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

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.