Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ ИМЕНИ ГУМАРБЕКА ДАУКЕЕВА

Кафедра IT-инжиниринга

АЛГОРИТМЫ И ИХ СЛОЖНОСТЬ

Методические указания к выполнению лабораторных работ для магистрантов специальности 7М06103 – «Вычислительная техника и программное обеспечение»

Алматы 2021

Составитель: З.К. Куралбаев. Методические указания к выполнению лабораторных работ по дисциплине «Алгоритмы и их сложность» для магистрантов специальности 7М06103 – «Вычислительная техника и программное обеспечение». – Алматы: АУЭС, 2021. – 34 с.

Методические указания по дисциплине «Алгоритмы и их сложность» предназначены для выполнения лабораторных работ магистрантами специальности 7М06103 – «Вычислительная техника и программное обеспечение» с целью закрепления теоретических знаний, полученных на лекциях. Предусмотрено выполнение десяти работ по различным разделам дисциплины.

Список литературы – 12 наименований.

Рецензент: к.ф-м.н., доцент А.К. Искакова

Печатается по плану издания некоммерческого акционерного общества «Алматинский университет энергетики и связи имени Гумарбека Даукеева» на 2020 г., поз. 103.

Содержание

Введение

Цели и задачи дисциплины. Целью изучения дисциплины «Алгоритмы и их сложности» является изучение применяемых в программировании (и информатике) структур данных, их спецификации и реализации, а также алгоритмов обработки данных и анализ этих алгоритмов, взаимосвязь алгоритмов и структур.

В результате изучения дисциплины магистрант должен:

а) иметь представление об основных тенденциях в создании структур данных, методах оптимального использования памяти и времени для обработки структур данных и управления процессами обработки данных;

б) знать и использовать различные структуры данных в соответствии с запросами алгоритмов;

в) создавать списковые и древообразные структуры и управлять организацией этих структур (изменение списков и деревьев посредством включения–исключения, замены элементов структур), знать, использовать оптимальные методы поиска и сортировки данных;

г) знать и использовать основные алгоритмы решения классических задач информатики;

д) иметь представление о математических методах анализа алгоритмов; классификации алгоритмических задач по сложности, сводимости алгоритмических задач к известным задачам определенного класса сложности;

е) иметь опыт работы с алгоритмическими языками программирования.

Методы и формы организации обучения. Процесс изучения дисциплины «Алгоритмы и их сложности» направлен на формирование следующих компетенций.

Общекультурные компетенции:

– способность использовать углубленные теоретические и практические знания в области прикладной математики и информатики;

- способность самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения, в том числе, в новых областях знаний, непосредственно не связанных со сферой деятельности, расширять и углублять свое научное мировоззрение;

– способность порождать новые идеи и демонстрировать навыки самостоятельной научно-исследовательской работы и работы в научном коллективе.

Профессиональные компетенции:

– способность проводить научные исследования и получать новые научные и прикладные результаты;

–- способность разрабатывать концептуальные и теоретические модели решаемых научных проблем и задач;

– способность углубленного анализа проблем, постановки и обоснования задач научной и проектно-технологической деятельности.

Для успешного освоения дисциплины применяются различные образовательные технологии, которые обеспечивают достижение планируемых результатов обучения согласно основной образовательной программе, с учетом требований к объему занятий в интерактивной форме.

Интерактивные формы обучения, которые используются в данном курсе, включают: «Работу в команде» и «Поисковый метод». Для контроля освоения компетенций используются следующие формы контроля: защита лабораторных работ, опрос по изучаемым разделам дисциплины, тесты.

Место дисциплины в структуре образовательной программы. Дисциплина «Алгоритмы и их сложности» относится к вариативной части математического и естественнонаучного цикла образовательной программы. Успешное овладение дисциплиной предполагает предварительные знания математического анализа, вычислительных методов, дискретной математики в объеме, предусмотренном специальностью «Вычислительная техника и программное обеспечение», а также навыки программирования на языках высокого уровня.

Лабораторный практикум дисциплины «Алгоритмы и их сложности» позволяет получить практические навыки использования изучаемых структур данных и эффективных алгоритмов решения различных задач.

Методические рекомендации. При выполнении лабораторной работы рекомендуется следующая последовательность действий:

1. Изучить теоретические и методические материалы по теме лабораторной работы.
2. Получить задание на выполнение лабораторной работы.
3. Используя один из алгоритмических языков, разработать программу и выполнить задание.
4. Подготовить отчет по лабораторной работе.
5. Защитить выполненную лабораторную работу.

Результаты выполнения лабораторных работ представляются в виде отчета, который состоит из следующих пунктов:

1.Тема и цель работы.
2. Формулировка задания.
3. Алгоритм решения задачи.
4. Блок-схема алгоритма.
5. Текст (листинг) программы.
6. Распечатка результатов выполнения программы.
7. Выводы: обсуждение и анализ сложности алгоритма.

Лабораторная работа № 1. Алгоритм прямого линейного выбора

***Цель работы*:** Изучение одного из алгоритмов построения статистических (экспериментальных) рядов – прямого линейного выбора для упорядочения одномерных массивов.

Методические указания. Проблема сортировки массивов, списков, файлов и др. является одной из важных в программировании. Рассматриваются алгоритмы внутренней и внешней сортировки. В данной работе рассматривается один из фундаментальных алгоритмов внутренней сортировки – алгоритм прямого линейного выбора. Здесь рассматривается задача упорядочения заданного исходного массива (списка) по возрастанию, алгоритм решения которой представлен ниже. Алгоритм состоит из просмотров списка.

Один просмотр (проход) в данном случае включает:

– выбор из сортируемого списка элемента с наименьшим ключом (значением);

– размещение его в формируемом списке.

Просмотры выполняются до тех пор, пока список вывода не будет сформирован полностью. В готовом виде список вывода представляет собой упорядоченный исходный список.

Реализация алгоритма прямого линейного выбора состоит из следующих этапов:

1. Ввод элементов исходного списка.

2. Выполнение цикла: количество повторений равно количеству элементов списка.

3. Определение наименьшего элемента исходного списка и его позиции.

4. Найденный элемент становится первым элементом упорядоченного массива. Он исключается из исходного массива; его место в списке занимает фиктивный элемент – z.

5. Все перечисленные операции повторяются для текущего массива: определяются второй, третий и т.д. элементы.

В результате этих операций будет получен упорядоченный массив по возрастанию элементов.

Замечание. Следует отметить, что подобным образом можно определить также упорядоченный массив по убыванию его элементов.

Пример. Пусть рассматривается список элементов, представленных их ключами (значениями).

Исходный список:

Позиция	Ключ
1	3
2	11
3	6
4	4
5	9
6	5
7	7
8	8
9	10
10	2
11	1

На следующем рисунке показан процесс 5 просмотров списка:

Исходный список		Просмотры (проходы)					Вывод (упорядоченный список)
Позиция	Ключ	1	2	3	4	5	
1	3	3	3	z	z	z	1
2	11	11	11	11	11	11	2
3	6	6	6	6	6	6	3
4	4	4	4	4	z	z	4
5	9	9	9	9	9	9	5
6	5	5	5	5	5	z	
7	7	7	7	7	7	7	
8	8	8	8	8	8	8	
9	10	10	10	10	10	10	
10	2	2	z	z	z	z	
11	1	z	z	z	z	z	

Продолжая эти операции, можно получить окончательный список – упорядоченный по возрастанию его элементов. Для анализа сложности данного алгоритма должно быть обсуждение результатов его выполнения.

Обсуждение результатов выполнения алгоритма. Каждый просмотр добавляет в список вывода по одному элементу. Тогда количество просмотров будет равняться $N-$ количеству элементов в списке. Каждый просмотр потребует $N-1$ сравнений. Общее количество сравнений равно $N \cdot (N-1)$. Число пересылок из исходного списка в список вывода равно N. При

линейном выборе для данных выделяется в два раза больше памяти, чем для исходного списка. При этом упорядоченный список увеличивается от просмотра к просмотру.

Итак, получены следующие результаты, характеризующие данный алгоритм:

– количество просмотров N;

– количество сравнений $N \cdot (N-1)$;

– количество пересылок N.

