Лабораторная работа № 1 по курсу дискретного анализа: сортировка за линейное время

Выполнил студент группы М8О-207Б-21 Меркулов Фёдор.

Условие

Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.
Вариант задания определяется типом ключа (и соответствующим ему методом сортировки) и типом значения: Сортировка подсчётом.

Тип ключа: почтовые индексы.

Тип значения: строки переменной длины (до 2048 символов).

Метод решения

Вместо того, чтобы использовать префиксные суммы и потом с конца в начало вставлять строки в ответ, я решил использовать структуру в которой хранился бы целый список строк и после создания массива таких структур из 1000000 элементов у меня стало иметься хранилище, в котором индекс (от 0 до 999999) отвечает за ключ и под этим индексом находится структура, хранящая список строк. После ввода всех строк в соответствующие списки, остаётся пробежать по массиву структур и вывести все имеющиеся строки, тем самым получив отсортированное множество строк по ключу.

Описание программы

Использовался единственный файл, который и решал поставленную задачу. Данную задачу легко можно было бы решить с помощью использования таких типов данных, как vector, string, но так как STL и прочие библиотеки нельзя было использовать, типом данным, играющим основополагающую роль стал own_vector, который является подобием вектора со списком массивов char (аналог строк). Внешние функции не использовались.

Дневник отладки

Когда в первый раз отослал код, поймал TL на последнем тесте и пытался оптимизировать код, первой идеей было убрать и сократить всё что можно, собственно, что я и сделал, вынеся некоторые части кода из блоков ветвлений и оставив их в единственном экземпляре. Второй идеей стало нахождение и использование минимальных и максимальных ключей для оптимизации в среднем (чтобы не рассматривать весь 1000000 ключей, а только диапазон от меньшего к большему) и самой финальной оптимизацией стало определение буфера из 6 символов не в цикле с 1000000 повторений, а вне его, тем самым я сократил использование памяти на 30% и ускорил работу программы. После чего код прошёл все тесты по памяти и по времени.

Тест производительности

Я создал Benchmark “DA1time.cpp”, который измерял время работы сортировки в миллисекундах, на некоторых входных файлах. Входные данные: “1.txt”, “2.txt”, “3.txt”, “4.txt”, “5.txt”, “6.txt”.

Данные файлы представляют собой случайно сгенерированные данные типа “почтовый индекс – строка переменной длины (до 2048 символов)”, цифра N в названии которых говорит о количестве строк во входном файле, соответственно равном 10^N.

papik@papik-VirtualBox:~$ g++ DA1time.cpp

papik@papik-VirtualBox:~$ ./a.out < 1.txt

Time: 116 ms

papik@papik-VirtualBox:~$ ./a.out < 2.txt

Time: 260 ms

papik@papik-VirtualBox:~$ ./a.out < 3.txt

Time: 881 ms

papik@papik-VirtualBox:~$ ./a.out < 4.txt

Time: 8574 ms

papik@papik-VirtualBox:~$ ./a.out < 5.txt

Time: 66545 ms

papik@papik-VirtualBox:~$ ./a.out < 6.txt

Time: 649684 ms

Согласно данным измерениям, можно сказать, что с ростом количества входных данных время работы согласуется с заявленной сложностью, то есть растёт линейно.

Недочёты

Скорее всего мой код написан не самым оптимальным образом, так как он зашёл впритык, 22 миллисекунды отделяли его от TL. Скорее всего из-за того, что я использую массив структур с размером в 1000000, структуры с индексами, неиспользуемых ключей понапрасну тратят память и увеличивают время обработки данных.

Выводы

Областью применения данного алгоритма являются задачи, в которых количество запросов значительно превышает диапазон сортируемых ключевых значений. Данный алгоритм работает за O(n). Этот алгоритм крайне удобен, когда известен диапазон сортируемых значений, что в то же время является его недостатком, так как в задачах, где ничего неизвестно об исходном состоянии сортируемых данных этому алгоритму потребуется дополнительное время для нахождения минимального и максимального значений сортируемых данных.