Олимпиада СПбГУ по информатике 2019/20 учебного года

A	B	C	D	E	F	Sum
100	100	100	40	0	0	340

Task A ()

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

typedef int32_t i32;
typedef int64_t i64;

const i32 INF32 = 1000000000;
const i64 INF64 = 1000000000000000000;

using namespace std;

i32 main() {
    #ifdef DEBUG
    freopen("input.txt", "r", stdin);
    #endif // DEBUG
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    i64 count;
    cin >> count;

    i64 a = 0;
    while (true) {
        i64 new_a = (a + count) / 2;
        if (a == new_a) {
            break;
        }
        a = new_a;
    }

    cout << a << "\n";

    return 0;
}
```

Task B ()

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

typedef int32_t i32;
typedef int64_t i64;
typedef long double ld;

const i32 INF32 = 1000000000;
const i64 INF64 = 1000000000000000000;

using namespace std;

void sort_points(i32 count, pair<ld, ld>* points) {
    pair<ld, ld> min_point = points[0];
    i32 index = 0;
    for (i32 i = 1; i < count; i++) {
        if (points[i].second < min_point.second || (points[i].second == min_point.second && points
            [i].first < min_point.first)) {
            min_point = points[i];
            index = i;
        }
    }

    swap(points[0], points[index]);

    for (i32 i = 1; i < count; i++) {
        index = i;
        for (i32 j = i + 1; j < count; j++) {
            ld x1 = points[index].first - points[i - 1].first;
            ld y1 = points[index].second - points[i - 1].second;
            ld x2 = points[j].first - points[i - 1].first;
            ld y2 = points[j].second - points[i - 1].second;

            ld product = x2 * y1 - x1 * y2;
            if (product > 0) {
                index = j;
            }
        }

        swap(points[i], points[index]);
    }
}

pair<ld, ld> get_point(pair<ld, ld> p1, pair<ld, ld> p2) {
    pair<ld, ld> vec;
    vec.first = (p2.second - p1.second) / 2.0;
    vec.second = -(p2.first - p1.first) / 2.0;

    const ld INVSQRT3 = 1.0 / sqrtl(3);

    vec.first *= INVSQRT3;
    vec.second *= INVSQRT3;

    pair<ld, ld> result;
    result.first = (p1.first + p2.first) / 2.0 + vec.first;
    result.second = (p1.second + p2.second) / 2.0 + vec.second;
    return result;
}

i32 main() {
    #ifdef DEBUG
    freopen("input.txt", "r", stdin);
    #endif // DEBUG
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    i32 count;
    cin >> count;

    cout.precision(10);

    if (count == 6) {
```

```
        pair<ld, ld> points[6];
        for (i32 i = 0; i < 6; i++) {
            cin >> points[i].first >> points[i].second;
        }

        sort_points(6, points);

        for (i32 i = 0; i < 3; i++) {
            cout << fixed << points[2 * i].first << " " << fixed << points[2 * i].second << "\n";
        }
    } else {
        pair<ld, ld> points[3];
        for (i32 i = 0; i < 3; i++) {
            cin >> points[i].first >> points[i].second;
        }

        sort_points(3, points);

        pair<ld, ld> new_points[3];
        for (i32 i = 0; i < 3; i++) {
            new_points[i] = get_point(points[i], points[(i + 1) % 3]);
        }

        for (i32 i = 0; i < 3; i++) {
            cout << fixed << points[i].first << " " << fixed << points[i].second << "\n";
            cout << fixed << new_points[i].first << " " << fixed << new_points[i].second << "\n";
        }
    }

    return 0;
}
```

Task C ()

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

typedef int32_t i32;
typedef int64_t i64;

const i32 INF32 = 1000000000;
const i64 INF64 = 1000000000000000000;

using namespace std;

i32 main() {
    #ifdef DEBUG
    freopen("input.txt", "r", stdin);
    #endif // DEBUG
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    string pat;
    cin >> pat;

    bool in_pat[26];
    memset(in_pat, 0, sizeof(in_pat));
    for (i32 i = 0; i < (i32) pat.size(); i++) {
        in_pat[pat[i] - 'a'] = true;
    }

    i32 count;
    cin >> count;

    i32 result = 0;
    for (i32 i = 0; i < count; i++) {
        string str;
        cin >> str;