***Контрольные вопросы**:*

1. Алгоритм и его роль в решении задач практики.
2. Каковы основные требования, предъявляемые к алгоритмам?
3. Что собой представляет сложность алгоритма?
4. Как определяется эффективность алгоритма?
5. Что такое модель алгоритма?
6. Где размещаются данные?
7. В каких задачах могут быть использованы алгоритмы сортировки?
8. В чем разница между внешней и внутренней сортировками.
9. Что собой представляет вычислительный процесс?
10. Как определяется сложность алгоритма прямого линейного выбора?

Варианты задания для самостоятельного выполнения:

№ 1.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	3,2	1,5	4,6	2,8	3,6	1,2	4,8	5,2	7,3	6,5	3,4	1,9	7,1	2,5

№ 2.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	4,2	2,5	3,6	2,8	3,3	1,5	1,8	5,0	4,3	6,5	3,4	2,9	1,1	2,5

№ 3.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	5,2	3,5	4,5	1,8	3,6	1,2	4,4	5,7	1,3	5,5	4,4	2,9	4,1	2,5

№ 4.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	6,2	2,1	3,6	2,3	4,3	1,7	2,8	3,0	4,8	6,5	3,4	2,0	1,5	3,5

№ 5.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	4,2	2,5	3,6	1,8	4,6	5,2	4,8	1,2	4,3	1,5	3,8	1,9	4,0	0,5

№ 6.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	5,2	1,5	4,6	2,8	3,3	1,7	1,8	5,0	4,1	5,5	3,5	2,9	1,1	2,5

№ 7.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	6,2	2,5	1,5	1,8	3,4	1,2	4,4	5,7	1,3	5,5	4,4	2,9	4,1	2,5

№ 8.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	3,2	2,0	3,0	2,5	4,6	1,7	2,8	3,9	4,8	6,5	3,4	2,8	1,5	3,5

№ 9.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	1,2	2,5	3,5	2,8	3,3	1,5	1,8	4,0	2,3	1,1	3,4	2,9	1,9	1,6

№ 10.

Позиции	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ключи	5,0	3,0	4,0	1,7	3,6	1,2	5,4	3,7	1,9	5,4	4,2	2,7	4,9	1,5

Лабораторная работа № 2. Алгоритмы линейного выбора с обменом и подсчетом

Цель работы: Изучение двух алгоритмов построения статистических (экспериментальных) рядов; алгоритмы линейного выбора с обменом и подсчетом.

1 Алгоритм линейного выбора с обменом. *Обменом* называется перестановка позиций двух записей в списке в зависимости от результата проверки их относительной величины.

Производительность метода обмена зависит от сложности определения последовательности сравнений и обменов. Для сокращения числа обменов используется метод, который будет изложен здесь. В начале первого просмотра предполагается, что первый элемент исходного списка имеет наименьший ключ. Затем он сравнивается со всеми элементами до тех пор, пока не встретится меньший ключ. Меньший ключ и его адрес становятся содержимым рабочей области памяти.

Сравнение продолжается при новом содержимом рабочей памяти. Просмотр завершается по достижении конца списка.

Порядок просмотра списка в алгоритме линейного выбора с обменом. По окончании первого просмотра запись, ключ которой расположен в рабочей памяти, переставляется с записью на вершине списка. Теперь наименьшая величина в списке занимает первую позицию. Второй просмотр идентичен первому с той разницей, что вторым по величине считается второй ключ.

Первая позиция исключается из процесса. По окончании второго просмотра вторая запись с наименьшим ключом помещается на вторую позицию. Третий просмотр начинается сравнением ключа на третьей позиции и т.д. Эта процедура заканчивается, когда свое место занимает $(N-1)-$ая запись.

Этапы алгоритма линейного выбора с обменом:

– Ввод N элементов исходного списка в память компьютера.

– Цикл: количество повторений равно $N-1$.

– Определение наименьшего элемента исходного списка и его позиции.

– Найденный элемент становится первым элементом в списке.

– Первый элемент исходного списка занимает прежнюю позицию наименьшего элемента.

– Все операции повторяются для полученного списка. В результате определяется второй элемент упорядоченного массива и т.д.

– В результате этих операций будет получен упорядоченный массив по возрастанию элементов.

Просмотры элементов списка; результаты после каждого просмотра:

Исходный список		**Просмотры (проходы)**									
Позиция	Ключ	1	2	3	4	5	6	7	8	9	10
1	3	1	1	1	1	1	1	1	1	1	1
2	11	11	2	2	2	2	2	2	2	2	2
3	6	6	6	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4
5	9	9	9	9	9	5	5	5	5	5	5
6	5	5	5	5	5	9	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8
9	10	10	10	10	10	10	10	10	10	9	9
10	2	2	11	11	11	11	11	11	11	11	10
11	1	3	3	6	6	6	9	9	9	10	11

Основное преимущество данного метода сортировки заключается в том, что сокращается количество элементов в списке от просмотра к просмотру. На каждом последующем просмотре выполняется на одно сравнение меньше.

Выполняются последовательно $N-1, N-2, \ldots, 2,1$ сравнений. Общее число сравнений равно $N \cdot (N-1)/2$.

Количество обменов равно количеству просмотров.

Не учитывается количество пересылок в рабочую память.

К увеличению количества обменов может привести побочный эффект, который заключается в том, что сам список подвергается промежуточному упорядочению.

2. Идея метода сортировки с подсчетом. Метод сортировки с подсчетом описывается как процедура упорядочения внутреннего списка чисел. Он является формой индексирования, в которой счетчик относительной позиции каждого элемента корректируется в течение процесса сравнения.

Память, используемая линейным выбором с подсчетом, будет включать область вывода для хранения окончательно упорядоченного списка. Размер области вывода будет таким же, как при линейном выборе. Дополнительно должна быть память под счетчик для каждого элемента списка.

В результате действий над значениями этих счетчиков образуется множество индексов позиций для элементов в упорядоченном списке. При каждом просмотре ключ сравнивается со своим линейным преемником.

Каждый раз, когда находится больший ключ, его счетчик увеличивается на единицу. Если найденный ключ меньше или равен, то увеличивается счетчик, соответствующий большему из сравниваемых ключей.

Порядок выполнения алгоритма. При первом просмотре первый ключ в списке сравнивается со всеми остальными ключами. В его счетчике подсчитывается количество меньших ключей. В счетчиках больших ключей будет 1. Этот процесс продолжается N раз. При втором просмотре первый ключ не рассматривается. Второй ключ сравнивается с ключами всех преемников. Подсчеты фиксируются. Этот процесс продолжается $N-1$ раз. На этом этапе известна относительная позиция каждого элемента.

Помещая ключи в выходной список в соответствии со значениями их счетчиков, можно получить упорядоченный список.

Пример выполнения алгоритма. Рассматривается тот же пример, который был обсужден в предыдущей лабораторной работе:

Позиция		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Ключ		3	11	6	4	9	5	7	8	10	2	1
Счетчик в каждом просмотре	1	2	1	1	1	1	1	1	1	1	0	0
	2	2	10	1	1	1	1	1	1	1	0	0
	3	2	10	5	1	2	1	2	2	2	0	0
	10	2	10	5	3	8	4	6	7	9	1	0

Пояснения полученной таблицы. В таблице показаны результаты всех просмотров:

– в течение первого просмотра ключ 3 сравнивается с ключами всех линейных преемников; все ключи, значения которых больше значения ключа 3, следовательно, в их счетчиках будет 1; в счетчике ключа 3 будет 2, так как это ключ больше двух ключей из списка (1 и2);

– при втором просмотре сравнивается ключ 11, определяется его счетчик 10, так как он больше девяти в списке ключей;

– при третьем просмотре проводятся сравнения с ключом 6, и определяется его счетчик, равный 5, т.к. он больше четырех ключей списка;

все, кроме 1 и 2, больше ключа 3, поэтому в счетчике ключа 3 будет 2;
повторяя таким образом, можно найти счетчики всех ключей, и последний, 10-й просмотр определяет окончательное количество счетчиков каждого ключа;

– найденные счетчики определяют позиции каждого из ключей.

Обсуждение. Число сравнений равно $N \cdot (N-1)/2$. Выполняется $N-1$ просмотров с $N/2$сравнениями в среднем на каждом просмотре. Значения счетчиков заменяются столько раз, сколько раз выполняются сравнения, потому что каждое сравнение – это изменение значения счетчика. Количество пересылок равно N.

