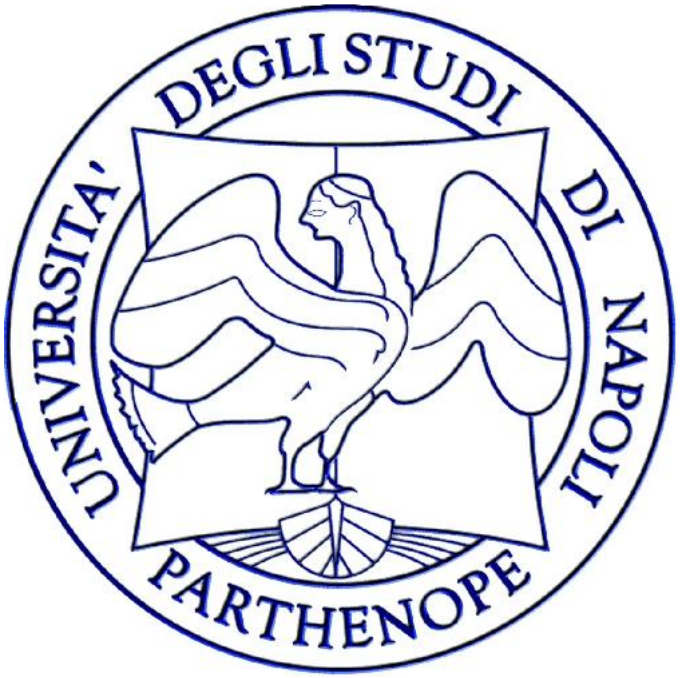
Università degli Studi di Napoli "Parthenope"
Dipartimento di Scienze e Tecnologie
Corso di laurea Triennale in Informatica



Algoritmi e Strutture Dati e Laboratorio di Algoritmi e Strutture Dati

Traccia 2

Docenti
Prof. Alessio Ferone
Prof. Francesco Camastra

Candidato
Vittorio Fones 0124/1384

Anno Accademico 2019-2020

Indice

Elenco delle figure

Albero red-black di hash table

1 Descrizione problema

Il problema in analisi prevede di creare una struttura dati, che d'ora in avanti chiameremo **red-black hash**, in grado di immagazzinare delle stringhe alfanumeriche. Tale struttura è l'unione di un albero binario di ricerca bilanciato, **albero rosso-nero** o **albero red-black**, e delle **hash table**: in particolar modo, all'interno di ogni nodo di tale albero, vi è presente una hash table, struttura dati che associa per ogni chiave un singolo valore, al cui interno sono presenti delle stringhe. La traccia prevedeva di poter effettuare operazioni **C.R.D.**[1] su tuple nel formato: `chiave1:chiave2:stringa` . In particolar modo: la chiave 1 indicizza un nodo dell'albero red black, il quale puntando ad una hash table utilizza la chiave 2 per associare la stringa. Vi è quindi una relazione `1:1` per i nodi dell'albero e l'hash table, e `1:M` tra l'hash table e le stringhe, dove **M** è la dimensione massima dell'hash table.

[1] Create Retrieve Delete. Operazioni tipiche delle basi di dati, ma senza la possibilità di effettuare Updates.

Viaggi Galattici

8 Descrizione problema

Il problema in esame prevede di trovare, in un **grafo non orientato**, il **percorso più breve** tra due nodi specifici, ovvero quel percorso tale per cui la somma dei costi associati all' attraversamento degli archi che collegano un punto A ad un punto B è

minima [2]. Il problema prevede la possibilità di poter usare alcuni nodi speciali, detti **wormholes**: ogni wormhole nel grafo è collegato ad ogni altro wormhole, inoltre il costo di percorrenza wormhole - wormhole ha peso 1.

9 Descrizione strutture dati

9.1 Grafi

Le informazioni circa il nome, ovvero la chiave numerica usata come identificativo univoco, ed eventuali dati satelliti, sono salvate in una struttura dati **nodo**, o **vertice**. Per salvare i percorsi dei cammini minimi, ogni vertice mantiene un riferimento al nodo precedente nel cammino. La struttura dati in cui vengono salvati i nodi facenti parte del problema, è un **grafo non orientato**. Per implementare tale struttura dati si è scelto di mantenere per ogni vertice un listato contenente un riferimento agli altri nodi adiacenti ed il relativo peso numerico atto a riportare il costo effettivo dell' attraversamento. Nel caso del grafo non orientato per ogni arco inserito non vi è distinzione tra **arco uscente** o **arco entrante**, per cui si andrà ad inserire due volte volte il sopracitato arco: un arco di peso w tra A e B equivale ad un arco da A a B e un altro da B ad A.

