Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Пояснительная записка к курсовой работе по теме «Графы»**

**по дисциплине**

**“Алгоритмы и структуры данных”**

**Вариант 26**

|  |  |
| --- | --- |
| Выполнили студенты гр.9308: | Яловега Н.В. |
| Проверил: | Колинько П.Г. |

Санкт-Петербург, 2020 г.

Оглавление

[Введение 3](#_Toc57574927)

[1. Задание 3](#_Toc57574928)

[2. Формализация задания 3](#_Toc57574929)

[3. Тестирование программы 4](#_Toc57574930)

[4. Пример работы программы 4](#_Toc57574931)

[5. Временная сложность функций 5](#_Toc57574932)

[6. Результаты прогона программы c некоторыми контрольными тестами. 5](#_Toc57574933)

[Вывод 7](#_Toc57574934)

[Список используемых источников 8](#_Toc57574935)

[Приложение 1 (Исходный текст программы) 9](#_Toc57574936)

## Введение

Исследование алгоритмов, реализуемых с помощью графов.

## 1. Задание

Построение ширинного стягивающего леса для неориентированного графа.

## 2. Формализация задания

Стягивающее дерево - ациклический связный подграф данного связного [неориентированного графа](https://ru.wikipedia.org/wiki/Неориентированный_граф), в который входят все его [вершины](https://ru.wikipedia.org/wiki/Вершина_(граф)).

Т.к в задании не сказано, что граф должен быть связным, то мы будем находить стягивающий лес - подграф, состоящий из объединения стягивающих деревьев для каждой его [компоненты связности](https://ru.wikipedia.org/wiki/Компонента_связности).

Ширинный стягивающий лес — стягивающий лес, который построен при помощи обхода графа серией поисков в ширину.

Таким образом, для каждой компоненты связности, запускаем обход в ширину, строя стягивающее дерево для данной компоненты связности.

Для реализации данного алгоритма было принято представлять граф в памяти в виде списов смежности. Т.к для составления стягивающего дерева, нам нужно обходить граф, то сложность обхода для такого представления будет линейной O(n+m). Если же использовать матрицу смежности, то сложность будет O(n^2).

НАПИСАТЬ ПРО ПАМЯТЬ.

Для хранения стягиващего леса в памяти ЭВМ было принято иcпользовать массив ребер. Основная операция, которая используется в алгоритме — вставка ребра. Сложность вставки ребра, используя массив ребер O(1). Кроме того, это экономая структура для хранения дерева. Оценим количество ребер, которое необходимо хранить. Для каждой компоненты связности графа мы вычисляем стягивающее дерево. Количество ребер этого дерева будет на 1 меньше числа вершин в компоненте. Тогда если в графе k компонент связности, то количество ребер в стягивающем лесе будет n-k. Граф разреженный, т.к количество ребер меньше чем количество вершин в квадрате. Поэтому, если использовать матрицу смежности, которая тоже обладает константной сложностью вставки ребра в дерево, потребуется хранить n^2 элементов, что не является хорошим способом для хранения разреженного графа. Используя массив ребер нам достаточно хранить m пар чисел, где m — количество ребер, и число вершин леса n (для рассмотрения случаев, когда лес содержит вершины, которые не соединены ребрами с другими). Если использовать списки смежности, то для вставки ребра временная сложность будет линейной, а для хранения потребуется примерно n+m (с учетом оценки m: 2n - k) элементов.

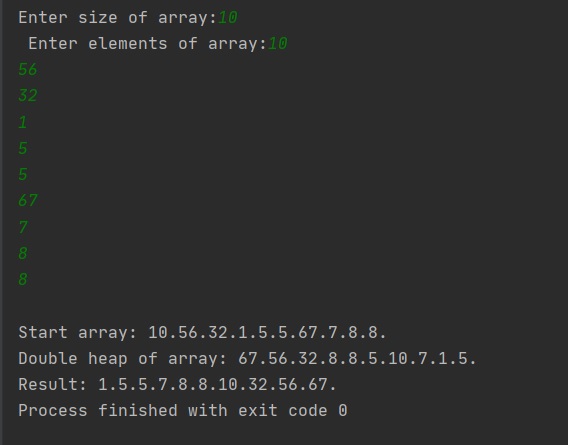
## 3. Тестирование программы

В таблице представлены контрольные примеры, которые использовались для тестирования программы.

