Федеральное государственное автономное образовательное учреждение

высшего образования

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа «Киберфизические системы и управление»

**Отчет по лабораторной работе**

по дисциплине «Системный подход к разработке программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 3530902/00201 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Позолотин О. В. |
|  | <*подпись*> |  |
|  |  |  |
|  |  |  |
| Руководитель:  Кандидат т.н. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Нестеров С. А. |
|  | <*подпись*> |  |

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 г.

Санкт-Петербург

2023

**Tasks from presentation**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**3-3 Collection**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text, application

Description automatically generated

Deque, Node, Map, HashMap, LinkedList, Queue

1. What is the difference between a Queue and a Stack? Give an example of each.

Queue – FIFO

Stack – LIFO

1. Is it possible to add nodes to the beginning of a LinkedList? If so, how? What about adding a node to the end of a LinkedList? If this can be done, what method would be used

Graphical user interface, text

Description automatically generated

1. What is the purpose of implementing the Comparable interface in one of our classes?

Чтобы переопределить метод compareTo(), позволяющий сравнивать 2 объекта класса, реализующего интерфейс Comparable.

**Стивенс Р. Алгоритмы**

**Глава 3. Связные списки**

Связные списки — возможно, самые простые структуры данных, которые вам придется создавать. Тем не менее некоторые методы их построения применимы для формирования более сложных структур, описанных в книге. Для использования связных списков вам необходимо иметь представление о ссылках и ячейках, включая способы их нахождения, вставки и удаления. Эти же понятия фигурируют при построении сложных сетей и деревьев, в том числе сбалансированных.

Связный список построен из объектов, обычно называемых ячейками. Этот класс содержит все данные, которые должны храниться в списке, и ссылку на другую ячейку. Ссылка представляет собой справку или указатель на объект такого же класса. Поле типа «указатель» в ячейке часто называется Next.

В однонаправленном связном списке (см. рис. 3.1) каждая ячейка связана со следующей с помощью одинарной ссылки. Чтобы использовать такой список, вам понадобятся алгоритмы для передвижения по списку, поиска, добавления и удаления элементов. Рассмотрим некоторые из них.

**Передвижение по спискам**

Данный алгоритм начинается с цикла While, который работает до тех пор, пока верхний указатель ячейки не станет null. Внутри цикла алгоритм сперва вызывает метод Print, чтобы показать значение ячейки top, а затем с ее помощью указывает на следующую ячейку связного списка. Процесс продолжается до тех пор, пока top не станет указывать на null в конце списка и цикл While не остановится.

**Нахождение элемента**

Поиск ячейки выглядит так: алгоритм передвигается по связному списку и останавливается, найдя нужную.

**Использование ограничителей**

Если внимательно изучить приведенный выше алгоритм, легко представить случай, при котором он даст сбой. Например, когда искомое значение содержится в первой ячейке связного списка, то перед ней не окажется ячейки и алгоритм не сможет ничего вернуть. Первое значение, которое он станет рассматривать, будет находиться во второй ячейке списка, а алгоритм никогда не возвращается назад. Один из способов справиться с описанной проблемой — добавить специальный код, который отыщет искомое значение непосредственно в первой ячейке. Но если программе придется рассматривать эту ситуацию в качестве особого случая, она может запутаться. Другой подход — создать ограничитель в начале списка. Он представляет собой ячейку, которая является частью связного списка, но не содержит какие-либо значимые данные. Ограничитель используется только в качестве метки-заполнителя, поэтому алгоритмы могут свободно обращаться к нему.

**Добавление ячеек в начало списка**

Алгоритм устанавливает новый указатель ячейки Next таким образом, чтобы он указывал вначале на ту ячейку, которая идет первой после ограничителя в списке, а затем на новую. Получается, что новая ячейка размещается после ограничителя и становится первой в связном списке.

**Diagram

Description automatically generated**

**Добавление ячеек в конец списка**

**Diagram

Description automatically generated**

**Вставка ячеек**

**Diagram

Description automatically generated**

**Удаление ячеек**

**Diagram

Description automatically generated**

**Двунаправленные связные списки**

В двунаправленных списках ссылки ячеек указывают на следующие и предыдущие ячейки. Вторые часто называют Prev, или Previous. В подобных структурах данных удобно иметь верхний и нижний ограничители (рис. 3.6), чтобы программа легко управляла списком с обоих концов, например добавляла и удаляла элементы. Такой подход позволяет это сделать за время, равное O(1).

**Chart, box and whisker chart

Description automatically generated**

Алгоритмы для работы с двунаправленными и однонаправленными списками очень похожи, за исключением того, что первые должны выполнять дополнительные действия для управления еще одним набором ссылок. Вот как выглядит псевдокод, добавляющий ячейку после выбранной.