9.2 Coda di priorità

La coda di priotià è una particolare coda il cui criterio di inserimento dei vari elementi che compongono la coda è dato non più dall'ordine FIFO [3], ma dalla priorità associata ad ogni elemento. Nel caso della **coda a priorità minima** , gli elementi con priorità minima saranno inseriti all' inizio. L'operazione di ricerca del minimo, viene eseguita in $O(1)$, il che rende particolarmente utile la coda di priorità nelle applicazioni che fanno un grande uso della suddetta operazione. Tale ADT[4], è usata nell' algoritmo di **Dijkstra**, per la rapida estrazione degli elementi con minor distanza dalla sorgente. [5]

[2] Definizione formale del **cammino minimo** nella teoria dei Grafi
[3] FIFO: first in, first out.
[4] ADT: abstract data type.
[5] Algoritmo per il calcolo dei cammini minimi da sorgente unica.

9.3 Min Heap Binario

La struttura dati su cui si basa la coda a priorità minima, sebbene non sia la più efficiente[6], è l' **heap binario**, una struttura dati basata su albero binario completo sviluppato come vettore che gode della **proprietà heap**: nel caso del **min heap** *il padre di un nodo ha come chiave un valore minore di quella del figlio sinistro e destro.* È stata scelta questa struttura dati rispetto ad un **Heap di Fibonacci**, nonostante abbia complessità $O(logn)$ nelle operazione di estrazione del minimo, inserimento e abbassamento priorità, poiché di facile implementazione.

10 Formato di input e di output

10.1 Input

I dati in input del problema sono:

- V: numero intero di Vertici del grafo
- E: numero intero di Archi del grafo
- W: numero intero di Wormholes presenti nel grafo
- Tuple rappresentante archi: NodoA, NodoB, Peso [7]

Tali dati sono immagazzinati in un file di testo non binario contenente nel primo rigo i primi tre dati elencati, mentre nei successivi sono presenti le **tuple**. Per rappresentare i wormhole il programma prende gli **ultimi W NodiB** contenuti nel file e li va a salvare in un vettore di vertici.

10.2 Output

Il programma sviluppato restituisce in output i nodi che collegano la coppia **sorgente - destinazione** nel minor "tempo" possibile e il relativo costo di tale cammino minino, **se esiste**: può capitare, come vedremo nel paragrafo *"Test effettuati"*, che il grafo non sia connesso e che il nodo destinazione sia raggiungibile solo attraverso i vertici di tipo wormhole. Inoltre il programma restituisce, il cammino minimo (vertici da attraversare e costo totale) facendo uso dei nodi speciali wormhole. L'output secondario può mancare nel caso in cui non si incontrino wormhole, oppure il wormhole di partenza è uguale a quello di destinazione.

[6] Heap di Fibonacci ha complessità $O(1)$ nelle funzioni utili all' algoritmo

[7] ndr: NodoA e NodoB sono le chiavi intere dei vertici e Peso è un intero usato per rappresentare il peso di tale arco.

11 Descrizione algoritmo

11.1 Pseudo codice

Per calcolare i cammini minimi[8] da una sorgente si è scelto di utilizzare l'algoritmo greedy proposto dall'olandese **Edsger Dijkstra** che va a scegliere localmente un nodo adiacente più vicino a quello analizzato. Per definizione della sottostruttura ottima di un cammino minimo, il **primo wormhole** aggiunto nell'albero dei cammini minimi, è quello che si può raggiungere più velocemente da una data sorgente. Per tanto è stato modificato Dijkstra per salvare i womrhole che incontra durante la creazione del cammino, e per fermarsi una volta raggiunta la destinazione.

L'algoritmo che verrà mostrato di seguito è il cuore del programma: in input riceve i nodi sorgente e destinazione e in fase di elaborazione restituisce **i cammini minimi** dal nodo sorgente a quello destinazione. È stato usato il plurale in quanto, potrebbe esserci un secondo cammino che fa uso dei wormhole, o viceversa se il grafo non è connesso, esserci solamente il cammino con i wormhole.

