ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент		В. А. Кузнецов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 4.2

РАЗРАБОТКА АЛГОРИТМА ИСЧЕРПЫВАЮЩЕГО ПОИСКА (ПОЛНОГО ПЕРЕБОРА)

по курсу:

ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Постановка задачи

Задача: реализовать алгоритм на языке C/C++, выполняющий поставленную задачу. Алгоритм должен быть реализован путем перебора всех возможных вариантов решения. Вариант задания, пример входных и выходных данных представлен в таблице 1. Глобальные параметры использовать запрещено; допустимо использование дополнительных функций.

- Разработанный алгоритм должен быть реализован в виде цельной программной функции (или нескольких функций) так, чтобы мог быть многократно применим с различными исходными данными и при этом не включал команды, не относящиеся к решаемой задаче, например, ввод и вывод исходных данных на консоль или в файл.
- Произвести экспериментальную проверку времени работы разработанного алгоритма, определив его класс сложности для среднего случая. Измерить среднее время для Test_Count повторений при различных размерностях входных данных.

Таблица 1 – Вариант

N	Текст задания	Вход	Выход
2	**Данные:** • Множество претендентов на вакансию P_i, количество претендентов N. • Каждый претендент обладает множеством навыков $\{S_j\}$, j – индекс навыка. • Количество не повторяющихся общих навыков M. • У разных претендентов могут быть пересекающиеся множества навыков. • Любой навык из общего множества навыков присутствует по крайней мере в одном множестве навыков претендента. **Задача:** Выбрать как можно меньше претендентов таким образом, чтобы все навыки были охвачены.		

