|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7 «ГРАФЫ»**

Студент Иванов Павел Александрович

Группа ИУ7 – 35Б

Вариант №6

Проверил

**Оглавление**

[Описание условия задачи 1](#_Toc90668586)

[Описание технического задания 2](#_Toc90668587)

[Описание структуры данных 3](#_Toc90668588)

[Основные функции 4](#_Toc90668589)

[Описание алгоритма 5](#_Toc90668590)

[Набор тестов 6](#_Toc90668591)

[Сравнение эффективности 7](#_Toc90668592)

[Ответы на контрольные вопросы 8](#_Toc90668593)

[Вывод 9](#_Toc90668594)

# **Описание условия задачи**

Обработать графовую структуру в соответствии с заданным вариантом. Обосновать выбор необходимого алгоритма и выбор структуры для представления графов. Ввод данных осуществить на усмотрение программиста. Результат выдать в графической форме.

Найти минимальное (по количеству ребер) подмножество ребер, удаление которых превращает заданный связный граф в несвязный.

# **Описание технического задания**

Входные данные:

*Целые числа*: номер команды, число вершин, число рёбер, номера вершин рёбер графа.

Выходные данные:

*Изображение*: граф в формате PNG.

*Целые числа*: минимальное количество рёбер, которые нужно удалить чтобы граф распался, начала и концы рёбер для удаления.

Действие программы:

Ввод графа с клавиатуры (в формате перечисления множества рёбер), вывод графа в изображение формата PNG, поиск минимального подмножества рёбер, удаление которых превращает введенных граф в несвязный.

Обращение к программе:

Программа открывается через команду ./app.exe. Если исполняемый файл отсутствует, вызвать make из папки проекта.

Аварийные ситуации:

1. Вызов построения графа при его фактическом отсутствии.

Сообщение: “No graph!”

1. Вызов поиска рёбер при отсутствии графа.

Сообщение: “No graph!”

1. Неправильный ввод команды.

Сообщение: “Wrong input!”

1. Неправильный ввод числа рёбер или вершин.

Сообщение: “Wrong input!”

1. Неправильный ввод концов ребра.

Сообщение: “Wrong input!”

**Меню программы:**

1 – Ввод графа с клавиатуры;

2 – Просмотр графа в формате изображения;

3 – Поиск минимального (по количеству ребер) подмножество ребер, удаление которых превращает заданный связный граф в несвязный;

0 – Выход.

# **Описание структуры данных**

Структура данных для графа:

typedef struct graph\_t

{

    int V;

    int E;

    edge\_t\* edge;

} graph\_t;

В этой структуре V – число вершин, E – число рёбер, edge – массив из E элементов, в котором хранятся все рёбра. В свою очередь ребро представляет из себя следующую структуру:

**typedef struct edge\_t**

**{**

**int src;**

**int dest;**

**} edge\_t;**

В ней src – первый конец ребра, dest – оставшийся конец ребра.

Для хранения списка рёбер, которые нужно удалить, используется связный список типа edgelist\_t:

**typedef struct edgelist\_t**

**{**

**edge\_t edge;**

**struct edgelist\_t \*next;**

**} edgelist\_t;**

Список устроен классическим способом. Edge – ребро, next – следующий элемент списка.

Использование таких структур данных обоснованно выбранным алгоритмом. На каждой итерации алгоритма происходит слияние двух вершин в одну и удаление некоторых рёбер. Это значит, что в случае матрицы смежности, например, пришлось бы удалять столбцы и строки матрицы, в списке смежности аналогично — пришлось бы постоянно удалять элементы списка (вершины) и добавлять новые, а также постоянно обрабатывать полный список смежности, чтобы точно удалить все рёбра. Для того, чтобы не было излишних трат на изменение структур, используется структура непересекающегося множества:

**struct subset\_t**

**{**

**int parent;**

**int rank;**

**};**

Изначально каждая вершина — подмножество, содержащее номер родителя (если она уже куда-то включена, иначе себя) и ранг — число включенных в неё вершин. Таким образом, мы не добавляем новые вершины, а с помощью массива элементов subset\_t определяем, каким образом происходило слияние элементов.

