Tipi di dato di base

Liste (o “successioni finite”)

Algebra eterogenea in cui le sort sono *List, N, Bool, Elem* (dove *Elem* è una sort non ulteriormente specificata) e le operazioni, con loro arietà, sono

- *emptyList:→List*
- *set: N x Elem x List → List*
- *add: N x Elem x List → List*
- *addBack: Elem x List → List*
- *addFront: Elem x List → List*
- *removePos: N x List → List*
- *get: N x List → Elem*
- *isEmpty: List → Bool*
- *size: List → N*

Liste (o “successioni finite”)

emptyList: → List
/ emptyList è la lista vuota */*

set: N x Elem x List → List
/ set(pos, e, l) modifica l'elemento in posizione pos di l, sostituendolo con e */*

add: N x Elem x List → List
/ add(pos, e, l) inserisce l'elemento e nella lista l in posizione pos, shiftando a destra gli altri elementi */*

addBack: Elem x List → List
/ addBack(e, l) inserisce l'elemento e alla fine della lista l */*

Liste (o “successioni finite”)

addFront: Elem x List → List
/ addFront(e, l) inserisce l'elemento e all'inizio della lista l */*

removePos: N x List → List
/ removePos(pos, l) cancella l'elemento in posizione pos dalla lista l */*

get: N x List →Elem
/ get(pos, l) restituisce l'elemento in posizione pos nella lista l */*

isEmpty: List → Bool
/ isEmpty(l) è vero se la lista è vuota, falso altrimenti */*

size: List → N
/ size(l) è il numero di elementi nella lista */*

Riflessione #1

Le liste (“List”) sono un TDD, ma indichiamo con la parola “lista” anche la famiglia di strutture dati che si ottengono collegando struct mediante puntatori: liste semplici, liste doppiamente collegate, liste doppiamente collegate circolari, liste doppiamente collegate circolari con sentinella. Un'altra fonte di ambiguità a cui prestare attenzione...

Riflessione #2

Consideriamo

addFront: Elem x List → List

Vuole dire che l'inserimento MODIFICA la lista a cui si applica, oppure che LA LASCIA INALTERATA e restituisce una lista nuova, uguale a quella su cui abbiamo chiamato addFront nella quale però abbiamo aggiunto il nuovo elemento in testa?

Voting time

- Supponiamo di dover votare rispetto alla domanda posta prima
- Chi vota per MODIFICA ha ragione, se volevamo implementare un'**operazione “stateful”** (ovvero, un'operazione che modifica gli argomenti a cui è applicata == un'operazione con side effect)
- Chi vota per LASCIA INALTERATA LA LISTA ARGOMENTO E RESTITUISCE UNA LISTA NUOVA ha ragione, se volevamo implementare una **funzione**

  Come abbiamo constatato altre volte, spesso non esiste una soluzione giusta o sbagliata in assoluto.

Strutture dati per implementare liste

- Strutture dati collegate
- Array dinamici con size, maxsize e aggiunta/rimozione di spazio quando necessario

- Il codice reso disponibile dai docenti è parte integrante del materiale didattico

Riflessione #3

- Quando usiamo i vector in C++, usiamo qualcosa di molto simile alle liste implementate con array dinamico, size e maxsize.
- Anche nei vector è necessario riallocare quando lo spazio nell'array non basta più. Le operazioni di inserimento in un vector C++, ad esempio la push_back, nel peggiore dei casi non hanno costo costante, esattamente come la nostra addBack!
- http://codefreakr.com/how-is-c-stl-implemented-internally/

Pile (Stack; slide dal libro di testo)

tipo `Pila`:
dati:
una sequenza S di n elementi.

operazioni:

`isEmpty`$() \rightarrow result$
restituisce `true` se S è vuota, e `false` altrimenti.

`push`$(elem\ e)$
aggiunge e come ultimo elemento di S.

`pop`$() \rightarrow elem$
toglie da S l'ultimo elemento e lo restituisce.

`top`$() \rightarrow elem$
restituisce l'ultimo elemento di S (senza toglierlo da S).

Code (Queue; slide dal libro di testo)

tipo `Coda`:
dati:
una sequenza S di n elementi.

operazioni:

`isEmpty`$() \rightarrow result$
restituisce `true` se S è vuota, e `false` altrimenti.

`enqueue`$(elem\ e)$
aggiunge e come ultimo elemento di S.

`dequeue`$() \rightarrow elem$
toglie da S il primo elemento e lo restituisce.

`first`$() \rightarrow elem$
restituisce il primo elemento di S (senza toglierlo da S).

