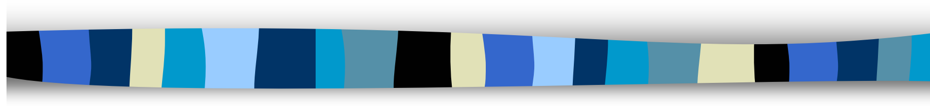
Il TDA Set



Union-Find

Il TDA Set

- Un insieme è un contenitore di oggetti distinti
  - Non esiste un'esplicita nozione di chiavi
  - Non esiste un ordinamento degli elementi
- Se esiste una relazione d'ordine totale sugli elementi di un insieme, possiamo implementare in maniera efficiente le operazioni sugli insiemi

Operazioni sul TDA Set

- Le operazioni che possiamo fare su di un insieme A sono:
  - union(A,B)
    - Rimpiazza A con l'unione di A e B
  - intersect(A,B)
    - Rimpiazza A con l'intersezione di A e B
  - subtract(A,B)
    - Rimpiazza A con la differenza di A e B

Interfaccia Set

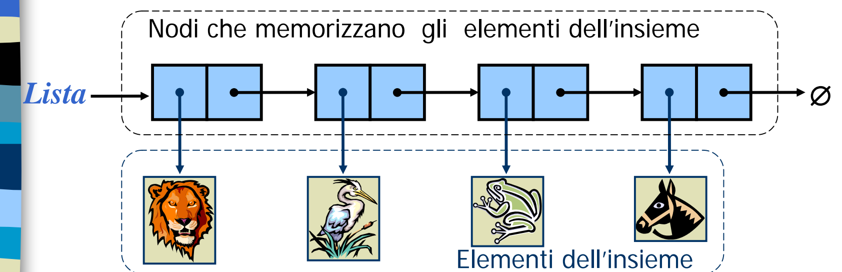
```
public interface Set {
// Restituisce il numero degli elementi nell'insieme
public int size();

// Restituisce true se l'insieme è vuoto
  public boolean isEmpty();

// Rimpiazza this con l'unione di this e B
public Set union(Set B);

// Rimpiazza this con l'intersezione di this e B
public Set intersect(Set B);

// Rimpiazza this con la differenza di this e B
 public Set subtract(Set B);
}
```

Implementazione di Set – 1

- Il modo più semplice per implementare il TDA insieme è quello di memorizzare gli elementi dell'insieme in una sequenza (List, Vector, Sequence)
  - Ad esempio, la classe che implementa Set avrà una variabile istanza di tipo Vector (conterrà gli elementi dell'insieme) ed una di tipo EqualityTester (usata per confrontare elementi dell'insieme)

Implementazione di Set – 2

- Implementando l'interfaccia Set tramite una lista, lo spazio usato è O(n)



Metodi ulteriori – 1

- L'implementazione oltre a contenere i metodi indicati nell'interfaccia Set, potrà contenere i metodi:
  - public boolean contains(Object element)
    - Restituisce true se element appartiene all'insieme
    - Restituisce false se element non appartiene all'insieme

Metodi ulteriori – 2

- public Object insert(Object element)
  - Inserisce element nell'insieme restituendolo
  - Se element è già presente nell'insieme non lo inserisce restituendo null
- public Object remove(Object element)
  - Cancella element dall'insieme restituendolo
  - Restituisce null se element non è presente
- public Object remove()
  - Cancella l'ultimo elemento dell'insieme restituendolo, restituisce null se l'insieme è vuoto

Possibile implementazione

```
public class ListSet implements Set {
   EqualityTester eq;
   List L;

   ListSet() {
        L = new NodeList();
        eq = new DefaultEqualityTester();
   }
 ...
}
```


Complessità

- Indichiamo con n ed m il numero degli elementi degli insiemi A e B, rispettivamente
- La complessità dei metodi A.insert(e), A.contains(e) e A.remove(e) è O(n)
- La complessità dei metodi A.union(B), A.intersect(B) e A.subtract(B) è O(n·m)

Sequenze ordinate

- Se gli elementi che inseriamo negli insiemi soddisfano una relazione di ordine totale, allora possiamo memorizzarli in una sequenza ordinata
- L'implementazione dovrà utilizzare un'istanza di Comparator al posto di EqualityTester
- È possibile usare lo schema della procedura merge usata nell'ordinamenento per *fusione* (MergeSort) per implementare i metodi: union, intersect, subtract
- Come cambia la complessità dei metodi delle slide precedenti?

