Il TDA Graph

Definizione

- Un grafo è una coppia (V, E), dove
  - V è un insieme di elementi, chiamati vertici (nodi)
  - E è un insieme di coppie di nodi, chiamati archi
- Nella nostra implementazione, vertici ed archi sono posizioni e memorizzano elementi
  - Saranno rappresentati dalle interfacce Vertex e Edge

Interfacce Vertex ed Edge

```
public interface Vertex extends Position{ }

public interface Edge extends Position{ }
```


Note – 1

- element nell'implementazione di Vertex l'elemento potrà rappresenterà il nome del vertice
- element nell'implementazione di Edge l'elemento contenuto rappresenterà
  - Il nome dell'arco, ad esempio (u,v), oppure il peso dell'arco
- Tramite il design pattern Decorator possiamo aggiungere una qualsiasi proprietà (attributo) a Edge e Vertex

Note – 2

- Rappresenteremo i grafi pesati e quelli non pesati nello stesso modo
- Rappresenteremo i grafi direzionati e quelli non direzionati nello stesso modo
  - Cambierà l'interfaccia per accedere alla rappresentazione del grafo
  - Nel caso di grafo orientati, ad esempio, vogliamo conoscere il numero degli archi entranti e quello degli archi uscenti

Metodi di accesso del TDA Graph

- isEmpty()
- numVertices()
  - restituisce il numero di vertici del grafo
- numEdges()
  - restituisce il numero di archi del grafo
- endVertices(e):
  - Restituisce un array contenente i due vertici dell'arco e
- opposite(v, e)
  - Restituisce il vertice incidente sull'arco e opposto al vertice v
- areAdjacent(v, w)
  - Restituisce true se e solo i vertici v e w sono adiacenti

Metodi di aggiornamento del TDA Graph

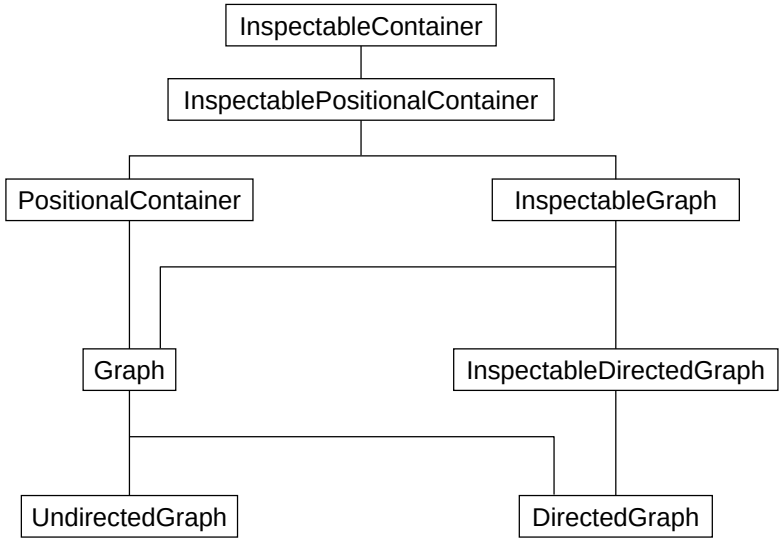
- replace(v, x)
  - Sostituisce l'elemento presente nel vertice v con x
- replace(e, x)
  - Sostituisce l'elemento presente nell'arco e con x
- insertVertex(x)
  - Inserisce un vertice che memorizza l'elemento x
- insertEdge(v, w, x)
  - Inserisce l'arco (v,w) che memorizza l'elemento x
- removeVertex(v)
  - Cancella il vertice v ed i suoi archi incidenti
- removeEdge(e)
  - Cancella l'arco e

Metodi che restituiscono iteratori

- incidentEdges(v)
  - Restituisce un iteratore sugli archi incidenti su v
- vertices()
  - Restituisce un iteratore sui vertici nel grafo
- edges()
  - Restituisce un iteratore sugli archi nel grafo
- outEdges(v)
  - Restituisce un iteratore sugli archi uscenti da v
    - Si applica solo ai grafi direzionati
- inEdges(v)
  - Restituisce un iteratore sugli archi entranti in v
    - Si applica solo ai grafi direzionati

