Университет ИТМО

Факультет программной инженерии и компьютерной техники

# Лабораторная работа №3

по «Алгоритмам и структурам данных»

Timus

Выполнил:

Студент группы P3233

Перевозчиков И. С.

Преподаватели:

Косяков М. С.

Тараканов Д. С.

Санкт-Петербург

2022

Задача 1067. Структура папок

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

using namespace std;

vector <string> parse_path(const string s) {
	vector <string> full_path;
	int i = 0;
	int j = 0;
	while (i < s.size()) {
		if (s[i] == '\\') {
			full_path.push_back(s.substr(j, i - j));
			j = ++i;
		}
		i++;
	}
	full_path.push_back(s.substr(j, i - j));

	return full_path;
}

void print_spaces(const int i) {
	for (int j = 0; j < i; j++) {
		cout << ' ';
	}
}

void print_first_path(const vector <string>& first_path) {
	for (int i = 0; i < first_path.size(); i++) {
		print_spaces(i);
		cout << first_path[i] << '\n';
	}
}

void print_folder_structure(const vector <vector <string>>& folder_structure) {
	print_first_path(folder_structure[0]);
	for (int i = 1; i < folder_structure.size(); i++) {
		for (int j = 0; j < folder_structure[i].size(); j++) {
			if (j >= folder_structure[i - 1].size() || folder_structure[i][j] != 
folder_structure[i - 1][j]) {
				for (int k = j; k < folder_structure[i].size(); k++) {
					print_spaces(k);
					cout << folder_structure[i][k] << '\n';
				}
				break;
			}
		}
	}
}

int main()
{
	int n;
	cin >> n;
	vector <vector <string>> folder_structure;
	for (int i = 0; i < n; i++) {
		string s;
		cin >> s;
		folder_structure.push_back(parse_path(s));
	}

	sort(folder_structure.begin(), folder_structure.end());
```

```
        print_folder_structure(folder_structure);
}
```

При вводе данных нужно разделить их на отдельные папки. Так как каждая строка представляет собой полный путь, то после сортировки получится полный список всех папок в лексикографическом порядке. Остается только вывести в нужном формате.

Алгоритмическая сложность: $O(l \cdot logl)$, где $l -$ длина всех входных данных.

Задача 1628. Белые полосы

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

using namespace std;

bool isolated_day(int x, int y, int m, int n, set <pair<int, int>>& bad_days) {
	int isolation = 0;

	if (x == 0) {
		isolation++;
	}

	if (x == m - 1) {
		isolation++;
	}

	if (x != 0 && bad_days.find({ x - 1, y }) != bad_days.end()) {
		isolation++;
	}

	if (x != m - 1 && bad_days.find({ x + 1, y }) != bad_days.end()) {
		isolation++;
	}

	if (y == 0) {
		isolation++;
	}

	if (y == n - 1) {
		isolation++;
	}

	if (y != 0 && bad_days.find({ x, y - 1 }) != bad_days.end()) {
		isolation++;
	}

	if (y != n - 1 && bad_days.find({ x, y + 1 }) != bad_days.end()) {
		isolation++;
	}

	return isolation == 4;
}

int gorizontal_white_lines(int m, int n, set <pair<int, int>>& bad_days) {
	int num_of_white_stripes = 0;
	auto bad_day = bad_days.begin();
	int previous_bad_day = n;

	for (int i = 0; i < m + 1; i++) {
		if (n - previous_bad_day > 2) {
			num_of_white_stripes++;
		}
		else if (n - previous_bad_day == 2) {
			if (isolated_day(i - 1, n - 1, m, n, bad_days)) {
				num_of_white_stripes++;
			}
		}

		previous_bad_day = -1;

		while (bad_day != bad_days.end() && bad_day->first == i) {
```

```
                        if (bad_day->second - previous_bad_day > 2) {
                                num_of_white_stripes++;
                        }
                        else if (bad_day->second - previous_bad_day == 2) {
                                if (isolated_day(i, previous_bad_day + 1, m, n, bad_days)) {
                                        num_of_white_stripes++;
                                }
                        }
                        previous_bad_day = bad_day->second;
                        bad_day++;
                }
        }

        return num_of_white_stripes;
}

int vertical_white_lines(int m, int n, set <pair<int, int>>& reversed_bad_days) {
        int num_of_white_stripes = 0;
        auto bad_day = reversed_bad_days.begin();
        int previous_bad_day = m;

        for (int j = 0; j < n + 1; j++) {
                if (m - previous_bad_day > 2) {
                        num_of_white_stripes++;
                }

                previous_bad_day = -1;

                while (bad_day != reversed_bad_days.end() && bad_day->first == j) {
                        if (bad_day->second - previous_bad_day > 2) {
                                num_of_white_stripes++;
                        }
                        previous_bad_day = bad_day->second;
                        bad_day++;
                }
        }

        return num_of_white_stripes;
}

int main()
{
        int m, n, k;
        cin >> m >> n >> k;
        set <pair<int, int>> bad_days;
        set <pair<int, int>> reversed_bad_days;
        for (int i = 0; i < k; i++) {
                int x, y;
                cin >> x >> y;
                bad_days.insert({ x - 1, y - 1 });
                reversed_bad_days.insert({ y - 1, x - 1 });
        }
        cout << gorizontal_white_lines(m, n, bad_days) + vertical_white_lines(m, n,
reversed_bad_days);
}
```