Этот метод может быть модифицирован. Если за исходное значение каждого счетчика взять адрес начала списка вывода и при каждом последующем изменении увеличивать его на длину записи, то значение счетчика будет представлять собой итоговый адрес соответствующей записи.

***Контрольные вопросы**:*

1. Какие факторы должен учитывать программист при выборе метода сортировки?
2. От чего зависит производительность сортировки?
3. Почему размер списка влияет на производительность сортировки?
4. Какое множество будет получено в результате использования метода сортировки с подсчетом?
5. Сколько раз повторяются подсчеты?
6. Какие ключи исключаются при третьем просмотре?
7. Что определяют позиции упорядоченного списка?
8. Назовите разницу в подходах к трем методам сортировки.
9. Сделайте сравнительный анализ трех алгоритмов, рассмотренных в первых двух лабораторных работах.
10. Какие алгоритмы, по вашему мнению, требуют наименьшей оперативной памяти компьютера?

Варианты задания для самостоятельного выполнения те же, которые были предложены в предыдущей работе.

Лабораторная работа № 3. Алгоритм парного и стандартного обмена

Цель лабораторной работы – изучение основных особенностей применения алгоритма стандартного обмена.

Методические материалы. Для выполнения лабораторной работы необходимо ознакомиться с основными методами сортировки элементов списка. Изучить основы разработки алгоритмов методов парного и стандартного обмена. Метод парного обмена основан на нечетном и четном просмотрах элементов списка. Метод стандартного обмена, или метод «пузырька», перемещает один элемент списка в соответствующую позицию при каждом просмотре. В ходе изучения теоретического материала обратить внимание на эффективность алгоритма каждого из методов сортировки.

Метод парного обмена состоит из различных «нечетных» и «четных» просмотров. При «нечетных» просмотрах каждый элемент из нечетной позиции сравнивается со своим соседом из четной позиции, и больший из них

занимает четную позицию. Нисходящий просмотр списка продолжается до тех пор, пока последний нечетный элемент в списке не сравнивается с последним четным элементом. Если в списке нечетное число элементов, то последний элемент не будет участвовать в сравнении. В течение каждого просмотра ведется подсчет обмена.

Рекомендуется разобрать следующий пример.

Пример. Дан некоторый исходный список. Требуется сортировка по возрастанию элементов списка.

Парный обмен: первый просмотр

Номер	Исходный список	Сравнение позиций				
		1 : 2	3 : 4	5 : 6	7 : 8	9 : 10
1	3	3*	3	3	3	3
2	11	11*	11	11	11	11
3	6	6	4**	4	4	4
4	4	4	6**	6	6	6
5	9	9	9	5**	5	5
6	5	5	5	9**	9	9
7	7	7	7	7	7*	7
8	8	8	8	8	8*	8
9	10	10	10	9	10	2**
10	2	2	2	2	2	10**
11	1	1	1	1	1	1
EXCOUNT		0	1	2	2	3

EXCOUNT – суммарный счетчик обменов. Звездочкой * отмечены сравниваемые элементы; двойными звездочками ** отмечены обмениваемые элементы.

Парный обмен: второй просмотр

Номер	Исходный список	Сравнение позиций				
		2 : 3	4 : 5	6 : 7	8 : 9	10 : 11
1	3	3*	3	3	3	3
2	11	4**	4	4	4	4
3	6	11**	11	11	11	11
4	4	6	5**	5	5	5
5	9	9	6**	6	6	6
6	5	5	9	7**	7	7
7	7	7	7	9**	9	9
8	8	8	8	8	2**	2
9	10	2	2	2	8**	8
10	2	10	10	10	10	1**
11	1	1	1	1	1	10**
EXCOUNT		1	2	3	4	5

Парный обмен: следующие просмотры

№	После 2-ого просмотра	Просмотры									
		3	4	5	6	7	8	9	10	11	12
1	3	3	3	3	3	3	3	2^{**}	2	1^{**}	1
2	4	4	4	4	4	4	2^{**}	3^{**}	1^{**}	2^{**}	2
3	11	5^{**}	5	5	5	2^{**}	4^{**}	1^{**}	3^{**}	3	3
4	5	11^{**}	6^{**}	6	2^{**}	5^{**}	1^{**}	4^{**}	4	4	4
5	6	6	11^{**}	2^{**}	6^{**}	1^{**}	5^{**}	5	5	5	5
6	7	7	2^{**}	11^{**}	1^{**}	6^{**}	6	6	6	6	6
7	9	2^{**}	7^{**}	1^{**}	11^{**}	7^{**}	7	7	7	7	7
8	2	9**	1^{**}	7^{**}	7	11^{**}	8^{**}	8	8	8	8
9	8	1^{**}	9^{**}	8^{**}	8	8	11^{**}	9^{**}	9	9	9
10	1	8**	8	9^{**}	9	9	9	11^{**}	10^{**}	10	10
11	10	10	10	10	10	10	10	10	11^{**}	11	11
EXCOUNT		3	3	3	2	3	3	3	2	1	0

Метод стандартного обмена перемещает один элемент списка в соответствующую позицию при каждом просмотре. Таким образом, при первом просмотре запись с наибольшим ключом помещается в последнюю позицию, при втором просмотре запись со следующим по величине ключом перемещается в предпоследнюю позицию и т.д. Метод может быть преобразован так, что будет размещать элементы по убыванию.

При первом просмотре первый элемент списка сравнивается со своим непосредственным преемником, и, если этот преемник меньше, сравнивается с элементом из третьей позиции. Обмен происходит, если необходимо больший из них разместить в третьей позиции. Затем позиция 3 сравнивается с позицией 4, позиция 4 с позицией 5 и т.д. Когда позиция $\boldsymbol{N-1}$ сравнивается с позицией $\boldsymbol{N}$, просмотр заканчивается.

Если в списке элемент $k-1$ расположен раньше k, то при каждом сравнении $(k-1)$: k больший элемент попадает в позицию k. Элемент, перемещаемый вниз, считается в данный момент наибольшим в списке. Когда при сравнении обнаруживается, что -й элемент больше, обмена не происходит. Следовательно, каждый раз сравниваются элемент, считающийся наибольшим $(k-1)$, и его непосредственный преемник k.

Второй просмотр идентичен первому с той разницей, что уже установленная позиция исключается из последовательности. Каждый последующий просмотр исключает очередную установленную позицию, укорачивая список.

Подсчет обмена ведется для каждого просмотра. Просмотр, в результате которого число обмена не увеличилось, заканчивает сортировку. Для

понимания сущности алгоритма стандартного обмена нужно разобрать следующий пример.

П*ример*. Задан список из чисел:

3, 11, 6, 4, 9, 5, 7, 8, 10, 2, 1.

Стандартный обмен: результаты первого просмотра:

2:3	3:4	4:5	5:6	6:7	7:8	8:9	9:10	10:11
3	3	3	3	3	3	3	3	3
6*	6	6	6	6	6	6	6	6
11*	4*	4	4	4	4	4	4	4
4	11*	9*	9	9	9	9	9	9
9	9	11*	5*	5	5	5	5	5
5	5	5	11*	7*	7	7	7	7
7	7	7	7	11*	8*	8	8	8
8	8	8	8	8	11*	10*	10	10
10	10	10	10	10	10	11*	2*	2
2	2	2	2	2	2	2	11*	1*
1	1	1	1	1	1	1	1	11*
1	2	3	4	5	6	7	8	9

Примечание: последняя строка EXCOUNT – суммарный счетчик обмена. Звездочка * означает, что между данными числами происходит обмен.

В результате первого просмотра ключ 11 попадает на последнюю позицию. Число обменов равно EXCOUNT=9. К концу второго просмотра два наибольших элемента занимают соответствующие позиции, к концу третьего – три наибольших и т.д.

Требуется составить программу на основе данного алгоритма.

Контрольные вопросы:

1. Какие элементы сравниваются между собой в методе сортировки обменом?
2. С чем сравнивается каждый элемент из нечетной позиции при нечетных просмотрах?
3. В чем заключается идея парного обмена?
4. Чему равно число сравнений на каждом просмотре?
5. От чего зависит число просмотров?
6. Куда перемещается запись с большим значением при первом просмотре?
7. В чем разница между первым и вторым просмотрами?
8. Как определяется количество сравнений?
9. От чего зависит число сравнений в данном методе?
10. Какой алгоритм вы считаете более удобным для пользователя из двух рассмотренных здесь?