Esercizi

- Implementare la classe ListSet
- Implementare l'interfaccia Set usando come rappresentazione dell'insieme
  - Il TDA Sequence
  - Il TDA Vector
- Sviluppare la classe OrderedListSet che implementa l'interfaccia Set usando una lista ordinata

Rappresentazione degli elementi – 1

- Invece di rappresentare un elemento di un insieme con la classe Object possiamo farlo attraverso una classe che contiene due campi
  - Object element
    - Rappresenta l'elemento stesso
  - Set set
    - È il riferimento all'insieme che contiene l'elemento
    - Se set è null, allora l'elemento non appartiene ad alcun insieme.

Rappresentazione degli elementi – 2

- Un elemento conosce a quale insieme appartiene
- Alcune operazioni possono essere implementate più efficientemente rispetto alla rappresentazione precedente

Interfaccia Element

```
public interface Element {

    public Set set();

    public Object element();

}
```


Classe SetAwareElement

```
public class SetAwareElement
                implements Element {

  Object element;
  Set set;
...
}
```

Metodi di SetAwareElement

- public SetAwareElement(Object el)
- public SetAwareElement(Set s, Object el)
- public Set set()
- public Object element()
- public Set assignSet(Set s)
- public Object assignElement(Object el)

Esercizi

- Implementare la classe SetAwareElement
- Implementare l'interfaccia Set utilizzando la classe SetAwareElement per rappresentare gli elementi dell'insieme (non ordinato)
  - Sviluppare la classe ListAwareSet

Note su ListAwareSet

- Metodi modificati di ListAwareSet
  - public boolean contains(Element el)
  - public Object insert(Element el)
  - public Object remove(Element el)
- La classe deve implementare anche il metodo protetto checkElement

Metodo checkElement

```
protected SetAwareElement checkElement(Element el)
                    throws InvalidElementException
{
   if(el == null || !(el instanceof SetAwareElement))
     throw new InvalidElementException("Elemento non
   valido");
   return (SetAwareElement) el;
 }
```

Il TDA Partition

- Una partizione è una collezione di insiemi $S_1,\dots,S_k$ a due a due disgiunti
  - $S_i \cap S_j = \emptyset$ per ogni $i \neq j$
- TDA Partition supporta i seguenti metodi:
  - makeSet(x)
    - Crea l'insieme contenente il solo elemento x
  - union(A, B)
    - Unisce gli insiemi A e B distruggendo B e restituendo A
  - find(x)
    - Restituisce l'insieme che contiene l'elemento x

Interfaccia Partition – 1

```
public interface Partition {
// Restituisce il numero degli insiemi nella partizione
public int size();

// Restituisce true se la partizione è vuota
  public boolean isEmpty();

// Restituisce l'insieme contenente il solo elemento x
  public Set makeSet(Object x);
```


Interfaccia Partition – 2

```
// Sostituisce A con l'unione di A e B, distruggendo B
// e restituendo A
public Set union(Set A, Set B);

// restituisce l'insieme che contiene l'elemento x
public Set find(Object x);
}
```


Implementazione del TDA Partition

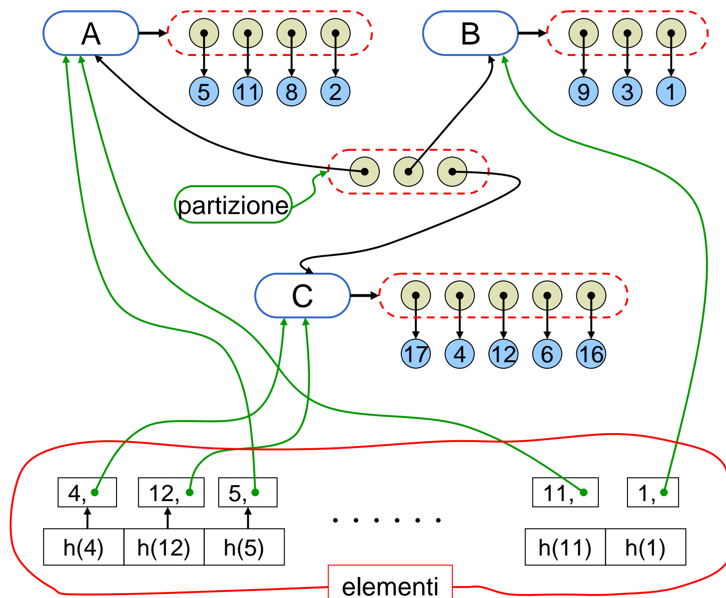
- Ogni insieme nella partizione è rappresentato da una sequenza (Vector, List, Sequence)
  - Oppure da una classe che implementa il TDA Set
- In una sequenza (Vector, List, Sequence) inseriamo tutti i riferimenti agli insiemi nella partizione
  - Ogni insieme creato nella partizione verrà inserito in questa sequenza

