Laboratorio Di Algoritmi e Strutture Dati - Anno 2020/2021

Stefano Peddoni

Matr 839138

Relazione Esercizio 1

Nell'Esercizio 1 è stata implementata una libreria generica con alcune funzioni:

```
// Create new Generic Array
GenericArray* GenericArray_create(GenericArrayCmp);

// Returns 1 if the array is empty, 0 if the array is not empty
int GenericArray_empty(GenericArray* generic_array);

// Returns the capacity of the Generic Array
int GenericArray_capacity(GenericArray* generic_array);

// Returns the number of elements currently stored in the Generic Array
unsigned long GenericArray_size(GenericArray* generic_array);

// Deallocates the Generic Array
void GenericArray_free(GenericArray* generic_array);

// It accepts as input a pointer to an Generic Array and an integer "i" and it
returns the pointer to the i-th element of the Generic Array
void* GenericArray_get(GenericArray* generic_array, int i);

// Method that doubles the size of the Generic Array
void GenericArray_check_and_realloc(GenericArray* generic_array);

// Insert an element inside the Generic Array
void GenericArray_insert(GenericArray* generic_array, void* element);

// Reverses the order of elements in the generic array
void GenericArray_reverse(GenericArray* generic_array);
```

Dopo aver creato il generic_array con il suo header file; ho creato l'applicativo per poter eseguire l'algoritmo ibrido secondo un certo valore K che andremo ad esaminare in seguito. A seconda del valore di K fornito, viene utilizzato l'Insertion Sort o il Merge Sort.

I due algoritmi presi in questione sono caratterizzati in questa maniera:

ALGORITMO	CASO PEGGIORE	CASO MIGLIORE
MergeSort	O(nlogn)	O(nlogn)
InsertionSort	O(n^2)	O(nlogn)

Possiamo notare che l'algoritmo del MergeSort sia nel caso peggiore che nel caso migliore ha una complessità asintotica di O(nlogn)

Considerando l'applicativo, ho notato che i valori di K, considerando ad esempio K = 40; possono cambiare di qualche secondo di volta in volta per via del carico del processore.

Il valore di K se aumentato in maniera significativa porta quindi ad un aumento del valore di lettura, arrivando quindi a metterci diversi minuti.

Relazione Esercizio 2

Frase:

Quando avevo cinqve anni, mia made mi perpeteva sempre che la felicita e la chiave della vita. Quando andai a squola mi domandrono come vuolessi essere da grande. Io scrissi: selice. Mi dissero che non avevo capito il corpito, e io dissi loro che non avevano capito la wita.

Osservazioni:

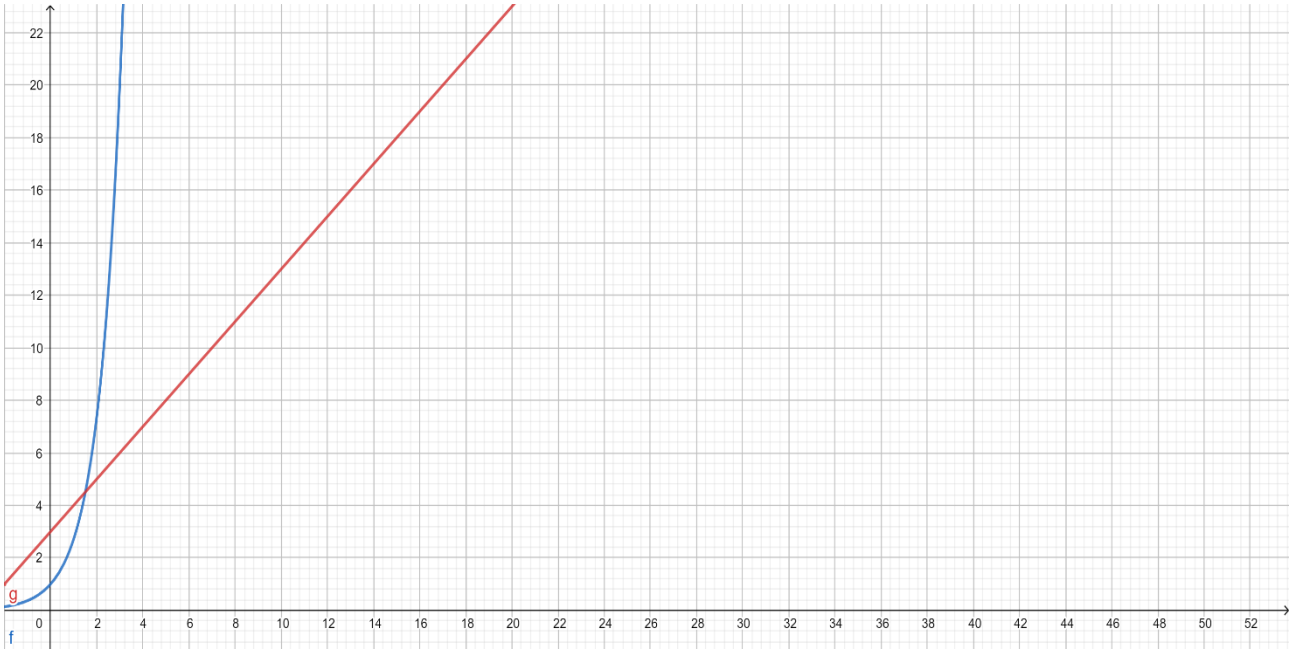
Nel momento della correzione, ho potuto notare che le parole: “made” e “selice” rimangono tali, essendo due parole corrette nel dizionario italiano; mentre le parole “squola” e “corpito” vengono corrette in maniera sbagliata.

Dopo aver eseguito l’esercizio, mi sono reso conto che l’edit_distance è molto lento a leggere tutte le parole del file, a differenza dell’edit_distance_dyn che ci mette molto meno tempo.

Quindi, per rendere più efficiente il calcolo della distanza tra le parole, l’algoritmo utilizza una tecnica di programmazione dinamica producendo il lavoro in tempo minore rispetto all’edit distance che produce il lavoro in tempo esponenziale.

Grafico delle funzioni:

Qui possiamo trovare un esempio di come si comportano i due algoritmi all’interno del programma.



Sull’asse delle ascisse troviamo la lunghezza delle parole, mentre sull’asse delle ordinate troviamo il tempo di esecuzione.

La funzione f (blu) rappresenta l’algoritmo classico, quindi l’edit distance (crescita esponenziale).

La funzione g (rossa) rappresenta l’algoritmo dinamico, quindi l’edit distance dyn (crescita lineare e decisamente migliore rispetto al metodo classico).

Metodo Dinamico:

Utilizzando l'algoritmo dinamico (tecnica basata sulla divisione del problema in sotto problemi), viene richiesta più memoria rispetto al metodo classico. Questo è molto favorevole, poiché l'aumento della memoria è trascurabile rispetto al tempo di calcolo.

Relazione Esercizio 3 – 4

Nell'esercizio 3 è stata implementata la struttura dati Union-Find-Set, per la cui realizzazione è stata utilizzata la class Tree. Come struttura di supporto è stata implementata una hashmap che permette l'accesso ai dati in tempo costante.

Nell'esercizio 4 è stata implementata la struttura di un Grafo.

Qui di seguito elencherò i metodi usati dall'HashMap all'interno di Graph:

```
private HashMap<V, ArrayList<Edge<V, E>>> graph;
```

METODO	COMPLESSITA'	DESCRIZIONE
graph.put	O(1)	Viene utilizzata per inserire la coppia <chiave, valore>.
graph.containsKey	O(1)	Viene utilizzata per verificare se la chiave è contenuta nel grafo e ritorna se è presente
graph.get	O(1)	Viene utilizzata per verificare la chiave e restituisce il valore associato.
graph.size	O(1)	Restituisce la dimensione del grafo
graph.remove	O(1)	Viene utilizzata per rimuovere l'elemento dal grafo
graph.keySet	O(1)	Restituisce una vista delle chiavi

Inoltre, è stata utilizzata anche un'ArrayList che consente di rappresentare sequenze di oggetti di lunghezza variabile all'interno dell'esercizio.