        i32 min_added = INF32;
        i32 start = 0;
        for (i32 finish = 1; finish <= (i32) str.size(); finish++) {
            if (finish == (i32) str.size() || !in_pat[str[finish] - 'a']) {
//                cout << "Result for ";
//                for (i32 i = start; i < finish; i++) {
//                    cout << str[i];
//                }
//                cout << ": ";

                for (i32 begin = start; begin < finish; begin++) {
                    i32 i = 0;
                    i32 j = 0;
                    i32 added = 0;
                    while (j < (i32) pat.size()) {
                        if (str[begin + i] == pat[j]) {
                            i++;
                        } else {
                            added++;
                        }
                        j++;
                    }
                    min_added = min(min_added, added);
                }
//
//                cout << min_added << "\n";

                start = finish;
            }
        }

        result += min_added;
    }

    cout << result << "\n";

    return 0;
```

```
}
```

Task D ()

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

typedef int32_t i32;
typedef int64_t i64;

const i32 INF32 = 1000000000;
const i64 INF64 = 1000000000000000000;

using namespace std;

void dijkstra2(i32 height, i32 width, pair<i32, i32>* deltas, i32 row_from, i32 col_from, i32*
    min_path) {
    fill(min_path, min_path + width * height, INF32);
    min_path[width * row_from + col_from] = 0;

    set<pair<i32, i32> > min_path_set;
    for (i32 i = 0; i < width * height; i++) {
        min_path_set.insert(make_pair(min_path[i], i));
    }

    while (!min_path_set.empty()) {
        set<pair<i32, i32> >::iterator iter = min_path_set.begin();
        i32 curr_path = iter->first;
        i32 curr = iter->second;
        i32 curr_row = curr / width;
        i32 curr_col = curr % width;
        min_path_set.erase(iter);

        for (i32 next = 0; next < width * height; next++) {
            if (next == curr) {
                continue;
            }

            i32 next_row = next / width;
            i32 next_col = next % width;

            i32 path = abs(next_col - curr_col - deltas[curr].second) + abs(next_row - curr_row -
                deltas[curr].first);
            if (curr_path + path < min_path[next]) {
                min_path[next] = curr_path + path;
                min_path_set.insert(make_pair(min_path[next], next));
            }
        }
    }
}

void dijkstra(i32 count, pair<i32, i32>* deltas, i32 from, i32* min_path) {
    fill(min_path, min_path + count, INF32);
    min_path[from] = 0;

    set<pair<i32, i32> > min_path_set;
    for (i32 i = 0; i < count; i++) {
        min_path_set.insert(make_pair(min_path[i], i));
    }

    while (!min_path_set.empty()) {
        i32 curr_path = min_path_set.begin()->first;
        i32 curr = min_path_set.begin()->second;
        min_path_set.erase(min_path_set.begin());

        for (i32 next = 0; next < count; next++) {
            if (next == curr) {
                continue;
            }

            i32 path = abs(next - curr - deltas[curr].second) + abs(deltas[curr].first);
            if (curr_path + path < min_path[next]) {
                min_path[next] = curr_path + path;
                min_path_set.insert(make_pair(min_path[next], next));
            }
        }
    }
```

```
}

i32 main() {
    #ifdef DEBUG
    freopen("input.txt", "r", stdin);
    #endif // DEBUG
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    i32 height, width;
    cin >> height >> width;

    // Tests in statement
//    if (height == 3 && width == 3) {
//        cout << "1\n";
//        return 0;
//    } else if (height == 3 && width == 5) {
//        cout << "4\n";
//        return 0;
//    }

    // Groups 3, 4
//    if (height > 1) {
//        cout << "0\n";
//        return 0;
//    }

    // Now height == 1

    i32 start_row, start_col, finish_row, finish_col;
    cin >> start_row >> start_col >> finish_row >> finish_col;

    start_row--;
    start_col--;
    finish_row--;
    finish_col--;

    pair<i32, i32> deltas[width * height];
    for (i32 i = 0; i < width * height; i++) {
        cin >> deltas[i].first >> deltas[i].second;
    }

    i32 distances[width * height];
    dijkstra2(height, width, deltas, start_row, start_col, distances);

    cout << distances[finish_row * width + finish_col] << "\n";

    return 0;
}
```

Task E ()

Task F ()