**A screenshot of a computer

Description automatically generated with medium confidence**

**Копирование**

**Text

Description automatically generated**

**Сортировка вставкой**

**Text

Description automatically generated**

**Сортировка методом выбора**

**Text

Description automatically generated**

**Многопотоковые связные списки**

В однонаправленном связном списке каждая ячейка содержит ссылку на следующую ячейку, в двунаправленном — на следующую и предыдущую. Это необходимо для того, чтобы обеспечить два способа перемещения — прямой и обратный. Однако к ячейкам списка можно добавлять и другие ссылки, чтобы предусмотреть иные способы перемещения. Предположим, вы построили класс Planet для хранения информации о планетах Солнечной системы и назначили ему поле с именем NextDistance, в котором находятся сведения о расстояние до Солнца. С учетом этого список будет организован в следующем порядке: Меркурий, Венера, Земля, Марс, Юпитер, Сатурн, Уран, Нептун (и Плутон, если захотите включить и его). Аналогичным образом можно добавить и другие поля, чтобы отсортировать планеты по их массе, диаметру и т. д. Каждое такое прохождение через ячейки, определенные набором ссылок, называется потоком. С одним потоком работать легко, если представить его как простой связный список, но визуализировать все потоки одновременно довольно сложно. На рисунке 3.8 показан связный список планет с тремя потоками. Ссылки, обозначенные тонкой линией, указывают на сортировку по удаленности от Солнца, пунктирной — по массе, а толстой — по диаметру

**Diagram, schematic

Description automatically generated**

**Связные списки с циклами**

Кольцевой связный список — это связный список, в котором последняя ссылка указывает на первый элемент списка (рис. 3.9). Он может быть полезен в том случае, когда в рамках цикла нужно бесконечно проходить через последовательность элементов. Так, операционная система способна повторять цикл процессов, чтобы запустить каждый из них. Если начался новый процесс, он может быть добавлен в любое место списка, например за ограничителем, что позволит ему сразу запуститься.

**Engineering drawing

Description automatically generated with medium confidence**

На рисунке 3.10 показан кольцевой связный список, в который включены не все ячейки. В этом случае возникают два интересных вопроса. Во-первых, как определить, содержит ли связный список такой цикл? Во-вторых, если связный список содержит подобный цикл, как узнать, где он начинается и как его прервать? Последний вопрос, по сути, связан с определением конца списка.

**Diagram

Description automatically generated**

**Маркировка ячеек**

Возможно, самый легкий способ определить, имеет ли связный список цикл, — пройти через его ячейки и пометить каждую. Если очередная рассматриваемая ячейка окажется помеченной, значит, в списке есть цикл и он начинается в этом месте. Приведенный ниже алгоритм представляет следующий псевдокод

**Text

Description automatically generated**

**Использование хэш-таблиц**

Более подробно хеш-таблицы описываются в главе 8. Все, что вам необходимо знать о них сейчас, — такие таблицы помогают очень быстро сохранять и находить элементы, а также выявлять их наличие в принципе. Представленный ниже алгоритм перемещается по списку и добавляет каждую ячейку в хеш-таблицу, предварительно проверяя, нет ли ее уже там. Если он доходит до ячейки, которая присутствует в хеш-таблице, значит, с нее в списке начинается цикл.

**Text, letter

Description automatically generated**

**Повторная трассировка списка**

Представленный ниже алгоритм проводит по списку один объект, а затем в его поисках через каждую посещенную ячейку проходит второй объект. Это слегка запутанное объяснение легче понять, если взглянуть на следующий псевдокод.

**Text

Description automatically generated**

Работу этого алгоритма демонстрирует программа BreakLoopHashtable, которую можно скачать на сайте с материалами книги. Предположим, что в списке N ячеек. Когда объект алгоритма cell проверяет K-ю ячейку, объект tracer должен пройти вверх по списку до этой точки. Таким образом, он выполнит K шагов. Это значит, что общее время работы алгоритма составит 1 + 2 + 3 + ... + N = N (N – 1)/2 = O(N2 ). Оно меньше, чем у предыдущих алгоритмов, но зато не требуется дополнительной памяти.

Следующий алгоритм также не нуждается в дополнительных ресурсах, но работает в течение времени O(N).

**Реверсирование списка**

Алгоритм проходит по ячейкам списка, меняя каждую ссылку таким образом, чтобы она указывала на предыдущую ячейку, а не на последующую. Если алгоритм обнаруживает значение null до ограничителя, цикла в списке нет. Конечно, описанное действие преобразует ссылки, поэтому алгоритм опять проходит по списку и восстанавливает их, чтобы они снова указывали на изначальные ячейки. Прояснит подобный принцип рисунок 3.11. Верхний фрагмент — это оригинальный список, а закрашенная ячейка — та, через которую проходит алгоритм в настоящее время, реверсируя ссылки. На среднем фрагменте видно, что алгоритм достиг ячейки I (измененные ссылки показаны толстыми стрелками) и должен проследовать в ячейку D, а из нее по реверсированным ссылкам в ячейки C, B и A. В результате совершенных действий ссылки изменятся снова, что отражено на нижнем фрагменте пунктиром. В этом месте алгоритм возвращается к первой ячейке и определяет, что список содержит цикл. Обратите внимание: новый список выглядит так же, как и старый, только ссылки в цикле имеют обратное направление.

