Лабораторная работа №1 Построение идеальной хеш-таблицы (без коллизий).

Целью данной работы является сравнение следующих подходов:
1) Квадратичный подход к построению идеальной хеш-таблицы; (см. Кормен, 3-е издание, 11.5 Идеальное хеширование, или видео запись)
2) Двух-уровневый подход к построению идеальной хеш-таблицы; (см. Кормен, 3-е издание, 11.5 Идеальное хеширование, или видео запись)
3) (опционально) Графовый подход к построению идеальной хеш-таблицы
(см. https://habrahabr.ru/post/254431/, вот исходная статья, обязательно посмотрите http://cmph.sourceforge.net/papers/chm92.pdf).

Что измеряется?
1) Время построения хеш-таблицы;
2) Время операции поиска элемента;
3) Затрачиваемая память

Замечание: В качестве хеш-функций используйте только функции из универсальных семейств, про них рассказывалось на семинарах. Список универсальных хеш-функций можно найти здесь https://en.wikipedia.org/wiki/Universal_hashing.

Замечание: Для двух-уровневой хеш-таблицы нужно обязательно провести эксперименты с разлиными значениями параметров, разобраться как различные значения параметров влияют на производительность и найти оптимальный выбор параметров. Без этих экспериментов работа не будет зачтена. Про какие именно параметры идет речь смотрите в записи лекции.

Входные данные:
а) Случайные натуральные числа.
б) Случайные вектора или строки.
с) Очень бы хотелось увидеть как поведут себя таблицы на real life данных, например на словарях или словах какого нибудь литературного произведения.

Ваш вывод должен содержать:
1) Графики сравнения скорости построения и количества занимаемой памяти для подхода 1) при различных значениях параметров. Выбор оптимального, с Вашей точки зрения, набора параметров.
2) График сравнения скорости построения хеш-таблиц для подходов 1) 2) 3), причем для подхода 2) нужно использовать оптимальный набор параметров.
3) График сравнения скорости поиска для обоих подходов.
4) Все графики нужно продублировать, если Вы используете разные входные данные: случайные строки и real life данные.

Итого, должно быть как минимум 4 картинки с несколькими кривыми на каждой.

Лабораторная работа № 2. Сравнение различных подходов к хешированию.

Целью данной работы является сравнение различных методов борьбы с коллизиями:
1) Метод цепочек;
2) Метод открытой адресации (см. Кормен, 3-е издание, 11.4 Открытая адреация);
3) Метод кукушки (см. https://en.wikipedia.org/wiki/Cuckoo_hashing).
4) Для интереса, предлагается также сделать сравнения со стандартными средствами языка c++: std::map, std::hash_map. Вдруг получится их обогнать.

Замечание: В качестве хеш-функций используйте только функции из универсальных семейств, про них рассказывалось на семинарах. Список универсальных хеш-функций можно найти здесь https://en.wikipedia.org/wiki/Universal_hashing.

Что измеряется?
1) Время вставки;
2) Время удаления;
3) Время поиска.

Более конкретно об измерении:
Нужно выбрать какое-то стартовое значение N, скажем 100, выбрать шаг step, скажем пусть step = 100, и выбрать максимальное значение, скажем 100 000. После чего нужно для каждого N с шагом step от минимального значения до максимального построить таблицу размера N (из случайных элементов, или сделать выборку из заранее подготовленной базы) и произвести **одну или несколько операций** (если несколько, скажем 10, то нужно усреднить). Измеряем именно время **одной операции**. Некоторые допускают ошибку и делают N вставок с замером времени, но не понятно, что в итоге Вы измерили.

Входные данные:
а) Случайные натуральные числа.
б) Случайные вектора или строки.
с) Очень бы хотелось увидеть как поведут себя таблицы на real life данных, например на словарях или словах какого нибудь литературного произведения.

Ваш вывод должен содержать:
1) График зависимости скорости вставки от количества элементов в таблице;
2) График зависимости скорости удаления от количества элементов в таблице;
3) График зависимости скорости поиска от количества элементов в таблице;

На каждом графике должно быть несколько кривых, по одной или больше для каждого подхода. Заметим также, что таблицы из подходов 1) и 2) имеют дополнительный параметр m – ёмкость таблицы. Хорошо бы построить на графиках кривые для разных значений m, например m = 2n, m = n, m = ½ n. Но это не обязательно.

Лабораторная работа №3. Сравнение поисковых деревьев.

