
#include <cstring>
#include <cassert>

using namespace std;

const int MAXN = 1000;

const int SEEN = 1;
const int AVAILABLE = 2;
const int NOT_AVAILABLE = 4;
const int USED = 8;

int field[MAXN * 2 + 1][MAXN * 2 + 1];
const int DX[] = {-1, 0, 1, 0};
const int DY[] = {0, 1, 0, -1};
```

```
const char *DC = "LURD";

bool used[MAXN * 2 + 1][MAXN * 2 + 1];

int fly_cnt;

bool
check_pos(int x, int y, int op)
{
   return field[x][y] & op;
}

void
set_pos(int x, int y, int op)
{
   field[x][y] |= op;
}

void
analyze_cur_position(int x, int y)
{
   for (int dy = 1; dy >= -1; --dy) {
       for (int dx = -1; dx <= 1; ++dx) {
           int cur_x = x + dx, cur_y = y + dy;
           int cur_bit;
           set_pos(cur_x, cur_y, SEEN);
           cin >> cur_bit;
           if (cur_bit) {
               assert(!check_pos(cur_x, cur_y, NOT_AVAILABLE));
               set_pos(cur_x, cur_y, AVAILABLE);
           } else {
               assert(!check_pos(cur_x, cur_y, AVAILABLE));
               set_pos(cur_x, cur_y, NOT_AVAILABLE);
           }
       }
   }
}

void
land(int x, int y)
{
   if (fly_cnt > 0) {
       fly_cnt = 0;
       cout << 'P' << endl;
```

```
        analyze_cur_position(x, y);
    }
}

void
make_move(int dir, int x, int y)
{
    cout << DC[dir] << endl;
    analyze_cur_position(x, y);
    ++fly_cnt;
    fly_cnt %= 8;
}

int
get_reverse_dir(int cur_dir)
{
    return (cur_dir + 2) % 4;
}

void
dfs(int x, int y, int last_move_dir = -1)
{
    set_pos(x, y, USED);
    for (int dir = 0; dir < 4; ++dir) {
        int cur_x = x + DX[dir];
        int cur_y = y + DY[dir];
        assert(check_pos(cur_x, cur_y, SEEN));
        if (check_pos(cur_x, cur_y, AVAILABLE) && !check_pos(cur_x, cur_y, USED)) {
            make_move(dir, cur_x, cur_y);
            dfs(cur_x, cur_y, dir);
        }
    }
    if (last_move_dir != -1) {
        int back_dir = get_reverse_dir(last_move_dir);
        x += DX[back_dir], y += DY[back_dir];
        make_move(back_dir, x, y);
    }
}

int
get_dir(char char_dir)
{
    return find(DC, DC + 4, char_dir) - DC;
```

```
}

bool
fly_diagonal(int x, int y, const char *fly_pattern)
{
   for (int i = 0; i < 4; ++i) {
       int cur_dir = get_dir(fly_pattern[i]);
       x += DX[cur_dir];
       y += DY[cur_dir];
       make_move(cur_dir, x, y);
   }
   for (int dir = 0; dir < 4; ++dir) {
       int next_x = x + DX[dir], next_y = y + DY[dir];
       if (!used[next_x][next_y] && !check_pos(next_x, next_y, USED) &&
check_pos(next_x, next_y, AVAILABLE)) {
           // second continent! YAHOO!!!
           make_move(dir, next_x, next_y);
           land(next_x, next_y);
           return true;
       }
   }
   for (int i = 3; i >= 0; --i) {
       int cur_dir = get_reverse_dir(get_dir(fly_pattern[i]));
       x += DX[cur_dir];
       y += DY[cur_dir];
       make_move(cur_dir, x, y);
   }
   return false;
}

bool
fly_straight(int x, int y, const int dir)
{
   for (int i = 0; i < 4; ++i) {
       x += DX[dir];
       y += DY[dir];
       make_move(dir, x, y);
       for (int ndir = 0; ndir < 4; ++ndir) {
           int next_x = x + DX[ndir], next_y = y + DY[ndir];
           if (!used[next_x][next_y] && !check_pos(next_x, next_y, USED) &&
check_pos(next_x, next_y, AVAILABLE)) {
               // second continent! YAHOO!!!
               make_move(ndir, next_x, next_y);
```

```
                land(next_x, next_y);
                return true;
            }
        }
    }
    for (int i = 0; i < 4; ++i) {
        int back_dir = get_reverse_dir(dir);
        x += DX[back_dir];
        y += DY[back_dir];
        make_move(back_dir, x, y);
    }
    return false;
}

bool
search_dfs(int x, int y, int last_move_dir = -1)
{
    used[x][y] = 1;
    for (int dir = 0; dir < 4; ++dir) {
        int next_x = x + DX[dir], next_y = y + DY[dir];
        if (check_pos(next_x, next_y, USED) && !used[next_x][next_y]) {
            make_move(dir, next_x, next_y);
            bool ret = search_dfs(next_x, next_y, dir);
            if (ret) {
                return true;
            }
        }
    }
    for (int dir = 0; dir < 4; ++dir) {
        land(x, y);
        if (fly_straight(x, y, dir)) {
            return true;
        }
    }
    const char *moves[] =
    {
        "LLDD",
        "LLUU",
        "RRDD",
        "RRUU"
    };
    for (int i = 0; i < 4; ++i) {
        land(x, y);
```

```
        if (fly_diagonal(x, y, moves[i])) {
            return true;
        }
    }
    if (last_move_dir != -1) {
        int back_dir = get_reverse_dir(last_move_dir);
        x += DX[back_dir], y += DY[back_dir];
        make_move(back_dir, x, y);
    }
    return false;
}

int
main()
{
    ios_base::sync_with_stdio(false);
    for (int i = 0; i < MAXN * 2 + 1; ++i) {
        for (int j = 0; j < MAXN * 2 + 1; ++j) {
            field[i][j] = 0;
        }
    }
    // make start point in the center of the field
    int root_x = MAXN, root_y = MAXN;
    fly_cnt = 0;
    // traverse all available cells - cells of the first continent
    analyze_cur_position(root_x, root_y);
    dfs(root_x, root_y);
    // land kvadrocopter if it in the air now
    land(root_x, root_y);
    bool res = search_dfs(root_x, root_y);
    assert(res);
    cout << 'W' << endl;
    return 0;
}
```

Задача E.

Данная функция возвращает -1 и числа от 2 до N, квадраты которых меньше 1024. Поэтому правильный ответ: 31. За ответ 32 и 33 ставились частичные баллы (не учтено, что функция не возвращает 0 и не возвращает 1). Для получения полного балла требовалось обосновать ответ.

Задача F.

Ответ: acbzefd.

Данная программа выводит следующую перестановку.

```
#include <iostream>
#include <algorithm>

int main() {
   std::string s;
   std::getline(std::cin, s);
   std::next_permutation(s.begin(), s.end());
   std::cout << s << std::endl;
}
```

Частичные баллы давались за неполное описание работы программы.