**Diagram

Description automatically generated**

**Черепаха и кролик**

Алгоритм кролика и черепахи, который также называется алгоритмом нахождения циклов Флойда, придуман Робертом Флойдом в 1960-х гг. Лежащий в его основе принцип не слишком сложный, но объяснить его непросто, поэтому если вы не любите математические формулы, то нижеприведенное описание можно пропустить. Алгоритм запускает два объекта: «черепаху» и «кролика». Они двигаются из начала списка, но с разными скоростями: «черепаха» преодолевает за один шаг одну ячейку, а «кролик» — две. Когда последний достигает ссылки со значением null — список окончен и цикла в нем нет. Если же цикл есть, «кролик» входит в него и начинает бегать по кругу. Тем временем «черепаха» ползет, пока не дойдет до цикла и в какой-то момент не окажется в нем вместе с «кроликом». Пусть L — количество ячеек внутри цикла, T — количество шагов, пройденных «черепахой», чтобы попасть в цикл, а H — расстояние от начала цикла до того положения, в котором за это же количество шагов оказался «кролик». Тогда, согласно рисунку 3.12, L = 5, T = 4, H = 4.

**Diagram

Description automatically generated**

Поскольку «кролик» перемещается в два раза быстрее «черепахи», он доходит до цикла через T ячеек и преодолевает еще T ячеек внутри цикла, чтобы оказаться в позиции, как на рисунке 3.12. Это приводит к факту № 1.

Обратите внимание, что «кролик» мог пробежать несколько кругов по циклу, если L намного меньше T. Например, если L = 5, а T = 102, черепаха дойдет до цикла за 102 шага. «Кролик» же добирается до цикла за 51 шаг, совершает 50 шагов (преодолевает 100 ячеек), пробегая 20 кругов по циклу, а затем перемещается еще на один шаг (2 ячейки) внутри цикла. В таком случае H = 2. Возникает вопрос: когда же «кролик» догонит «черепаху»? Как только «черепаха» попадет в цикл, «кролик» окажется на H шагов впереди нее (см. рис. 3.12). Однако поскольку «черепаха» и «кролик» пребывают в цикле, вы можете рассматривать последнего как отстающего на L – H ячеек. Из-за того, что «кролик» перемещается на две ячейки, а «черепаха» на одну, он нагоняет по одной ячейке за шаг. Это означает, что «кролик» доберется до «черепахи» за L – H шагов. На рисунке 3.12, где H = 4, а L = 5, «кролик» догонит «черепаху» за 5 – 4 = = 1 шаг. Они встретятся в ячейке E. В этот момент «черепаха» передвинется в цикле на L – H ячеек, и оба объекта будут на L – (L – H) = H ячеек ближе к началу цикла. Отсюда факт № 2.

Если бы вы могли переместить объект «черепаха» на H ячеек от точки встречи, он оказался бы как раз в начале цикла. К сожалению, это значение вам неизвестно. Тем не менее из факта № 1 вы знаете, что если черепаха проходит через T ячеек по циклу, она закончит свой путь на расстоянии H ячеек до того места, откуда стартовала, то есть остановится в начале цикла! Значение T тоже для вас тайна, поэтому переместить «черепаху» на указанное расстояние вы опять не можете. Однако если запустить «кролика» от начала связного списка и позволить ему проходить за раз только одну ячейку вместо двух (возможно, он устал, бегая по циклу), он также окажется в начале цикла после того, как пересечет T ячеек, то есть оба объекта встретятся снова. Следующий псевдокод демонстрирует алгоритм высокого уровня. 1. Запускаем «черепаху» из начала списка со скоростью одна ячейка за шаг и «кролика» со скоростью две ячейки за шаг. 2. Если «кролик» найдет ссылку null, список не содержит цикла. 3. Если «кролик» догонит «черепаху», перезапускаем его из начала списка со скоростью одна ячейка за шаг, в то время как «черепаха» продолжает двигаться с в прежнем темпе. 4. Когда «кролик» и «черепаха» снова встретятся, они будут находиться в начале цикла. Оставляем «кролика» в этом месте, чтобы он мог «отдохнуть», пока «черепаха» движется по циклу. Момент, когда указатель Next «черепахи» покажет на ячейку, где ждет «кролик», и будет означать конец цикла. 5. Чтобы прервать цикл, устанавливаем указатель «черепахи» Next на null.

**Циклы в двунаправленных связных списках**

Обнаружить циклы в двунаправленных списках очень просто: если они есть, указатель Next возвращается к ранней части списка. При этом Prev будет указывать на уже существующую ячейку, а не на ту, которую создал цикл. Таким образом, чтобы выявить цикл, просто пройдите через список и проверьте cell для каждой ячейки: Next.Prev == cell. Все это предполагает, что ячейки образуют двунаправленный связный список и цикл, если он существует, является обычным. Если списки Next и Prev полностью несогласованы, метод позволяет обнаружить путаницу, но не помогает исправить ее. Это больше похоже на случай с двумя потоками через одни и те же ячейки, чем на двунаправленный список с циклом.