Целью данной работы является сравнение следующих структур данных:
1) АВЛ — дерево;
2) Декартово дерево (по-другому: дуча, treap);
3) splay-дерево (см. например https://habrahabr.ru/company/spbau/blog/210296/ или https://en.wikipedia.org/wiki/Splay_tree).
4) Красно-черное дерево (не нужно реализовывать, используется в std::map или std::set);
5) Бинарный поиск в отсортированном массиве (сортировку реализовывать не нужно, используйте std::sort, бинарный поиск — по желанию, можете использовать std::find. Если мне не изменяет память, то именно эта функция).

Что измеряется?
1) Время вставки;
2) Время удаления;
3) Время поиска.

Входные данные:

а) Случайные натуральные числа.
б) Случайные вектора или строки.
с) Очень бы хотелось увидеть как поведут себя таблицы на real life данных, например на словарях или словах какого нибудь литературного произведения.

Ваш вывод должен содержать:
1) График зависимости скорости вставки от количества элементов в дереве;
2) График зависимости скорости удаления от количества элементов в дереве;
3) График зависимости скорости поиска от количества элементов в дереве.

На каждом графике должно быть несколько кривых, по одной или больше для каждой структуры.

Лабораторная работа №4. Сравнение приоритетных очередей.

Целью данной лабораторной работы является сравнение следующих структур данных:
1) Левосторонняя куча;
2) Косая куча;
3) Биномиальная куча;
4) d-куча.

Что сравниваем?
1) Время вставки;
2) Время удаления;
3) Время слияния;
4) Время удаления минимума/максимума;
5) Количество занимаемой памяти.

Замечание: для некоторых приоритетных очередей некоторые из перечисленных операций не реализуются эффективно. Например: для d-кучи нет эффективного слияния, а для левосторонней куче нет эффективного удаления. Для этих случаев измерения проводить не нужно.

Данные:
а) Случайные натуральные числа;
б) Случайные вектора или строки.

Ваш вывод должен содержать:
1) График зависимости скорости вставки от количества элементов в очереди;
2) График зависимости скорости удаления от количества элементов в очереди;
3) График зависимости скорости слияния от количества элементов в очереди.

На каждом графике должно быть несколько кривых, по одной или больше для каждой структуры. Заметим, что у d-кучи есть параметр d. Проведите несколько экспериментов для разных значений d, например d = 2, 3, 10, 100. Найдите экспериментально оптимальное значение параметра d.

Лабораторная работа № 5. Сортировки.

Целью данной работы является сравнение следующих алгоритмов сортировки:
0) Сортировка подсчетом (counting sort, есть в книге Кормена)
1) Поразрядная сортировка (radix sort, есть в книге Кормена);

2) Карманная сортировка (bucket sort, есть в книге Кормена);
3) Быстрая сортировка (quick sort, есть в книге Кормена);
4) Сортировка слиянием (merge sort, есть в книге Кормена);
5) Сортировка кучей (heap sort на основе d-кучи, есть в книге Кормена);
6) Сортировка Шелла (https://habr.com/ru/post/133996/ , смотрите также англ. Вики)
7) std::sort

Замечание 1:
Поразрядную сортировку следует так, как это было рассказано на лекции. Это даст наибольшую скорость. А именно, если вы сортируете 32-разрядные целые числа, то их стоит разбивать либо на два числа по 16 бит, либо на 4 числа по 8 бит, а потом делаем counting sort. Не возможно предсказать какой вариант на разных системах будет быстрее. Нужны эксперименты. С числами 64 бит действуем аналогично.

Замечание 2:
Рекурсивный вариант быстрой сортировки нужно реализовать правильно, то есть должна осуществляться элиминация хвостовой рекурсии. В противном случае сортировка будет очень редко, но падать.

Замечание 3:
Рекурсивный вариант merge sort должен выделять доп. память только!!! один раз, а не в каждом узле рекурсии!

Замечание 4:
Сортировку d-кучей следует реализовать in-place, тоесть так, чтобы ей не требовалась дополнительная память. Иными словами все перестановки, которые делает эта сортировка можно делать внутри входного массива без выделения дополнительной памяти для формирования выходного массива.

Что измеряем?
1) Время сортировки;
2) Количество занимаемой дополнительной памяти (память для входного массива не учитывается).