Nota

- Ad ogni elemento dobbiamo associare l'insieme a cui appartiene
  - Rappresentiamo gli elementi con la classe SetAwareElement
  - Nella partizione c'è una mappa che ha come voci le coppie (elemento, insieme)
    - Rappresentiamo mappa con una tabella hash
    - Serve per implementare efficientemente find

Interfaccia PartitionAware

```
public interface PartitionAware {
    public int size();
    public boolean isEmpty();
    public Set makeSet(Element x);
    public Set union(Set A, Set B);
    public Set find(Element x);
}
```

La classe HashTablePartition

```
public class HashTablePartition
            implements Partition{
  Map elementi; //implementata con una
                //tabella hash
  Vector partizione;
…
}
```

Nota su Partition – 1

- Dato che in una partizione gli insiemi sono due a due disgiunti, siamo sicuri che un elemento apparterrà solo ad un insieme. Di conseguenza possiamo aggiungere all'implementazione di Partition i metodi
  - public Set fastUnion(Set A, Set B)
  - public Object fastInsert(Object x)

Nota su Partition – 2

- public Object fastInsert(Object x)
  - Inserisce, all'inizio dell'insieme A, l'elemento x senza verificare se x appartiene ad A
  - Complessità O(1)
- public Set fastUnion(Set A, Set B)
  - Unisce gli insiemi A e B senza verificare se gli elementi di B appartengono ad A
  - Complessità O(|B|), se si usa fastInsert

Note su union(Set A, Set B) – 1

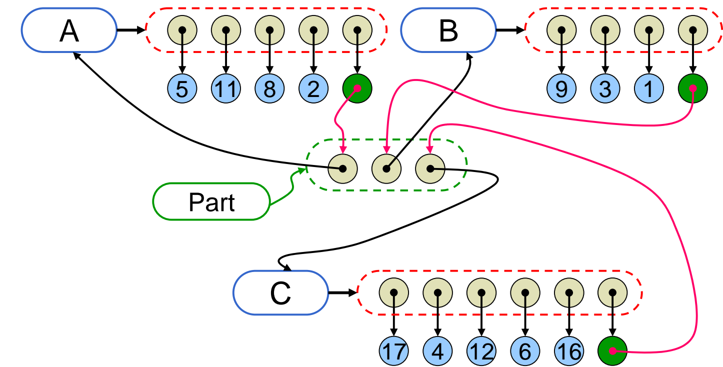
- Quando si esegue un'unione, spostiamo sempre gli elementi dall'insieme più piccolo a quello più grande
  - Ogni volta che spostiamo un elemento, esso va a finire in un insieme che è almeno due volte più grande dell'insieme da cui proviene
  - In questo modo, un elemento può essere mosso al più O(log n) volte
- Per ogni elemento spostato aggiorniamo la mappa (insieme a cui appartiene)

Note su union(Set A, Set B) – 2

- Non cancelliamo tutti e due gli insiemi ma solo quello più piccolo
- Per cancellare un insieme dovremmo conoscere la sua posizione nella sequenza (partizione) che memorizza i riferimenti di tutti gli insiemi nella partizione
  - Se la partizione è rappresentata con un Vector o Sequence possiamo scorrere tutto il Vector o tutta la Sequence per cercare l'insieme e poi rimuoverlo (non c'è