Gerarchie di interfacce

- Tutti i metodi vengono suddivisi in una gerarchia di interfacce così come abbiamo fatto per gli alberi
- Alcuni metodi verranno inseriti nell'implementazione delle posizioni Edge e Vertex e richiamati nell'implementazione di Graph
- Il codice relativo alle interfacce è sul sito



InspectableGraph – 1

```
public interface InspectableGraph
    extends InspectablePositionalContainer
  {
   public boolean isEmpty();
  //restituisce il numero di vertici del grafo
  public int numVertices();

  //restituisce il numero di archi del grafo
  public int numEdges();
```


InspectableGraph – 2

```
//restituisce un vertice arbitrario del grafo
public Vertex aVertex();

//restituisce un arco arbitrario del grafo
public Edge anEdge();

//restituisce un iteratore sugli archi del grafo
 public Iterator edges();


//restituisce true se v1 e v2 sono adiacenti
public boolean areAdjacent(Vertex v1, Vertex v2);
```

InspectableGraph – 3

```
//restituisce il grado del vertice v
 public int degree(Vertex v);

 //restituisce un iteratore sui vertici adiacenti al vertice v
 public Iterator adjacentVertices(Vertex v);

 //restituisce un iteratore sugli archi incidenti sul vertice v
 public Iterator incidentEdges(Vertex v);
```


InspectableGraph – 4

```
//restituisce un array di grandezza due contenente
  // i vertici incidenti su e
  public Vertex[] endVertices(Edge e);

//restituisce l'estremità di e distinta da v
// (l'altro vertice incidente su e)
  public Vertex opposite(Vertex v, Edge e);

}
```


Graph – 1

```
public interface Graph
    extends PositionalContainer, InspectableGraph {

  // Inserisce un vertice isolato nel grafo che
  // contiene element
  public Vertex insertVertex(Object element);

// Inserisce un arco contenente element tra i vertici v1
// e v2. Se il grafo è orientato v1 è l'origine e v2 è la
// destinazione
  public Edge insertEdge (Vertex v1, Vertex v2,
                          Object element);
```


Graph – 2

```
  //Cancella il vertice v e tutti gli archi incidenti su esso
  public void removeVertex(Vertex v);

  //Cancella l'arco e
  public void removeEdge(Edge e);

  //Cancella l'arco tra i nodi v1 e v2
  public void removeEdge(Vertex v1, Vertex v2);

  //Sostituisce l'elemento in v con x
 public void replace(Vertex v, Object x);

 //Sostituisce l'elemento in e con x
 public void replace(Edge e, Object x); }
```

UndirectedGraph

public interface UndirectedGraph
extends Graph { }

- Le interfacce InspectableDirectedGraph e DirectedGraph sono a disposizione sul sito
- Aggiungere tutte le eccezioni necessarie

Implementazioni di Edge e Vertex

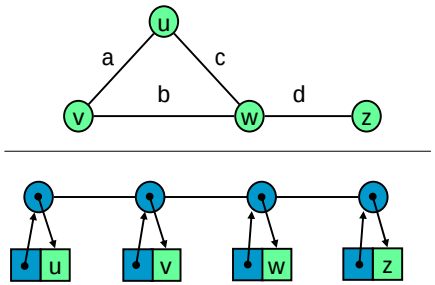
- Le loro implementazioni cambiano a seconda della rappresentazione che utilizzeremo per il grafo
  - Lista di Incidenza (*Incidence List*)
  - Matrice delle Adiacenze (*Adjacent Matrix*)
  - Lista delle Adiacenze (*Adjacent List*)

Lista di incidenza

- Il grafo è rappresentato tramite due liste
- Una lista V (NodeList) conserva i vertici del grafo
  - All'interno di una posizione nella lista c'è una posizione che rappresenta il vertice
- Una lista E (NodeList) conserva gli archi del grafo
- Per rappresentare l'insieme dei vertici e l'insieme degli archi avremmo potuto utilizzare un contenitore qualsiasi