Le seguenti operazioni richieste sono:

Creazione di un grafo vuoto

In **graph**, è stato creato un grafo vuoto attraverso una HashMap, un parametro direct che essendo boolean determina se il grafo menzionato è diretto (TRUE) oppure non diretto (FALSE). La complessità del problema è di O(1) non avendo cicli.

Aggiunta di un nodo

In **addVertex**, aggiungo un nodo all'interno del grafo, ritorno TRUE se il parametro v viene aggiunto al grafo, FALSE nel caso contrario. La complessità asintotica del problema è di O(1) non avendo cicli.

Aggiunta di un arco

In **addEdge**, aggiungo un arco all'interno del grafo. In questo caso nel metodo addEdge vado a verificare se il grafo è orientato o non orientato; perché nel primo caso il grafo conterrà (a,b), mentre nel secondo caso (grafo non orientato) conterrà (b,a). La complessità del problema è di O(1) non avendo cicli.

Verifica se il grafo è diretto

In **isDirect**, verifico se il grafo in questione è diretto o no. Nel primo caso ritorna TRUE, nel secondo caso FALSE. La complessità del problema è di O(1) non avendo cicli.

Verifica se il grafo contiene un dato nodo

In **containsVertex**, verifico se un dato nodo è contenuto all'interno del grafo. In questo caso verifico se a è contenuto all'interno di esso. La complessità del problema è di O(1) non avendo cicli.

Verifica se il grafo contiene un dato arco

In **containsEdge**, verifico se un dato arco è contenuto all'interno del grafo. In questo caso verifico se a e b sono contenuti all'interno di esso. La complessità del problema è di O(1) non avendo cicli.

Cancellazione di un nodo

In **deleteVertex**, passiamo come parametro a l'elemento che vogliamo cancellare. La complessità asintotica del problema è O(n) poiché all'interno del metodo abbiamo un ciclo for.

Cancellazione di un arco

In **deleteEdge**, passiamo come parametro a e b gli elementi che vogliamo cancellare. La complessità del problema è di O(1) non avendo cicli.

Determinazione del numero di nodi

In **sizeVertex** determino il numero di nodi presenti nel grafo. La complessità asintotica è O(1) non avendo cicli.

Determinazione del numero di archi

In **sizeEdge** determino il numero di archi presenti nel grafo. La complessità asintotica del problema è di O(n) avendo un ciclo while all'interno.

Recupero dei nodi del grafo

In **getAllVertex**, recupero tutti i nodi presenti nel grafo. La complessità asintotica del problema è O(n) avendo un ciclo for all'interno.

Recupero degli archi del grafo

In **getAllEdge**, recupero tutti gli archi presenti nel grafo. La complessità asintotica del problema è O(n) avendo un ciclo for all'interno.

Recupero nodi adiacenti di un dato nodo

In **adjVertex**, recupero tutti i nodi adiacenti presenti nel grafo. La complessità asintotica del problema è O(1).

Recupero etichetta associata a una coppia di nodi

In **getWeigth**, recupero l'etichetta a,b associata ad una coppia di nodi presenti nel grafo. La complessità asintotica del problema è O(1).

KRUSKAL

KRUSKAL ordina gli archi secondo costi crescenti e costruisce un insieme ottimo di archi scegliendo di volta in volta un arco di peso minimo che non forma cicli con gli archi già scelti.

Inoltre, è stata richiesta un'implementazione dell'algoritmo di Kruskal e del suo applicativo, che, prendendo in input il file italian_dist_graph legge e stampa il numero di nodi, di archi e il peso.

È stato inserito anche un tempo di esecuzione, utilizzando timestamp, che mi stampa il numero di millisecondi che ci vogliono per stampare a video il risultato richiesto.