Варианты задания для самостоятельного выполнения те же, которые были предложены в предыдущей работе.

Лабораторная работа № 4. Алгоритм метода просеивания

Цель лабораторной работы – изучение одного из популярных методов обработки информации – алгоритма метода просеивания.

Методические материалы. По мнению специалистов, метод просеивания считается одним из лучших методов сортировки. Его отличие от других методов обмена заключается в том, что не сохраняет фиксированной последовательности сравнений. Исчезает разделение на отдельные просмотры как следствие схемы последовательностей сравнений. Метод просеивания работает так же, как стандартный обмен до тех пор, пока не надо выполнять перестановку.

Порядок выполнения алгоритма. Сравниваемая величина с меньшим ключом поднимается, насколько это возможно, вверх по списку. Она сравнивается «в обратном порядке» со всеми ее линейными предшественниками по направлению к вершине списка. Если ее ключ меньше, чем у предшественника, то выполняется обмен и начинается очередное сравнение. Когда элемент, передвигаемый вверх, встречает элемент с меньшим ключом, этот процесс прекращается и нисходящее сравнение возобновляется с той позиции, с которой выполнялся первый обмен.

Первичное и вторичное сравнения. Предполагается, что нисходящее сравнение называется *первичным*, а восходящее – *вторичным*. Любое первичное сравнение может увеличить число вторичных сравнений. Допустим, если первичное сравнение охватывает позиции 6 и 7, то цепочка вторичных сравнений может иметь самое большее 5 сравнений. Этот максимум достигается, если исходный ключ на позиции 7 меньше всех ключей в списке, расположенных выше этой позиции. Фактическая длина последовательности вторичных сравнений зависит от величины двигающегося вверх элемента относительно величины каждого элемента на предшествующей упорядоченной части списка. Сортировка заканчивается, когда первичное сравнение охватит $(N + 1)$-й элемент.

Пример выполнения алгоритма просеивания. Пусть рассматривается тот же пример, что и в предыдущих работах.

Позиция	1	2	3	4	5	6	7	8	9	10	11
Ключ	3	11	6	4	9	5	7	8	10	2	1

Порядок выполнения алгоритма. Процесс будет начинаться с первичного сравнения позиций 1 и 2. Здесь обмен не нужен, потому что ключ на позиции 1 меньше, чем ключ на позиции 2. Следующий шаг: первичное сравнение позиций 2 и 3. Здесь производится обмен ключей 6 и 11. Теперь последовательность вторичных сравнений начинает продвигать ключ 6 вверх по списку, насколько это возможно. Теперь сравниваются позиции 2 и 1. Так как ключ 3 меньше, чем ключ 6, то обмен не выполняется. Здесь вторичная последовательность кончается, потому что была бы обнаружена вершина списка.

Следующий шаг возобновляет первичное сравнение. Так как последним первичным сравнением было 2 и 3, то следующим будет 3 и 4. В позиции 3 находится ключ 11, который был помещен туда при последнем первичном обмене. Первичное сравнение 3 и 4 обнаруживает инверсию, выполняется

обмен с последующей вторичной последовательностью 2 и 3, 1 и 2 для размещения ключа 4. Заметим, что выше позиции последнего первичного сравнения список упорядочен.

Позиция	Исходный список	ПС 1 и 2	ПС 2 и 3	ВС 1 и 2	ПС 3 и 4	ВС 2 и 3	ВС 1 и 2
1	3	3*	3*	3*	3	3	3*
2	11	11*	6*←	6*	6	4*←	4*
3	6	6	11*←	11	4*←	6*←	6
4	4	4	4	4	11*←	11	11
5	9	9	9	9	9	9	9
6	5	5	5	5	5	5	5
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	10	10	10	10	10	10	10
10	2	2	2	2	2	2	2
11	1	1	1	1	1	1	1
Позиция	ПС 4 и 5	ВС 3 и 4	ПС 5 и 6	ВС 4 и 5	ВС 3 и 4	ВС 2 и 3	ПС 6 и 7
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
Позиция	ВС 5 и 6	ВС 4 и 5	ПС 7 и 8	ВС 6 и 7	ВС 5 и 6	ПС 8 и 9	ВС 7 и 8
1					3	3	3
2					4	4	4
3					5	5	5
4					6	6	6
5					7*	7	7
6					8*	8	8
7					9	9	9*
8					11	10*←	10*
9					10	11*←	11
10					2	2	2
11					1	1	1

Позиция	ПС 9 и 10	ВС 8 и 9	ВС 7 и 8	ВС 6 и 7
1	3	3	3	3
2	4	4	4	4
3	5	5	5	5
4	6	6	6	6
5	7	7	7	7
6	8	8	8	2*
7	9	9	2*←	8*
8	10	2*←	9*←	9
9	2*←	10*←	10	10
10	11*←	11	11	11
11	1	1	1	1

В качестве примера использован тот же список, который рассматривался на предыдущих работах. На следующих рисунках показаны первичные и вторичные сравнения до того момента, как ключ 2 достигает позиции 6. Эта вторичная цепочка будет содержать более пяти сравнений, оканчиваясь размещением ключа 2 в позиции 1 в результате сравнения 1 и 2. Затем будет выполнено последнее первичное сравнение 10 и 11, а вторичная последовательность из девяти сравнений и обменов переместит ключ 1 в позицию 1, упорядочив тем самым список.

Анализ алгоритма. Необходимо определить количество сравнений и количество обменов при выполнении данного алгоритма.

Количество сравнений минимально, оно равно $N-1$, когда исходный список полностью упорядочен. Количество сравнений максимально, когда список имеет обратный порядок. В этом случае каждое из $N-1$ первичных сравнений порождает вторичную последовательность, что увеличивает общее расстояние от точки первичного сравнения до вершины списка. Средняя длина вторичной последовательности будет равна $(N-1)/2$. Следовательно, общее количество сравнений равно $(N-1)^2/2$. Поэтому можно принять в качестве оценки минимального и максимального числа сравнений соответственно N и $N^2/2$.

Для того чтобы определить ожидаемое сравнение, необходимо учесть, что средняя длина поиска любой позиции списка равна половине длины списка. Процесс «просеивания» аналогичен поиску в упорядоченном списке чисел нужной позиции для нового числа. Целесообразно предположить, что новый элемент будет располагаться у центра списка. Следовательно, средняя длина вторичной последовательности приблизительно равна $(N-1)/4$, потому что каждая последовательность будет равна половине всей максимальной длины. В случае $N-1$ первичного сравнения будет приблизительно $(N-1)^2/4$ вторичных сравнений. К этой величине необходимо добавить $N-1$ первичных сравнений. Тогда общее число

ожидаемых сравнений будет равно $N-1+((N-1)^2/4)$. Тогда оценка $N+N^2/4$ не приведет к ошибке.

Количество обменов минимальное, когда равно 0, если список упорядочен; максимальное – равно числу сравнений, если список имеет обратный порядок. Ожидаемое число обменов очень близко к ожидаемому числу сравнений. Самое большее, может быть $2 \cdot N-2$ сравнений, которые не сопровождаются обменами. Это – первичные и последние вторичные сравнения. Если каждое первичное сравнение вызывает обмен, то имеется максимальное число обменов. Так как ожидается примерно $N^2/4$ вторичных сравнений, то можно ожидать примерно $N^2/4$ обменов.

Контрольные вопросы:

1. Назовите основные отличия метода просеивания от других методов, рассмотренных ранее.
2. Какое сравнение называется первичным?
3. Какое сравнение называется вторичным?
4. В каких случаях достигается максимум цепочки вторичных сравнений?
5. Чему равно минимальное число сравнений, когда список полностью упорядочен?
6. Когда достигается максимальное число сравнений?
7. Назовите оценку минимального и максимального числа сравнений.
8. В каких случаях имеется максимальное число обменов?
9. Назовите примерное число обменов.
10. Каким преимуществом, по вашему мнению, обладает алгоритм метода просеивания перед другими?

Варианты задания для самостоятельного выполнения те же, что были предложены в предыдущей работе.

Лабораторная работа № 5. Алгоритм сортировки вставками

Цель лабораторной работы – изучение основных особенностей применения алгоритма Шелла, являющегося расширением сортировки просеивания.

