ASD

alessandro.manfucci@studenti.unitn.it

1 Pseudo Linguaggio

- a = b
- a $\leftrightarrow$ b $\equiv$ tmp = a; a = b; b = tmp
- T [ ] A = new T [1…n]
- T [ ][ ] B = new T [1…n][1…m]
- int, float, boolean
- and, or, not
- ==, $\neq$, $\leq$, $\geq$
- +, −, $\cdot$, /, $\lfloor x \rfloor$, $\lceil x \rceil$, log, x^2, mod, …
- iif(condizione, v_1, v_2)
- if condizione then istruzione
- if condizione then istruzione1 else istruzione2
- while condizione do istruzione
- foreach elemento $\in$ insieme do istruzione
- return
- % commento
- for indice = estremoInf to estremoSup do istruzione

- int indice = estremoInf
  while indice $\leq$ estremoSup do
  | istruzione
  | indice = indice + 1
  -

- for indice = estremoSup downto estremoInf do istruzione

- int indice = estremoSup
  while indice $\geq$ estremoInf do
  | istruzione
  | indice = indice − 1
  -

RETTANGOLO
int lunghezza
int altezza

- rettangolo r = new rettangolo
- r.altezza = 10
- delete r
- r = nil

2 Strutture dati

2.1 Pila

```
STACK
---
% Restituisce true se la pila è vuota
boolean isEmpty()
% Inserisce v in cima alla pila
push(ITEM v)
% Rimuove l'elemento in cima alla pila e lo restituisce
ITEM pop()
% Legge l'elemento in cima alla pila
ITEM top()
```

2.2 Coda

```
QUEUE
---
% Restituisce true se la coda è vuota
boolean isEmpty()
% Inserisce v in fondo alla coda
enqueue(ITEM v)
% Estrae l'elemento in testa alla coda e lo restituisce al chiamante
ITEM dequeue()
% Legge l'elemento in testa alla coda
ITEM top()
```

2.3 Sequenza

```
SEQUENCE
---
% Restituisce true se la sequenza è vuota
boolean isEmpty()
% Restituisce true se p = pos0 o se p = posn+1
boolean finished(POS p)
% Restituisce la posizione del primo elemento
POS head()
% Restituisce la posizione dell'ultimo elemento
POS tail()
% Restituisce la posizione dell'elemento che segue p
POS next(POS p)
% Restituisce la posizione dell'elemento che precede p
POS prev(POS p)
% Inserisce l'elemento v di tipo ITEM nella posizione p e restituisce
% la posizione del nuovo elemento, che diviene il predecessore di p
POS insert(POS p, ITEM v)
% Rimuove l'elemento contenuto nella posizione p e restituisce la posizione
% del successore di p, che diviene il successore del predecessore di p
POS remove(POS p)
% Legge l'elemento di tipo ITEM contenuto nella posizione p
ITEM read(POS p)
% Scrive l'elemento v di tipo ITEM nella posizione p
write(POS p, ITEM v)
```

2.4 Insieme

```
SET
---
% Restituisce la cardinalità dell'insieme
int size()
% Restituisce true se x è contenuto nell'insieme
boolean contains(ITEM x)
% Inserisce x nell'insieme, se non è giù presente
insert(ITEM x)
% Rimuove x dall'insieme, se è presente
remove(ITEM x)
% Restituisce un nuovo insieme che è l'unione di A e B
Set union(Set A, Set B)
% Restituisce un nuovo insieme che è l'intersezione di A e B
Set intersection(Set A, Set B)
% Restituisce un nuovo insieme che è la differenza di A e B
Set difference(Set A, Set B)
```

Sia n il numero di elementi nell'insieme e m la capacità dell'insieme.

Implementazione	contains()	insert()	remove()	min()	foreach()
Array booleano	O(1)	O(1)	O(1)	O(m)	O(m)
Lista non ordinata	O(n)	O(n)	O(n)	O(n)	O(n)
Lista ordinata	O(n)	O(n)	O(n)	O(1)	O(n)
Array ordinato	O(log n)	O(n)	O(n)	O(1)	O(n)
RB Tree	O(log n)	O(log n)	O(log n)	O(log n)	O(n)
Hash Table	O(1)	O(1)	O(1)	O(m)	O(m)

2.5 Dizionario

```
DICTIONARY
---
% Restituisce il valore associato alla chiave k se presente, nil altrimenti
ITEM lookup(ITEM k)
% Associa il valore v alla chiave k
insert(ITEM k, ITEM v)
% Rimuove l'associazione della chiave k
remove(ITEM k)
```

