Fundação Getulio Vargas
Estruturas de Dados e Algoritmos
Prof. Jorge Poco
Aluno: Davi Sales Barreira

Homework 2
Due:: August 13, 2019

Homework 2

1 Red-Black Trees

I am attaching a binary tree source code (`bst-0.0.cpp`) with the methods insert, delete and print. Your job would be to implement a Red-Black Tree with the functions insert, remove and print.
To test your code you can follow the examples described in the document `anexo1.pdf`. In addition, you might be interested in the document `anexo2.pdf` for a more detailed description of this tree, there is also some Java code that might be useful.
Note, your code must be implemented in C++ and based in the BST class I'm providing you. Grading would be as follow:

(a) **(2.5pts)** insert

(b) **(2.5pts)** remove

An example of the main function is:

```
int main() {
  // this constructor must call the function insert multiple times
  // respecting the order
  RBTree tree(41, 38, 31, 12, 19, 8);
  tree.print();

  // testing the remove function
  tree.remove(8);
  tree.print();
}
```

2 Radix Sort

(2pts) Your job is to implement the radix sort algorithm in Python. The following code is going to be used to test your implementation. You have to submit a notebook with your code.

```
def radix_sort(A, d, k):
  # A consists of n d-digit ints, with digits ranging 0 -> k-1
  #
  # implement your code here
  # return A_sorted


# Testing your function
A = [201, 10, 3, 100]
A_sorted = radix_sort(A, 3, 10)
print(A_sorted)
# output: [3, 10, 100, 201]
```

3 Sorting in Place in Linear Time

(1.5pts) Suppose that we have an array of n data records to sort and that the key of each record has the value 0 or 1. An algorithm for sorting such a set of records might possess some subset of the following three desirable characteristics:

1. The algorithm runs in $O(n)$ time.
2. The algorithm is stable.
3. The algorithm sorts in place, using no more than a constant amount of storage space in addition to the original array.

(a) Give an algorithm that satisfies criteria 1 and 2 above.

(b) Give an algorithm that satisfies criteria 1 and 3 above.

(c) Give an algorithm that satisfies criteria 2 and 3 above.

(d) Can any of your sorting algorithms from parts(a)–(c) be used to sort n records with b-bit keys using radix sort in $O(bn)$ time? Explain how or why not.

(e) Suppose that the n records have keys in the range from 1 to k. Show how to modify counting sort so that the records can be sorted in place in $O(n+k)$ time. You may use $O(k)$ storage outside the input array. Is your algorithm stable? (Hint: How would you do it for $k = 3$?)

4 Alternative Quicksort Analysis

(1.5pts) An alternative analysis of the running time of randomized quicksort focuses on the expected running time of each individual recursive call to QUICKSORT, rather than on the number of comparisons performed.

(a) Argue that, given an array of size n, the probability that any particular element is chosen as the pivot is $1/n$. Use this to define indicator random variables $X_i = I\{i\text{th smallest element is chosen as the pivot}\}$. What is $E[X_i]$?

(b) Let $T(n)$ be a random variable denoting the running time of quicksort on an array of size n. Argue that

$$E[T(n)] = E\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right] \tag{1}$$

(c) Show that equation 1 simplifies to

$$E[T(n)] = \frac{2}{n}\sum_{q=0}^{n-1} E[T(q)] + \Theta(n) \tag{2}$$

(d) Show that

$$\sum_{k=1}^{n-1} k \lg k \leq \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2 \tag{3}$$

(Hint: Split the summation into two parts, one for $k = 1, 2, \ldots, \lceil n/2 \rceil - 1$ and one for $k = \lceil n/2 \rceil, \ldots, n-1$.)

(e) Using the bound from equation 3, show that the recurrence in equation 2 has the solution $E[T(n)] = \Theta(n \lg n)$. (Hint: Show, by substitution, that $E[T(n)] \leq an \log n - bn$ for some positive constants a and b.)

Respostas

Questão 3

(a) Algoritmo Counting Sort satisfaz essas duas condições.

(b) Como temos keys com somente 0 e 1, podemos utilizar o algoritmo de PARTITION do Quicksort, que realiza a partição inplace em O(n) e com uso constante de espaço de armazenamento. Assim, cumpre as duas condições.

(c) Para esse caso podemos utilizar o Insert Sort, pois é estável e inplace.

(d) Somente o algoritmos de Counting Sort, pois é o único dos três que é stable e performa em O(n).

(e) Abaixo está o algoritmo de Counting sort modificado:

Algorithm 1: COUNTING_SORT(A,k)

```
counts = [0 * k]
for a = 1 in A do
    counts[a] = counts[a] + 1
end
c = 0
for i = 0 to k do
    for j = 1 to C[i] do
        c = c + 1
        A[c] = i
    end
end
```

O algoritmo não é estável.

Questão 4

(a) Sendo X_i um variável aleatório da definida como:

$$X_i = \begin{cases} 1, & \text{se o pivot for o menor elemento.} \\ 0, & \text{caso contrário.} \end{cases} \tag{4}$$

Assim, $\mathbb{E}[X_i] = P(X_i = 1) \cdot 1 + P(X_i = 0) \cdot 0 = P(X_i = 1) = \frac{1}{n}$

(b) O algoritmo do Quicksort realiza uma partição de complexidade $\Theta(n)$ em um elemento q, quebrando a lista em duas. Em seguida, o algoritmo é chamado recursivamente em cada uma dessas listas. Dessa forma, a complexidade com Quicksort pode ser escrita como:

$$T(n) = T(q-1) + T(n-q) + \Theta(n)$$

Calculando o valor esperado, temos que:

$$E[T(n)] = \sum_{q=1}^{n} T(n) \cdot P(pivot = q)$$

$$E[T(n)] = \sum_{q=1}^{n} (T(q-1) + T(n-q) + \Theta(n)) P(pivot = q) = \sum_{q=1}^{n} (T(q-1) + T(n-q) + \Theta(n)) E(X_q)$$

E finalmente, pela linearidade da esperança, temos que:

$$E[T(n)] = E\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right]$$

(c) Utilizando a linearidade da esperança e o fato que $E[X_i] = \frac{1}{n}$, temos:

$$E[T(n)] = E\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right]$$

$$E[T(n)] = \sum_{q=1}^{n} E\left[X_q(T(q-1) + T(n-q) + \Theta(n))\right] = \sum_{q=1}^{n} \frac{1}{n}[T(q-1) + T(n-q) + \Theta(n)]$$

$$E[T(n)] = \Theta(n) \cdot \frac{n}{n} + \frac{1}{n}\sum_{q=1}^{n}[T(q-1) + T(n-q)]$$

Perceba que:

$$\sum_{q=1}^{n}[T(q-1)] = T(0) + T(1) + ... + T(n-1) = \sum_{q=1}^{n}[T(n-q)] = \sum_{q=0}^{n-1} T(q)$$

Portanto:

$$E[T(n)] = \Theta(n) + \frac{2}{n}\sum_{q=0}^{n-1} T(q)$$

(d) Utilizaremos a soma por partes (ou transformação de Abel). Assim, para $S_n = \sum_{k=1}^{n-1} k$:

$$\sum_{k=2}^{n-1} k \lg k = \sum_{k=2}^{n-1}(S_k - S_{k-1}\lg k) = \sum_{k=2}^{n-1} S_k \lg k - \sum_{k=2}^{n-1} S_{k-1}\lg k =$$

$$= S_{n-1}\lg(n) - S_0 \lg 1 - \sum_{k=2}^{n-2}[S_k \lg(1 + 1/k)]$$

Note que $S_{n-1} = \frac{n^2-n}{2}$, logo:

$$\sum_{k=2}^{n-1} k \lg k = \frac{n^2-n}{2}\lg(n) - \sum_{k=2}^{n-2}\frac{k^2-k}{2}\lg(1+1/k) = \frac{n^2-n}{2}\lg(n) - \sum_{k=2}^{n-2}\frac{k-1}{2}k\lg(1+1/k)$$

Como a função $k \lg k$ é crescente, então:

$$\frac{n^2-n}{2}\lg(n) - \sum_{k=2}^{n-2}\frac{k-1}{2}k\lg(1+1/k) \leq \frac{n^2-n}{2}\lg(n) - \sum_{k=2}^{n-2}\frac{k-1}{2}\cdot 2\lg(3/2) =$$

$$= \frac{n^2}{2}\lg(n) - \frac{n}{2}\lg(n) - \frac{(n-2)(n-3)}{2}\lg(3/2) =$$

$$= \frac{n^2}{2}\lg(n) - \frac{n^2}{2}\lg n - \frac{n}{2} + \frac{5n}{2}\lg(3/2) - 3\lg(3/2) \leq \frac{n^2}{2}\lg(n) - \frac{n^2}{8}$$

Provando assim a desigualdade desejada.

(e) Prova por substituição. Assumimos que $T(q) \leq q \lg(q) + \Theta(n)$. Logo:

$$E[T(n)] = \Theta(n) + \frac{2}{n}\sum_{q=1}^{n-1} T(q) \leq \Theta(n) + \frac{2}{n}\sum_{q=1}^{n-1}(q\ lg(q) + \Theta(n)) =$$

$$= \Theta(n) + \frac{2}{n}\sum_{q=1}^{n-1}(q\ lg(q)) + \frac{(n-2)\cdot 2}{n}\Theta(n)$$

Aplicando a desigualdade do item (d), temos que:

$$E[T(n)] \leq \frac{2}{n}(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2) + \Theta(n) = n \lg n - \frac{1}{4}n + \Theta(n) = n \lg n + \Theta(n)$$

Portanto, $E[T(n)] = \Theta(n \lg n)$