Фонд оценочных средств  
по дисциплине  
«Языки программирования и   
методы программирования»

Приложение 4

Лабораторная работа №4

(информатика, 3 семестр)

Варианты заданий

Постановка задачи

Написать программу на C++, реализующую алгоритмы (поиска) на графах. Написать краткое техническое задание (ТЗ). Выполнить реализацию. Написать для нее тесты.

**Минимальные требования к программе**. В программе должен быть реализован, по меньшей мере, один алгоритм, реализующей решение заданной задачи с помощью графовых структур данных. В таблице ниже перечислен перечень задач, структур данных для их решения, а также дополнительных задач. При этом не структуры данных и доп. задачи применимы для каждой из основных задач. Рейтинг является оценкой сложности и трудоемкости.

При реализации необходимо придерживаться принципа общности: алгоритмы и структуры данных должны быть достаточно общими для решения задач данного класса (в разумных пределах). Например, в некоторых задачах в роли весов дуг могут выступать как целые числа, так и вещественные и дата/время.

При реализации алгоритмов и структур данных следует пользоваться структурами данных (и алгоритмами), реализованным в рамках предыдущих лабораторных работ.

Основные реализованные алгоритмы необходимо покрыть тестами. Программа должна позволять выбрать любой из реализованных алгоритмов поиска и запустить его на (достаточно произвольных) исходных данных. При этом должна быть возможность как автоматической, так и ручной проверки корректности работы алгоритмов (в т.ч. должна быть возможность просмотра как исходных данных, так и результата). Программа должна обладать пользовательским интерфейсом (консольным или графическим). Пользовательский интерфейс, в особенности, графический, тестировать не требуется. Программа должна предоставлять функцию измерения времени выполнения алгоритма. Должна быть функция сравнения алгоритмов – по времени выполнения на одних и тех же входных данных[[1]](#footnote-1) (программу желательно писать, исходя из предположения, что таких алгоритмов может быть создано несколько). Программа должна предоставлять функционал по построению графиков зависимостей, либо по выгрузке необходимых данных в открытых форматах (например, csv).

Методы поиска и их модификации

|  |  |  |
| --- | --- | --- |
|  | **Задача/модификация** | **Рейтинг** |
|  | Структуры данных |  |
|  | Ориентированный граф | 5 |
|  | Неориентированный | 6 |
|  | Алгоритмы |  |
|  | Раскраска | 7 |
|  | Поиск кратчайших путей |  |
|  | На ориентированном графе | 5 |
|  | На неориентированном графе | 5 |
|  | На большом графе с учетом географического расположения | 8 |
|  | С расстоянием, зависящим от времени[[2]](#footnote-2) | 8 |
|  | Поиск остова графа | 8 |
|  | Топологическое упорядочение | 6 |
|  | Поиск пути с наибольшей пропускной способностью |  |
|  | Поиск одного наилучшего пути | 7 |
|  | Поиск нескольких путей для достижения заданной пропускной способности | 9 |
|  | Решение задачи об оптимальном размещении | 8 |
|  | Построение частичного порядка, определение экстремальных характеристик | 7 |
|  | «Оптимальный конфигуратор» | 7 |
|  | Дополнительные задачи |  |
|  | Генерация графов заданной топологии и размера | 10 |
|  | Материализация графа «по требованию» | 3 |
|  | Реализация алгоритма динамического программирования | 10 |
|  | Реализация очереди с приоритетами в форме наследника Sequence<T>[[3]](#footnote-3) | 7 |

Студент самостоятельно выбирает состав решаемой им задачи. Выбирать его следует выбирать таким образом, чтобы суммарный рейтинг равнялся, как минимум, 12.

**Пояснения**.

1. **Топологическое упорядочение**. Пусть имеется направленных ациклический граф , вершинами которого являются элементы частично упорядоченного множества. Сам граф является диаграммой Хассе. Требуется выстроить такую последовательность элементов , что , т.е. каждый последующий элемент либо «больше» каждого предыдущего, либо несравним с ним.
2. **Построение частичного порядка и определение экстремальных характеристик**. Пусть имеется некоторое (конечное) множество элементов и задано отношение частичного порядка. Требуется построить диаграмму Хассе и с ее помощью найти минимальные и максимальные элементы.
3. **Задача об оптимальном размещении**. В задачах такого вида граф порождается динамически и заранее неизвестен. Каждая дуга обозначает некоторый вариант выбора, а поиск наикратчайшего пути является реализацией направленного перебора.
4. **«Оптимальный конфигуратор»**. Является вариацией на тему задачи об оптимальном размещении. Пусть имеется набор компонентов различного типа – например, компьютерных. Требуется подобрать такую конфигурацию, наилучшую в некотором смысле, но укладывающую в заданные ограничения. Например: (1) самую дешевую, обеспечивающую, по крайней мере, некоторый заданный уровень производительности; (2) самую производительную при заданном лимите стоимости. Вариацией этой задачи является поиск оптимального сочетания между стоимостью и производительностью; основная сложность состоит в том, чтобы правильно подобрать целевую функцию (т.е. как именно, по какой формуле, определять соотношение между производительностью и стоимостью). Следует учитывать, что некоторые компоненты являются обязательными, даже если не влияют на целевую функцию, например, видеокарта, мышь, монитор, и т.п.
5. **Очереди с приоритетами**. Задача состоит в том, чтобы написать наследник класса Sequence – PriorityQueue. При этом бОльшую часть методов Sequence в PriorityQueue нужно скрыть:

|  |  |  |
| --- | --- | --- |
| **Название** | **Сигнатура** | **Назначение** |
| Методы, унаследованные от Sequence | | |
| Get | <TElement> Get(int index) | Сделать private |
| GetFirst | <TElement> GetFirst() | Сделать private |
| GetLast | <TElement> GetLast | Сделать private |
| GetSubsequence | Sequence<TElement> GetSubsequence(int startIndex, int endIndex) |  |
| Append | void Append(<TElement> item) | Сделать private  Вызвать Enqueue |
| Prepend | void Prepend(<TElement> item) | Сделать private  Вызвать Enqueue |
| InsertAt | void InsertAt(int index, <TElement> item) | Сделать private  throw new exception(“invalid operation”) |
| Remove | void Remove(<TElement> item> | Сделать private  throw new exception(“invalid operation”) |
| Методы, которые необходимо реализовать в PriorityQueue | | |
| Peek | T Peek(const int i) | Просто вызов Get |
| PeekFirst | T PeekFirst() | Просто вызов GetFirst |
| PeekLast | T PeekLast() | Просто вызов GetLast |
| Enqueue | void Enqueue(T item, K priority) | Добавить в очередь с учетом приоритета |
| Dequeue | T Dequeue() | Достать первый в очереди |

Критерии оценки

|  |  |  |  |
| --- | --- | --- | --- |
|  | Качество программного кода: | * стиль (в т.ч.: имена, отступы и проч.) (0-2) * структурированность (напр. декомпозиция сложных функций на более простые) (0-2) * качество основных и второстепенных алгоритмов (напр. обработка граничных случаев и некорректных исходных данных и т.п.) (0-3) | 0-7  баллов |
|  | Качество пользовательского интерфейса: | * предоставляемые им возможности (0-2) * наличие ручного/автоматического ввода исходных данных (0-2) * настройка параметров для автоматического режима   отображение исходных данных и промежуточных и конечных результатов и др. (0-2) | 0-6  баллов |
|  | Качество тестов | * степень покрытия * читаемость * качество проверки (граничные и некорректные значения, и др.) | 0-3  баллов |
|  | Полнота выполнения задания и качество ТЗ | Оценивается качество подготовки ТЗ, полнота выполнений минимальных требований | 0-5  баллов |
|  | Владение теорией | знание алгоритмов, области их применимости, умение сравнивать с аналогами, оценить сложность, корректность реализации | 0-3  баллов |
|  | Оригинальность реализации | оцениваются отличительные особенности конкретной реализации – например, общность структур данных, наличие продвинутых графических средств, средств ввода-вывода, интеграции с внешними системами и др. | 0-9  баллов |
|  | Итого | | 0-33  баллов |

Для получения зачета за выполнения лабораторной работы необходимо соблюдение всех перечисленных условий:

* оценка за п. 1 должна быть не менее 3 баллов
* оценка за п. 4 должна быть не менее 3 баллов
* оценка за п. 5 должна быть больше 0
* суммарная оценка за работу без учета п. 6 должна быть не менее 20 баллов

1. Следует рассматривать три основных случая: последовательность уже отсортирована в нужном направлении; последовательность отсортирована в обратном направлении; последовательность не отсортирована. В случае деревьев имеется ввиду сравнение времени построения дерева, а не время поиска в уже построенном дереве. [↑](#footnote-ref-1)
2. Т.е. вес дуги меняется по мере перехода от одной вершины до другой по какому-то детерминированному правилу. Подобным образом работает приложения типа Яндекс.Такси: пока машина доедет до определенного участка, пройдет некоторое время, и дорожная ситуация на этом участке может измениться – движение может стать быстрее или, наоборот, медленнее. [↑](#footnote-ref-2)
3. Очередь с приоритетами полезна в ряде задач на графы, особенно при поиске кратчайших путей. [↑](#footnote-ref-3)