Implementazione	lookup()	insert()	remove()	foreach()
Array non ordinato	O(n)	O(1),O(n)	O(1)	O(n)
Array ordinato	O(log n)	O(n)	O(n)	O(n)
Lista non ordinata	O(n)	O(1),O(n)	O(n)	O(n)
RB Tree	O(log n)	O(log n)	O(log n)	O(n)
Hash Table	O(1)	O(1)	O(1)	O(n)

2.6 Priority Queue

```
MIN-PRIORITYQUEUE
---
% Crea una coda a priorità con capacità n
PRIORITYQUEUE PriorityQueue(int n)
```

```
% Restituisce true se la coda a priorità è vuota
boolean isEmpty()
% Restituisce l'elemento minimo di una coda a priorità non vuota
ITEM min()
% Rimuove e restituisce l'elemento minimo di una coda a priorità non vuota
deleteMin()
% Inserisce l'elemento x con priorità p nella coda a priorità e restituisce
% un oggetto PRIORITYITEM che identifica x all'interno della coda
PRIORITYITEM insert(ITEM x, int p)
% Diminuisce la priorità dell'oggetto identificato da y portandola a p
decrease(PRIORITYITEM y, int p)
```

Implementazione	min()	deleteMin()	insert()	decrease()
Array/Lista non ordinato	O(n)	O(n)	O(n)	O(n)
Array ordinato	O(1)	O(n)	O(n)	O(log n)
Lista ordinata	O(1)	O(1)	O(n)	O(n)
RB Tree	O(log n)	O(log n)	O(log n)	O(log n)
Heap Tree	O(1)	O(log n)	O(log n)	O(log n)

2.7 Albero binario

```
TREE
---
% Costituisce un nuovo nodo, contenente v, senza figli o genitori
Tree(ITEM v)
% Legge il valore memorizzato nel nodo
ITEM read()
% Modifica il valore memorizzato nel nodo
write(ITEM v)
% Restituisce il padre, oppure nil se questo è il nodo radice
TREE parent()
% Restituisce il figlio sinistro di questo nodo, oppure nil se è assente
TREE left()
% Restituisce il figlio destro di questo nodo, oppure nil se è assente
TREE right()
% Inserisce il sottoalbero radicato t come figlio sinistro di questo nodo
insertLeft(TREE t)
% Inserisce il sottoalbero radicato t come figlio destro di questo nodo
insertRight(TREE t)
% Distrugge ricorsivamente il figlio sinistro di questo nodo (in O(n) con punt.)
deleteLeft()
% Distrugge ricorsivamente il figlio destro di questo nodo (in O(n) con punt.)
deleteRight()
```

2.8 Albero generico

```
TREE
---
% Costituisce un nuovo nodo, contenente v, senza figli o genitori
Tree(ITEM v)
% Legge il valore memorizzato nel nodo
ITEM read()
% Modifica il valore memorizzato nel nodo
```

```
write(ITEM v)
% Restituisce il padre, oppure nil se questo è il nodo radice
TREE parent()
% Restituisce il primo figlio da sinistra, oppure nil se questo nodo
% è una foglia
TREE leftmostChild()
% Restituisce il primo fratello sulla destra, oppure nil se è assente
TREE rightSibling()
% Inserisce il sottoalbero t come primo figlio di questo nodo
insertChild(TREE t)
% Inserisce il sottoalbero t come prossimo fratello di questo nodo
insertSibling(TREE t)
% Distrugge l'albero radicato identificato dal primo figlio
deleteChild()
% Distrugge l'albero radicato identificato dal prossimo fratello
deleteSibling()
% Distrugge l'albero radicato identificato dal nodo
delete(TREE t)
```

2.9 Grafo

```
GRAPH
---
% Crea un nuovo grafo
GRAPH Graph()
% Restituisce l'insieme di tutti i vertici
SET V()
% Restituisce il numero di nodi
int size()
% Restituisce l'insieme dei nodi adiacenti ad u
SET adj(NODE u)
% Aggiunge un nodo u al grafo
insertNode(NODE u)
% Aggiunge l'arco (u, v) al grafo
insertEdge(NODE u, NODE v)
% Rimuove il nodo u dal grafo (in O(n) con vettori/liste adiacenza)
removeNode(NODE u)
% Rimuove l'arco (u, v) dal grafo
removeEdge(NODE u, NODE v)
```