Частное учреждение образования

«Колледж бизнеса и права»

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Ведущий методист колледжа  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.В. Паскал  « \_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 года |
| Специальность 2-40 01 01 «Программное обеспечение информационных технологий» | Учебная дисциплина «Основы алгоритмизации и программирование» |

**Лабораторная работа № 42**

**Инструкционно-технологическая карта**

Тема: Разработка постановки задачи приложения, реализующего алгоритм с возвращением.

Цель: Научиться разрабатывать постановку задачи приложения, реализующего алгоритм с возвращением.

Время выполнения: 2 часа.

1. **Порядок выполнения работы**
2. Изучить теоретические сведения к лабораторной работе.
3. Разработать на языке С++ программу вывода на экран решения задачи в соответствии с вариантом индивидуального задания, указанным преподавателем.
4. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
5. Ответить на контрольные вопросы.
6. **Теоретические сведения**

**Алгоритм с возвращением (возвратом)**

Во многих практических задачах из различных предметных областей требуется найти общее количество вариантов решения, число элементов в полном наборе решений. Иногда, исходя из постановки задачи, достаточно найти один из вариантов, соответствующих условию задачи. В некоторых задачах изучается вопрос о существовании решения как такового.

Ответы на поставленные вопросы, как правило, требуют проведения исчерпывающего поиска в некотором множестве всех возможных вариантов, среди которых находятся решения конкретной задачи. Существуют два общих метода организации исчерпывающего поиска: перебор с возвратом (backtracking) и его естественное логическое дополнение – метод решета.

Решение задачи методом перебора с возвратом строится конструктивно последовательным расширением частичного решения. Если на конкретном шаге такое расширение провести не удается, то происходит возврат к более короткому частичному решению, и попытки его расширить продолжаются. Для ускорения перебора с возвратом вычисления всегда стараются организовать так, чтобы была возможность отказаться как можно раньше от как можно большего числа заведомо неподходящих вариантов. Незначительные модификации метода перебора с возвратом, связанные с представлением данных или особенностями реализации, имеют и иные названия: метод ветвей и границ (branch and bound), поиск в глубину (depth first search), метод проб и ошибок и т.д. Перебор с возвратом практически одновременно и независимо был изобретен многими исследователями еще до его формального описания. Он находит применение при решении различных комбинаторных задач в области искусственного интеллекта.

При использовании метода решета вместо конструктивного построения решений задачи из множества возможных вариантов исключаются все элементы, не являющиеся решениями. Методы решета нашли широкое применение в теоретико-числовых задачах.

Метод перебора с возвратом и метод решета, строго говоря, не являются ни методами, ни алгоритмами решения задач. Их следует воспринимать как некоторые общие схемы, которые применяются для решения той или иной задачи. Реализация этих схем в виде конкретных алгоритмов часто требует значительных дополнительных усилий в представлении данных и описании зависимостей между ними.

Соединение метода перебора с возвратом и рекурсии определяет специфический способ реализации рекурсивных вычислений и называется **возвратной рекурсией**. Это соединение двух эффективных методов реализации переборных алгоритмов.

При использовании возвратной рекурсии отпадает необходимость непосредственно организовывать возвраты и отслеживать правильность их осуществления. Они, как правило, становятся встроенной частью механизма выполнения рекурсивных вызовов.

**Вычислительная схема перебора с возвратом**

Опишем общую постановку класса задач, к которым заведомо применим алгоритм перебора с возвратом. Пусть M0, M1, ..., Mn-1 – n конечных линейно упорядоченных множеств и G – совокупность ограничений (условий), ставящих в соответствие векторам вида v = (v0, v1, …, vk)T(vj ϵ Mj; j = 0, 1, …, k; k <= n-1), булево значение G(v) ϵ {истина, ложь}. Векторы v = (v0, v1, ..., vk)T, для которых G(v) = истина, назовем частичными решениями. Пусть, далее, существует конкретное правило P, в соответствии с которым некоторые из частичных решений могут объявляться полными решениями. Тогда возможна постановка следующих поисковых задач: 1) найти все полные решения или установить отсутствие таковых; 2) найти хотя бы одно полное решение или установить его отсутствие. Общий метод решения приведенных задач состоит в последовательном покомпонентном наращивании вектора v слева направо, начиная с v0, и последующих проверках его ограничениями G и правилом P. В общем случае этот метод приводит к алгоритмам с экспоненциальной временной сложностью, а применяется он в основном к классу так называемых Np-полных задач (задача коммивояжера, задача о рюкзаке и т.д.). Задачи этого класса эквивалентны друг другу в том смысле, что все они разрешимы недетерминированными алгоритмами полиномиальной сложности. Для них известно, что, либо все они разрешимы, либо ни одна из них не разрешима детерминированными алгоритмами полиномиальной сложности. Иными словами, если хотя бы для одной из этих задач не существует детерминированного алгоритма, имеющего в худшем случае полиномиальную трудоемкость, то такие алгоритмы не должны существовать и для остальных задач этого класса. Наоборот, если хотя бы для одной из этих задач удалось найти детерминированный алгоритм, имеющий в худшем случае полиномиальную трудоемкость, то подобные алгоритмы существовали бы и для остальных задач этого класса и, более того, их можно было бы построить.