Данные:
а) Случаные, 32-битные, целые числа (для поразрядной сортировки);
б) Случаные, 64-битные, целые числа (для поразрядной сортировки);
в) Случайные double числа из интервала [0,1] (или другого фиксированного интервала);
г) Случайные float числа из интервала [0,1] (или другого фиксированного интервала);
д) Случайные строки (сравниваем по лексикографии);
е) Очень бы хотелось увидеть как поведут себя сортировки на real life данных, например на словарях или словах какого нибудь литературного произведения.

Ваш вывод должен содержать:
1) Графики зависимости скорости сортировок от размера массива для разных видов входных данных. По одному графику для каждого вида входных данных;
2) Поэкспериментируйте с radix sort и bucket sort (в зависимости от варианта). Для radix sort попробуйте разные основания системы счисления (16 бит, 8 бит и т.д.). Для bucket sort попробуйте разные варианты простых сортировок для сортировки карманов (сортировка пузырьком, сортировка вставками, попробуйте встроенную сортировку std::sort). Не забудьте построить графики!!!

На каждом графике должно быть несколько кривых, по одной или больше для каждого алгоритма.

Лабораторная работа № 6. Алгоритмы поиска кратчайших путей в графах.

Замечание: в данной работе рассматриваются алгоритмы поиска кратчайших путей ои заданной вершины до всех остальных (single source multiple sink shortest paths algorithms).

Целью данной работы являются следующие алгоритмы поиска кратчайших путей в графах:
1) Алгоритм Беллмана-Форда;
2) Алгоритм Дейкстры на d-куче;
3) Алгоритм Дейкстры на массиве;
4) Алгоритм Дейкстры на биномиальной куче.

Что измеряем?
Время поиска кратчайших путей от заданной вершины до всех остальных.

Данные:
а) Плотные случайные графы: например $m = n^2/4$, $m = n^2/3$, $m = n^2/\log(n)$ и т.д. (достаточно трех вариантов)
б) Разреженные случайные графы: например $m=2n$, $m = 5n$, $m = \log(n)\, n$, $m = \log^2(n)\, n$ и т. д. (достаточно трех вариантов)
с) Очень бы хотелось увидеть как поведут себя алгоритмы на real life графах, например на графах дорог или графах интернет сетей (графы сайтов, соц-сети, граф ссылок википедии).

Ваш вывод должен содержать:
1) Графики зависимости скорости поиска от количества вершин. По одному графику для каждого вида входных данных и плотности по количеству ребер. То есть как минимум 6 графиков и на каждом графике несколько кривых, по одной для каждого подхода;
2) Поэкспериментируйте с d-кучей. Выберите несколько вариантов для параметра d, например d = 3, 10, 100 и т. д., экспериментально найдите лучший вариант и именно его сравнивайте с отсальными алгоритмами.

На каждом графике должно быть несколько кривых, по одной или больше для каждого алгоритма.

Лабораторная работа № 7. Поиск компонент связности в графах.

Целью данной работы является сравнение различных алгоритмов поиска компонент связности в графах:
1) Поиск в ширину;
2) Поиск в глубину;
3) Использование структуры данных разделенное множество:
    3.1) Разделенное множество на массиве;
    3.2) Разделенное множество на на древовидной структуре с рангами вершин;
    3.3) Предыдущий пункт + эвристика сжатия путей.

Замечания: данная лабораторная работа состоит из экспериментов двух типов: а) on-line б) off-line. В off-line эксперименте Вы должны проверить скорость работы всех трех алгоритмов на заранее заданных больших графах; В on-line эксперименте Вы должны

проверить скорость пересчета компонент связности после добавления нового ребра. В случае а) должны выигрывать подходы 1) и 2), в случае же б), должен выигрывать подход 3).

Что измеряем?
1) Зависимость поиска всех компонент связности от количества вершин в графе (количество ребер задается значением плотности ребер графа, смотрите раздел **данные**);
2) Зависимость времени пересчета компонент связности после добавления нового ребра от количества вершин в графе. Чтобы проделать данные измерения, создайте граф без ребер из n вершин, добавляйте новые ребра пока их не станет m. Найдите среднее время добавления ребра и время в худшем случае.

Данные:
а) Плотные случайные графы: например $m = n^2/4$, $m = n^2/3$, $m = n^2/\log(n)$ и т.д. (достаточно двух вариантов)
б) Разреженные случайные графы: например $m=2n$, $m = 5n$, $m = \log(n)\, n$, $m = \log^2(n)\, n$ и т. д. (достаточно двух вариантов)
с) Очень бы хотелось увидеть как поведут себя алгоритмы на real life графах, например на графах дорог или графах интернет сетей (графы сайтов, соц-сети, граф ссылок википедии).