Implementazione di Vertex

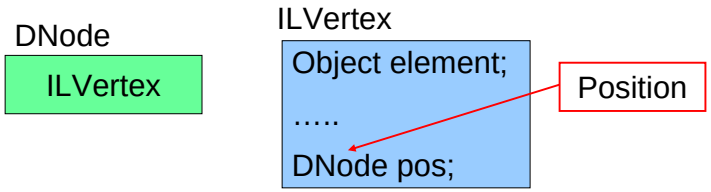
- L'implementazione di Vertex conserva sia l'elemento contenuto nel vertice sia il riferimento alla posizione che conserva il vertice nella sequenza dei vertici



- In questo modo alcune operazioni (e.g., remove(v)) saranno semplificate

Inserire il riferimento alla posizione?

- Nell'implementazione di Vertex (Edge) si potrebbe inserire un riferimento alla posizione del contenitore (NodeList) che conserva, come elemento, il vertice (l'arco)
  - L'implementazione di alcuni metodi sarà più efficiente



La classe ILVertex

- È la nostra implementazione di Vertex per i grafi rappresentati attraverso una lista di incidenza
- Per semplicità, usiamo la stessa rappresentazione sia per i grafi direzionati sia per quelli non direzionati
  - I vertici dei grafi direzionati hanno outDegree e inDegree
- Avremmo potuto avere una classe base da cui poi derivavamo le classi che rappresentano il vertice di un grafo non direzionato e quelli di uno direzionato

La classe ILVertex – 1

```
public class ILVertex  implements Vertex {

   protected Object _element;


   protected int inDegree;

   //serve solo per i grafi direzionati
   protected int  outDegree;
```


La classe ILVertex – 2

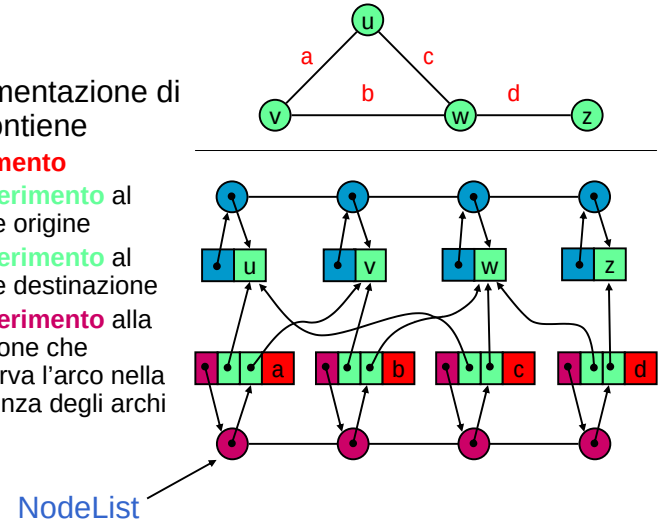
```
public ILVertex() {
                (String) _element = new String();}
public ILVertex(Object o) { _element = o; }

public Object element() {   return _element; }

public Object setElement(Object o) {
   Object tmp = _element;
   _element = o;  return tmp; }
```

La classe ILVertex – 3

```
public int inDegree() { return inDegree; }

public int setInDegree(int d) {
   inDegree = d;  return inDegree; }

public int outDegree() { return outDegree; }

public int setOutDegree(int d) {
   outDegree = d;  return outDegree; }

   }
```


Implementazione di Edge

- L'implementazione di Edge contiene
  - **L'elemento**
  - Un riferimento al vertice origine
  - Un riferimento al vertice destinazione
  - Un **riferimento** alla posizione che conserva l'arco nella sequenza degli archi



UEdge

- È la nostra implementazione di Edge
- Rappresenta un arco non direzionato
- La classe DEdge, rappresenta l'arco direzionato, sarà derivata da UEgde e
  - Verranno aggiunti solo dei metodi specifici per un arco direzionato