La procedura **Galactic Dijkstra** applica una prima volta Dijkstra dalla sorgente fino a che non trova la destinazione e nel mentre salva tutti i wormhole che incontra, estraendone solo il primo (riga 1). Se esiste un wormhole nell'albero dei cammini minimi radicato in S, allora si procede ad una seconda applicazione di Dijkstra, usando come sorgente il nodo di destinazione. Se questi due wormhole sono diversi allora si calcola il percorso minimo tra i due e si aggiunge un arco simbolico di peso 1.

input : Source node $\in V$, Destination node $\in V$
result: print fast path from s to d, w/ **and** w/o wormholes if any

1 w_1 = *apply Dijkstra from S and save first wormhole encountered*;
2 $distance = printPath(s, d)$;
3 **if** $\exists\ w_1$ **:**
4 $\quad w_2$ = *apply Dijkstra from D and save first wormhole encountered*;
5 $\quad$ **if** $\exists\ w2$ **and** $w1 \neq w2$ **:**
6 $\quad\quad dw_1 = printPath(s, w_1)$;
7 $\quad\quad dw_2 = printPath(w_2, d)$;
8 $\quad\quad d_{dw_1+dw_2} = dw_1 + 1 + dw_2$;

Algorithm 1: Galactic Dijkstra

[8] Anche definiti come albero dei cammini minimi radicati in una sorgente.

11.2 Diagrammi delle classe e dettagli architetturali

11.2.1 Priority Queue e Heap

La coda di priorità è stata sviluppata come detto in precedenza facendo uso di un **Heap Binario**, per la precisione un Min Heap. Si poteva creare una classe generica priority queue e usare un **pattern comportamentale** (e.g. Strategy) per sfruttare la possibilità di cambiare comportamento (minima priorità o massima) in base ad un flag in fase di creazione della classe. Non è stato usato tale approccio in quanto nell'algoritmo di Dijkstra si fa uso solamente di una coda a minima priotità.

Un' altra nota riguarda l'uso della classe Min Heap all'interno della Priority Queue: sarebbe stato utile usare un' **interfaccia** (i.e. classe astratta) per l'heap dando la possibilità al programmatore di usare un altra tipologia (e.g.: Fibonacci, Brodal, Binomiale, etc.). Non avendo usato tale approccio la classe Min Priority Queue è dipendente dall' Heap Binario.