Опишем схему выполнения недетерминированного алгоритма. Пусть алгоритм выполняется до тех пор, пока не доходит до места, с которого должен быть сделан выбор из нескольких альтернатив. Детерминированный алгоритм однозначно осуществит выбор конкретной альтернативы и продолжит работать в соответствии с этим выбором. Недетерминированный алгоритм исследует все возможности одновременно, как бы копируя себя для реализации вычислений по всем альтернативам одновременно. Далее все копии работают независимо друг от друга и по мере необходимости продолжают создавать новые копии. Копия, сделавшая неправильный или безрезультатный выбор, прекращает свою работу. Копия, нашедшая решение задачи, объявляет об этом, давая тем самым сигнал другим копиям о прекращении вычислений (уже ставших ненужными, решение ведь найдено). Недетерминированные алгоритмы, являясь весьма полезной и продуктивной абстракцией, рекурсивны по сути, ибо при реализации оператора выбора фактически обращаются сами к себе.

Возможно, что многие задачи, решаемые методом перебора с возвратом, могут быть решены более эффективно другими способами. Однако ценность метода перебора с возвратом в соединении с рекурсией неоспорима. Во-первых, программы решения многих задач строятся по единой схеме, а во-вторых, они компактны и тем самым просты для понимания и усвоения соответствующих идей.

**Алгоритм с возвращением на графах**

Общая задача нахождения кратчайших путей заключается в нахождении для каждой упорядоченной пары вершин (v, u) такого пути от вершины v в вершину u, что его длина будет минимальна среди всех возможных путей от v к u. Эта задача может возникнуть, например, если имеется компьютерная сеть и известно время прохождения сетевого пакета между соседними компьютерами. Требуется узнать время прохождения пакета от каждого компьютера к каждому или максимальный временной интервал для прохождения пакета. Или, допустим, дан взвешенный **орграф** (ориентированный граф), который содержит время полета по маршрутам, связывающим определенные города, и необходимо построить таблицу, где приводилось бы минимальное время перелета из одного города в любой другой.

Подобные задачи можно решить, последовательно применяя **алгоритм Дейкстры** для каждой вершины, объявляемой в качестве начальной. Но существует прямой способ решения, использующий **алгоритм Флойда**. Алгоритм Флойда использует матрицу Distance размера n \* n, где n – количество вершин в графе, в которой вычисляются длины кратчайших путей. Вначале каждый элемент матрицы Distance равен соответствующему элементу матрицы смежности, диагональные элементы равны 0. Если в графе дуга между вершинами u и v отсутствует, то Distance[u, v] = ∞ (бесконечности). Над матрицей Distance выполняется n итераций. После k-й итерации Distance[i, j] содержит значение наименьшей длины путей из вершины i в вершину j, которые не проходят через вершины с номером, большим k. Другими словами, между концевыми вершинами пути i и j могут находиться только вершины, номера которых меньше или равны k. На k-й итерации для вычисления матрицы Distance применяется следующая формула: Distance[i, j] = min(Distance[i, j], Distance[i, k] + Distance[k, j]). Одновременно с матрицей расстояний ведется построение матрицы предков parent, которая необходима для восстановления последовательности вершин, составляющих кратчайший путь. В элемент parent[i, j] записывается номер последней промежуточной вершины на пути от i к j.

Время исполнения алгоритма Флойда равно O(n3). Это ничуть не лучше, чем n вызовов алгоритма Дейкстры, но на практике алгоритм Флойда работает эффективнее, так как циклы этого алгоритма очень короткие. Алгоритм Флойда примечателен еще и тем, что быстрее работает при представлении графа с помощью матрицы смежности.

