TABELLE COMPLESSITA’

OPERAZIONI SU LISTE

- Operazioni sulle liste doppiamente collegate, circolari e con sentinella:

Funzione	CASO MIGLIORE	CASO PEGGIORE
set	1 -> (min(n,pos))	n -> (min(n,pos))
add	1 -> dipende da pos	n
addBack	1	1
addFront	1	1
RemovePos	1->(min(n,pos))	n->(min(n,pos))
get	1	n
isEmpty	1	1
size	n	n
emptyList	1	1

- Operazioni sulle liste collegate semplicemente:

Funzione	CASO MIGLIORE	CASO PEGGIORE
set	1 -> (min(n,pos))	n -> (min(n,pos))
add	1 -> dipende da pos	n
addBack	n	n
addFront	1	1
RemovePos	1->(min(n,pos))	n
get	1	n
isEmpty	1	1
size	n	n
emptyList	1	1

- Operazioni su array dinamico + size e maxsize:

!NB: rispetto alle liste semplici cambiano le operazioni che richiedono un ridimensionamento dell’array e quelle che posso semplificare tramite l’ accesso diretto

Funzione	CASO MIGLIORE	CASO PEGGIORE
set	1 -> (min(n,pos))	1 -> (min(n,pos))
add	1 -> dipende da pos	n
addBack	1	n
addFront	n -> shift elem	n
RemovePos	1 -> tolgo dal fondo e non ridimensiono	n

get	1	1
isEmpty	1	1
size	1	1 -> campo size
emptyList	1	1

OPERAZIONI SU STACK

- Operazioni su liste doppiamente collegate, circolari e con sentinella:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
push	1	1
pop	1	1
top	1	1
emptyStack	1	1
isEmpty	1	1

- Operazioni su liste collegate semplicemente:

!NB:Le operazioni push,pop e top vengono effettuate in testa

Funzioni	CASO MIGLIORE	CASO PEGGIORE
push	1	1
pop	1	1
top	1	1
emptyStack	1	1
isEmpty	1	1

- Operazioni su array dinamico + size e maxsize:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
push	1	N-> se devo ridimensionare
pop	1	1
top	1	1
emptyStack	1	1
isEmpty	1	1

OPERAZIONI SU CODE

- Operazioni su liste doppiamente collegate, circolari e con sentinella:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
enqueue	1	1
dequeue	1	1
first	1	1
emptyQueue	1	1
isEmpy	1	1

- Operazioni su liste collegate semplicemente (inserisco in coda e prendo in testa):

Funzioni	CASO MIGLIORE	CASO PEGGIORE
enqueue	n	n
dequeue	1	1
first	1	1
emptyQueue	1	1
isEmpy	1	1

- Operazioni su array dinamico + size e maxsize (o rendo efficiente la enqueue e inefficiente la dequeue o viceversa, suppongo di mettere in pos size e prendere da pos 0):

Funzioni	CASO MIGLIORE	CASO PEGGIORE
enqueue	1	n-> se devo ridimenionare
dequeue	n->shifto gli elementi a sinistra	n
first	1	1
emptyQueue	1	1
isEmpy	1	1

OPERAZIONI SU INSIEMI

- Operazioni su liste doppiamente collegate, circolari e con sentinella:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
emptySet	1	1
insertElem	1->controllo se l' elemento è già presente e se lo trovo subito ho il caso migliore	n

deleteElem	1	n
setUnion n = dim primo, m = dim secondo	max(n,m,n*m) 0	max(n,m.n*m)
setIntersection	n*m	n*m
isEmpty	1	1
size	n	n
member	1	n

- Operazioni su liste collegate semplicemente:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
emptySet	1	1
insertElem	1->controllo se l' elemento è già presente e se lo trovo subito ho il caso migliore	n
deleteElem	1	n
setUnion n = dim primo, m = dim secondo	max(n,m,n*m)	max(n,m.n*m)
setIntersection	n*m	n*m
isEmpty	1	1
size	n	n
member	1	n

- Operazioni su array dinamico + size, maxsize e ridimensionamento:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
emptySet	1	1
insertElem	1->controllo se l' elemento è già presente e se lo trovo subito ho il caso migliore	n
deleteElem	1->se è il primo sposto l' ultimo al suo posto e decremento size senza shiftare gli elementi	n
setUnion	max(n,m,n*m)	max(n,m.n*m)

n = dim primo, m = dim secondo		
setIntersection	n*m	n*m
isEmpty	1	1
size	1	1
member	1	n