Кроме того, в ответе нам важно получить список рёбер на удаление, а значит — нельзя вводить новые номера вершин (и удалять те же столбцы матриц или элементы списков смежности): конечный ответ не будет согласован с исходными данными. Именно поэтому граф хранится как множество рёбер, которое не изменяется по ходу программы.

# **Основные функции**

// Функция создания графа

struct graph\_t\* create\_graph(int V, int E);

// Функция освобождения списка рёбер

void graph\_free(graph\_t \*\*graph);

// Функция ввода графа с клавиатуры

graph\_t \*input\_graph(void);

// Функция поиска минимального разреза алгоритмом Каргера

int karger\_mincut(graph\_t \*graph, edgelist\_t \*\*out\_list);

// Функция, повторяющая алгоритм Каргера некоторое число раз для увеличения вероятности верного ответа

void karger(graph\_t \*graph);

//Функция поиска родителя вершины (в каком множестве она сейчас)

int find\_parent(struct subset\_t subsets[], int i);

// Функция объединения вершин в одну

void union\_sets(struct subset\_t subsets[], int x, int y);

# **Описание алгоритма**

Для решения поставленной задачи выбран алгоритм Каргера. Эта задача известна в литературе как «minimum cut problem» или «проблема минимального разреза». Разрез — есть разбиение множества вершин графа на два непересекающихся подмножества. Если обозначить вес разреза как число рёбер между подмножествами, то поставленная задача переформатируется так: найти разрез минимального веса.

Алгоритм Каргера является вероятностным (то есть результат при вызове алгоритма над одними и теми же данными может отличатся), поэтому процедуру поиска разреза будем выполнять некоторое число раз. Строго доказано, что если повторить процедуру поиска разреза алгоритмом Каргера n2ln(n) раз и запоминать минимум, то ответ вероятность неверного ответа нестрого меньше, чем 1/n2. Таким образом, уже при 10 вершинах вероятность неверного ответа меньше 0.01 (а вероятность успеха 0.99). Значит, будем считать алгоритм верным.

Выясним, как работает процедура поиска минимального разреза. Пока в графе есть хотя бы две вершины, будем стягивать случайные вершины. Стягивание происходит удалением рёбер между вершинами и объединением вершин в множество (мультивершину) — одна из вершин становится родителем другой. В следующий раз при обращении к ней будет видно, что у неё есть родитель, и она не будет считаться самостоятельной вершиной. Таким образом, стягивание происходит до тех пор, пока не останется две мультивершины. Рёбра между ними и будут искомыми рёбрами на удаление.

Очевидно, что все вершины в мультивершине взаимно достижимы, а значит одна мультивершина образует компоненту связности. Дойдя до последнего шага, получается, мы разделили граф на две компоненты связности. Так как вершины для стягивания всегда выбираются случайно, ответ не всегда будет оптимальным — поэтому прогоняем этот алгоритм n2ln(n) и ищем минимум.

# **Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовый случай | Ввод | Результат |
| 1 | Неверный ввод команды | -1 | Wrong input! |
| 2 | Неверный ввод команды | a | Wrong input! |
| 3 | Ввод неверного числа вершин | 1 | Wrong input! |
| 4 | Ввод неверного числа вершин | a | Wrong input! |
| 5 | Ввод неверного числа вершин | -2 | Wrong input! |
| 6 | Ввод неверного числа рёбер | 0 | Wrong input! |
| 7 | Ввод неверного числа рёбер | a | Wrong input! |
| 8 | Ввод неверного числа рёбер | -10 | Wrong input! |
| 9 | Ввод неверного числа в концах ребра | 3, 2, 3, a 1 | Wrong input! |
| 10 | Ввод неверного числа в концах ребра | 3, 2, 3, 1 a | Wrong input! |
| 11 | Ввод неверного числа в концах ребра | 3, 2, 3, 3 1 | Wrong input! |
| 12 | Вызов вывода графа без его ввода | 2 | No graph! |
| 13 | Выполнение вычислительной части без введенного графа | 3 | No graph! |
| 14 | Ввод графа | 1  3  3  0 1  1 2  0 2 | Введен граф. |
| 15 | Вывод введенного графа | 2 | Открылась фотография графа. |
| 16 | Вычислительная часть | 1  6 12  0 1  0 1  0 2  0 2  2 1  1 2  2 3  3 4  3 4  3 5  3 5  3 | Answer: 1  2 - 3 |
| 17 | Вычислительная часть | 1  5 8  0 1  1 2  2 3  3 4  4 0  0 2  1 4  3 4  3 | Answer: 3  0 – 4  0 – 2  0 – 1 |
| 18 | Вычислительная часть | 1  3 2  1 0  1 2 | Answer: 0  Graph is not well-connected. |