Во многих задачах интерес представляет только сам факт существования пути, длиной не меньше единицы, от вершины i до вершины j. В качестве примера можно привести задачу о шантажисте – «**граф шантажиста**», в котором наличие дуги от i к j означает, что лицо i владеет компроматом на персону j и может принудить ее сделать что угодно. Для лоббирования выгоднее нанять то лицо, которое имеет компромат на наибольшее количество людей. Этому человеку будет соответствовать та вершина в графе, из которой достижимо наибольшее количество других вершин. На языке теории графов такая задача называется построением транзитивного замыкания. **Транзитивным замыканием** графа G называется такой ориентированный граф, в котором дуга между вершинами i и j существует тогда и только тогда, когда в графе существует путь от вершины i к j. Алгоритм Флойда можно применить для нахождения транзитивного замыкания. Но полученный в результате алгоритм еще до Флойда разработал С. Варшалл, поэтому в литературе часто алгоритм Флойда называют **алгоритмом Флойда – Варшалла**. Если вес каждой дуги считать равным единице и выполнить алгоритм Флойда, то по матрице расстояний легко строится транзитивное замыкание: если Distance[i, j] < ∞, то дуга от вершины i к вершине j существует. Также легко выяснить, из какой вершины достижимо наибольшее количество вершин, для задачи о шантажисте.

Рассматриваемый алгоритм иногда называют алгоритмом Флойда-Уоршелла. Алгоритм Флойда-Уоршелла является алгоритмом на графах, который разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом (Варшаллом). Он служит для нахождения кратчайших путей между всеми парами вершин графа.

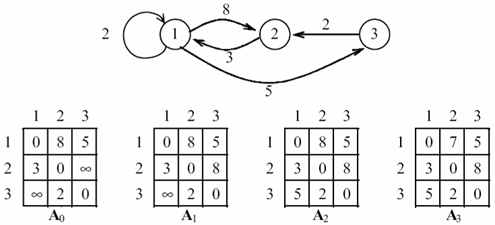
Метод Флойда непосредственно основывается на том факте, что в графе с положительными весами ребер всякий неэлементарный (содержащий более 1 ребра) кратчайший путь состоит из других кратчайших путей.

Этот алгоритм более общий по сравнению с алгоритмом Дейкстры, так как он находит кратчайшие пути между любыми двумя вершинами графа. В алгоритме Флойда используется матрица A размером n на n, в которой вычисляются длины кратчайших путей. Элемент A[i, j] равен расстоянию от вершины i к вершине j, которое имеет конечное значение, если существует ребро (i, j), и равен бесконечности в противном случае.

Основная идея алгоритма Флойда: пусть есть три вершины i, j, k и заданы расстояния между ними. Если выполняется неравенство A[i, k] + A[k, j] < A[i, j], то целесообразно заменить путь i->j путем i->k->j. Такая замена выполняется систематически в процессе выполнения данного алгоритма.

Шаг 0. Определяем начальную матрицу расстояния A0 и матрицу последовательности вершин S0. Каждый диагональный элемент обеих матриц равен 0, таким образом показывая, что эти элементы в вычислениях не участвуют. Полагаем k = 1.

Основной шаг k. Задаем строку k и столбец k как ведущую строку и ведущий столбец. Рассматриваем возможность применения замены описанной выше, ко всем элементам A[i,j] матрицы Ak-1. Если выполняется неравенство A[i, k] + A[k, j] < A[i, j], (i != k, j != k, i != j), тогда выполняем следующие действия: 1) создаем матрицу Ak путем замены в матрице Ak-1 элемента A[i, j] на сумму A[i,k] + A[k,j]; 2) создаем матрицу Sk путем замены в матрице Sk-1 элемента S[i,j] на k. Полагаем k = k + 1 и повторяем шаг k. Таким образом, алгоритм Флойда делает n итераций, после i-й итерации матрица А будет содержать длины кратчайших путей между любыми двумя парами вершин при условии, что эти пути проходят через вершины от первой до i-й. На каждой итерации перебираются все пары вершин и путь между ними сокращается при помощи i-й.

