INGEGNERIA DEGLI ALGORITMI 2018/2019

SECONDA PROVA PRATICA IN ITINERE
PROGETTO 1

RELAZIONE

AUTORI:

- *GABRIELE ANSELMI*

  MATRICOLA: 0253209 *E-MAIL: gabriele@giulianoanselmi.it*

- *MICHELE TERENZI*

  MATRICOLA: 0254291 *E-MAIL: micheleterenzi@gmail.com*

NOTE:
Per lo svolgimento e la realizzazione del progetto è stato utilizzato l'IDE "PyCharm".
Per la realizzazione delle tabelle nella relazione è stato utilizzato "Excel"; per la realizzazione dei grafici nella relazione è stato utilizzato l'IDE "PyCharm", con un codice apposito (fornito a lezione e non riportato nel progetto), che utilizza i moduli "matplotlib" (per salvare il grafico come file immagine) , "numpy" (per la variazione del colore dei risultati da rappresentare) e "pandas" (che tramite una funzione permette di caricare in una matrice file contenenti i risultati dei test effettuati).

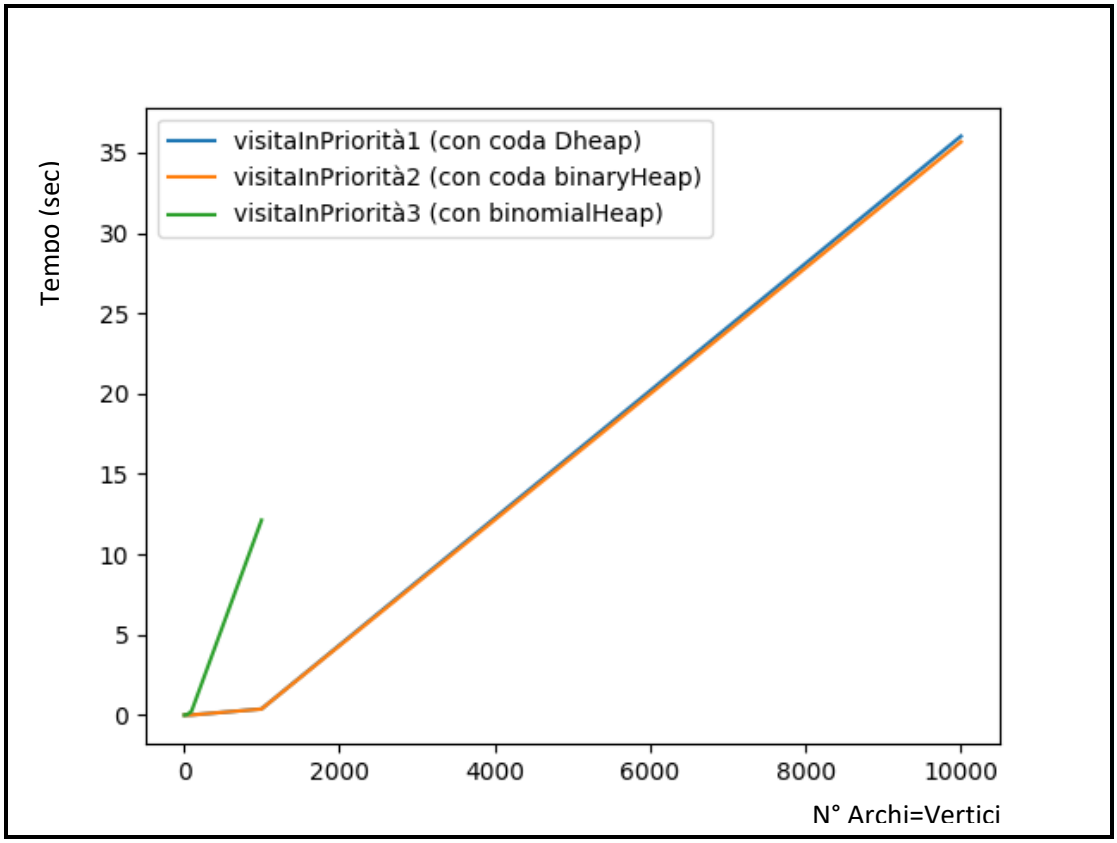
LOCATION DEI FILE:
I test effettuati si trovano nella cartella "Testing", nel file "testing.py" (che è il "main" del progetto; https://github.com/mikired100/AnselmiTerenzIa/blob/master/Testing/testing.py).
L'Algoritmo implementato, richiesto dalla traccia del progetto, ("visitaInPrioritaX", con X ∈{1,2,3} a seconda della coda con priorità implementata: 1 per Dheap, 2 per BinaryHeap, 3 per BinomialHeap) si trova nella cartella "graph", nel file "Graph.py"
(https://github.com/mikired100/AnselmiTerenzIa/blob/master/graph/Graph.py , righe 286-368).

