ALGORITMI E STRUTTURE DATI
RELAZIONE DI LABORATORIO

Composizione del gruppo:
- Ghione Alessio
- Giorgi Jessica
- Placenti Laura

ESERCIZIO 1
QUICK SORT

La scelta dell'elemento pivot per l'algoritmo di QuickSort può essere effettuata in vari modi, al fine di ottimizzare le prestazioni e i tempi di esecuzione. Le più comuni utilizzano il primo elemento, l'ultimo o quello centrale dell'array da ordinare. Esistono anche versioni in cui l'elemento perno viene scelto tramite una funzione random.

***** STRINGHE *****

FIRST ELEMENT	LAST ELEMENT	MIDDLE ELEMENT	RANDOM
18.112688	20.784760	19.122868	17.476566
19.534552	20.447556	18.977156	21.160969
17.950291	20.452377	17.323908	17.405613
18.036877	20.378706	17.295279	18.951864
19.938805	18.880634	17.336107	18.130659
19.499372	18.645063	17.399372	18.977468
19.544744	20.790125	17.951332	18.780218
17.938444	20.427820	17.528376	17.641804
17.886415	18.712263	17.351112	17.288006
18.101875	20.442553	17.308628	19.177841

- **FIRST ELEMENT** mean: 18.65441
- **LAST ELEMENT** mean: 19.99619
- **MIDDLE ELEMENT** mean: 17.75941
- **RANDOM ELEMENT** mean: 18.4991

***** INTERI *****

FIRST ELEMENT	LAST ELEMENT	MIDDLE ELEMENT	RANDOM
13.656793	12.952303	11.525797	11.912206
13.839086	12.946976	12.877149	13.443836
13.908467	14.421749	12.844172	13.058432
12.525144	12.977984	12.844172	12.484239
12.446080	12.867813	12.916329	11.672873
12.476605	14.346616	12.916329	11.716152
13.874069	12.815818	11.519870	11.976317
13.885532	12.887404	11.549896	13.141059
12.490275	14.562078	11.536001	12.006298
12.513278	14.431750	12.858062	11.789536

- **FIRST ELEMENT** mean: 13.16153
- **LAST ELEMENT** mean: 13.52105
- **MIDDLE ELEMENT** mean: 12.33878
- **RANDOM ELEMENT** mean: 12.32009

***** FLOAT *****

FIRST ELEMENT	LAST ELEMENT	MIDDLE ELEMENT	RANDOM
14.248706	14.769923	13.396974	12.221505
13.878471	14.809275	13.458639	12.413009
13.883660	13.522076	13.422063	12.468021
13.924189	13.507707	13.406198	13.638110
13.923907	14.766398	12.278106	12.223482
12.679525	13.498059	12.343132	13.271958
13.876578	14.766695	12.263491	13.309401
12.631569	13.484761	12.303925	12.100323
13.907083	14.782839	13.415822	13.506286
12.712106	13.488884	13.395634	13.436866

- **FIRST ELEMENT** mean: 13.56658
- **LAST ELEMENT** mean: 14.13966
- **MIDDLE ELEMENT** mean: 12.9684
- **RANDOM ELEMENT** mean: 12.8589

PIVOT	MEAN
First element	15.12751
Last element	15.88563
Middle element	14.35553
Random element	14.55936

L'algoritmo da noi implementato utilizza l'elemento centrale come pivot in quanto impiega, in media, il minor tempo.

BINARY INSERTION SORT

RECORDS	SEC	SEC 2nd TRIAL
500.000	37.061928	
750.000	93.790672	100.290123
850.000	128.507706	130.006317
1.000.000	192.467041	
1.250.000	370.171997	
1.500.000	546.127441	544.230164
1.500.250	576.297180	551.871216
1.550.000	619.192078	

L'algoritmo di BinaryInsertionSort impiega più di 12 minuti ad ordinare 20.000.000 di record, viene quindi riportato un fallimento dell'operazione.
L'esecuzione viene limitata a 1.550.000 record in quanto 10,33 minuti [619.192078 sec] è un tempo ragionevole.
Mediamente l'algoritmo di QuickSort riordina i campi del record in un tempo minore rispetto al BinaryInsertionSort. Questo è dovuto al fatto che il BinaryInsertionSort richiama la procedura di BinarySearch per trovare la posizione e, una volta restituita, vengono eseguiti gli scambi tra i valori per consentire l'ordinamento.
Anche confrontando il BinaryInsertionSort con il QuickSort che utilizza il pivot peggiore (last element) i tempi rimangono comunque più lunghi.

