
Лабораторная работа №3

по курсу

«Информатика (организация и поиск данных)»

(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, реализующий 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 данных¹ (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
М-1.	Структуры данных	

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.

М-1.1.	Ориентированный граф	5
М-1.2.	Неориентированный граф	6
М-2.	Алгоритмы	
М-2.1.	Раскраска	7
М-2.2.	Поиск кратчайших путей	
М-2.2.1.	На ориентированном/неориентированном графе	5
М-2.2.2.	На большом графе с учетом географического расположения	8
М-2.2.3.	Многокритериальный поиск (н-р., транспортная сеть с несколькими видами транспорта и штрафом за пересадку)	12
М-2.2.4.	С расстоянием, зависящим от времени ²	8
М-2.3.	Задача коммивояжера	
М-2.3.1.	Базовая	9
М-2.3.2.	Многокритериальный поиск	12
М-2.4.	Реализация АТД Решетка	
М-2.4.1.	Диаграмма Хассе задана явно	9
М-2.4.2.	Диаграмма Хассе задана неявно	12
М-2.5.	Поиск остова графа	6
М-2.6.	Поиск компонент связности не-ор-графа	4
М-2.7.	Поиск компонент сильной связности ор-графа	5,5
М-2.8.	Топологическое упорядочение	6
М-2.9.	Поиск пути с наибольшей пропускной способностью	
М-2.9.1.	Поиск одного наилучшего пути	7
М-2.9.2.	Поиск нескольких путей для достижения заданной суммарной пропускной способности	9
М-2.10.	Оптимизация плана задач	10
М-2.11.	Задача о рюкзаке	10
М-2.12.	Задача об оптимальном поселении в гостинице	9
М-2.13.	Построение частичного порядка, определение экстремальных характеристик	7
М-2.14.	«Оптимальный конфигуратор»	7
М-2.15.	«Крестики-нолики»	12
М-2.15.1.	На бесконечном поле	3
М-2.16.	Разработка менеджера пакетов	
М-2.17.	Генеалогия (направленный ациклический граф)	
М-2.18.	Категория объектов как граф. Функтор как отображение между графами.	
М-2.19.	Оптимальное размещение сети ретрансляторов на карте	
М-3.	Дополнительные задачи	
М-3.1.	Генерация графов заданной топологии и размера	10
М-3.2.	Материализация графа «по требованию»	3
М-3.3.	Реализация алгоритма динамического программирования	10
М-3.4.	Реализация очереди с приоритетами в форме наследника Sequence<T> ³	7

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.

³ Очередь с приоритетами полезна в ряде задач на графы, особенно при поиске кратчайших путей.

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

Пояснения.

1. **Задача коммивояжёра.** В базовом варианте предполагает построение плана посещения заданного набора пунктов, при котором некоторая целевая характеристика окажется оптимальной. Доступные маршруты между пунктами задаются графом, в котором, как правило нет «посторонних» вершин, т.е. путь между двумя пунктами, если он существует, задается единственной дугой (редко – несколькими параллельными дугами разного типа). Однако можно рассматривать расширение этой задачи, когда задан некоторый граф, в котором обязательными для посещения являются лишь подмножество вершин, а остальные просто являются частью транспортной сети. Такая задача не сводится к построению плана посещения с использованием кратчайших путей между вершинами, если дуги характеризуются набором характеристик, которые используются для определения составного критерия оптимизации, так и для задания ограничений. Например, транспортная сеть включает несколько видов транспорта и требуется найти баланс между минимизацией времени в пути, минимизации стоимости (которая, как правило, тем выше, чем быстрее транспортное средство) и числом пересадок (которое желательно минимизировать, но, например, 1-2 являются вполне приемлемыми, а больше уже нежелательно).
2. **Топологическое упорядочение.** Пусть имеется направленный ациклический граф G , вершинами которого являются элементы s_i частично упорядоченного множества. Сам граф G является диаграммой Хассе. Требуется выстроить такую последовательность элементов s_{i_1}, \dots, s_{i_n} , что $\forall j, k (s_j \leq s_k \vee \neg(s_j \leq s_k \wedge s_k \leq s_j))$, т.е. каждый последующий элемент либо «больше» каждого предыдущего, либо несравним с ним.
3. **Оптимизация плана задач.** В базовом варианте, имеется частично упорядоченное множество задач, упорядочение отражает зависимости, т.е. то, какие задачи не могут быть выполнены до выполнения некоторых других. Имеется один или несколько исполнителей. Планом исполнителя называется последовательность задач, которые этот исполнитель будет выполнять. Планом задач называется совокупность планов исполнителей. С помощью динамического программирования, требуется построить оптимальный по времени план.
В более сложном варианте, критериев оптимизации может быть несколько, задачи могут быть разнотипными и каждый исполнитель способ исполнять только задачи некоторых, но не всех типов, а также зависимости между задачами могут быть заданы неявно. Подробности следует обсуждать и согласовать с преподавателем.
4. **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.
5. **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.
6. **«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.

7. **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

8. **Разработка менеджера пакетов.** Задача аналогична варианту из ЛР-3, но вместо дерева зависимостей строится направленный ациклический граф.
9. **Планирование оптимального размещения сети ретрансляторов.** В дискретном варианте карта представляет собой граф с привязкой вершин к координатам, узлы в этом графе –
- 10.

Критерии оценки

1.	Качество программного кода:	<ul style="list-style-type: none"> – 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) 	0-6 баллов
2.	Качество тестов	<ul style="list-style-type: none"> – степень покрытия – читаемость – качество проверки (граничные и некорректные значения, и др.) 	0-5 баллов
3.	Качество пользовательского интерфейса:	<ul style="list-style-type: none"> – 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
4.	Полнота выполнения задания	Оценивается полнота выполнения минимальных требований	0-3 баллов
5.	Владение теорией	знание алгоритмов, области их применимости, умение сравнивать с аналогами, оценить сложность, корректность реализации	0-3 баллов
6.	Оригинальность реализации	оцениваются отличительные особенности конкретной реализации – например, общность структур данных, наличие продвинутых графических средств, средств ввода-вывода, интеграции с внешними системами и др.	0-5 баллов
		Итого	0-30 баллов
7.	Объем выбранного задания	Оценивается объем работы, выполненный сверх минимально необходимого. Примерно 1 балл за 1-2 пункта рейтинга задания сверх 15 (при условии, что работа выполнена в надлежащем качестве).	0-10 баллов
		Итого	0-40 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. 2 должна быть не менее 3 баллов
- оценка за п. 4 должна быть больше 0
- оценка за п. 5 должна быть больше 0
- суммарная оценка за работу без учета п. 7 должна быть не менее 18 баллов