SCELTE IMPLEMENTATIVE:
Le code con priorità implementate sono le classi "Dheap", "BinaryHeap" e "BinomialHeap" nelle tre varianti di "visitaInPriorita" ("visitaInPriorita1", "visitaInPriorita2", "visitaInPriorita3"). Ogni nodo è stato poi inserito con il metodo "insert()", utilizzando come chiave il suo peso.
In ogni classe di coda con priorità è stata inoltre implementata una funzione aggiuntiva per estrarre il "figlio (inesplorato)" con chiave massima (e quindi priorità e peso maggiore) di un determinato nodo.
Nel caso del "Dheap" e del "Binaryheap", per trovare il "figlio" con chiave massima, è stata utilizzata la funzione "maxSon(nodeId)" (variante della funzione "minSon(nodeId)", usata per trovare il "figlio" con chiave minima).
Nel caso del "BinomialHeap", per trovare il massimo ed estrarlo, è stata utilizzata la funzione "findMax() (variante della funzione "findMin()").
È stata implementata la funzione "genGraph(numVertici, numArchi, inputType)" che permette, attraverso le funzioni preesistenti "insertNode()" e "insertEdge()", la costruzione del grafo.
Per la stampa del grafo è stata utilizzata la funzione "print()" che scorrendo fra gli "i" nodi, stampa per ciascuno il suo adiacente.

RISULTATI SPERIMENTALI (TABULARI E GRAFICI):

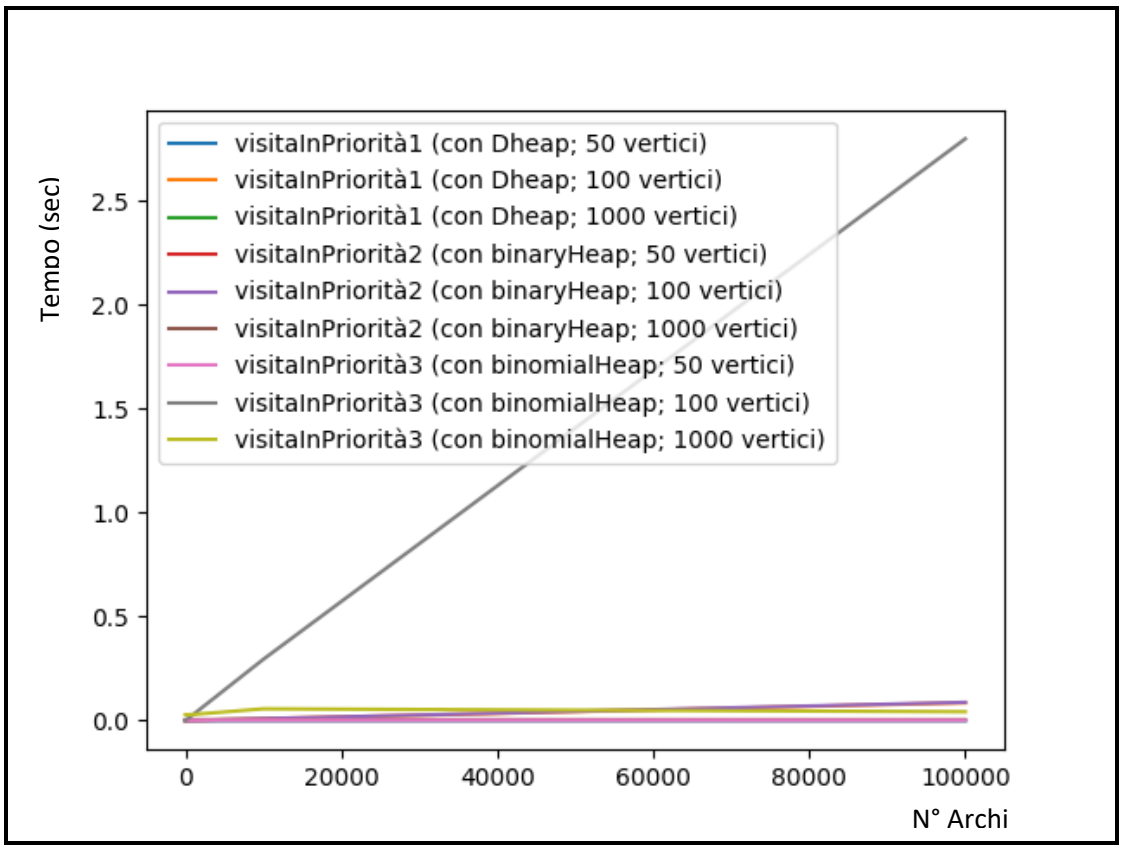
- TEST 1: Numero di archi uguale a quello dei vertici