|  |  |
| --- | --- |
| ***Исходные данные*** | ***Результат*** |
| 10.56.32.1.5.5.67.7.8.8 | 1.5.5.7.8.8.10.32.56.67 |
| 12.12.12.1.3.4.5.6 | 1.3.4.5.6.12.12.12 |
| -6.-7.-8.-12.14.0.-1 | -12.-8.-7.-6.-1.0.14 |
| 1.2.3.4.5.6.7.8 | 1.2.3.4.5.6.7.8 |
| 8.7.6.5.4.3.2.1 | 1.2.3.4.5.6.7.8 |
| -1.3.4.5.1.1.1. | -1.1.1.1.3.4.5 |

## 4. Пример работы программы

Пример выполнения программы представлен на рисунке 1.



*Рисунок 1. Пример выполнения программы.*

Производится ввод размера множества и само множество. Далее программа строит двоичную кучу на основании полученного массива данных. После этого производится сортировка на основе двоичной кучи.

## 5. Временная сложность функций

1. **Ввод дерева.**

Сложность: O(*n^2*). …

1. **Генерация случайного дерева.**

Сложность: O(*n^2*). ...

1. **Построение стягивающего дерева.** Сложность: O(*n+m*). ...
2. **Построение стягивающего леса.**

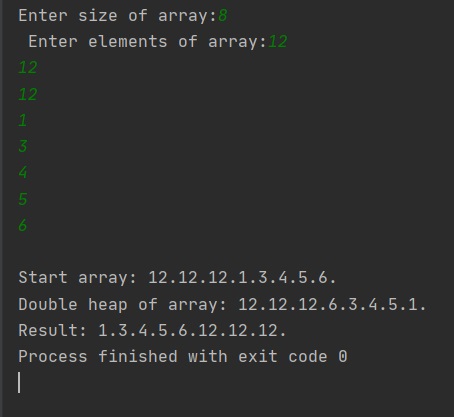
Сложность: O(*n+m*). Так как мы храним массив посещенный вершин, то мы не будем начинать обход из уже посещенных вершин, следовательно временная сложность …

1. **Вывод леса.**

Сложность: O(*n*). ...

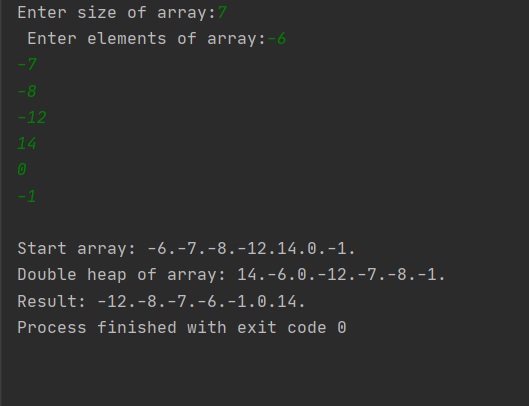
## 6. Результаты прогона программы c некоторыми контрольными тестами.

На скриншотах ниже приведены различные тесты программы.



*Рис 2. Первый тест программы.*

*Рис 3. Второй тест программы.*



Все результаты тестирования программы совпали с ожидаемыми результатами, которые приведены в таблице в пункте 3.

## Вывод

* + - * 1. Алгоритмы, реализуемые с помощью графов, отличаются своим быстродействием и оптимальностью. Так же немаловажным фактом является то, насколько много алгоритмов мы можем реализовать с помощью данной структуры данных. Особо хочется отметить простоту и понятность реализации какого-либо алгоритма, связанного с графами.

## Список используемых источников

1. Колинько П.Г. Пользовательские структуры данных / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.
2. Операции над множествами. <https://www.cyberforum.ru/cpp-beginners/thread2191195.html>
3. Ахо Дж., Хопкрофт А., Ульман Дж. Структуры данных и алгоритмы. – СПб.: И. Д. Вильямс, 2001. – 382 с.
4. Курс на сайте Stepic «Алгоритмы: теория и практика. Структуры данных». <https://stepik.org/lesson/41235/step/6?auth=registration&unit=19819>

## Приложение 1 (Исходный текст программы)