Corso di Paradigmi di Programmazione

Prof. Vittorio Maniezzo

***NOTA BENE**:*

- *le seguenti specifiche verranno controllate in automatico. E' fondamentale rispettarle nel dettaglio, con particolare riferimento ai nomi proposti. In caso contrario probabilmente il progetto risulterà non funzionante.*
- *Il mancato funzionamento dell'implementazione di alcune specifiche comporterà una corrispondente riduzione della valutazione finale, fino a un livello di richiesta di ripetere la prova d'esame.*

Si richiede di sviluppare una applicazione di tipo Class Library chiamata ASDlib (in un unico file ASDlib.cs) e contenente il namespace ASDlib.

La libreria espone le seguenti interfacce:

```
interface IPriorityQueue
{  bool insert(int i);
   int findMin();
   int extractMin();
}

interface IGraphSearch
{  void depthFirst();
   void breadthFirst(int s);
}

interface ICandidate
{  string nome {get;}
   string cognome {get;}
   string matricola {get;}
}
```

Al proprio interno gestisce un grafo memorizzato come:

```
public struct Nodo
{  public int id;
   public int x,y;
   public Nodo(int p1, int p2, int p3){id=p1;x=p2;y=p3;}
}

public struct Arco
{  public int id;
   public int end1;
   public int end2;
   public int w;
   public Arco(int p1, int p2, int p3, int p4)
   {  id = p1; end1 = p2; end2 = p3; w=p4;
   }
}
```

Vengono richieste:

- una **classe pubblica Ordinamenti** avente come metodi gli overloading per int, double e string degli algoritmi *insertionSort*, *quickSort* (signature: public void insertionSort(int[] A), public void quickSort(int[] arr) e relativi overloading).
  Implementare inoltre la *countingSort* (public void countingSort(int[] A, out int[] B) solo per array di interi.
  Sono da preferire implementazioni non ricorsive per evitare overflow dello stack, ma il client non farà verifiche in merito.

- Una **classe pubblica Grafo**, con il grafo rappresentato come List<Nodo> nodi e List<Arco> archi. La classe espone i metodi *readXMLgraph* (*public void readXMLgraph*(*string fpath*)), *dijkstra* (*public int[] dijkstra(int s)*), e i metodi virtuali *kruskal (public virtual List<int> kruskal()) e prim(public int[] prim (int r))*, oltre alle properties *numNodi* , *numArchi* e alla booleana *isOriented*. La classe Grafo deve poi essere specializzata in due classi derivate, **GrafoOrientato** e **GrafoNonOrientato** che ridefiniscono (***solo dove ha senso***) in overriding i metodi della classe base, restituendo *null* oppure un array di int con i predecessori (Dijkstra, Prim) o una List<int> (Kuskal).
  Nota: Kruskal al suo interno farà uso di un'istanza della ***classe pubblica UpTree*** (costruttore public UpTree(int n), da definire NON internamente a Grafo) che espone la *findSet* (public int findSet(int x)) *, makeSet* (public void makeSet(int x)) e *union* (public void union(int x, int y)).

- Una **classe pubblica MyHeap** che espone le proprietà *HeapInt*, *HeapDouble* e *HeapString* ***in sola lettura***, collegate ad opportuni array privati, e i metodi pubblici *buildHeap* (public void buildHeap(int[] A)), *insert* (public void insert(int x)), *extractMin* (public void extractMin(out int min)), con overloading per interi, double e stringhe. Dove necessario, è richiesto effettuare un ridimensionamento dell'array di base. I metodi buildHeap, insert ed extractMin lavorano su HeapInt, HeapDouble e HeapString.
  NOTA: la heap deve ordinare dal più piccolo al più grande!

- Una **classe pubblica MyHash** che che espone le ***proprietà in sola lettura*** *NumPos*, collegata a una var. privata *m* ***inizializzata dal costruttore***. Funzione di hash per divisione (k % m) in metodo privato. I dati non sono strutturati (il dato è anche la sua chiave) e corrispondono a numeri interi. Metodi pubblici: List<int> *showTableLine*(int k) che ritorna la lista corrispondente alla posizione k della tabella*. chainedHashInsert(int x), bool chainedHashSearch(int k)* e *bool chainedHashDelete(int x).*

L'interfaccia **IPriorityQueue** deve essere realizzata sia da una classe **ArrayPQ**, basata su un array privato inizializzato dal costruttore (no resize), che da una **HeapPQ** che implementano la coda di priorità rispettivamente con un array e con un oggetto di tipo MyHeap.

L'interfaccia **IGraphSearch** deve essere realizzata da una classe GraphSearch che lavora su un oggetto privato di tipo Grafo (direttamente o a seguito di upcast da classi derivate) contenuto in file chiamato "Grafo.xml", che può essere orientato o no, ed espone tre array di interi p, d ed f, ed un array *nColor[] nodeColor*, dove nColor è una *enum* definita sull'insieme di valori {white, gray, black}

L'interfaccia **ICandidate** deve essere realizzata della classe Ordinamenti.