Implementazioni di pile e code

Il codice messo a disposizione dai docenti è parte integrante del materiale didattico su pile e code

Insiemi

- Insiemi: un aggregato di elementi a valori omogenei (provenienti dallo stesso dominio) e distinti (senza ripetizioni).
- In informatica: discreti e finiti

Insiemi: TDD

emptySet: → Set

insertElem: Elem x Set → Set

deleteElem: Elem x Set → Set

setUnion: Set x Set → Set

setIntersection: Set x Set → Set

setDifference: Set x Set → Set

isEmpty: Set → Bool

isSubset: Set x Set → Bool

size: Set → N

member: Elem xSet → Bool

/ Il significato delle operazioni è quello ovvio in campo insiemistico: non lo specifichiamo */*

Strutture dati per implementare insiemi

- Oltre a quelle “ovvie” (liste collegate, liste semplici, array), un'altra struttura dati adatta all'implementazione degli insiemi è il bit vector (anche detto array vector o bitmap)
- Sia v un array di bit di dimensione DIM
- Come implemento l'insieme A, t.c. A è un sottoinsieme di [0, DIM]?
- v[i] == 1 sse i appartiene ad A
- v[i] == 0 sse i non appartiene ad A

Bit vector

In the case where the set values fall within a limited range, we can represent the set using a bit array with a bit position allocated for each potential member. Those members actually in the set store a value of 1 in their corresponding bit; those members not in the set store a value of 0 in their corresponding bit. For example, consider the set of primes between 0 and 15. Figure 9.1 shows the corresponding bit array. To determine if a particular value is prime, we simply check the corresponding bit. This representation scheme is called a bit vector or a bitmap.

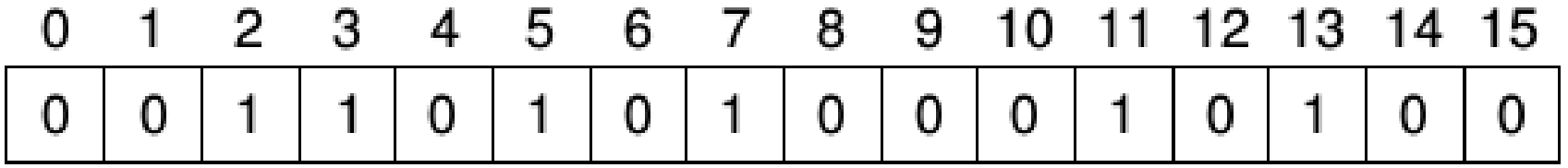

Figure 9.1 The bit array for the set of primes in the range 0 to 15. The bit at position i is set to 1 if and only if i is prime.

Bit vector

If the set fits within a single computer word, then set union, intersection, and difference can be performed by logical bit-wise operations. The union of sets A and B is the bit-wise OR function (whose symbol is | in C ++). The intersection of sets A and B is the bit-wise AND function (whose symbol is & in C ++). For example, if we would like to compute the set of numbers between 0 and 15 that are both prime and odd numbers, we need only compute the expression 0011010100010100 & 0101010101010101.

The set difference A − B can be implemented in C ++ using the expression A& ˜B (˜is the symbol for bit-wise negation). For larger sets that do not fit into a single computer word, the equivalent operations can be performed in turn on the series of words making up the entire bit vector.

Bit vector e operazioni &, ~ ed |

```
unsigned int x1, x2;
x1 = 0;
x2 = ~0;
cout << "0: " << x1 << endl;
cout << "~0: " << x2 << endl;
cout << "~~0: " << ~x2 << endl;

x1 = 2;
x2 = 4;
cout << x1 << " | " <<  x2 << " = " << (x1|x2) << endl;
```

Bit vector e operazioni &, ~ ed |

```
x1 = 2;
x2 = 4;
cout << x1 << " & " << x2 << " = " << (x1&x2) << endl;

x1 = 6;
x2 = 4;
cout << x1 << " | " << x2 << " = " << (x1|x2) << endl;

x1 = 6;
x2 = 4;
cout << x1 << " & " << x2 << " = " << (x1&x2) << endl;
```

Array ordinato

- Un'altra struttura dati che possiamo usare, è un array (con size e maxsize) che manteniamo **ordinato**.
- L'ordinamento rende meno efficienti alcune operazioni, ma più efficiente la ricerca, l'unione e l'intersezione.

Altre implementazioni

Il codice messo a disposizione dai docenti è parte integrante del materiale didattico sugli insiemi