Implementazione di UEdge – 1

```
public class UEdge
   implements Edge {

   protected Object   _element;
   protected Vertex   _origin;
   protected Vertex   _destination;
```

Implementazione di UEdge – 2

```
public UEdge () {
            (String) _element = new
  String(); }
public UEdge (Object o) { _element = o; }

public Object element () { return
  _element; }

 public Object setElement (Object o) {
   Object tmp = _element;
```


Implementazione di UEdge – 3

```
public void setEndpoints (Vertex v1, Vertex v2) {
  _origin = v1;  _destination = v2; }

public Vertex[] endpoints () {
  Vertex[] ep = new ILVertex[2];
  ep[0] = _origin;
  ep[1] = _destination;
  return ep;
}
```


Implementazione di UEdge – 4

```
public Vertex oppostite(Vertex v) {
   if(v.equals(_origin))
     return _destination;
   else if(v.equals(_destination))
     return _origin;
   else throw new
  Errore("L'arco non è incidente al vertice");
  }
}
```


Esercizio

- Aggiungere equals alla classe ILVertex
- Implementare la classe DEdge
- Aggiungere
  - getOrigin();
  - getDestination();
  - reverseDirection();
  - setDirectionFrom(Vertex v);
  - setDirectionTo(Vertex v);

Implementazione del TDA Graph

- Solo per semplicità, implementeremo separatamente le intefacce UndirectedGraph e DirectedGraph scrivendo le classi ILUndirectedGraph e ILDirectedGraph
- La cosa migliore da fare sarebbe quella di implementare una gerarchia di classi raggruppando le funzionalità comuni ai due tipi di grafi così come abbiamo fatto per le interfacce
- Le classi al top di questa gerarchia dovrebbero essere classi astratte

ILUndirectedGraph

- È la classe che implementa l'interfaccia UndirectedGraph
- Si basa sulle liste di incidenza (NodeList)
  - Una per i vertici
  - Una per gli archi
- L'implementazione fornita è parziale, completarla per esercizio

Bozza della classe

```
public class ILUndirectedGraph
    implements UndirectedGraph {

  NodeList _vertices;
  NodeList _edges;

  public ILUndirectedGraph() {
    _vertices = new NodeList();
    _edges = new NodeList();
  }
```


Alcuni metodi – 1

```
public int numVertices() {
  return _vertices.size(); }

public int numEdges() {
  return _edges.size(); }

public int degree(Vertex v) {
    ILVertex tmp = checkVertex(v);
  return tmp.inDegree();
 }
```

Alcuni metodi – 2

```
public Vertex aVertex() throws
                    GraphEmptyException {
 if(_vertices.size())
    return (Vertex)
   (_vertices.first()).element();
 else
   throw new
        GraphEmptyException("Grafo
   vuoto");
```


Nota

- Nell'implementazione di ILUndirectedGraph aggiungere le funzioni checkEdge e checkVertex
- Sono simili a checkPosition di NodeList
- Verificano che la posizione Vertex o Edge passata è valida
  - Convertono una posizione generica Vertex o Edge al tipo ILVertex o UEdge, rispettivamente, eseguendo un cast, se non ci riescono lanciano un'eccezione
  - Bisogna stabilire quando una posizione (Vertex ed Edge) non è valida

Iteratori

- Nelle interfacce precedenti sono previsti dei metodi che restituiscono un iteratore
- Possiamo gestire iteratori su archi e vertici implementando due classi
  - VertexIterator
  - EgdeIterator
- Entrambe implementano Iterator
- Per l'implementazione utilizzeremo un *Adapter Pattern*

VertexIterator – EdgeIterator

```
public class VertexIterator
                    implements Iterator {
        …..
  public Vertex nextVertex() {….}
}

public class EdgeIterator
                    implements Iterator {
        …..
  public Edge nextEdge() {….}
}
```