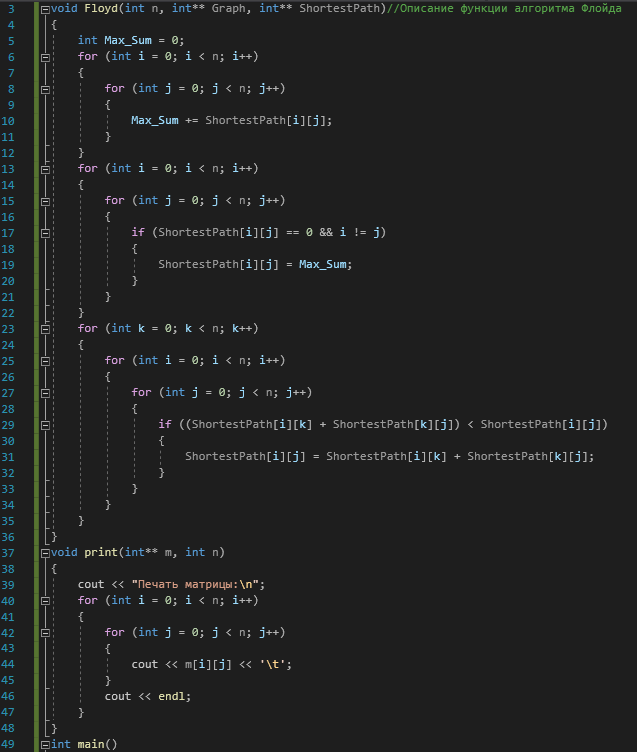


Заметим, что если граф неориентированный, то все матрицы, получаемые в результате преобразований симметричны и, следовательно, достаточно вычислять только элементы, расположенные выше главной диагонали.

Если граф представлен матрицей смежности, то время выполнения этого алгоритма имеет порядок O(n3), поскольку в нем присутствуют вложенные друг в друга три цикла.

1. **Пример выполнения программы**

Демонстрация алгоритма Флойда:



1. **Порядок выполнения работы**

Создать и отобразить (в письменном или электронном виде) свой взвешенный граф, состоящий из заданного количества вершин, дуг и их весов. Уметь проиллюстрировать работу алгоритма Флойда-Варшалла по поиску кратчайшего пути между любыми двумя вершинами взвешенного графа.

Граф должен быть проиллюстрирован в графическом виде и в виде матрицы, например, как показано ниже:

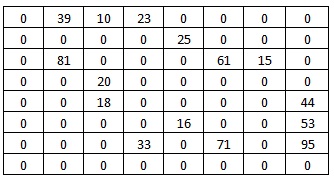


Рисунок 1 – Пример матрицы для хранения весов ребер графа. Названия строк и столбцов проименуйте соответствующими номерами вершин графа

Таблица 1 – Варианты индивидуальных заданий

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант №** | **Количество**  **вершин** | **Количество**  **ребер** | **Диапазон весов**  **существующих ребер** |
| 1 | 7 | 12 | 15 – 99 |
| 2 | 8 | 11 | 14 – 90 |
| 3 | 9 | 10 | 13 – 80 |
| 4 | 7 | 13 | 12 – 70 |
| 5 | 9 | 11 | 10 – 50 |
| 6 | 8 | 12 | 11 – 60 |
| 7 | 7 | 14 | 9 – 40 |
| 8 | 8 | 13 | 8 – 30 |
| 9 | 9 | 12 | 7 – 20 |
| 10 | 7 | 15 | 6 – 65 |
| 11 | 8 | 14 | 5 – 55 |
| 12 | 9 | 13 | 4 – 45 |
| 13 | 7 | 14 | 3 – 35 |
| 14 | 8 | 15 | 2 – 25 |
| 15 | 9 | 14 | 1 – 15 |

1. **Контрольные вопросы**
2. Что такое «алгоритм с возвращением»?
3. Дайте определение понятию «взвешенный граф».
4. Что такое «кратчайший путь в графе»?
5. Что такое «транзитивное замыкание»?
6. Опишите алгоритм Флойда-Варшалла. Почему он так назван?
7. Что такое «граф шантажиста»? Объясните суть задачи о шантажисте.

**Литература**

**Дейтел,** Х.М. Как программировать на С++ / Х.М. Дейтел, П.Дж. Дейтел . – М. : Бином-Пресс , 2018 . – 1456 с.

**Павловская**, Т.А. С++. Объектно-ориентированное программирование : практикум / Т.А. Павловская, Ю.А. Щупак . – СПб. : Питер , 2019 . – 265 с.

**Страуструп**, Б. Язык программирования С++ / Б. Страуструп . – СПб. : Бином-Пресс , 2019 . – 1054 с.

Преподаватель Шаляпин Ю.В.

|  |
| --- |
| Рассмотрено на заседании цикловой  комиссии ПОИТ № 10  Протокол №\_\_\_\_от «\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г.  Председатель ЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |