Fundação Getulio Vargas
Estruturas de Dados e Algoritmos
Aluno: Marcelo Bianchi Barata Ribeiro
Prof. Jorge Poco

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();
}
```

Resposta:
Ver arquivo.

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]
```

Resposta:

Ver arquivo: radix_sort.ipynb.

Como referências, usei:

- Livro "Introduction to Algorithms", capítulo 8.4 (CORMEN, Thomas et al).
- Slide L3 do curso de Standford que está sendo utilizado pelo professor.
  - Link: http://web.stanford.edu/class/archive/cs/cs161/cs161.1178/

Além do exemplo de teste fiz mais alguns com números naturais. O radix sort faz recursão em cada dígito do conjunto de elementos da lista e chama bucket sort, que por sua vez cria uma chave-valor e usa o dígito como chave, chamando em seguida um stable sort em cada bucket gerado. Escolhi o insert sort dado que o foco não está na escolha do stable sort por algum critétio específico. Nesse caso, o insertion sort ocupa menos linhas, tornando o código mais conciso de modo a visualizar suas outras partes.

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 $k = 3$?)

Resposta:

a) Nesse caso, podemos aplicar counting sort, dado que sabemos o número distinto de valores. Counting sort tem complexidade O(n) e é um stable sort, tal como demonstrado no livro.

b) Nesse caso específico, Quicksort pode ser aplicado uma única vez se escolhermos 0 como pivô, dado que isso vai separar a lista, deixando todos os 0s do lado esquerdo e todos os 1s do lado direito. Quicksort é

sempre inplace. Sua complexidade em geral é O(nlogn), mas será O(n) neste caso.

c) Insertion sort é in place e também é conhecido como um stable sort.

d) Counting sort.

e)

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:

ver livro , 118, 179

a)
Tal como explicado no livro "Introduction to Algorithms", capítulo 5 (CORMEN, Thomas et al), dado um espaço amostral S e um evento A, a Indicadora da variável aleatória I{A} associada ao evento A será definida por:

$$I\{A\} = \begin{cases} 1 & \text{se A ocorrer} \\ 0 & \text{se A não ocorrer} \end{cases}$$

Neste caso, queremos encontrar o valor esperado do *i-ésimo menor elemento a ser selecionado como pivô* (A_i). O espaço amostral é dado por S = $\{A_1, A_2, ..., A_n\}$, sendo a probabilidade de $A_i = \frac{1}{n}, \forall 1 \leq i \leq n$, pois

as probabilidades de cada evento são todas iguais e, somadas, totalizam 1. Podemos definir uma variável aleatória Xi associada ao evento A_i. Essa variável conta o número de vezes que o pivô 'i' será selecionado ao fazer a seleção, que será 1 em caso afirmativo, ou 0 caso contrário. Escrevemos então:

$$X_i = \begin{cases} 1 & \text{se } A_i \text{ ocorrer} \\ 0 & \text{se } A_i \text{ não ocorrer} \end{cases}$$

O número esperado de vezes que A_i será selecionado como pivô é simplesmente o valor esperado da variável indicadora Xi:

$$\begin{aligned} &E[X_i] = E[I\{A_i\}] \\ &= 1 \cdot Pr\{A_i\} + 0 \cdot Pr\{A_i\} \\ &= 1 \cdot \frac{1}{n} + 0 \cdot \frac{n-1}{n} \\ &= \frac{1}{n} \end{aligned}$$

Portanto, o número esperado de vezes que A_i será selecionado como pivô é $\frac{1}{n}$

b)
Primeiro, definindo T(n):
Relembrando o algoritmo quicksort:
QUICKSORT(A, p, r)
1. if p < r:
2. —- q = PARTITION(A, p, r)
3. —- QUICKSORT(A, p, q-1)
4. —- QUICKSORT(A, q+1, r)
Sendo que a chamada inicial da função é: QUICKSORT(A,1,A.length).
Não precisamos entrar em muitos detalhes sobre o PARTITION. Essencialmente, será definido um pivô "q" por meio do qual o array A será dividido entre os elementos menores do que q e os maiores do que q, formando duas sublistas onde se dará nova recursão. Tal tarefa de particionamento tem custo linear, $\Theta(n)$
Portanto, a recursão de quicksort é $T(n) = T(q-1) + T(n-q) + \Theta(n)$

A partir daqui podemos We can apply linearity of expectation over all of the events X_i. Suppose we have a particular X_i be true, then, we will have one of the sub arrays be length $i-1$, and the other be $n-i$, and will of course still need linear time to run the partition procedure. This corresponds exactly to the summand in equation (7.5).
tt

c)rever para ficar exatamente igual. entender linha 6

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

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

$$= \sum_{q=1}^{n} \frac{(T(q-1) + T(n-q) + \Theta(n))}{n} \text{ Usando o que foi demonstrado na letra 'a'}$$

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

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

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

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

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

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

d)

e)

Utilizando o método de substituição, estipulamos que $T(q) \leq q\lg(q) - bn$ e assumimos por indução que a propriedade se mantém. Pela proposição "c", temos que:

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

$$\leq \frac{2}{n}\sum_{q=0}^{n-1} aq\lg q - bq + \Theta(n) \quad \text{Aplicando substituição.}$$

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

$$\leq \frac{2}{n}(\frac{1}{2}an^2\lg n - \frac{a}{8}n^2 - \frac{bn^2}{2}) + \Theta(n) \text{ Aplicando o resultado de "d".}$$

$$= an\lg n - \frac{an}{4} - \frac{2}{n}\frac{bn^2}{2} + \Theta(n)$$

$$= an\lg n - bn - \frac{an}{4} + \Theta(n)$$

$$\leq an\lg n - bn + \Theta(n),\ \forall a, b > 0$$

$$= \Theta(n\lg n).$$