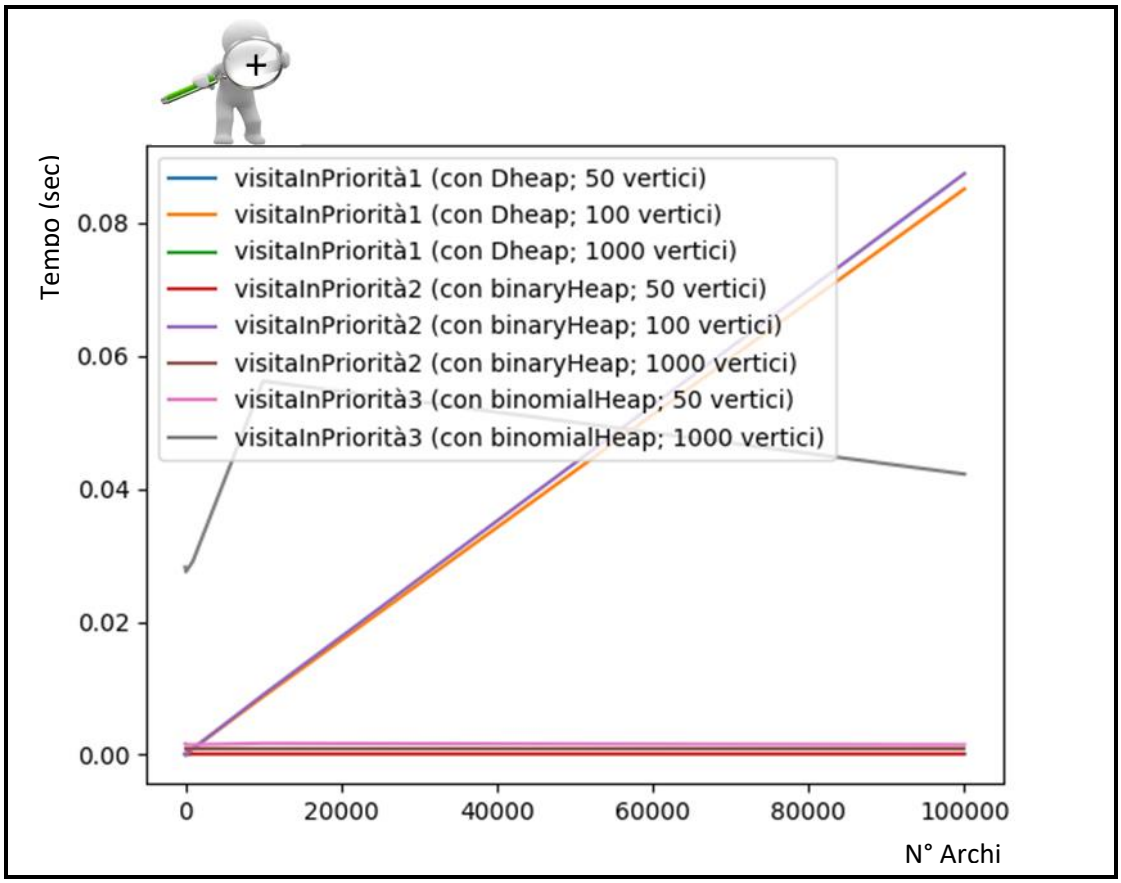
TEST 1 "visitaInPriorità"	TEMPI DI ESECUZIONE AL VARIARE DELL'INPUT (in sec)					
CODE CON PRIORITA' (insieme F)	**INPUT: N° DI ARCHI = N° DI VERTICI**					
	10	**20**	**50**	**100**	**1000**	**10000**
visitaInPriorità1 (con Dheap)	0,0001	0,0003	0,0015	0,0074	0,3816	36,0253
visitaInPriorità2 (con binaryHeap)	0,0001	0,0002	0,0017	0,0074	0,3755	35,6708
visitaInPriorità3 (con binomialHeap) *	0,0013	0,0075	0,0486	0,2418	12,1299	— — —
* Con 10000 vertici il tempo è eccessivamente elevato, per questo il relativo grafico non riporta il risultato oltre Archi=Vertici=1000						

