Comprendiamo i dati sperimentali

## **Organizzazione della tabella di hash**

1. Quando si memorizzano i dati presenti nel file ascii.txt in una tabella di dimensione 20, la deviazione standard che otteniamo usando h1 e h2 è la stessa. Analogamente, se la tabella è di dimensione 100 la deviazione standard risulta la stessa, a prescindere dal fatto che si sia usata h1 o h2. Lo stesso accade se la dimensione è 1000. Per quale ragione?

**Perché nel file ascii.txt sono contenute solo chiavi (stringhe) composte da un solo carattere, quindi le due funzioni ritornano gli stessi valori.**

2. A differenza di quanto accade con i dati nel file ascii.txt, quando si memorizzano i dati presenti nel file d.txt la deviazione standard varia sia con la dimensione della tabella, sia con il variare della funzione di hash usata. Per quale ragione?

**Perché la funzione h1 su una lista di parole che iniziano sempre con la stessa lettera (in questo caso d) ritornerà sempre lo stesso indice e quindi le parole saranno distribuite in modo meno efficiente rispetto ad h2.**

3. Qual è la funzione di hash migliore, tra h1 e h2, per memorizzare dati come quelli nei file a.txt, b.txt, etc? Per quale ragione?

**La funzione h2 è migliore di h1 per memorizzare dati che iniziano con lo stesso carattere perché altrimenti h1 li inserirebbe tutti nello stesso indice**

## **Efficienza delle operazioni del TDD “Dizionario” usando strutture dati diverse**

1. Perché ricercare “azure” in una tabella di hash in cui sono stati inseriti i dati nel file a.txt risulta sempre un'operazione molto efficiente, a prescindere dalle caratteristiche (dimensione, funzione di hash usata) della tabella stessa?

**Dato che l’inserimento dei valori nel file a.txt procede in ordine alfabetico ma questi vengono inseriti in testa al Bucket, le ultime chiavi nel file a.txt si troveranno sempre all’inizio della lista in posizione h(k)**

2. Perché invece ricercare “avulso” in una tabella di hash in cui sono stati inseriti i dati nel file a.txt dipende fortemente dalle caratteristiche (dimensione, funzione di hash usata) della tabella stessa?

**Perché la parola “avulso” non esiste nel file a.txt quindi la funzione di ricerca scannerizza un’intera lista senza successo. In generale, l’efficienza della funzione di ricerca dipende dalle caratteristiche della tabella di hash.**

3. Quale struttura dati supporta in modo più efficiente l'operazione di inserimento? Perché?

**La struttura dati che supporta in modo più efficiente l’operazione di inserimento è la tabella di hash *perfetta* perché avendo la funzione di hash iniettiva, l’inserimento avrà Θ(1) sia nel caso migliore sia nel caso peggiore.**

4. Quale struttura dati supporta in modo più efficiente l'operazione di ricerca? Perché?

**La struttura dati che supporta in modo più efficiente l’operazione di ricerca è la tabella di hash *perfetta* perché avendo la funzione di hash iniettiva, si può ricercare ogni elemento direttamente all’indice h(k) mantenendo la complessità Θ(1).**

5. Che cosa notate, rispetto al tempo richiesto da qualunque operazione, quando usate h2?

**Il tempo è minore rispetto a h1;**

**Questo è dovuto allo scarto quadratico medio di h2, che è inferiore a quello di h1.**

## **La vostra funzione di hash, h3**

Che funzione di hash avete implementato? Quali sono i criteri che avete seguito? Che prestazioni offre rispetto ad h1 e h2?

**La funzione di hash implementata è la seguente:**

**int h3(Key s)**

**{**

**int somma\_ascii = 0;**

**for(unsigned int i = 0; i < s.size(); ++i)**

**somma\_ascii += (int)(s[i]);**

**somma\_ascii \*= (5/s.size());**

**return somma\_ascii % tableDim;**

**}**

**Il criterio seguito è che la lunghezza media delle parole inglesi presenti nei file txt è 5-6 caratteri. Quindi h3 dovrebbe distribuire più omogeneamente i dati in base alla loro lunghezza. In realtà, generalmente le prestazioni sono peggiori rispetto ad h1 e h2 perchè quando s.size() < 5, il valore di somma\_ascii diventa 0, il che aumenta notevolmente lo scarto quadratico medio.**