ESERCIZIO 2

MAX HEIGHT	TEMPO TRIAL 1	TEMPO TRIAL 2	TEMPO TRIAL 3	MEAN TRIALS
8	72.437508	71.859375	66.390625	70.22917
10	11.92187	11.531250	11.328125	11.59375
15	1.953125	1.812500	1.984375	1.916667
20	1.687500	1.703125	1.843750	1.744792
25	1.734375	1.578125	1.937500	1.750000
30	1.671875	2.093750	1.687500	1.817708
45	1.656250	1.984375	2.031250	1.890625
50	1.593750	1.765625	1.609375	1.65625
60	1.734375	1.906250	1.796875	1.8125
70	1.671875	1.921875	1.750000	1.78125
75	1.734375	1.656250	1.765625	1.71875
85	1.859375	1.906250	2.156250	1.973958
90	1.875000	1.703125	1.750000	1.776042
100	1.906250	1.812500	1.687500	1.802083
125	1.750000	1.937500	1.843750	1.84375
150	1.718750	2.015625	1.828125	1.854167

A seguito dei test effettuati, MAX_HEIGHT viene impostato con il valore 50 per ottenere il tempo medio (tra le prove effettuate) migliore di esecuzione (1.65625). Si nota anche che superando il MAX_HEIGHT = 15 si è intorno ai 1.6 / 2 secondi. Si noti che i valori dei tempi ottenuti sono diversi a causa della funzione random all'interno del codice per il calcolo di max level.
I test vengono conclusi con un valore di MAX_HEIGHT non inferiore a 8 in quanto i tempi calcolati con, ad esempio, MAX_HEIGHT = 5 sono superiori ai 10 minuti.

ESERCIZIO 3

L'heap è implementato come un arraylist. Viene inoltre utilizzata una hashmap per memorizzare l'indice corrispondente ad ogni elemento.

ESERCIZIO 4

Il nostro grafo è implementato come una hashmap di hashmap: la prima hashmap contiene, nel campo key, i nodi, mentre in item contiene una seconda hashmap che, a sua volta, nel campo key contiene i nodi, mentre in item contiene il peso dell'arco (tra il nodo nella prima hash e quello nella seconda).
All'interno della class Edge si può trovare la definizione dell'oggetto Edge. Essa contiene tre campi: due nodi e una label, che corrisponde al peso dell'arco tra il node1 e il node2.
Nella classe WeightedNode, invece, si trova la definizione dell'oggetto omonimo, che comprende un nodo e il suo peso. Contiene, inoltre, il predecessore del nodo considerato. Il peso corrisponde alla somma dei pesi (degli archi) che collegano la source con il nodo in questione. È impostato ad infinito se si tratta di un nodo non ancora visitato, quindi di cui non se ne conosce il predecessore. Questo oggetto viene utilizzato all'interno del metodo Dijkstra.
Il metodo Dijkstra viene implementato all'interno della classe Main_ex4. Per la sua implementazione, ci siamo serviti di:
- un grafo shortestPath dichiarato all'interno del metodo che sarà il grafo (che verrà restituito a fine esecuzione del metodo) dei cammini minimi a partire dalla source
- una priorityQueue in cui in ogni posizione viene memorizzato un oggetto WeightedNode
- una hashmap weightedNodeMap che ritorna il nodo WeightedNode nell'heap per ritrovare l'indice di posizione
- un arraylist nodesList che contiene tutti i nodi del grafo
- un arraylist adjList che contiene la lista di adiacenza di un nodo

Dijkstra ritorna il grafo shortestPath del quale ne viene fatta una visita in profondità dal metodo BFS.