2 Схема алгоритма решения

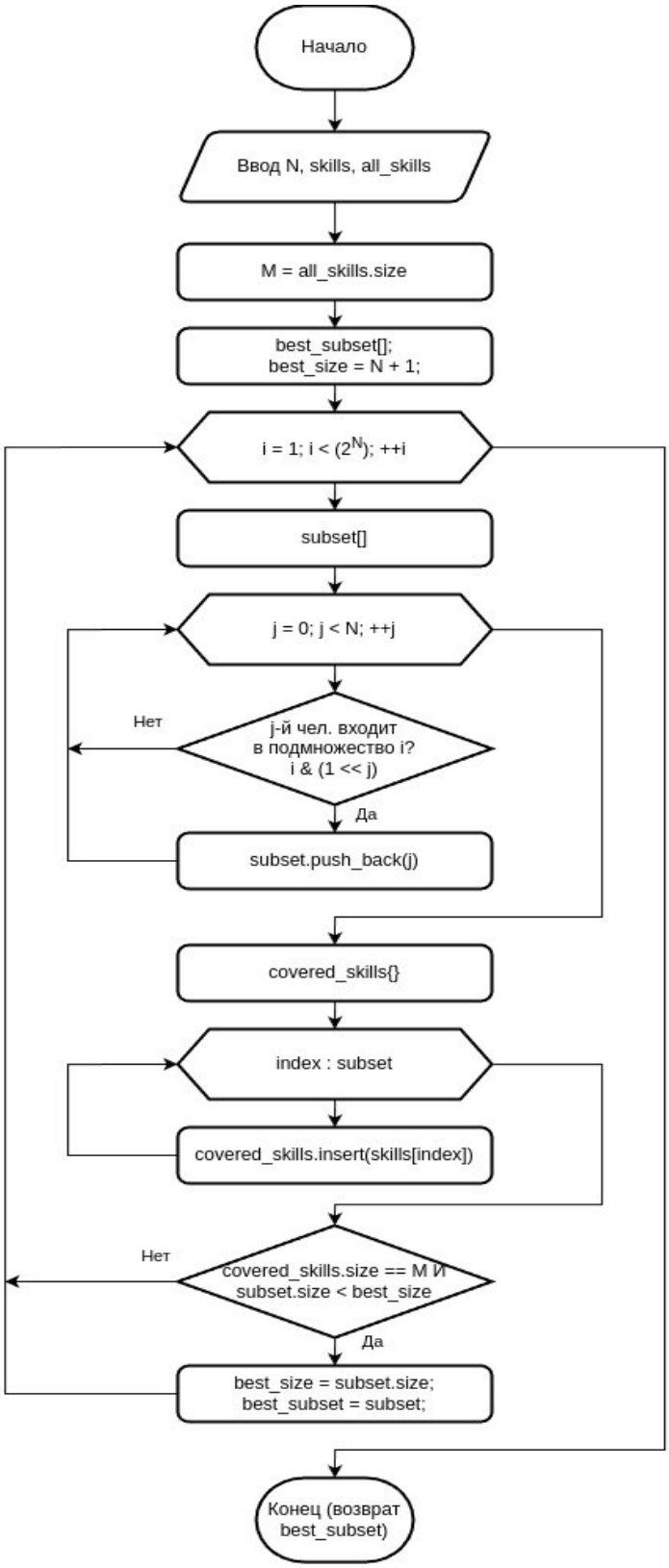

Рисунок 1 – Блок-схема алгоритма

3 Полное описание реализованных функций

3.1 unite_skills

Функция unite_skills объединяет навыки нескольких претендентов в одно множество. Принимает следующие аргументы:

1. const std::vector<std::set<std::string>>& skills — вектор, содержащий множества навыков всех претендентов. Каждое множество представляет навыки одного претендента.
2. const std::vector<int>& indices — вектор, содержащий индексы претендентов, навыки которых нужно объединить.

Возвращает std::set<std::string> — множество, содержащее все уникальные навыки претендентов, указанных в indices. Работа функции происходит следующим образом:

1. Создается пустое множество result.
2. Проходит по каждому индексу в indices.
3. Для каждого индекса добавляет все навыки соответствующего претендента в множество result.
4. Возвращает объединенное множество всех навыков.

3.2 find_min_set_cover

Функция find_min_set_cover находит минимальное подмножество претендентов, которое покрывает все уникальные навыки. Принимает следующие аргументы:

1. const std::vector<std::set<std::string>>& skills — вектор, содержащий множества навыков всех претендентов.
2. int M — количество уникальных навыков.

Возвращает std::vector<int> — вектор, содержащий индексы претендентов, составляющих минимальное подмножество, покрывающее все уникальные навыки. Работа функции происходит следующим образом:

1. Определяет количество претендентов N.
2. Инициализирует пустой вектор best_subset для хранения лучшего подмножества.

3. Устанавливает best_size на значение большее, чем количество претендентов (для сравнения).
4. Перебирает все возможные подмножества претендентов (от 1 до 2^N - 1 — общее количество подмножеств множества из N элементов без пустого подмножества). Число i используется как битовая маска для представления подмножества претендентов.
5. Для каждого подмножества создает вектор subset, содержащий индексы претендентов, входящих в подмножество. Условие if (i & (1 << j)) проверяет, включен ли j-й претендент в подмножество, соответствующее числу i.
6. Проверяет, покрывают ли выбранные претенденты все уникальные навыки, используя функцию unite_skills.
7. Если подмножество покрывает все навыки и его размер меньше текущего лучшего размера, обновляет best_size и best_subset.

3.3 main

1. Запрашивает у пользователя количество претендентов N.
2. Создает вектор skills для хранения множеств навыков претендентов.
3. Запрашивает у пользователя навыки каждого претендента, разделенные пробелами, и заполняет вектор skills.
4. Определяет количество уникальных навыков M.
5. Вызывает функцию find_min_set_cover для нахождения минимального подмножества претендентов, покрывающего все навыки.
6. Выводит количество претендентов в минимальном подмножестве и их порядковые номера.

4 Листинг программы

Листинг 1

```
#include <iostream>
#include <vector>
#include <set>
#include <string>
#include <sstream>

// Функция для объединения навыков нескольких претендентов
std::set<std::string> unite_skills(const std::vector<std::set<std::string>>
&skills, const std::vector<int> &indices) {
    std::set<std::string> result;
    for (int index : indices)
        result.insert(skills[index].begin(), skills[index].end());
    return result;
}

// Основная функция для решения задачи полного перебора
std::vector<int> find_min_set_cover(const std::vector<std::set<std::string>>
&skills, int M) {
    int N = skills.size();
    std::vector<int> best_subset;
    int best_size = N + 1;

    // Перебираем все возможные подмножества претендентов
    for (int i = 1; i < (1 << N); ++i) {
        std::vector<int> subset;
        for (int j = 0; j < N; ++j)
            if (i & (1 << j)) subset.push_back(j);

        // Проверяем, покрывают ли выбранные претенденты все навыки
        std::set<std::string> covered_skills = unite_skills(skills, subset);
        if (covered_skills.size() == M && subset.size() < best_size) {
            best_size = subset.size();
            best_subset = subset;
        }
    }

    return best_subset;
}

int main() {
    int N;
    std::cout << "Введите количество претендентов: ";
    std::cin >> N;
    std::cin.ignore(); // Для игнорирования символа новой строки после ввода
числа

    std::vector<std::set<std::string>> skills(N);
    std::set<std::string> all_skills;

    for (int i = 0; i < N; ++i) {
        std::cout << "Введите навыки претендента " << i + 1 << " через пробел:
";
        std::string line;
        getline(std::cin, line);
```