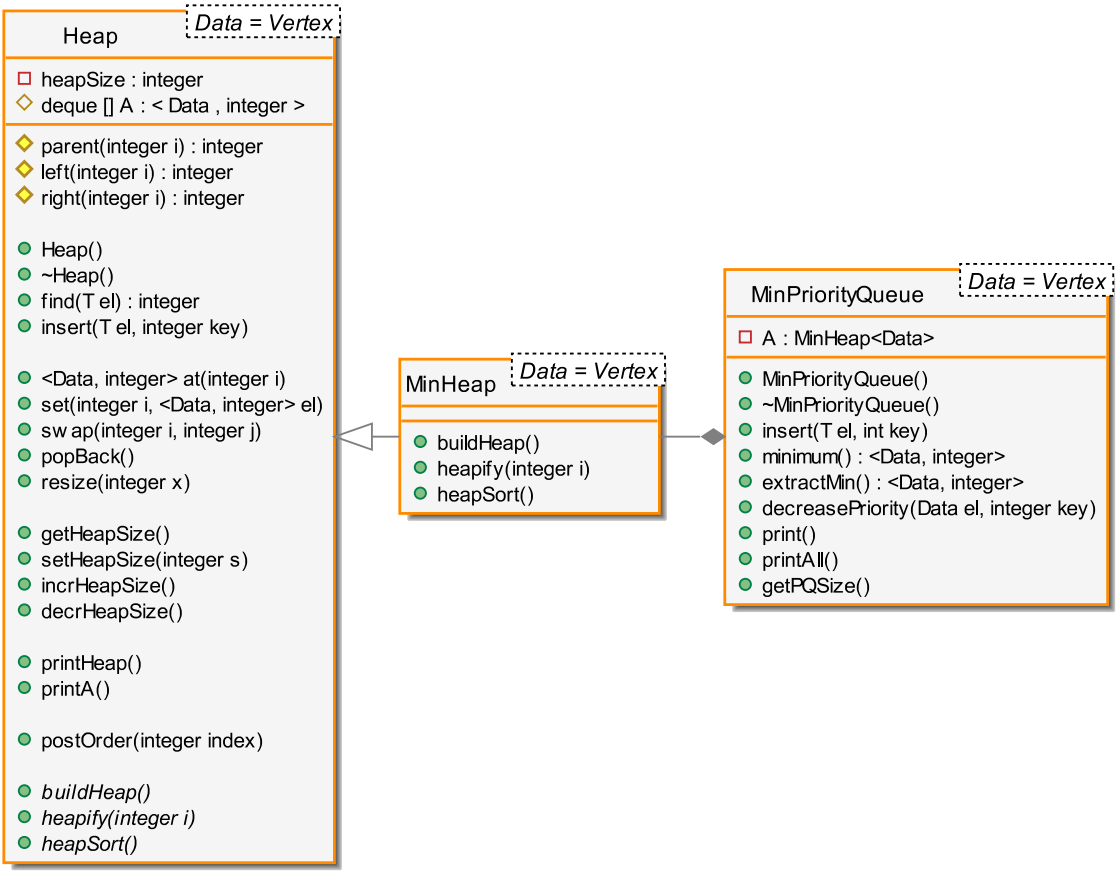


Figura 1: Min priority Queue

N.d.r.: ogni classe ha un parametro template Data, in questo caso specifico viene usato un puntatore ad oggetto Vertex, mostrato di seguito.

11.2.2 Grafo e Vertici

Ogni elemento del grafo, i **Vertici**, sono dei nodi che ereditano da un generico "oggetto" **Item** la possibilità di inserire Dati indentificati da una chiave. A tal proposito non avendo bisogno di conservare nessun dato si è deciso di usare come parametro del template un **puntatore a void**. Ogni vertice ha una mappatura con i vertici **adiacenti** (realizzata tramite un hashtable di tipo unordered), un riferimento al padre nell' albero dei cammini minimi, e la distanza dal nodo radice.

Il grafo possiede un vettore di puntatori a vertici e ha metodi per creare un albero dei cammini minimi (Dijkstra), restituire o stampare il percorso da una sorgente e una destinazione. Il metodo dijkstra è stato ridefinito in modo tale da poter eseguire operazioni aggiuntive alla fine del rilassamento di un nodo estratto dalla coda: si può decidere di usare una funzione lambda oppure un puntatore a funzione per inserire un **criterio di stop** nell'algoritmo (e.g. raggiunto un nodo specifico), inoltre verrà restituito l'ultimo elemento estratto.
N.d.r.: il metodo dijkstra internamente fa uso delle subroutine *initSingleSource* e *relax* come da manuale[9] ma poichè tale implementazione può cambiare (non far uso della Min Priority Queue ad esempio) si è deciso di non inserirle come metodi privati durante la definizione della classe. Per cui verranno citate e mostrate solo per far capire la connessione con la min priority queue.

[9]Vedere riferimenti bibliografici.