- Operazioni su bit vector (i 2 vettori devono avere la stessa dimensione):

Funzioni	CASO MIGLIORE	CASO PEGGIORE
emptySet	n	n
insertElem	1	1
deleteElem	1	1
setUnion	n	n
setIntersection	n	n
isEmpty	1->se V[0] == 1 il vettore non è vuoto	n
size	n	n
member	1	1

- Operazioni su array dinamico ordinato + size, maxsize e senza ridimensionamento:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
emptySet	1	1
insertElem	1->lo trovo come primo o ultimo oppure è > dell' ultimo	n->log n per la ricerca binaria ma poi shift a destra per mantenere ordinato (n)
deleteElem	1->se è < del primo e > dell' ultimo	n
setUnion	n+m->sfrutto il fatto che sono ordinati e uso una tecnica simile a "merge" di mergesort	n+m
setIntersection	1->intersezione vuota	n+m
isEmpty	1	1
size	1	1
member	1	Log n->ricerca binaria

FUNZIONI DI ORDINAMENTO:

Funzioni	CASO MIGLIORE	CASO PEGIORE
Ricerca binaria	1	Log n
Merge sort	Log n	Log n
Quick sort	n log n	sommatoria -> n^2
Selection sort	sommatoria->n^2	sommatoria->n^2
Inserction sort	n	n^2
Bubble sort	n	n^2

- Dizionario implementato come BST on altezza h e num nodi n:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
search	1	h
insert	1	h
delete	1	h

TABELLE DI HASH:

- Ad accesso diretto:

Funzioni	COMPLESSITA’
insert	1
delete	1
createEmpty	m ->devo assegnare emptyElem a tutte le m celle della tabella
search	1
isEmpty	1

- Con liste di collisione (se rispettano le buone proprietà):

Funzioni	CASO MIGLIORE	CASO PEGGIORE
insert	1	1 + (n/m)
delete	1	
createEmpty	m ->devo assegnare emptyElem a tutte le m celle della tabella	
search	1	
isEmpty	1	

GRAFI:

- Liste di adiacenza con vertici memorizzati in un array (non orientato):

Per trovare un elemento nella lista dei vertici ci impiego massimo Theta (1)

Funzioni	CASO MIGLIORE	CASO PEGGIORE
addVertex	1	n ->se devo riallocare
addEdge	1	(min(grado(u), grado(v)))
removeVertex	grado(v)	m*
removeEdege	1	(grado(u) + grado(v))
incidentEdges	grado(v)	grado(v)
degree	grado(v)	grado(v)
AreAdjacent	1	min((grado(u), grado(v))

*Il testo semplifica in m ma in realtà la complessità nel caso peggiore è data dalla somma del numero di archi incidenti negli u, adiacenti a v sommo anche il numero di nodi adiacenti a v che coincide con il numero degli archi incidenti in v e al più ottengo (in un grafo non orientato) il doppio del numero m di tutti gli archi (non di più)

- Liste di adiacenza con vertici memorizzati in una lista (grafo non orientato):

Per trovare un elemento nella lista dei vertici ci impiego massimo Theta(n)

Funzioni	CASO MIGLIORE	CASO PEGGIORE
addVertex	n	n
addEdge	n	n
removeVertex		m+n
removeEdge		grado(u) + grado(v) +n = n
incidentEdges		grado v + n = n
degree		grado(v) + n = n
areAdjacent		min(grado(u),grado(v))+n = n

- Matrici di adiacenza (grafo non orientato):

Funzioni	CASO MIGLIORE	CASO PEGGIORE
addVertex	1	n^2
addEdge	1	1
removeVertex		n^2
removeEdge	1	1
incidentEdges	n	n
degree	n	n
areAdjacent	1	1

CODE CON PRIORITA':

- Implementate con heap binario:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
deleteMax	Log n	Log n
insertElem	Log n	Log n
findMax	1	1

- Implementate con lista semplice ordinata:

Funzioni	CASO MIGLIORE	CASO PEGGIORE
deleteMax	1	1
insertElem		n
findMax	1	1