Теоретические материалы. Под сортировку выделяется количество памяти, равное предполагаемой длине окончательного списка из всех элементов. Счетчик длины списка в самом начале устанавливается равным нулю. При помощи этого счетчика контролируется длина области поиска нужной позиции для элемента, вставляемого в список. Сортировка привлекается для каждого элемента. Один «вызов» сортирующей подпрограммы размещает один элемент в списке и увеличивает счетчик на единицу.

Первый элемент помещается в верхнюю позицию области вывода. Следующий элемент, присоединяемый к списку, сравнивается с первым. Если ключ нового элемента больше, он помещается в позицию, следующую за позицией первого элемента. Если ключ нового элемента меньше, то первый элемент перемещается в позицию 2, а новый элемент помещается в позицию 1.

В дальнейшем все новые элементы последовательно сравниваются с каждым элементом списка, начиная с первого, до тех пор, пока не встретится больший элемент. Этот больший элемент и все последующие элементы списка передвигаются на одну позицию вниз. Таким образом освобождается место, на которое вставляется новый элемент.

Пример. Рассматривается тот же пример, в котором исходным списком является (3, 11, 6, 4, 9, 5, 7, 8, 10, 2, 1), и производится сортировка его методом линейной вставки.

Номера обращений					
первое	второе	третье	четвертое	пятое	шестое
длина=0	длина=1	длина=2	длина=3	длина=4	длина=5
новый =3	новый =11	новый =6	новый =4	новый =3	новый =5
TOSORT=3**	TOSORT=3	TOSORT=3	TOSORT=3	TOSORT=3	TOSORT=3
0	11**	6**	4**	4	4
0	0	11*	6*	6	5**
0	0	0	11*	9**	6*
0	0	0	0	11*	9*
0	0	0	0	0	11*
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

седьмое	восьмое	девятое	десятое	одиннадцатое
длина=6	длина=7	длина=8	длина=9	длина=10
новый =7	новый =8	новый =10	новый =2	новый =1
TOSORT=3	TOSORT=3	TOSORT=3	TOSORT=2**	TOSORT=1**
4	4	4	3*	2*
5	5	5	4*	3*
6	6	6	5*	4*
7**	7	7	6*	5*
9*	8**	8	7*	6*
11*	9	9	8*	7*
0	11*	10**	9*	8*
0	0	11*	10*	9*
0	0	0	11*	10*
0	0	0	0	11*

Примечание: последняя строка EXCOUNT – суммарный счетчик обмена. Звездочка * означает, что между данными числами происходит обмен.

При первом обращении к сортировке длина списка вывода равна нулю. Первым элементом, помещаемым в список вывода, является ключ 3. В столбце, озаглавленном «Первое обращение», ключ 3 отмечен двойной звездочкой, установленной на начале области вывода – ячейке TOSORT. Длина списка увеличивается на 1. Как обычно, все элементы, помещаемые в список, указываются двойной звездочкой, а все перемещаемые – одной звездочкой.

При втором обращении к сортировке – длина +1. Новый ключ 11 сравнивается с ключом 3, ищется больший и помещается во вторую позицию. Для третьего обращения длина списка устанавливается равной 2. Ключ 6, больший ключа 3, сравнивается затем с ключом 11, и выясняется, что ключ 6 меньше. Поскольку ключ 6 должен быть между ключом 3 и ключом 11, то позиция, ныне занимаемая ключом 11, должна быть освобождена для ключа 6. Ключ 11 опускается на одну позицию, а ключ 6 вставляется в список на нужное место.

Все последующие обращения к сортировке происходят аналогичным образом. Ключ 4 вызывает перемещение вниз ключей 11 и 6, так что ключ 4 может быть вставлен между ключами 3 и 6, ключ 9 вызывает перемещение ключа 11 и т.д.

Требуется составить программу на основе данного алгоритма. Результаты выполнения должны быть анализированы с целью определения сложности алгоритма.

***Контрольные вопросы*:**

1. Какая идея является основой для метода вставки?
2. В каких случаях целесообразно использовать сортировку вставками?
3. Сколько раз осуществляется «вызов» сортирующей подпрограммы?
4. Что собой представляет сортирующая подпрограмма?
5. Какое количество памяти выделяется для окончательного списка?
6. Сколько сравнений осуществляется, если элементы поступают в нужной последовательности?
7. Чему равно число перемещений элементов при использовании метода сортировки линейной вставки?
8. В чем заключается преимущество метода вставки перед методами обмена?
9. Каковы недостатки данного метода вставки в сравнении с предыдущими методами.
10. Какому алгоритму вы отдадите предпочтение, если придется осуществлять выбор между рассмотренными выше алгоритмами?

Варианты задания для самостоятельного выполнения те же, что были предложены в предыдущей работе.

Лабораторная работа № 6. Алгоритм Шелла

Цель лабораторной работы – изучение одного из популярных среди пользователей алгоритмов, который известен как алгоритм Шелла.

Теоретические материалы. Метод сортировки Шелла, или метод слияния с обменом, построен на предположении, что предшествующие просмотры используют технический прием, в котором сравниваются и обмениваются не непосредственные соседи, а элементы, отстоящие на заданное расстояние. Например, позиция 1 сравнивается с позицией 5, позиция 5 с позицией 9, позиция 9 с позицией 13 и т.д. Когда обнаружена инверсия, цепочка вторичных сравнений охватывает те элементы, которые входили в последовательность первичных сравнений. Например, если обнаружена инверсия между позициями 9 и 13, то возможная вторичная последовательность состоит из сравнений 5:9 и 1:5.

На каждом просмотре шаг между сравниваемыми элементами определяется путем сокращения вычисленного исходного шага. На последнем просмотре он сокращается до 1. Целью метода на разных просмотрах является сокращение числа вторичных сравнений и обмена на поздних просмотрах. В этом методе элементы могут совершать большие скачки к нужным позициям на ранних просмотрах, а не передвигаться на одну позицию за один раз. На последнем просмотре числа будут стремиться располагаться близко к своим позициям и, следовательно, потребуют незначительных перемещений при окончательном размещении.

Пример. Пусть рассматривается тот же список, что использован в предыдущей работе. Здесь будет использован технический прием разбиения на части, описанный Шеллом. Список длины N разбивается на $N/2$ частей (если N является нечетным, то $(N-1)/2$). Каждая из частей содержит два элемента. Элементы каждой части отстоят друг от друга на $N/2$ позиций. В списке из 11 элементов будет 5 частей с расстоянием в 5 позиций между элементами одной части. Следовательно, первая часть содержит элементы 1, 6 и 11, а вторая часть – элементы из позиций 2 и 7, … , и последняя часть – элементы из позиций 5 и 10.

Первый просмотр в этом методе упорядочивает элементы каждой части. Последовательность сравнений передвигается от одной части к другой по ходу просмотра списка. Например, первичная последовательность сравнений 1:6, 2:7, 3:8, т.е. первый элемент части 1 сравнивается со вторым элементом этой же части, затем первый элемент части 2 сравнивается со вторым элементом части 2 и т.д. В следующей таблице показана последовательность сравнений, обмена и порядок в списке после нескольких сравнений и в конце первого просмотра.

Первый просмотр сортировки Шелла с числом частей, равным $N/2$, имеет следующий вид:

1		Первичные			Вторичные
		2	3	4	5
Исходный список		После 2:7	После 5:10	После 6:11	После 1:6
1	3	3	3	3	1←
2	11	7←	7	7	7
3	6	6	6	6	6
4	4	4	4	4	4
5	9	9	2←	2	2
6	5	5	5	1←	3←
7	7	11←	11	11	11
8	8	8	8	8	8
9	10	10	9	10	10
10	2	2	10←	9	9
11	1	1	1	5←	5

Примечание. Стрелки указывают обмены.

Последовательность сравнений и обменов (по позициям) имеет следующий вид:

Сравнения	Обмены
1 : 6	-
1 : 7	2, 7
3 : 8	-
4 : 9	-
5 : 10	5, 10
6 : 11	6, 11
1 : 6 (вторичное)	1 , 6

До сравнения 6:11 каждое сравнение охватывало новую часть, и при этом каждый элемент сравнивался только один раз. Сравнение 6:11 использует два элемента из первой части, один из которых – 6 был использован ранее. Следовательно, известно, что ключ, находящийся в позиции 6, – больший из двух предыдущих элементов. Если сравнение выявит, что содержимое позиции 11 больше, то часть (1, 6, 11) упорядочена. Однако если ключ из позиции 11 меньше, то необходимо выполнить обмен между позициями 6 и 11. Состояние списка после этого обмена (ключей 1 и 5) показано на 4-ом столбце таблицы.