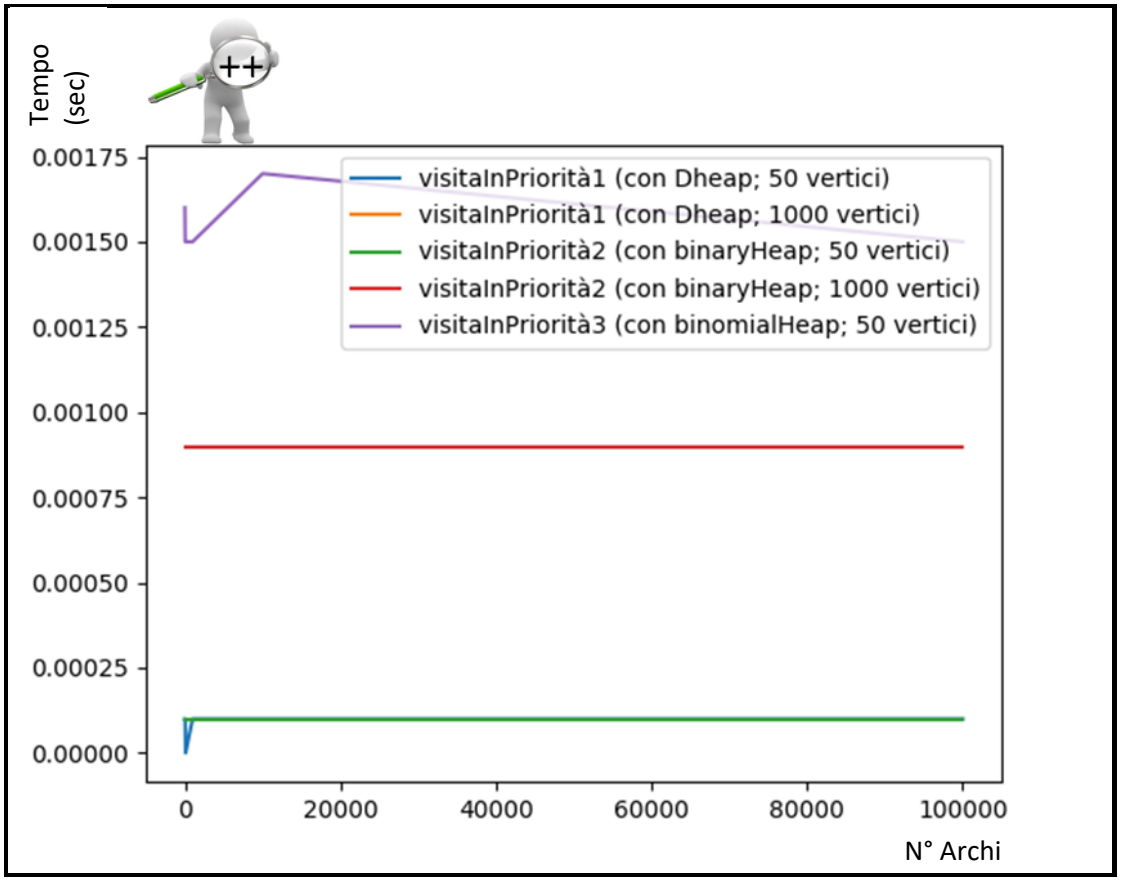


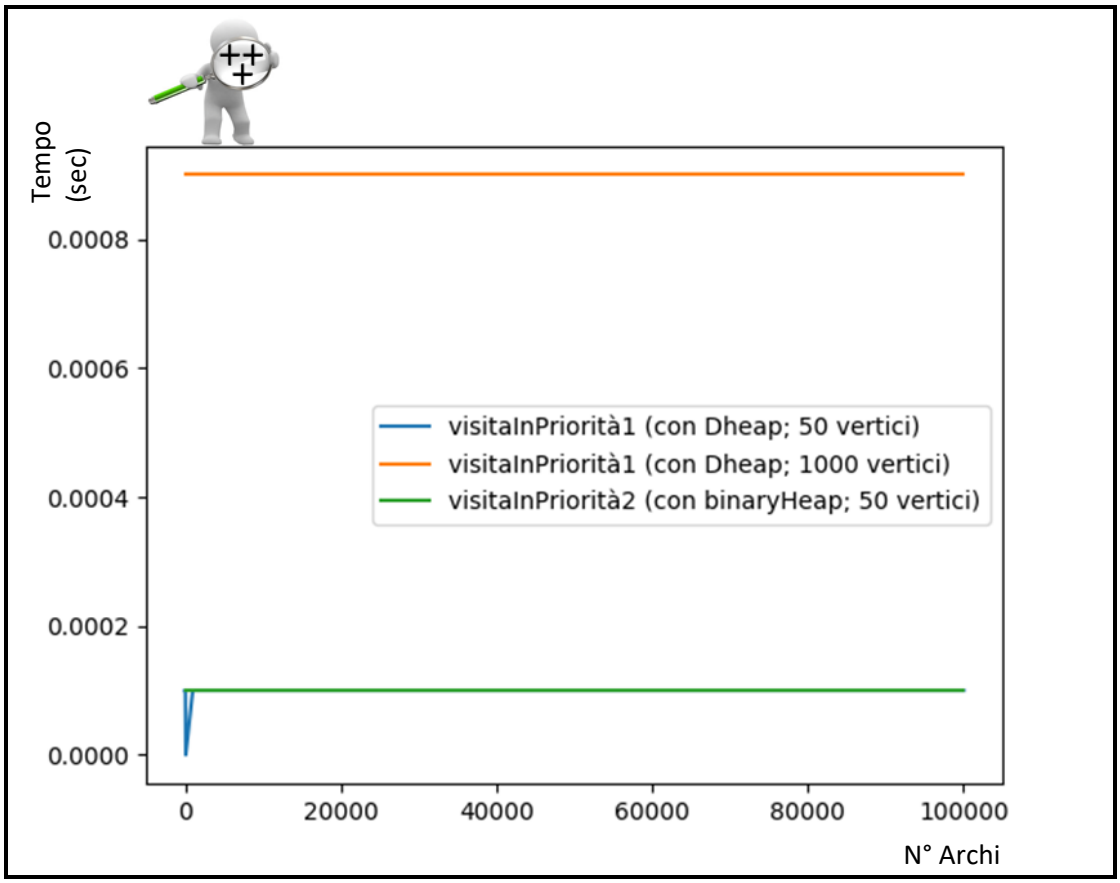
❖ TEST 2: Numero di vertici uguale a 50, 100, 1000 e numero di archi crescente

