ASD Homework 3

Francesco Iannaccone e Matteo Conti

19 novembre 2021

1 Homework Esercitativi

1.1 Hashing

L'ipotesi di hashing uniforme implica che tutte le chiavi siano mappate nelle posizioni dell'array con la stessa probabilità. La probabilità che due elementi dell'insieme delle chiavi collidano è quindi pari a $\frac{1}{m}$. Consideriamo una variabile aleatoria indicatrice $X_{ij} = I\{\text{l'elemento } i\text{-esimo collide con quello } j\text{-esimo}\}$. Il valore atteso di questa variabile, per quanto detto, è pari a $\frac{1}{m}$.

Il numero totale di collisioni è quindi indicato dalla variabile aleatoria

$$X = \sum_{i=1}^{n-1} \sum_{j=i-1}^{n} X_{ij}$$

.

Il valore atteso di questa può essere espresso quindi come:

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i-1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i-1}^{n} \frac{1}{m} = \frac{1}{m} \sum_{i=1}^{n-1} n - i = \frac{n(n-1)}{2m}$$

1.2 Alberi binari di ricerca

L'operazione di costruzione dell'albero richiede di richiamare n volte il metodo TREE-INSERT. Questa procedura ha complessità pari a $\Theta(h)$ con h pari all'altezza dell'albero binario. Nel caso peggiore, che si verifica nel caso di albero completamente sbilanciato, avremo che l'altezza h è pari a n. Si avrà quindi nel caso peggiore una complessità pari a $\sum_{i=1}^{n} n = \Theta(n^2)$. Nel caso migliore, ovvero quando l'albero è perfettamente bilanciato, avremo invece un'altezza dell'albero pari a $\lg_2(n)$. Ripetendo il calcolo, si avrà in questo caso una complessità pari

a $\sum_{i=1}^{n} \lg(n) = \Theta(n \lg_2(n))$. L'operazione di TREE-WALK, essendo a complessità lineare, non inficia sulle performance asintotiche. Avremo quindi, nel complesso una complessità pari a

- $\Theta(n^2)$ per il caso peggiore,
- $\Theta(n \lg_2(n))$ nel caso migliore.

2 Homework di verifica

2.1 Radix tree

Definiamo n la somma delle lunghezze delle stringhe da ordinare lessicograficamente e k il numero di queste. La procedura di costruzione del radix tree necessita di k inserimenti. Ogni inserimento di una stringa i di lunghezza l_i allo stesso tempo necessita di effettuare un numero di confronti pari alla sua lunghezza complessiva. Procedendo ad inserire k stringhe di lunghezza $l_1, l_2, \ldots, l_k$ otteniamo un numero complessivo di operazioni pari a:

$$\sum_{i=1}^{k} l_i = l_1 + l_2 + \cdots + l_k = n$$

. La complessità computazionale della costruzione del radix tree è quindi nel complesso $\Theta(n)$.

La fase successiva dell'ordinamento necessita poi di andare a percorrere l'albero con una procedura di TREE-WALK. In questo caso la complessità sarà pari al numero totale di nodi del radix tree. Questo valore è sempre minore o uguale alla somma n delle lunghezze di tutte le stringhe.

La complessità totale dell'ordinamento è quindi $T(n) = O(n)+\Theta(n) = \Theta(n)$.

2.2 Treaps

L'idea fondamentale su cui si basa l'algoritmo di inserimento all'interno del treap è quella di effettuare un normale inserimento dell'elemento in arrivo senza rispettare il vincolo di priorità procedendo poi a correggere la posizione dell'elemento inserito in modo tale che la priorità del suo nodo padre risulti minore della sua. Per riorganizzare l'heap, mantenendo le proprietà di ordinamento dell'albero binario di ricerca utilizziamo ripetutamente le rotazioni che, come noto, mantengono le proprietà dell'albero.

Algorithm 1: Treap_insert(T, x)

```
1 tree_insert(T, x);
2 while x.parent ≠ nil and x.parent.priority > x.priority do
3 |   if x = x.parent.left then
4 |   |   right_rotate(T, x.parent);
5 |   else
6 |   |   left_rotate(T, x.parent);
7 |   end
8 end
```

dove le procedure qui utilizzate sono le classiche tree_insert, right_rotate e left_rotate note in letteratura.

L'algoritmo consta di due fasi: la prima, ovvero quella di inserimento, che, supponendo il bilanciamento dell'albero, ha una complessità pari a $O(h) = O(\log(n))$, e la seconda, in cui si ripristina la priorità che, anche in questo caso, ha complessità pari a $O(h)$ dove h è l'altezza del treap. L'algoritmo di Treap_insert ha quindi complessità pari a $O(\log(n))$.