# **Сравнение эффективности**

Алгоритм единичного поиска разреза выполняется на O(n2), при этом мы повторяем его n2logn раз, а значит общая асимптотическая сложность — O(n4logn). В то же время, делая полный перебор, мы бы потратили O(2n).

# **Ответы на контрольные вопросы**

***1. Что такое граф?***

Граф — конечное множество вершин и рёбер, соединяющих их. То есть граф G = (V, E), где V – множество вершин, E – множество рёбер.

***2. Как представляются графы в памяти?***

Представление графа в памяти зависит от выбранного способа реализации. Граф может быть представлен матрицей смежности размера n x n (n – число вершин), в этом случае матрицу можно представить как на стеке, так и в куче в виде матрицы. Если граф представляется в виде списка смежности, который состоит из массива вершин, каждый элемент которого ссылается на список смежных вершин, то удобнее выделять память в куче по мере поступления вершин. Также можно хранить матрицу «по определению» — в виде списка вершин и списка рёбер.

***3. Какие операции возможны над графами?***

а) Поиск кратчайшего пути от одной вершины к другой

б) Поиск кратчайшего пути от одной вершины ко всем другим

в) Поиск кратчайших путей между всеми вершинами

г) Поиск эйлерова пути (при наличии)

д) Поиск гамильтонова пути (при наличии)

е) Выделение остова графа

***4. Какие способы обхода графов существуют?***

Основные два способа обхода графа: обход графа в глубину и в ширину. При обходе в глубину ищется ближайшая смежная вершина, для которой опять начинается поиск в глубину (так и получается «углубление»), пока не встретиться просмотренная вершина или не закончится список смежности.

При поиске в ширину сразу просматриваются все соседи текущей вершины, затем все соседи этих вершин (в порядке очереди).

***5. Где используются графовые структуры?***

Графы использовать для наглядного представления каких-то связей. Например, удобно представлять дороги в виде взвешенных орграфов. Часто графы применяются для решения задач о путях.

***6. Какие пути в графе Вы знаете?***

Эйлеров путь – путь, проходящий через каждое *ребро* ровно один раз. Если при этом путь проходит по некоторым вершинам несколько раз, то путь непростой. Для существования этого пути необходимо и достаточно, чтобы граф содержал не более двух вершин нечетной степени.

Если некоторый путь проходит через каждую *вершину* равно один раз, то он называется гамильтоновым. Он существует не для каждого графа.

**7. Что такое каркасы графа?**

Остов (каркас) – это подграф дерева, содержащий все его вершины и являющийся деревом (ациклическим связным графом). Некоторый каркас получается при обходе в глубину или ширину.

# **Вывод**

В ходе данной работы я познакомился с разными способами представления графов, а также реализовал один из них. Также я осознал, что задача подбора алгоритма для обработки графа — задача непростая, поскольку чтобы обойти всю информацию, представленную случайными связями, требуются большие затраты. А алгоритмы, предоставляющие простое или чуть более быстрое решение, зачастую не всегда детерминированы (как алгоритм Каргера), что приводит к необходимости неоднократного обращения к данным для проверки ответа. Тем не менее, даже моя задача может иметь применение в жизни (если, например, рассматривать граф как конструкцию и искать в ней минимальное число граней, при котором конструкция распадется на части). Поэтому графы — довольно часто применяемые абстракции для решения задач реальной жизни.