Теперь уже нет информации об относительной величине ключа, перемещенного в позицию 6, и ключа первого элемента этой части (из позиции 1). Для того чтобы определить, полностью ли упорядочена эта часть, необходимо еще раз выполнить сравнение 1:6. Сравнения этого типа, вызванные обменом меньших величин в этой части, называются вторичными.

Вторичное сравнение между позициями 1 и 6 вызывает обмен между соответствующими элементами, в результате чего часть 1 становится

упорядоченной. Все части теперь упорядочены. Они получены после пятого сравнения, т.е. к концу первого просмотра; результаты приведены на 5-ом столбце таблицы.

В конце этого просмотра части переопределяются путем изменения шага между элементами одной части. Здесь предложено устанавливать шаг для второго просмотра; он считается равным половине начального шага. Берется целая часть, т. е. шаг = 2.

Второй просмотр упорядочивает новые части тем же способом, что в первый просмотр; он (шаг=2) представлен в следующем виде:

Сравнения		Обмены
Первичные	Вторичные	
1 : 3		
2 : 4		2, 4
3 : 5		3, 5
	1 : 3	
4 : 6		4, 6
	2 : 4	2,4
5 : 7		
6 : 8		
7 : 9		7,9
	5 : 7	
8 : 10		
9 : 11		
	7 : 9	9,11
	5 : 7	7,9
	3 : 5	
		5,7

Окончательный список после просмотра 2:

Исходный список	1	2	3	4	5	6	7	8	9	10	11
	1	7	6	4	2	3	11	8	10	9	5
Окончательный список	1	3	2	4	5	7	6	8	10	9	11

В конце второго просмотра части переопределяются вновь. Шаг опять сокращается до половины предыдущего шага (шаг=1).

Последний просмотр сортировки Шелла (шаг=1):

Сравнения		Обмены
Первичные	Вторичные	
1 : 2		
2 : 3		2,3
	1 : 2	
3 : 4		
4 : 5		
5 : 6		
6 : 7		6,7
	5 : 6	
7 : 8		
8 : 9		
9 : 10		9,10
	8 : 9	
10 : 11		

Окончательный упорядоченный список

Исходный список	1	2	3	4	5	6	7	8	9	10	11
	1	7	6	4	2	3	11	8	10	9	5
Окончательный список	1	2	3	4	5	6	7	8	9	10	11

Итак, три просмотра упорядочивают список. Число просмотров равно числу делений на 2 для сведения N к 1. Когда N равно степени двойки, алгоритм требует $logN$ просмотров.

Контрольные вопросы:

1. Что собой представляют первичные сравнения?
2. Что такое вторичные сравнения?
3. Что такое «шаг просмотра» в методе Шелла?
4. В чем заключается цель метода Шелла?
5. Почему в методе Шелла шаг между сравниваемыми элементами сокращается?
6. Чему в методе Шелла будет равняться шаг на последнем просмотре?
7. Как выбирается начальный шаг в методе Шелла?
8. Каковы основные отличия метода Шелла от метода сравнения вставками?
9. Назовите основные преимущества метода Шелла.

Варианты задания для самостоятельного выполнения те же, что были предложены в предыдущей работе.

Лабораторная работа № 7. Алгоритм поиска в глубину и ширину

Цель лабораторной работы – изучение основных способов представления графов в оперативной памяти ЭВМ и практическая реализация алгоритма работы с графами.

Теоретические и методические материалы. Для выполнения данной лабораторной работы необходимы сведения из разделов теории графов. Из курса дискретной математики известно, что существуют различные способы представления графов в памяти компьютера, которые различаются объемом занимаемой памяти и скоростью выполнения операций над графами. Представление выбирается, исходя из потребностей конкретной задачи.

Обычно рассматриваются четыре наиболее часто используемых представления с указанием характеристики $P(n, m)$ – объема памяти для каждого представления. Здесь n – число вершин, m –число ребер графа. К ним относятся: матрица смежности, матрица инциденций, массив дуг и списки смежности.

Матрица смежности графа G с конечным числом вершин n (пронумерованных числами от 1 до n) — это квадратная матрица A размера n. Значение элемента $a[i, j]$ данной матрицы равно единице, если i-я и j-я вершины графа являются смежными, иначе значение равно нулю. Такая матрица называется *бинарной*. Для простого графа элементы главной диагонали равны 0. Симметрия матрицы смежности является её плюсом в плане экономии памяти, т. к. достаточно сохранить только половину матрицы.

Одной из популярных задач, связанных с представлением графа, является *обход графов* – некоторое систематическое перечисление его вершин или ребер. Наибольший интерес представляют обходы, использующие локальную информацию (списки смежности).

Обход графа (поиск на графе) – это процесс систематического просмотра всех ребер или вершин графа с целью отыскания ребер или вершин, удовлетворяющих некоторому условию. Для организации поиска на графах используются обходы в глубину и в ширину; среди всех обходов наиболее известны *поиск в ширину и в глубину*, потому что алгоритмы поиска в ширину и в глубину лежат в основе многих конкретных алгоритмов на графах.

При решении многих задач, в которых используются графы, необходимы методы регулярного обхода их вершин и ребер. К стандартным и наиболее распространенным методам относятся:

– поиск в глубину (*Depth First Search, DFS*);

– поиск в ширину (*Breadth First Search, BFS*).

Пусть для обозначения графа используется следующая формула: $G(U, V)$, где U – множество вершин, V – множество ребер, т.е. неупорядоченных пар различных элементов множества U.

Алгоритм поиска в ширину и в глубину имеет следующие этапы:

1. Ввод графа $G(U,V)$, представленный списками смежности, состоящей из массива указателей: $array\ [1..n]\ of\ integer$ на списки смежных вершин.

2. Цикл, который показывает, что все вершины не отмечены.

3. Начало обхода с произвольной вершины и помещение в выбранную структуру данных T; необходимо отметить эту вершину.

4. Извлечение эту вершины из структуры данных T и возвращение ее в качестве очередной пройденной вершины.

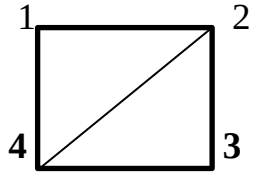
5. Цикл, который помещает очередную вершину в структуру данных T, если она не отмечена, и будет отмечена. Этот цикл выполняется до тех пор, пока T не станет пустым множеством.

Замечание: Если T является стеком, т.е. $LIFO - Last\ InFirst\ Out$, то обход называется *поиском в глубину*. Если T является очередью, т.е. $FIFO - First\ In\ First\ Out$, то обход называется *поиском в ширину*.

Обсуждение алгоритма. По алгоритму обходятся только вершины, которые попали в структуру данных T. В нее попадают только неотмеченные вершины, и при попадании вершина отмечается. Следовательно, любая вершина будет обойдена только один раз. В структуру данных T попадает не более n вершин, и на каждом шаге удаляется из нее одна вершина. Следовательно, всего будет выполнено не более n шагов.

Пример. Дан следующий простой граф. Требуется показать протоколы поиска в глубину и в ширину.

Диаграмма заданного графа:



Предполагается, что начальной вершиной является вершина 1. Протокол поиска в глубину для заданного графа представлен в следующей таблице:

u	1	4	3	2
T	2,4	2,3	2	0

Протокол поиска в ширину в следующей таблице:

u	1	2	4	3
T	2,4	4,3	3	0

Контрольные вопросы:

1. Опишите распространенные способы представления графов в памяти компьютера.
2. Для чего используется обход графов?
3. В каких случаях используется поиск в ширину?
4. В каких случаях используется поиск в глубину?
5. Какой из этих поисков происходит быстрее?

6. Почему вначале считается, что вершины графа не отмечены?
7. Для чего используется структура данных T?
8. Определите используемый объем памяти.

Задание для самостоятельного выполнения (для всех вариантов): Задана матрица смежности следующего вида

$$G = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

Требуется:
– составить матрицу инциденций;
– списки смежности;
– массив дуг;
– составить и выполнить программы для поиска в глубину и ширину;
– показать протоколы поиска в ширину и в глубину;
– провести анализ полученных результатов.

