МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Основы информатики»
II семестр

Задание 9
«Сортировка и поиск»

Группа: М80 – 107Б-18

Студент: Цапков Александр Максимович

Преподаватель: Ридли Александра Николаевна

Оценка: _____

Дата:_____

Москва, 2019

Содержание

Введение

В восьмом задание курсового проекта мне требуется составить программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице. В моем варианте нужно было реализовать пирамидальную сортировку просеиванием таблицы с 4х битным вещественным ключом.

2. Общее описание программ

Мне нужно было написать программу, которая сортирует таблицу и выполняет по ней поиск, но для удобства я написал еще одну вспомогательную программу. Эта программа генерирует нам таблицу размером заданным параметром при вызове через командную строку. В этой таблице совершенно случайным образом выбраны ключи. Вторая программа выполняет саму сортировку с помощью подпрограммы коротая принимает на вход 2 массива: 1 с ключами, а второй со значениями, соответствующие этим ключам. В саму же программу ввод выполняется с помощью таблицы, в начале которой указывается ее размер. После сортировки выполняется печать всей таблицы и ожидается ввод ключа, по которому будет осуществлен поиск. Поскольку мои ключи были вещественными, то я подумал, что гораздо удобнее будет, если поиск будет искать не просто идеальное совпадение ключа, а самое близкое совпадение ключа и соответствующее ему значение.

3. Подробное описание пирамидальной сортировки

Метод пирамидальной сортировки, изобретенный Д. Уилльямсом, является улучшением традиционных сортировок с помощью дерева. Общая идея пирамидальной сортировки заключается в том, что сначала строится пирамида из элементов исходного массива, а затем осуществляется сортировка элементов

Выполнение алгоритма разбивается на два этапа:

1 этап Построение пирамиды. Определяем правую часть дерева, начиная с **n/2-1** (нижний уровень дерева). Берем элемент левее этой части массива и просеиваем его сквозь пирамиду по пути, где находятся меньшие его элементы, которые одновременно поднимаются вверх; из двух возможных путей выбираете путь через меньший элемент.

2 этап Сортировка на построенной пирамиде. Берем последний элемент массива в качестве текущего. Меняем верхний (наименьший) элемент массива и текущий местами. Текущий элемент (он теперь верхний) просеиваем сквозь n-1 элементную пирамиду. Затем берем предпоследний элемент и т.д.

4. Анализ данных времени сортировки

Несмотря на некоторую внешнюю сложность, пирамидальная сортировка является одной из самых эффективных. Алгоритм сортировки эффективен для больших **n**. В худшем случае требуется **n·log2n** шагов, сдвигающих элементы. Среднее число перемещений примерно равно

(n/2)·log2n,

и отклонения от этого значения относительно невелики.

В моем случае сортировка полностью несортированной таблицы размером 10000 занимает в среднем 59 мс. (59 59 59 58), размером в 20000 занимает в среднем 124 мс. (124 124 125 124), размером в 30000 элементов в среднем 194 мс. (194 194 195 193). Проверим формулу по полученным результатам:

$$\left|\frac{10000\times\ln(2\times10000)}{20000\times\ln(2\times20000)}-\frac{59}{124}\right|\times100\approx0.851$$

Погрешность получилась примерно в 0.9%, что в принципе подтверждает правильность формулы. Сортировка полностью сортированного списка занимает чуть больше времени чем полностью несортированного.

Заключение

Пирамидальная сортировка является одним из самых эффективных способов сортировки особенно на большом количестве вхождений, однако она не является эффективной, если сортируемая таблица уже отсортирована или в ней есть много отсортированных элементов, так как занимает примерно то де количество итераций что и у несортированного.

key_sort.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>


int is_num(char c)
{
  return (c >= '0' && c <= '9') || c == '.' ? 1 : 0;
}
int is_space(char c)
{
  return c == '\n' || c == '\t' || c == ' ' ? 1 : 0;
}

void siftDown(float *numbers, size_t *values, int root, int bottom)
{
  int maxChild;
  int done = 0;
  while ((root * 2 <= bottom) && (!done))
  {
    if (root * 2 == bottom)
      maxChild = root * 2;
    else if (numbers[root * 2] > numbers[root * 2 + 1])
      maxChild = root * 2;
    else
      maxChild = root * 2 + 1;
    if (numbers[root] < numbers[maxChild])
    {
      float temp = numbers[root];
      size_t tempV = values[root];
      numbers[root] = numbers[maxChild];
      values[root] = values[maxChild];
      numbers[maxChild] = temp;
      values[maxChild] = tempV;
      root = maxChild;
    }
    else
      done = 1;
  }
}
void heapSort(float *numbers, size_t *values, int array_size)
{
  for (int i = (array_size / 2) - 1; i >= 0; i--)
    siftDown(numbers, values, i, array_size - 1);
  for (int i = array_size - 1; i >= 1; i--)
  {
    float temp = numbers[0];
    size_t tempV = values[0];
    numbers[0] = numbers[i];
    values[0] = values[i];
    numbers[i] = temp;
    values[i] = tempV;
    siftDown(numbers, values, 0, i - 1);
  }
}
```

```
int main(void)
{
    int N;
    scanf("%d", &N);
    float keys[N];
    size_t vals[N];

    for(int i = 0; i < N; i++) {
        vals[i] = (size_t)malloc(sizeof(char) * 100);
        scanf("%f %s",&(keys[i]), (char*)vals[i]);
    }
    puts("Unsorted:");
    for(int i = 0; i < N; i++) {
        printf("%f %s\n",keys[i], (char*)vals[i]);
    }
    struct timeval stop, start;
          gettimeofday(&start, NULL);
    heapSort(keys, vals, N);
    gettimeofday(&stop, NULL);
    long timeDiff = (long)((stop.tv_sec - start.tv_sec) * 1000.0f + (stop.tv_usec - start.tv_usec) /
1000.0f);
    printf("%ld\n", clock());
    puts("");
    puts("Sorted:");
    for(int i = 0; i < N; i++) {
        printf("%f %s\n",keys[i], (char*)vals[i]);
    }
    printf("%ld\n", timeDiff);
    float key;
    char c;
    int i = 0;
    char tmp_str[20];
    while((c = getchar()) != EOF) {
        if(is_num(c)) {
          tmp_str[i] = c;
          i++;
        } else if(is_space(c)) {
          tmp_str[i] = '\0';
          i = 0;
          key = atof(tmp_str);
          puts("Enter:");
          int a = 0;
          int b = N - 1;
          while(abs(a - b) > 1) {
              if (key > keys[(a + b) / 2]) {
                  a = (a + b) / 2;
              } else if (key < keys[(a + b) / 2]) {
                  b = (a + b) / 2;
              } else {
                  a = (a + b) / 2;
                  b = a;
              }
          }
          if (a != b) {
              if (fabsf(key - keys[a]) < fabsf(key - keys[b])) {
                  b = a;
              } else {
```

```
                a = b;
            }
        }
        printf("%f %s\n",keys[a], (char*)vals[a]);
      }
   }
   return 0;
}
```

key_gen.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

float d_gen(void)
{
   float out = (float)(rand() % 100);
   float len = (float)0.1;
   for(int i = 0; i < 5; i++) {
      out += (float)(rand() % 10) * len;
      len /= 10;
   }
   return out;
}

int main(int argc, char* argv[])
{
   srand(time(NULL));
   if(argc != 2) {
      return 1;
   }
   printf("%s\n", argv[1]);
   for(int i = 0; i < atoi(argv[1]); i++) {
      printf("%f %d%s\n", d_gen(),i ,"hello");
   }
   return 0;
}
```