Нужно считать все блоки, длина или ширина которых больше 1. Если блок размером 1 на 1, то нужно проверить, изолированный он или нет, если изолированный, то посчитать один раз, иначе не считать, так как в этом случае он входит в другой блок большей размерности. Для реализации можно пройтись по горизонтали, считая все блоки, а потом по вертикали, но уже не учитывая изолированные блоки, так как их учли при горизонтальном рассмотрении.

Пример:

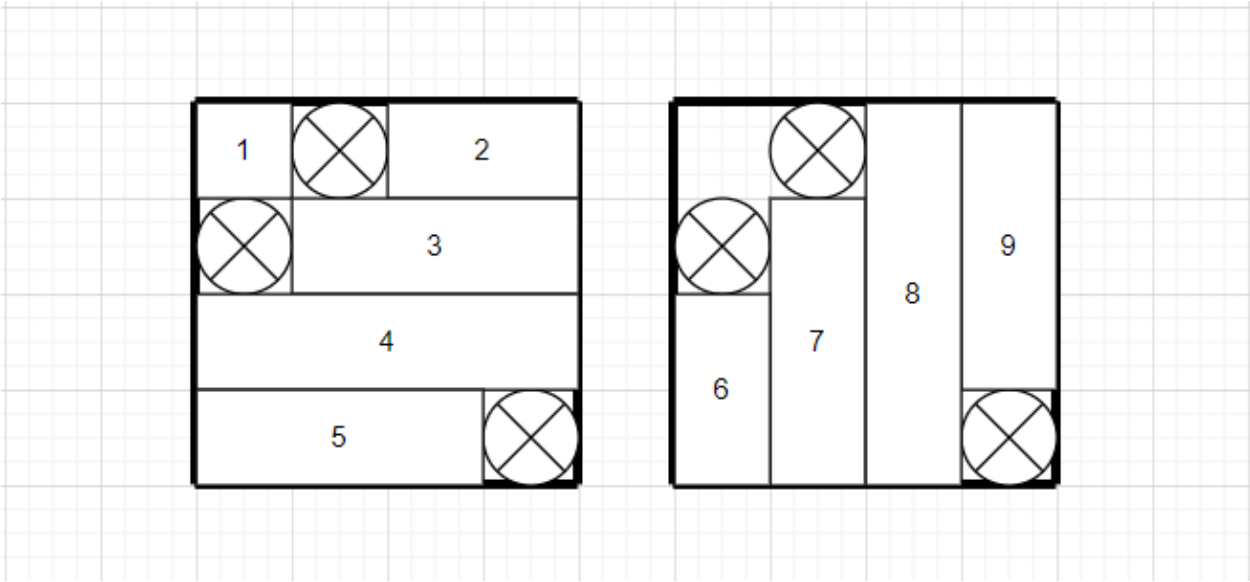
Входные данные:

4 4 3

1 2

2 1

4 4




Алгоритмическая сложность: $O(m \cdot k \cdot logk + n \cdot k)$.