Лабораторная работа № 8. Алгоритм Флойда – Уоршалла

Цель лабораторной работы – получение навыков применения одного из популярных алгоритмов поиска кратчайшего пути в графе – алгоритма Флойда.

Теоретические и методические материалы. Задача нахождения кратчайшего пути может иметь широкое практическое применение, поэтому умение использовать алгоритмы для ее решения является обязательным для программиста. В данной лабораторной работе рассматривается один из этих алгоритмов – алгоритм Флойда. В некоторых источниках этот алгоритм называется алгоритмом Уоршалла – Флойда. Для выполнения лабораторной работы необходимы знания по дискретной математике, в частности, по теории графов.

Известно, что задача о кратчайшем пути — задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов всех ребер, составляющих путь. Задача о кратчайшем пути является одной из важнейших классических задач теории графов. В настоящее время известно множество алгоритмов для её решения. У данной задачи существуют и другие названия: задача о минимальном пути или, в устаревшем варианте, задача о дилижансе.

Значимость данной задачи определяется её различным практическим применением. Например, в GPS-навигаторах осуществляется поиск кратчайшего пути между точкой отправления и точкой назначения. В качестве вершин выступают перекрёстки, а дороги являются рёбрами, которые лежат между ними. Если сумма длин дорог между перекрёстками минимальна, тогда

найденный путь – самый короткий. Другим классическим примером является задача о коммивояжере.

Алгоритм Флойда используется для определения кратчайшего пути между всеми парами вершин в графе. В нем для хранения информации о путях используется матрица $A[1..n, 1..n]$:

$$A[i,j] = \begin{cases} k, \text{если } k - \text{первая вершина, достигаемая на кратчайшем пути из } i \text{ в } j; \\ 0, \text{если из } i \text{ в } j \text{ нет пути.} \end{cases}$$

Здесь $n-$ количество вершин в графе; матрица $A[i,j]$ хранит информацию обо всех путях в графе. На пересечении -ой строки и j-го столбца матрицы стоит значение веса ребра из вершины i в вершину j. Главная диагональ матрицы смежности это всегда нули, потому что ребер из i-ой вершины в i-ую в графе быть не должно. В том случае, когда между вершинами нет ребра, в матрице будет стоять условная бесконечность (∞).

Непосредственное представление всех путей потребовало бы памяти объема n^3. С целью экономии памяти используется динамическое вычисление некоторой части информации вместо ее хранения в памяти. В данном случае любой конкретный путь $< u, v >$ извлекается из матрицы с помощью следующего алгоритма:

$w := u$; **yield** w {первая вершина }
while $w \neq v$ выполнить
  $w := A[w, v]$; **yield** w {следующая вершина }
end while

Алгоритм Флойда представляет собой простой перебор всех путей и выбор из них наименьшего. Реализуется поиск наименьшего расстояния последовательным перебором всех путей. В три цикла, внешний выбирает вершину, через которую пытаются найти лучший путь. Внутренние два цикла перебирают пары вершин, между которыми ищется минимальное расстояние.

Матрицу смежности удобнее всего хранить в файле в следующем виде: первой строкой – количество вершин, затем – непосредственно сама матрица. В качестве бесконечно большого расстояния (∞), когда вершины не связаны ребром, взять конкретную константу достаточно большой величины.

Контрольные вопросы:

1. Поясните сущность задачи о кратчайшем пути.
2. Какие практические задачи могут быть сведены к решению задачи о кратчайшем пути?
3. Объясните сущность алгоритма Флойда.
4. Почему ставится знак бесконечности ∞ вместо нуля, когда рассматривается расстояние от вершины i до вершины i?
5. Как осуществляется перебор путей для поиска наименьшего пути?
6. Какой критерий является для выбора оптимального решения данной задачи?
7. Какой объем памяти необходим для данного алгоритма?

8. По вашему мнению, можно ли использовать другие методы для решения данной задачи?

Задания для самостоятельного выполнения:

№ 1.

Вершины	1	2	3	4	5
1	0	4	5	7	3
2	4	0	3	2	7
3	5	3	0	8	7
4	6	3	2	0	7
5	6	8	3	4	0

№ 2.

Вершины	1	2	3	4	5
1	0	4	5	7	8
2	6	0	4	5	7
3	8	3	0	8	7
4	9	3	2	0	7
5	3	7	4	10	0

№ 3.

Вершины	1	2	3	4	5
1	0	11	4	8	4
2	4	0	3	2	10
3	3	3	0	6	3
4	6	3	5	0	7
5	9	8	9	4	0

№ 4.

Вершины	1	2	3	4	5
1	0	10	5	3	4
2	8	0	2	2	10
3	3	4	0	5	2
4	6	3	5	0	8
5	3	9	4	9	0

№5.

Вершины	1	2	3	4	5
1	0	7	3	5	3
2	3	0	5	4	5
3	6	2	0	8	7
4	4	9	8	0	9
5	8	2	6	3	0

9 Лабораторная работа № 9. Алгоритм Дейкстры

Цель лабораторной работы – получение навыков применения одного из популярных алгоритмов поиска кратчайшего пути в графе – алгоритма Дейкстры.

Теоретические и методические материалы. Алгоритм Дейкстры также используется для нахождения кратчайшего пути между вершинами в графе. Поэтому многие теоретические вопросы для данной работы совпадают с материалами, приведенными в предыдущей лабораторной работе. Для описания алгоритма Дейкстры нужно ввести следующие обозначения используемых параметров.

Обозначения параметров:

$A[i,j]-$ матрица, которая хранит информацию о путях;

$i,j-$ вершины графов;

$n-$ число вершин;

$T[u]-$ длина кратчайшего пути от i к u, если вершина u лежит на кратчайшем пути от i к j;

$S[u]-$ вершина, предшествующая u на кратчайшем пути.

После ввода данных, т.е. ввода элементов матрицы $A[i,j]$, выполняются этапы алгоритма.

Данный алгоритм Дейкстры имеет следующие этапы:

1. Начало цикла по счетчику u от 1 до n, где выполняются следующие операции:

1.1. Пока кратчайший путь неизвестен, $T[u] \coloneqq \infty$.

1.2. Все вершины не отмечены $X[u] \coloneqq 0$;

конец цикла.

2. Ничего не предшествует $S[i] \coloneqq 0$.

3. Длина кратчайшего пути равна нулю $T[i] \coloneqq 0$, и он неизвестен, $X[i] \coloneqq 1$.

4. Текущая вершина $u \coloneqq i$.

5. Цикл для всех вершин v:

если $X[v] = 0$ и $T[v] > T[u] + A[u,v]$, то $T[v] = T[u] + A[u,v]$, то найден более краткий путь из i в v через u; запомнить его $S[v] \coloneqq u$;

конец цикла.

6. Поиск конца кратчайшего пути $j \coloneqq \infty; u \coloneqq 0$.

7. Начало цикла по счетчику v от 1 до n, где выполняются следующие операции:

если $X[v] = 0$ и $T[v] < j$, то вершина u закачивает кратчайший путь из i;

конец цикла.

8. Если $u = 0$, то нет пути из i в j. Прекращаются дальнейшие операции (останов).

9. Если $u = j$, то найден кратчайший путь из i в j. Прекращаются дальнейшие операции (останов).

10. Найден кратчайший путь из i в u, $X[u] := 1$;

11. Идти к 5.

Вначале разобрать этапы выполнения алгоритма, ответить на контрольные вопросы и на основе алгоритма разработать программу для его реализации. Выполнить индивидуальное задание и провести сравнительный анализ с результатами выполнения алгоритма Флойда. Оценить сложности алгоритмов Флойда и Дейкстры для одного и того же варианта задания.

Контрольные вопросы:

1. Поясните сущность алгоритма Дейкстры.
2. Назовите основные отличия алгоритмов Флойда и Дейкстры.
3. Если кратчайший путь неизвестен, то почему используется оператор $T[u] := \infty$?
4. Если все вершины графа не отмечены, то почему используется оператор $X[u] := 0$?
5. Почему для поиска конца кратчайшего пути использованы операторы $j := \infty; u := 0$?
6. Укажите условие окончания процесса для данного алгоритма.
7. Определите, какой объем памяти необходим для данного алгоритма.
8. По вашему мнению, можно ли использовать другие методы для решения данной задачи?