Продолжение листинга 1

```
        std::stringstream ss(line);
        std::string skill;
        while (ss >> skill) {
            skills[i].insert(skill);
            all_skills.insert(skill);
        }
    }

    int M = all_skills.size(); // Количество уникальных навыков

    std::vector<int> result = find_min_set_cover(skills, M);

    std::cout << "Минимальное количество претендентов для покрытия всех
навыков: " << result.size() << std::endl;
    std::cout << "Претенденты: ";
    for (int index : result) {
        std::cout << index + 1 << " ";
    }

    return 0;
}
```

5 Результаты тестирования программы

```
Введите количество претендентов: 3
Введите навыки претендента 1 через пробел: ответственный общительный внимательный
Введите навыки претендента 2 через пробел: общительный креативный
Введите навыки претендента 3 через пробел: ответственный
Минимальное количество претендентов для покрытия всех навыков: 2
Претенденты: 1 2
Process finished with exit code 0
```

Рисунок 2

```
Введите количество претендентов: 5
Введите навыки претендента 1 через пробел: мотивированный амбициозный
Введите навыки претендента 2 через пробел: лидер уверенный мотивированный
Введите навыки претендента 3 через пробел: амбициозный лидер
Введите навыки претендента 4 через пробел: ответственный
Введите навыки претендента 5 через пробел: уверенный мотивированный амбициозный
Минимальное количество претендентов для покрытия всех навыков: 3
Претенденты: 1 2 4
Process finished with exit code 0
```

Рисунок 3

6 Результаты измерения времени работы и оценки сложности алгоритма

Количество всего возможных подмножеств множества из N элементов — 2^N. Для каждого подмножества необходимо объединить навыки претендентов и проверить их количество. Объединение навыков для каждого подмножества требует O(N) времени, так как нужно просмотреть до N претендентов и объединить их навыки. Проверка покрытия всех навыков требует проверки размера объединенного множества навыков, что требует O(1) времени. Таким образом, общая временная сложность алгоритма: $O(2^N \cdot N)$.

Для экспериментальной проверки времени работы были написаны две дополнительные функции: generate_random_skills — генерирует случайное количество случайно выбранных навыков для каждого претендента, test_find_min_set_cover — замеряет и выводит среднее время выполнения. Для измерения времени была использована библиотека <chrono>. Чтобы получить среднее время выполнения, функция вызывается test_count=5 раз.

Листинг 2

```
// Функция для генерации случайных данных
std::vector<std::set<std::string>> generate_random_skills(int N, int M) {
    std::vector<std::string> all_skills(M);
    std::vector<std::set<std::string>> skills(N);

    for (int i = 0; i < M; ++i)
        all_skills[i] = "skill" + std::to_string(i);

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> skill_quantity_dist(1, M);
    std::uniform_int_distribution<> skill_index_dist(0, M - 1);

    for (int i = 0; i < N; ++i) {
        int skills_quantity = skill_quantity_dist(gen);
        for (int j = 0; j < skills_quantity; ++j)
            skills[i].insert(all_skills[skill_index_dist(gen)]);
    }

    return skills;
}

void test_find_min_set_cover() {
    const int test_count = 5;
    std::vector<int> sizes = {5, 6, 7, 8, 9, 10, 15,
                              20, 21, 22, 23, 24}; // Размерности входных
данных (количество претендентов)
```

Продолжение листинга 2

```
    for (int size : sizes) {
        int N = size;
        int M = 10; // Количество уникальных навыков

        double total_duration = 0.0;

        for (int i = 0; i < test_count; ++i) {
            std::vector<std::set<std::string>> skills =
generate_random_skills(N, M);

            auto start = std::chrono::high_resolution_clock::now();
            std::vector<int> result = find_min_set_cover(skills, M);
            auto end = std::chrono::high_resolution_clock::now();
            std::chrono::duration<double> duration = end - start;
            total_duration += duration.count();
        }

        double average_duration = total_duration / test_count;
        std::cout << "Среднее время для N = " << N << ": " << std::fixed <<
std::setprecision(10) << average_duration
                  << " секунд." << std::endl;
    }
}
```