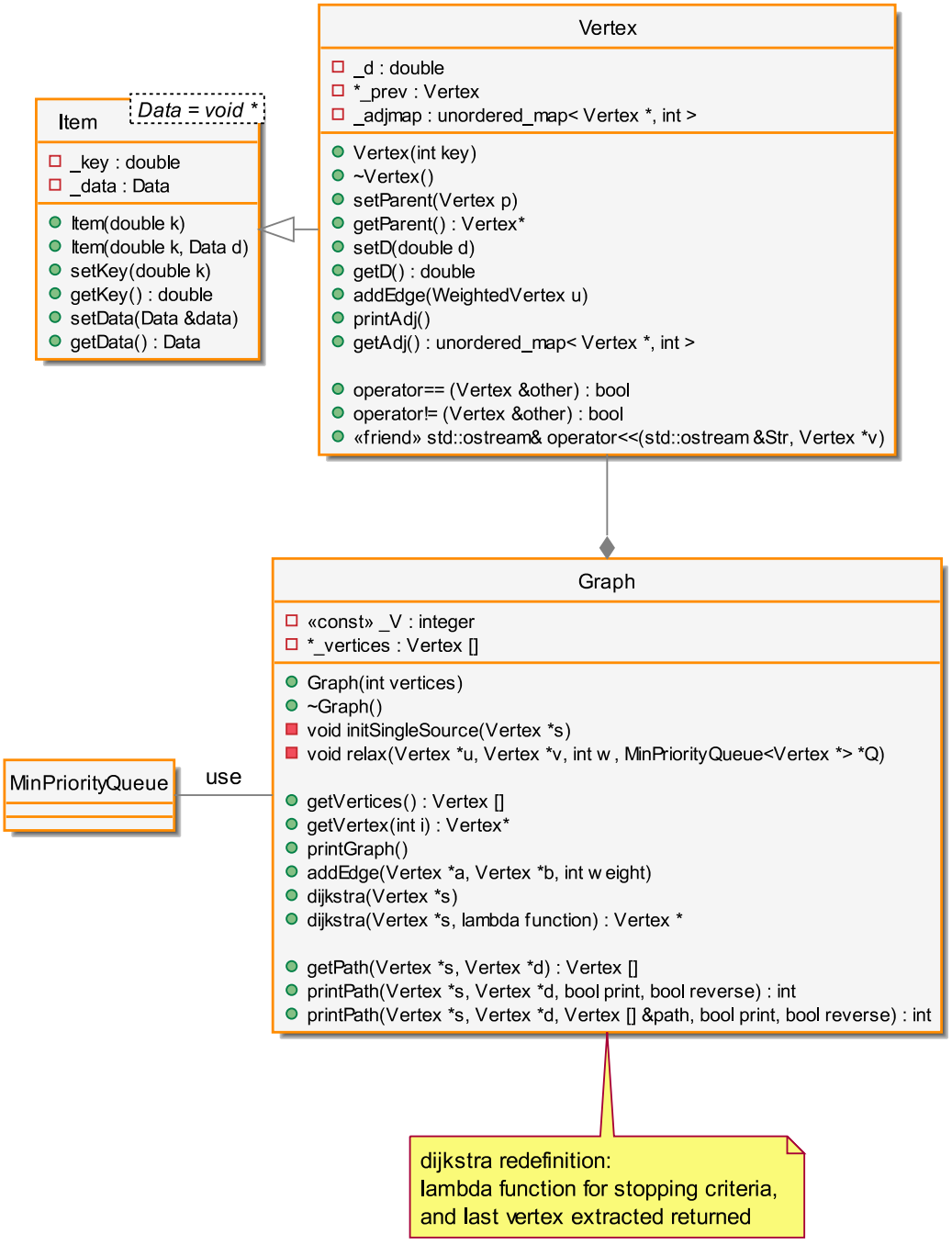


Figura 2: Vertici e Grafo

11.2.3 Parser e Grafo galattico

Il main del programma usera la classe Parser per creare e instanziare correttamente un oggetto di tipo GalacticGraph. Tale classe altro non è che una specializzazione del grafo base, con l'aggiunta di una mappatura dei wormhole del sistema caricato e dell' algortimo Galactic Dijkstra (par. 11.1).

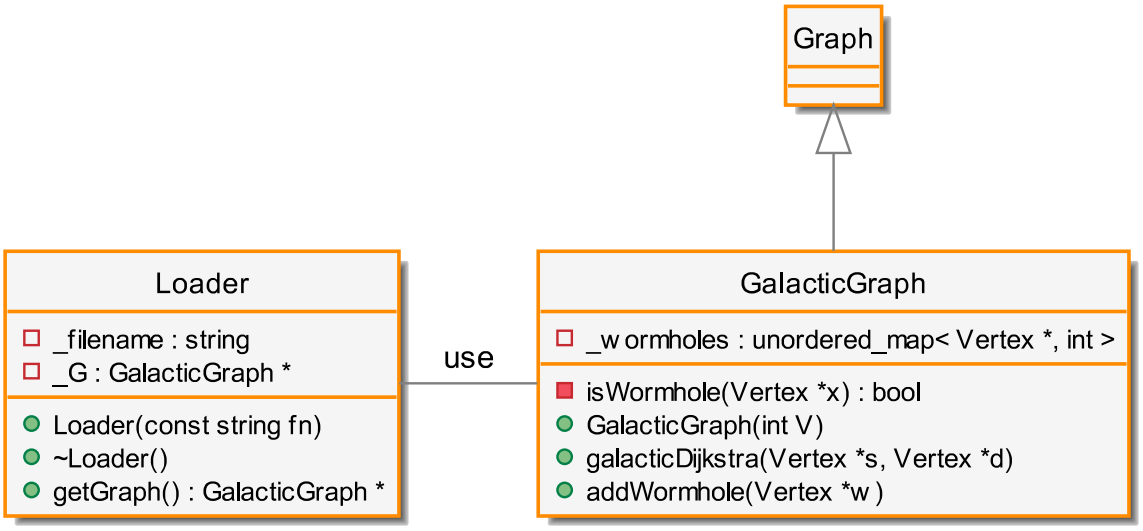


Figura 3: Loader e Grago galattico

12 Studio complessità

13 Test e risultati

13.1 Test effettuati

14 Codice sorgente