VertexIterator

```
public class VertexIterator implements Iterator {

  private Iterator V_iter;

  public VertexIterator (Iterator iter) { V_iter = iter; }

  public boolean hasNext () { return V_iter.hasNext(); }

  public Object next () {return V_iter.next(); }

 public Vertex nextVertex () { …  }
}
```


Utilizzo di VertexIterator

```
public Iterator vertices() {
    return new
        VertexIterator(_vertices.elements());
 }
```

- L'implementazione di

  public Iterator adjacentVertices(Vertex v);

  non è immediata come quella di vertices()
  - Come è possibile implementare vertices()?

elements() e positions()

- La nostra gerarchia di interfacce prevede i metodi
  - elements() e positions()
- Cosa dovremmo far restituire a tali metodi?
  - Solo archi; solo vertici; archi e vertici
- Come esercizio provate a fornire una possibile soluzione
  - Per il momento ignorate questi metodi (fate restituire un iteratore vuoto)

Come distinguiamo archi da vertici?

Algoritmi su grafi

- Ci sono degli algoritmi su grafi che hanno bisogno di associare delle informazioni ai vertici e/o archi del grafo
- BDF
  - Campi: predecessore, colore e distanza
- MST
  - Pesi sugli archi
  - Campi: predecessore e chiave

Mappe e grafi – 1

- Invece di aggiungere altri campi nell'implementazione di un vertice (arco), aggiungiamo solo un'istanza della classe che implementa Map
- La mappa associata al vertice (arco) conterrà delle informazioni aggiuntive (attributi) sul nodo che possono essere inserite a seconda dell'algoritmo che utilizzerà il grafo
  - Le chiavi della mappa saranno essere stringhe tipo: colore, distanza, genitore, …

Mappe e grafi – 2

- Dobbiamo estendere la classe che implementa il vertice aggiungendo
  - Un'istanza di Map
  - Dei metodi per leggere/settare le proprietà del vertice (arco)
- Questo è il design pattern Decorator

Interfaccia Decorable – 1

```
public interface Decorable {

//assegna il valore value all'attributo attr lancia
// un'eccezione se l'attributo non è del tipo giusto
public void set (Object attr, Object value) throws
                InvalidAttributeException;

//restituisce il valore associato all'attributo attr lancia
//un'eccezione se attr non esiste o non è del tipo giusto
public Object get (Object attr) throws
                InvalidAttributeException;
```


Interfaccia Decorable – 2

```
//rimuove l'attributo attr, lancia un'eccezione se attr
//non è del tipo giusto
public Object destroy (Object attr) throws
                InvalidAttributeException;

//restituisce true se se l'attributo attr esiste, lancia
//un'eccezione se attr non è del tipo giusto
public boolean has (Object attr) throws
    InvalidAttributeException;

//restituisce un iteratore sugli attributi
public Iterator attributes ();
}
```

Classe DecoratedILVertex – 1

```
public class DecoratedILVertex extends
            ILVertex implements Decorable {
  Map m;
  public DecoratedILVertex () {
       m = HashTableSCMap();
  }
  public DecoratedILVertex (Equality Tester eq) {
       m = HashTableSCMap(eq);
  }
```


Classe DecoratedILVertex – 2

```
public boolean has (Object attr) throws
        InvalidAttributeException {
   checkAttribute(attr);
   if(m.get(attr) == null)
        return false;
   else
        return true;
 }
```


Altre implementazioni per i grafi

- Se il grafo fosse rappresentato tramite una lista delle adiacenze, allora l'implementazione di Vertex potrebbe contenere un riferimento ad un contenitore che memorizza i vertici adiacenti al vertice stesso