Результаты измерения среднего времени работы представлены на рисунке 4. Из измеренного времени видно, что при незначительном увеличении количества претендентов N, время выполнения стремительно увеличивается. На рисунке 5 изображены графики, из которых видно, что экспериментально определенная сложность близко соотносится с теоретической.

```
Среднее время для N = 5: 0.0001020130 секунд.
Среднее время для N = 6: 0.0002451972 секунд.
Среднее время для N = 7: 0.0005398832 секунд.
Среднее время для N = 8: 0.0013223538 секунд.
Среднее время для N = 9: 0.0027854958 секунд.
Среднее время для N = 10: 0.0059432958 секунд.
Среднее время для N = 15: 0.2692368650 секунд.
Среднее время для N = 20: 11.7528219610 секунд.
Среднее время для N = 21: 23.3489711512 секунд.
Среднее время для N = 22: 47.7020807746 секунд.
Среднее время для N = 23: 93.8595349852 секунд.
Среднее время для N = 24: 212.5309610122 секунд.

Process finished with exit code 0
```

Рисунок 4

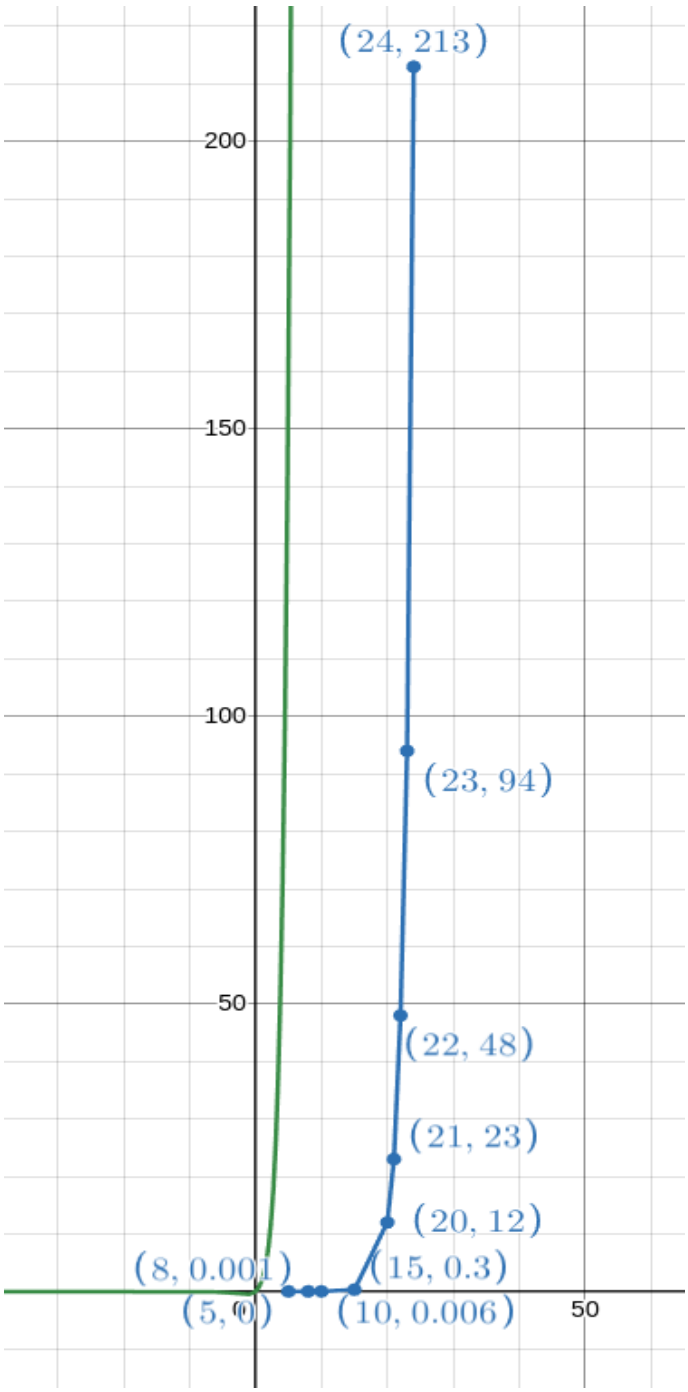

Рисунок 5 - Зеленый график — $2^x \cdot x$