Ваш вывод должен содержать:
1) Графики зависимости времени поиска компонент связности от количества вершин в графе. По одному графику для каждого значения плотности ребер.
2) Графики зависимости худшего времени пересчета компонент связности после вставки новых ребер от количества вершин в графе. По одному графику для каждого значения плотности ребер.
3) Графики зависимости среднего времени пересчета компонент связности после вставки новых ребер от количества вершин в графе. По одному графику для каждого значения плотности ребер.

На каждом графике должно быть несколько кривых, по одной или больше для каждого алгоритма.

Замечание: если не понятно, что нужно делать, пожалуйста обратитесь к преподавателю. Можете позвонить или написать письмо, узнайте контакты на занятии или обратитесь к старосте.

Лабораторная работа № 8. Порядковые статистики.

Целью данной работы является сравнение разных алгоритмов поиска k-ой порядковой статистики:
1) Рандомизированный линейный алгоритм (см. Кормен 3-е издание, порядковые статистики);
2) Детерминированный линейный алгоритм;
3) Наивный алгоритм со сложностью O(n log n), использующий сортировку.

Что измеряем?
Зависимость времени поиска k-ой порядковой статистики от размера массива.

Данные:
1) Случайные натуральные числа;
2) Случайные вектора или строки.

3) Очень бы хотелось увидеть как поведут себя алгоритмы на real life данных, например на словарях или словах какого нибудь литературного произведения.

Ваш вывод должен содержать:
Графики зависимости времени поиска k-ой порядковой статистики от размера массива. Нужно построить несколько графиков для разных значений k = 0,n-1,n/2, 3n/4, 2/3n и т. д. На каждом графике должно быть несколько кривых, по одной или больше для каждого алгоритма.

Лабораторная работа № 9. Поиск остова графа максимального веса.

Целью данной работы является сравнение разных алгоритмов поиска остовного подграфа максимального веса для некоторого взвешенного связного графа:
1) Алгоритм Краскала (алгоритм использует сортировку, реализовывать ее не нужно, используйте std::sort);
2) Алгоритм Борувки;
3) Алгоритм Прима.
Алгоритмы 1) и 2) используют структуру данных разделенное множество. Реализуйте несколько вариантов данной структуры для проведения экспериментов по каждой:
1) разделенное множество на массивах;
2) разделенное множество на древовидной структуре с использованием рангов узлов;
3) предыдущее + эвристика сжатия путей.

Что измеряем?
Зависимость времени поиска максимального остовного подграфа в некотором связном взвешенном графе от количества вершин. Количество ребер в графе задается значением плотности ребер, смотрите раздел **данные**.

Данные:
а) Плотные случайные графы: например $m = n^2/4$, $m = n^2/3$, $m = n^2/\log(n)$ и т.д. (достаточно двух вариантов)
б) Разреженные случайные графы: например $m=2n$, $m = 5n$, $m = \log(n)\, n$, $m = \log^2(n)\, n$ и т. д. (достаточно двух вариантов)

Ваш вывод должен содержать:
Графики зависимости времени поиска максимального остовного подграфа от количества вершин в графе. По одному графику для каждого значения плотности ребер.

На каждом графике должно быть несколько кривых, по одной или больше для каждого алгорима и для каждого варианта реализации структуры данных разделенное мносжество.

Лабораторная работа № 11. Алгоритмы поиска подстроки(образца) в строке.

Целью данной работы является сравнение следующих алгоритмов решения задачи поиска родстроки в строке:
1) Алгоритм Кнута-Морриса-Пратта (см. Кормен, 3-е издание);
2) Алгоритм Рабина-Карпа;
3) Наивный алгоритм.

Что измеряем?

Зависимость времени поиска образца в строке от суммы длин строки и образца.

Данные:
а) В качестве образца и строки выбираются случайные строки;
б) В качестве образца и строки выбираются подстроки некоторого текста естественного языка, например некоторого литературного произведения.

Ваш вывод должен содержать:
Графики зависимости времени поиска образца в строке от суммы длин образца и строки. По одному графику для каждого вида входных данных. Длину образца можете фиксировать (но она не должна быть слишком маленькой) или выбрать равной некоторой части строки, например 1/5 или 1/4 длины строки.

На каждом графике должно быть несколько кривых, по одной или больше для каждого алгоритма.