Merico Michele – Montesi Dennis – Turcan Boris

Relazione Esercizio 4

Per garantire le complessità richieste dalla consegna, abbiamo scelto di rappresentare la struttura dati GRAFO mediante una **HashMap** che permette di effettuare inserimento, ricerca e cancellazione in tempo O(1).

Creazione di un grafo vuoto in O(1)
Per creare un grafo vuoto, utilizziamo il costruttore offerto dalla API *HashMap* con chiave *String* e valore *List<String>* che rappresenta la lista delle adiacenze, ossia gli archi. Tale costruttore ha complessità O(1).

Aggiunta di un nodo in O(1)
Per aggiungere un nodo al grafo utilizziamo *map.put(key, new AdjList)* che ha complessità O(1).

Aggiunta di un arco in O(1)
Per aggiungere un arco al grafo controlliamo inizialmente l'esistenza dei nodi *begin* e *end*, se non sono presenti li aggiungiamo mediante il metodo *addNode* (il tutto ha complessità O(1)). In seguito aggiungiamo alla lista associata a *begin* il nuovo arco. In più, se il grafo è indiretto, si aggiunge l'arco anche alla lista associata a *end*. Quindi il metodo ha complessità totale O(1).

Verifica se il grafo è diretto in O(1)
La verifica è immediata, grazie alla presenza del flag *direct* all'interno della definizione della nostra struttura, quindi sia l'impostazione che recupero del suo valore si effettua in tempo costante O(1).

Verifica se il grafo contiene un dato nodo in O(1)
Per la verifica utilizziamo il metodo *map.containsKey(key)* che ha complessità O(1).

Verifica se il grafo contiene un dato arco in O(1)
Controlliamo che nella lista delle adiacenze del nodo *begin* sia presente il nodo *end* utilizzando il metodo map.*contains* che ha complessità O(1).

Cancellazione di un nodo in O(n)
Utilizziamo un *for* (complessità O(n)) per scorrere le chiavi della *HashMap:* se nella lista delle adiacenze del nodo corrente esiste un arco che coinvolge il nodo da eliminare, allora l'arco viene rimosso, infine rimoviamo la chiave dalla HashMap, quindi il nodo dal grafo. La complessità della cancellazione dell'arco è O(1) * O(n) = O(1*n) = O(n).

Cancellazione di un arco in O(1)
Per cancellare un arco, controlliamo che l'arco esista (complessità O(1)) e lo rimoviamo (compessità O(1)), mediante due operazione sulla *HashMap: map.get(key).remove(arco),* che per definizione della HashMap hanno complessità O(1). In più, se il grafo e' indiretto, effettuiamo le stesse operazioni in modo speculare rispetto al nodo *end*.

Determinazione del numero di nodi in O(1)
Immediato, pre definizione della API *HashMap, map.keySet.size()* – complessità O(1).

Merico Michele – Montesi Dennis – Turcan Boris

Determinazione del numero di archi in O(n)
Con l'ausilio del ciclo *for* (complessità O(n)) che scorre il set delle chiavi presenti, ovvero i nodi del grafo, sommiamo le lunghezze delle liste delle adiacenze di ciascun nodo. La complessità quindi è O(1) * O(n) = O(1*n) = O(n).

Recupero dei nodi del grafo in O(n)
Usiamo un ciclo *for* (complessità O(n)) per scorrere la HashMap e ad ogni iterazione del ciclo inseriamo in una lista il nodo corrente. La complessità quindi è O(1) * O(n) = O(1*n) = O(n).

Recupero degli archi del grafo in O(n)
Analogo al metodo precedente con l'inserimento delle liste al posto dei nodi.

Recupero nodi adiacenti di un dato nodo in O(1)
Sfruttando la definizione della HashMap, il recupero delle adiacenze avviene in tempo O(1) – *map.get(key)*.

Recupero etichetta associata a una coppia di nodi in O(1)
Recuperata la coppia dei nodi, il ritrovamento della etichetta è immediata per l'implementazione della struttura arco.