Note su union(Set A, Set B) – 2

- Se la partizione è rappresentata con List, allora c'è un metodo più efficiente per cancellare l'insieme
- Inseriamo come ultimo elemento dell'insieme X un elemento che conserva il riferimento alla posizione occupata da X in partizione
  - Il metodo makeSet inserisce come ultimo elemento il riferimento all'insieme appena creato

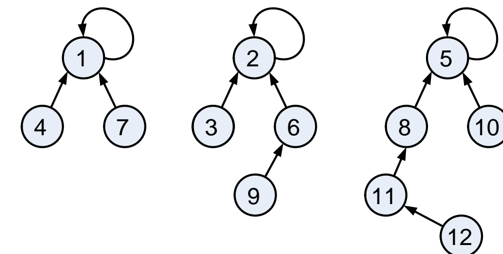


Metodo union(Set A, Set B)

```
union(Set A, Set B) {
  Position p = (Position) B.remove();
   while(!B.isEmpty()) {
      A.fastInsert(B.remove());
      // Aggiornamento mappa
   }
   partizione.remove(p);
}
```


Rappresentazione basata su albero – 1

- Ogni insieme nella partizione è visto come un albero
- Ogni elemento di un insieme è memorizzato in una posizione (nodo dell'albero)
- Un elemento appartiene all'insieme corrispondente alla radice (nodo che punta a se stesso) del suo albero
- Esempio degli insiemi "1", "2" e "5"

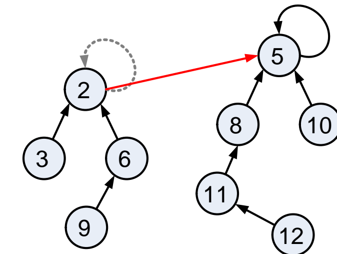


Rappresentazione basata su albero – 2

- Ogni posizione è implementata con un nodo contenente
  - element: riferimento all'elemento dell'insieme
  - parent: riferimento al nodo padre (a se stesso per la radice)
  - size numero di elementi nel sottoalbero, aggiornato solo per la radice
- Gli alberi usati sono specifici per Partition non costituiscono un'implementazione del TDA Tree
- Anche in questa implementazione possiamo usare una mappa o un SetAwareElement per conservare il riferimento all'insieme che contiene un dato elemento

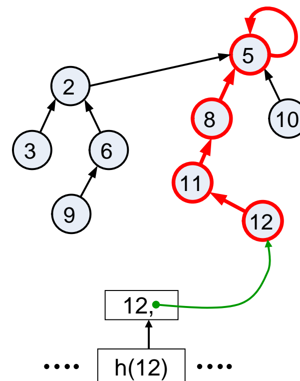
Il metodo union

- Per eseguire una union, si fa puntare semplicemente la radice di un albero alla radice dell'altro



Il metodo find

- Per eseuire una find, si segue il puntatore al genitore dal nodo che contiene l'elemento passato in input fino a quando non si arriva alla radice (nodo che punta a se stesso)

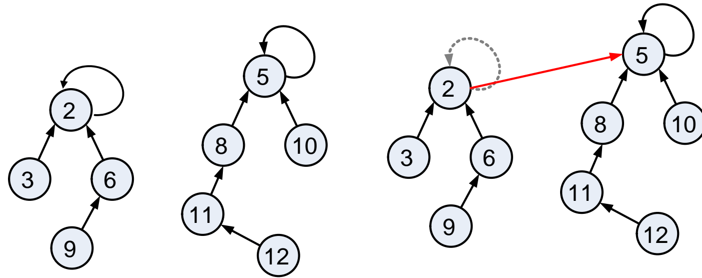


Esercizi

- Implementare l'interfaccia Partition tramite una classe TreePartition che rappresenta ogni insieme nella partizione con un albero i cui nodi contengono gli elementi dell'insieme ed ogni nodo ha un solo riferimento al nodo genitore
  - È necessario modificare l'interfaccia Partition?
- Scrivere un programma Java per testare la classe sviluppata al punto precedente

Miglioramenti – 1

- **Union-by-size**
  - nelle operazioni di unione rendi l'insieme più piccolo sottoalbero della radice dell'altro
  - Si usa un campo aggiuntivo nei nodi per memorizzare la taglia del sottoalbero



Miglioramenti – 2

- **Path Compression**
  - Dopo aver eseguito un'operazione find(x), tutti i nodi attraversati nella ricerca avranno come nodo genitore la radice