TEST 2 "visitaInPriorità"	TEMPI DI ESECUZIONE AL VARIARE DELL'INPUT (in sec)						
CODE CON PRIORITA' (insieme F)	N° DI VERTICI	N° DI ARCHI					
		10	50	100	1000	10000	100000
visitaInPriorità1 (con Dheap; 50 vertici)	50	0,0001	0,0001	0,0000	0,0001	0,0001	0,0001
visitaInPriorità1 (con Dheap; 100 vertici)	100	0,0001	0,0001	0,0002	0,0009	0,0087	0,0851
visitaInPriorità1 (con Dheap; 1000 vertici)	1000	0,0009	0,0009	0,0009	0,0009	0,0009	0,0009
visitaInPriorità2 (con binaryHeap; 50 vertici)	50	0,0001	0,0001	0,0001	0,0001	0,0001	0,0001
visitaInPriorità2 (con binaryHeap; 100 vertici)	100	0,0000	0,0001	0,0001	0,0009	0,0090	0,0874
visitaInPriorità2 (con binaryHeap; 1000 vertici)	1000	0,0009	0,0009	0,0009	0,0009	0,0009	0,0009
visitaInPriorità3 (con binomialHeap; 50 vertici)	50	0,0016	0,0015	0,0015	0,0015	0,0017	0,0015
visitaInPriorità3 (con binomialHeap; 100 vertici)	100	0,0005	0,0015	0,0028	0,0282	0,2943	2,7994
visitaInPriorità3 (con binomialHeap; 1000 vertici)	1000	0,0282	0,0281	0,0275	0,0291	0,0562	0,0422