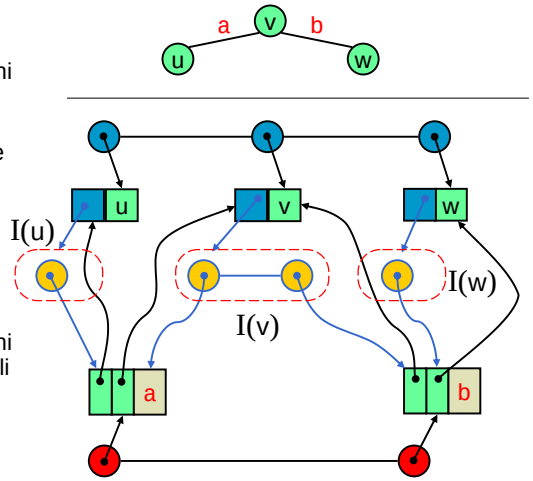
Altre rappresentazioni per grafi

- Per rappresentare i grafi potremmo utilizzare
  - La lista delle adiacenze
  - La matrice delle adiacenze
- Non rispecchiano esattamente le strutture viste nella lezione sui grafi

Lista delle adiacenze

I(u) è la "lista" delle adiacenze del vertice u

- Lista dei vertici
  - Sequenza di posizioni che rappresentano i vertici
  - Ogni vertice contiene un riferimento ad un contenitore di posizioni che rappresentano gli archi
- Lista degli archi
  - Sequenza di posizioni che rappresentano gli archi (non i vertici!!!)



Riferimento ai contenitori

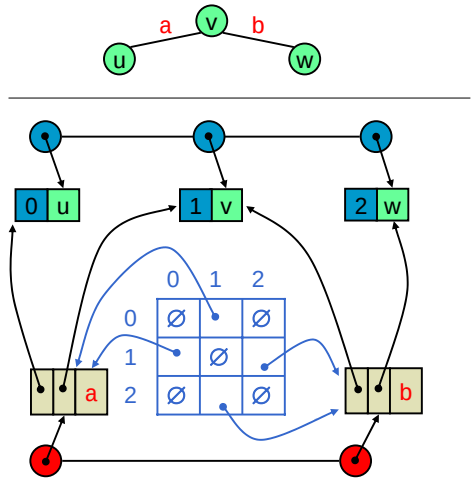
- Anche con questa rappresentazione la classe che rappresenta il vertice conserva sia l'elemento contenuto nel vertice sia il riferimento alla posizione che contiene il vertice nella sequenza dei vertici
- Lo stesso vale per la classe che rappresenta l'arco, inoltre la classe conserva i riferimenti alle posizioni nei contenitori dove è presente l'arco
  - Per l'arco (u,v) vengono conservati i riferimenti alle posizioni nei contenitori I(u) ed I(v)

Implementazione tramite lista delle adiacenze

- Nella classe che implementa Vertex possiamo aggiungere altre informazioni utili
  - Le informazioni presenti in ILVertex
  - Due contenitori (e.g., NodeList) uno contenente riferimenti agli archi uscenti, l'altro riferimento agli archi entranti
- Nella classe che implementa Edge aggiungiamo
  - Riferimenti ai vertici su cui l'arco insiste

Matrice delle adiacenze

- Lista degli archi
  - Come per il caso precedente
- Lista dei vertici
  - Sequenza di posizioni che rappresentano i vertici
  - Ogni vertice contiene una chiave intera (indice)
- Matrice di adiacenza
  - Contiene, per vertici adiacenti, un riferimento alle posizioni che rappresentano gli archi
  - Contiene null per vertici non adiacenti



	Lista di Incidenza	Lista delle Adiacenze	Matrice delle Adiacenze
Spazio	$n + m$	$n + m$	n^2
incidentEdges(v)	m	deg(v)	n
areAdjacent (v, w)	m	min(deg(v), deg(w))	1
insertVertex(o)	1	1	n^2
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	deg(v)	n^2
removeEdge(e)	1	1	1

n = # nodi —— m = #archi

Esercizi

- Implementare l'interfaccia DirectedGraph rappresentando il grafo con:
  - Una lista di incidenza
- Implementare le interfacce UndirectedGraph e DirectedGraph rappresentando il grafo con:
  - Una lista delle adiacenze
  - Una matrice delle adiacenze

Esercizi

- Implementare gli algoritmi:
  - BFS e DFS
  - MST
  - Prim e Kruskal
  - Dijkstra
  - Ford-Fulkerson