1. ADT – specification and interface

MAP

Domain: MP = {mp | mp is a map with elements key -> el, of type TKey -> TElem

Interface:

- init(mp)

  DESCR

  Initialises a new empty map

  PRE

  True

  POST

  mp is a valid map

- destroy(mp)

  DESCR

  Destroy a map

  PRE

  True

  POST

  mp was destroyed

- add(mp, key, el)

  DESCR

  Adds a new element with a given key to the map

  PRE

  mp is a valid Map, key is a valid TKey, el is a valid TElem

  POST

  MP' = MP + (key -> el)

- remove(mp, key, el)

  DESCR

  Removes an element with a given key from the map

  PRE

  mp is a valid Map, key is a valid TKey, el is a valid TElem

  POST

  True if element was removed, False otherwise

- search(mp, key, el)

  DESCR

  Searches an element with a given key in the map

  PRE

  mp is a valid Map, key is a valid TKey, el is a valid TElem

  POST

  search <- the element if it is in the map, NULL otherwise

- size(mp)

  DESCR

  Returns the number of key-value pairs from the map

  PRE

  mp is a valid Map,

  POST

  An integer number is returned (representing the number of key-value pair from the map)

- keys(mp)

  DESCR

  Returns the set of key from the map

  PRE

  mp is a valid Map,

  POST

  keys <-S (which is the set of all keys from mp)

- values(mp)

  DESCR

  Returns a bag with all the values from the map

  PRE

  mp is a valid Map

  POST

  keys <- B (which is a bag of all values from mp)

MAP ITERATOR

- init(mp, it)

  DESCR

  Intialises the iterator

  PRE

  mp is a valid MAP

  POST

  IT is a valid iterator

- valid(it)

  DESCR

  Check if a given iterator is valid or not

  PRE

  POST

  valid <- True if it is a valid Iterator, False otherwise

- getCurrent(it)

  DESCR

  Gets the current element

  PRE

  it is a valid Iterator

  POST

  post <- the element from current position of the iterator

- next(it)

  DESCR

  Makes the ‘iterator’ to point to the next element

  PRE

  it is a valid Iterator

  POST

  Iterator will point to the next element from container

Representation:

HashMapTElement

key: TKey
el: TElement

MAP

elems: HashMap <HashMapTElement>
(each element is of the form (TKey->TElement
.. so TKey (because are unique) are going to
be hashed)
len: Integer

MAP ITERATOR

map: *MAP
curentPos: TPos

HASH MAP

elems: TElement[]
next: Integer[]
size: Integer
firstFree: Integer
hash: Function

HASH MAP ITERATOR

hash_map: *Hash_Map
currentPos: TPos

Statement of the problem

Johnie drew on a map N points (in a cartesian coordinate system). He now ask himself how many squares can he draw using those points (as the corners of the square).

(http://www.infoarena.ro/problema/patrate3)

I'll store the coordinates as x -> y (x is x-axis coordinate, the key in map and y is the y-axis coordinate, the value in map).

In order to solve the problem efficiently, I'll take every combination of two points from the pool of points, suppose that the line between them is the diagonal of the square. I'll use some math formulas in order to determine the other two points and then I'll check to see if those two other points are really in the pool.

The idea is that due to the good complexity nature of Hash Maps in searching cases, I could do this final step of checking if those two points are really in the pool of points really fast.