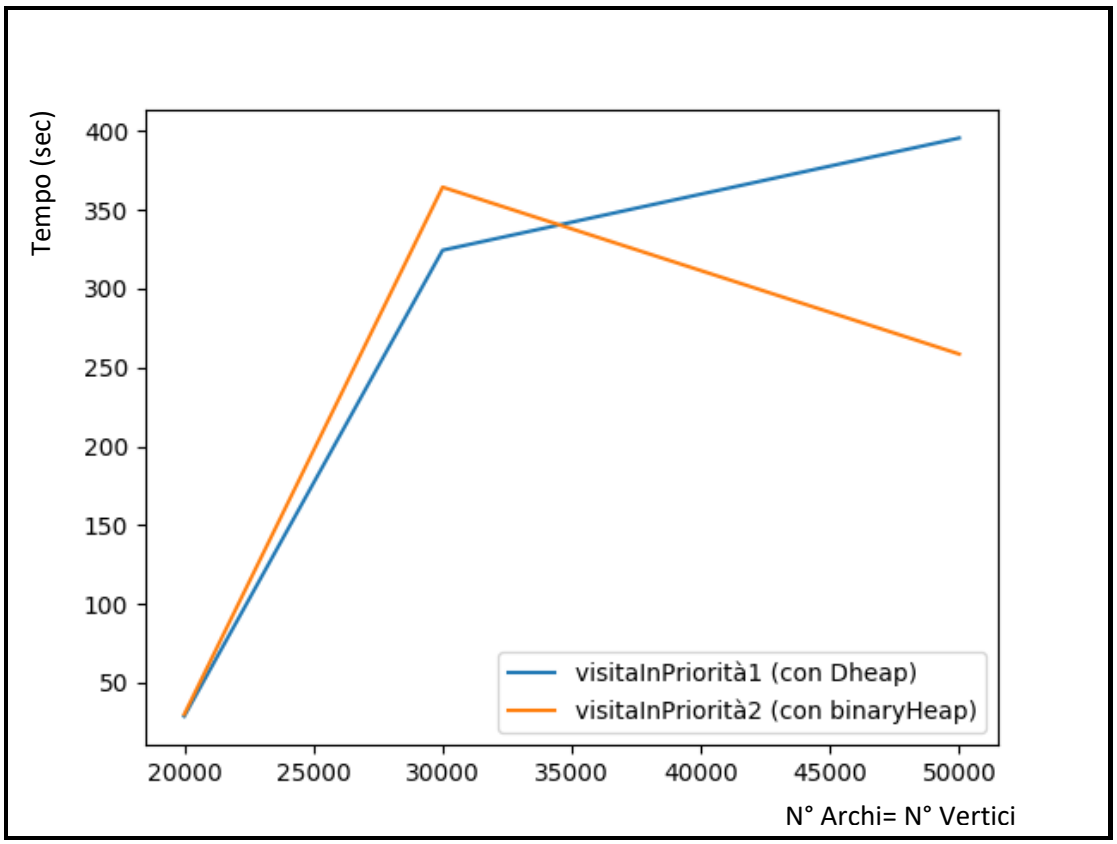

Tempo (sec)
0.08
0.06
0.04
0.02
0.00
visitaInPriorità1 (con Dheap; 50 vertici)
visitaInPriorità1 (con Dheap; 100 vertici)
visitaInPriorità1 (con Dheap; 1000 vertici)
visitaInPriorità2 (con binaryHeap; 50 vertici)
visitaInPriorità2 (con binaryHeap; 100 vertici)
visitaInPriorità2 (con binaryHeap; 1000 vertici)
visitaInPriorità3 (con binomialHeap; 50 vertici)
visitaInPriorità3 (con binomialHeap; 1000 vertici)
0
20000
40000
60000
80000
100000
N° Archi


Tempo (sec)
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
visitaInPriorità1 (con Dheap; 50 vertici)
visitaInPriorità1 (con Dheap; 1000 vertici)
visitaInPriorità2 (con binaryHeap; 50 vertici)
visitaInPriorità2 (con binaryHeap; 1000 vertici)
visitaInPriorità3 (con binomialHeap; 50 vertici)
0
20000
40000
60000
80000
100000
N° Archi


Tempo
(sec)
0.0008
0.0006
0.0004
0.0002
0.0000
visitaInPriorità1 (con Dheap; 50 vertici)
visitaInPriorità1 (con Dheap; 1000 vertici)
visitaInPriorità2 (con binaryHeap; 50 vertici)
0
20000
40000
60000
80000
100000
N° Archi

❖ TEST 3: Numero di archi uguale a quello dei vertici con “BinomialHeap” escluso

TEST 3 "visitaInPriorità"	TEMPI DI ESECUZIONE AL VARIARE DELL'INPUT (in sec)		
CODE CON PRIORITA' - binomialHeap escluso (insieme F)	INPUT: N° DI ARCHI = N° DI VERTICI		
	20000	30000	50000
visitaInPriorità1 (con Dheap)	28,7875	324,4205	395,6041
visitaInPriorità2 (con binaryHeap)	29,9705	364,4492	258,5500



COMMENTI DEI RISULTATI:

Dai test effettuati si può osservare che l'algoritmo “visitaInPriorita” con coda con priorità “BinomialHeap” è il meno performante.
Tra i restanti due algoritmi (con coda con priorità “Dheap” e “BinaryHeap”) si nota, come nel test effettuato con istanze di input piccole (numero di nodi e archi in input), che “visitaInPriorita” con coda “Dheap” risulta più veloce, ma al crescere della dimensione dell’input è il “BinaryHeap” ad essere più veloce (vedere “TEST 3: Numero di archi uguale a quello dei vertici con “BinomialHeap” escluso”).

FIRMA:

Gabriele Anselmi
Michele Terenzi