Индивидуальные задания для самостоятельного выполнения приведены в предыдущей работе.

10 Лабораторная работа № 10. Алгоритм симметричного обхода бинарного дерева

Цель лабораторной работы – изучение одного из популярного и использующего стека алгоритма симметричного обхода бинарного дерева.

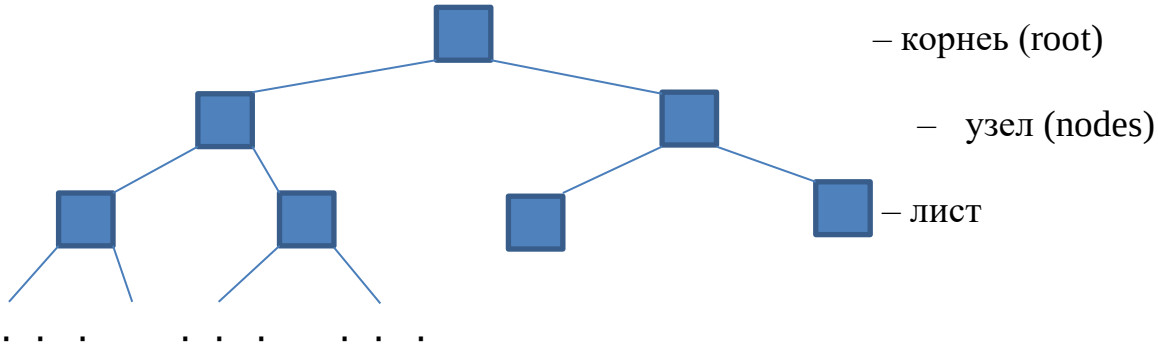
Теоретические и методические материалы. Одной из задач, которые оказывают большое влияние на практику программирования, является задача *представления деревьев* в ЭВМ. Во многих приложениях обнаруживается нелинейный порядок объектов, где элементы могут иметь нескольких наследников. Например, в фамильном дереве родитель может иметь нескольких потомков (детей). Древовидная структура характеризуется множеством узлов (nodes), происходящих от единственного начального узла, называемого корнем (root).

Известно, что всякое упорядоченное дерево можно представить *бинарным* деревом. Такие бинарные деревья (binary trees) имеют

унифицированную структуру, допускающую разнообразные алгоритмы прохождения и эффективный доступ к элементам. Изучение бинарных деревьев дает возможность решать наиболее общие задачи, связанные с деревьями, поскольку любое дерево общего вида можно представить эквивалентным ему бинарным деревом. Поэтому целесообразно рассматривать представление в ЭВМ бинарного дерева.

Представление бинарных деревьев в ЭВМ. Существуют различные представления бинарных деревьев, из которых часто используются *списочные структуры*. В них каждый узел представляется записью, содержащей два поля с указателями на левый (l, left) и правый (r, right) узлы, и еще одно поле i для хранения указателя на информацию об узле. Дерево представляется указателем на корень.

Бинарное дерево может быть изображено в следующем виде:



Большинство алгоритмов работы с деревьями основаны на обходах. Среди существующих обходов популярным является симметричный обход с использованием стека.

Обозначения, используемые в алгоритме:

$r -$ указатель на корень;

$i -$ указатель на узел;

$T -$ стек;

$q -$ указатель на текущий узел.

Алгоритм состоит из следующих этапов:

1. Ввод бинарного дерева, представленного списочной структурой, где $r -$ указатель на корень.

2. Вначале стек пуст $T := 0$.

3. Указать на корень дерева $q := r$.

4. Если $q = nil$, также $T = 0$, то обход завершен и $stop$.

5. В противном случае, если $q \neq nil$, то левое поддерево обойдено $q \leftarrow T$.

6. Очередной узел при симметричном обходе: $yield\ q$.

7. Начало обхода правого поддерева $q := q.r$, в противном случае необходимо запомнить текущий узел $q \rightarrow T$, затем осуществлять обход левого поддерева.

8. Идти к 2.

Контрольные вопросы:

1. Для чего используются деревья в программировании?
2. Являются ли деревья классами графов?
3. Что такое дерево в представлении программистов?
4. Какие деревья называются упорядоченными?
5. Что собой представляет бинарное дерево?
6. Назовите основные представления дерева в ЭВМ.
7. Что такое обходы деревьев?
8. Какие существуют обходы бинарных деревьев?
9. Что такое списочные структуры?

Индивидуальное задание для самостоятельного выполнения (для всех вариантов***)***

Требуется выполнить следующие виды работ:

1. Построение бинарного дерева. Сначала необходимо определить структуру для создания корня и узлов дерева:

```
template <class T.
struct TNode {
      T value;
      TNode *qleft, *qpright;
      //constructor
      TNode()  {
          qleft = qright = 0;
   }
};
```

Здесь поля qleft и qright являются указателями на потомков данного узла, а поле value предназначено для хранения информации.

2. Написать рекурсивную функцию, которая будет вызываться при создании дерева:

```
template<class T>
void makeTree(TNode<T>** qq, T x)   {
        if(!(*qq))   {
             TNode<T>* q =new TNode<T>(!);
             q->value = x;
             *qq = q;
}
Else  {
   If((*qq)->value > x)
      makeTree(&((*qq)->qleft),  x);
    else
            makeTree(&((*qq)->qright),  x);
     }
}
```

Эта функция добавляет элемент x к дереву, учитывая его величину. При этом создается новый узел дерева.

3. Составить программу симметричного обхода созданного вами бинарного дерева.

4. Выполнить программу и сделать анализ полученных результатов.

Примечание. Приведенные выше фрагменты программ могут быть написаны на другом языке, по желанию обучающегося (магистранта).

Список литературы

1 Коварцев А.Н. Алгоритмы и анализ сложности. Учебник. – Самара: Изд-во Самарского университета, 2016. – 128 с.

2 Инюшин В.И. Теория алгоритмов: Учебное пособие. – М.: ИНФРА-М, 2012. – 318 с.

3 Томас Кормен. Чарльз Лайзерсен, Рональд Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ. 3-изд.: пер. с англ. – М.: ООО «И.Д. Вильямс», 2013. – 1328 с.

4 Кнут Д. Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы. – М.: «Вильямс», 2012.

5 Альфред Ахо, Джон Хопкрофт, Джеффер Ульман. Структура данных и алгоритмы: пер. с англ. Учебное пособие. – М.: Изд. дом «Вильямс», 2010. – 400 с.

5 Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. – М.: «Вильямс», 2012.

6 Кнут Д. Искусство программирования для ЭВМ. Т. 4. Комбинаторные алгоритмы. – М.: «Вильямс», 2012.

7 Иванов Б.Н. Дискретная математика. Алгоритмы и программы: Учебное пособие. – М.: Лаборатория базовых знаний, 2003. – 288 с.

8 Новиков Ф.А. Дискретная математика для программистов. – СПб.: «Питер», 2002. – 303 с.

9 Стивен С. Скиена. Алгоритмы. Руководство и разработка. – 2-ое изд. - СПб.: БХВ-Петербург, 2011. – 720 с.

10 Крупский В.И., Плиско В.Е. Теория алгоритмов: Учебное пособие. – М.: Изд. центр «Академия», 2009. – 208 с.

11 Алексеев В.Е., Талаков В.А. Графы и алгоритмы. Структуры данных. Модели вычислений. – М.: БИНОМ. Лаборатория знаний. – 2012. – 320 с.

Сводный план 2020 г., поз.103

Зауытбек Куралбаевич Куралбаев

АЛГОРИТМЫ И ИХ СЛОЖНОСТЬ

Методические указания по выполнению лабораторных работ
для магистрантов специальности 7М06103 – «Вычислительная техника и программное обеспечение»

Редактор: Жанабаева Е.Б.

Специалист по стандартизации: Данько Е.Т.

Подписано в печать
Тираж 30 .
Объем 2,3 уч.-изд. л.

Формат 60х84 1/16
Бумага типографская №1
Заказ _. Цена 1100 тг.

Копировально-множительное бюро
некоммерческого акционерного общества
«Алматинский университет энергетики и связи имени Гумарбека Даукеева»
050013, Алматы, Байтурсынова, 126/1