Опис Person ADT:

Data:

* Стрічка з латинськими літерами та, можливо, цифрами
* Tuple з двома цілими числами
* Символ “m” або “f”

Operations:

* Person(name, living\_years, sex)
* \_\_str\_\_()
* \_\_eq\_\_()
* \_\_ne\_\_()
* get\_name()
* get\_sex()
* get\_birth\_year()
* get\_death\_year()
* \_\_repr\_\_()
* convert\_to\_dict

Опис FamilyGraph ADT:

Data:

Operations:

* FamilyGraph()
* \_validate\_vertex (v)
* vertex\_count()
* vertices ()
* edge\_count ()
* edges ()
* get\_edge(v, u)
* degree(v)
* incident\_edges(v)
* insert\_vertex(x)
* insert\_edge(u, v, type)
* adjacent\_vertices(v, type)

Опис Vertex ADT:

Data:

* об’єкт типу Person

Operations:

* Vertex(person)
* element()
* \_\_hash\_\_()
* \_\_str\_\_()
* \_\_repr\_\_()

Опис Edge ADT:

Data:

* Str

Operations:

* Edge(type)
* opposite(v)
* endpoints()
* \_\_hash\_\_()
* \_\_str\_\_()
* \_\_repr\_\_()

Опис RelationType ADT:

Data:

* Str

Operations:

* RelationType (type)
* \_\_eq\_\_()
* \_\_str\_\_()
* \_\_repr\_\_()
* \_\_invert\_\_()

class Person:  
 *"""  
 Class representing a person.  
 """* def \_\_init\_\_(self, name, living\_years, sex):  
 *"""  
 Initialization of person.* ***:param*** *name: string* ***:param*** *living\_years: tuple wih 2 integers* ***:param*** *sex: "f" or "m"  
 """*  
  
 def \_\_str\_\_(self):  
 *"""  
 Returns string representation of Person.* ***:return****: str  
 """*  
 def \_\_repr\_\_(self):  
 *"""  
 Returns string representation of Person.* ***:return****: str  
 """*  
 def \_\_eq\_\_(self, other):  
 *"""  
 Returns True if self equals other.* ***:param*** *other: Person* ***:return****: bool  
 """*def \_\_ne\_\_(self, other):  
 *"""  
 Returns True if self not equals other.* ***:param*** *other: Person* ***:return****: bool  
 """*def get\_name(self):  
 *"""  
 Returns name of Person.* ***:return****: str  
 """* def get\_sex(self):  
 *"""  
 Returns sex of Person.* ***:return****: str  
 """*  
 def get\_birth\_year(self):  
 *"""  
 Returns Person's birth year.* ***:return****: int  
 """*  
 def get\_death\_year(self):  
 *"""  
 Returns Person's death year.* ***:return****: int  
 """*

def convert\_to\_dict(self):

*"""  
 Converts Person to dictionary* ***:return****: dict  
 """*

class FamilyGraph:  
 *"""Representation of a family graph using an adjacency map."""* class Vertex:  
 *"""Lightweight vertex structure for a graph."""*  
  
 def \_\_init\_\_(self, person):  
 *"""Do not call constructor directly. Use Graph's insert\_vertex(x)."""*  
  
 def element(self):  
 *"""Return element associated with this vertex."""*  
  
 def \_\_hash\_\_(self): # will allow vertex to be a map/set key  
  
 def \_\_str\_\_(self):  
 def \_\_repr\_\_(self):  
  
 class Edge:  
 *"""Lightweight edge structure for a graph.""”*  
  
 class RelationType:  
 *"""  
 Class represents different relations between vertices.  
 """* def \_\_init\_\_(self, type):  
  
 def \_\_eq\_\_(self, other):  
 def \_\_str\_\_(self):  
 def \_\_repr\_\_(self):  
 def \_\_invert\_\_(self):  
  
 def \_\_init\_\_(self, u, v, type):  
 *"""Do not call constructor directly. Use Graph's insert\_edge(u,v,x)."""*  
  
 def endpoints(self):  
 *"""Return (u,v) tuple for vertices u and v."""*  
 def opposite(self, v):  
 *"""Return the vertex that is opposite v on this edge."""*  
 def type(self):  
 *"""Return element associated with this edge."""*  
 def \_\_hash\_\_(self): # will allow edge to be a map/set key  
 def \_\_str\_\_(self):  
 def \_\_repr\_\_(self):  
   
  
 # ------------------------- Graph methods -------------------------  
 def \_\_init\_\_(self):  
 *"""Create an empty directed graph.  
 ""”*  
 def \_validate\_vertex(self, v):  
 *"""Verify that v is a Vertex of this graph."""*  
  
 def vertex\_count(self):  
 *"""Return the number of vertices in the graph."""*  
  
 def vertices(self):  
 *"""Return an iteration of all vertices of the graph."""*  
  
 def edge\_count(self):  
 *"""Return the number of edges in the graph."""*  
  
 def edges(self):  
 *"""Return a set of all edges of the graph."""*  
  
 def get\_edge(self, u, v):  
 *"""Return the edge from u to v, or None if not adjacent."""*  
  
 def degree(self, v):  
 *"""Return number of outgoing edges incident to vertex v in the graph.  
 """*  
 def incident\_edges(self, v):  
 *"""Return all outgoing edges incident to vertex v in the graph.  
 """*  
  
 def insert\_vertex(self, x=None):  
 *"""Insert and return a new Vertex with element x."""*  
  
 def insert\_edge(self, u, v, type):  
 *"""Insert and return a new Edge from u to v with auxiliary element x.  
 Raise a ValueError if u and v are not vertices of the graph.  
 Raise a ValueError if u and v are already adjacent.  
 """*  
 def adjacent\_vertices(self, v, type=None):  
 *"""  
 Returns all adjacent vertices if type if None,  
 otherwise returns vertices, which connected with special type of edge.  
 """*