Relazione Es.4

Argomenti

Operazioni critiche

Prima di prendere qualsiasi scelta implementativa sono andato ad individuare le operazioni più critiche, ovvero quelle per cui avevo scarse opzioni implementative che soddisfassero la complessità richiesta.

Tali operazioni sono

- Verifica se il grafo contiene un dato nodo → $O(1)$
- Verifica se il grafo contiene un dato arco → $O(1)(*)$

Queste operazioni sono operazioni definibili di '*ricerca*' e tra le strutture studiate a lezione solamente una rispetta tale complessità: la **Hash Table**

Inoltre abbiamo che ogni nodo può apparire una sola volta all'interno del grafo per cui la Hash Table ci è perfetta in quanto la struttura non memorizza doppioni.

HashMap

Per rappresentare la Hash Table utilizziamo la classe HashMap fornitaci da Java la quale è molto simile alla Hashtable.

HashMap è composta da <key, value> dove ad ogni chiave possiamo associare un valore e la chiave deve comparire una sola volta.

Per cui avremo

- key = nodo
- value = lista di adiacenza

LinkedList

Per rappresentare le liste di adiacenza.

La classe LinkedList si basa sulle liste doppiamente concatenate ed è ottima da utilizzare quando ci sono molti inserimenti/rimozioni.

Avrei potuto anche utilizzare un ArrayList che conviene maggiormente per le letture.

Partendo da questa struttura tutte le complessità delle seguenti operazioni vengono rispettate

- *Creazione di un grafo vuoto* → $O(1)$

  La creazione della HashMap vuota ha costo $O(1)$
- *Aggiunta di un nodo* → $O(1)$

  Costo inserimento key HashMap = $O(1)$
- Aggiunta di un arco → $O(1)$

  Costo aggiunta di un elemento alla LinkedList associata ad una key = $O(1)$
- *Cancellazione di un nodo* → $O(n)$

  Quando andiamo a cancellare un nodo dobbiamo cancellare anche tutte le sue occorrenze negli archi.

  Costo cancellazione key HashMap = $O(1)$

  Costo cancellazione di tutti gli archi in cui compare = $O(n)$ dove n è il numero degli archi.
- *Cancellazione di un arco* → $O(1)(*)$

  Costo ricerca della LinkedList contenente l'arco = $O(1)$

  Costo ricerca dell'arco all'interno della LinkedList = $(*)$

- *Determinazione del numero dei nodi* → $O(1)$

  Costo calcolo numero di key uniche HashMap = $O(1)$
- *Determinazione del numero di archi* → $O(n)$

  Calcolare per ogni LinkedList quanti archi ha al suo interno = $O(n)$ dove n è il numero degli archi
- *Recupero dei nodi del grafo* → $O(n)$

  Costo ricerca di una key HashMap = $O(1)$

  Creazione di una lista contenente tutte le key = $O(n)$ dove n è il numero dei nodi.
- *Recupero degli archi del grafo* → $O(n)$

  Creazione di una lista contenente tutti gli archi di tutte le LinkedList = $O(n)$ dove n è il numero degli archi.
- *Recupero nodi adiacenti di un dato nodo* → $O(1)(*)$

  Costo ricerca della LinkedList = $O(1)$

  Costo creazione della lista con tutti gli archi di tale LinkedList = $(*)$
- Recupero etichetta associata a una coppia di nodi → $O(1)(*)$

  Costo ricerca della LinkedList = $O(1)$

  Costo ricerca dell'arco all'interno della LinkedList = $(*)$

Scelte implementative per le operazioni restanti

- Verifica se il grafo è diretto → $O(1)$

  Creazione di una variabile di classe che tenga traccia (dalla creazione dell'oggetto Grafo) di tale informazione.