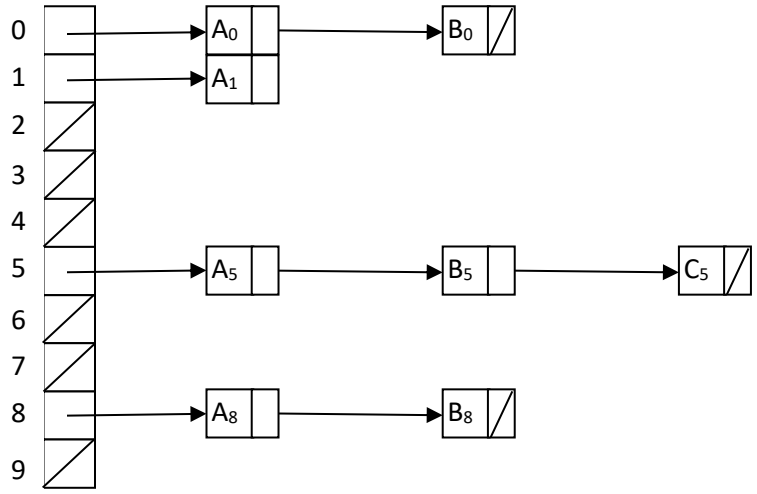
European Funds Knowledge Education Development
Wrocław University of Science and Technology
European Union European Social Fund

ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej

Laboratory – List 7

Introduction:

Hashtable is a data structure which has very fast two operations: insert a new element and find an element. This to operation base on a key and on a hash function. There are two main inner implementation: an array with open addressing or an array of lists. On the laboratory it will be used the second implementation.



The decision to which list it have to be inserted a new element depend on the hash function and the size of an array. For example if the key=1234, the hash function is h(x)=(x/100)*6+x and the size=10 then h(key)=12*6+1234=1306 and this value modulo size=10 is 6. So the element with the key 1234 have to be inserted into list on position 6 in an array.
This structure is fast if the list are very small (2-3 elements). To preserve that it is used a load factor limit constant which have to be smaller that 1 (usually between 0,7 and 0,8). The load factor is counted as number of elements stored in hashtable devided by the size of the array. If the load factor is bigger than a chosen limit, a new, bigger array have to be created and all elements from lists have to be moved to the new array.
Searching an element uses also hash function to determined which list can contain the wanted element. Then simple linear search is used in a selected list.

Remark:

There is a change on the main function. From now on, documents will be stored in hash table and an access to a document will be with its name. So the change in the main function is:

```
HashTable hashTable=new HashTable(8);
Document currentDoc=null;
```

And from now if the current document is equal **`null`**, the program has to write "`no current document".` Otherwise it has to perform the computation on a current document. It is also introduced an interface `IWithName` to give an access to a name of a document:

```
interface IWithName{
```

```
    String getName();
}
```

which you have to use in the function `toString()` of a class `HashTable`. Links for a document are still stored in the class `TwoWayCycledOrderedListWithSentinel<E>`.

List of tasks.

1. Implement class `HashTable` with functions specified in the template for the code. The inner representation of the hash table will be on the array of lists. It is proposed to use the library implementation of `LinkedList`, but you can use your own implementation from previous list. It is assumed that `add()` operation for a list in this array add a new link on the end of the list. A hash table stores its maximum load factor (default equals to 0.7).
2. The `HashTable` class uses the function `hashCode()` from an object so implement this function for a class `Document`. We assume, that the function works as follow: We will use number sequence 7,11,13,17,19,7,11,13,… so sequence 7,11,13,17,19 in a cycle. We take to a result a code of first letter and if there is a next letter, we multiply the first letter by the first number in a sequence and add it to the code for a second letter. If there is a next letter, the partial result we multiply by the next number in sequence and add code for next letter etc. For example for the string "`abcd`" the sequence of computation will be as follow:
   97*7+98=777
   777*11+99=8646
   8646*13+90=112488
3. If the number of elements exceed the load factor, the array capacity have to be doubled using function `doubleArray()` and element have to be moved to proper list base on the `hashCode()`.
4. The function `add()` for a `HashTable` cannot add a document with a name which is still in the hash table. In this case it has to return **`false`**.
5. You have to implement `equals()` function for a document, which will compare name's of documents.
6. The `toString()` function for a `HashTable` will print an array of list in the form: index of an array, colon. If the list is not empty print one space, then the name of the first document in a list. If there is a next document print semicolon, one space and the name for the second document etc. Every position in the array in separated line. To have an access to a name of the document cast the type of element to the type `IWithName`.

For 100 points present solutions for this list till Week 9.
For 80 points present solutions for this list till Week 10.
For 50 points present solutions for this list till Week 11.
After Week 11 the list is closed.

There is a next page…

Appendix

The solution will be automated tested with tests from console of presented below format. The test assumes, that there are up to X different documents, which there are created as the first operation in the test. Each document can be constructed separately.

If a line is empty or starts from `'#'` sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

If the current document is equal **`null`**, the program has to write "no current document".

If a line has a format:
`ch <str>`
your program has to choose a document with a name `str`, and all next functions will operate on this document.

If a line has a format:
`ld <name>`
your program has to call a constructor of a document with parameters `name` and `scan` for current position. The `name` have to be a correct identificator (in the constructor the `name` have to be stored in lower case letters). If it is not true, write in one line "`incorrect ID`". If the identificator is correct, the loaded document have to be now the current document. Try to add it to the hashtable. If it is impossible write "error" on the output.

If a line has a format:
`add <str>`
your program has to call `add(new Link(str))` for the current document. The result of the addition has to be written in one line. The `str` can be in the same format like a link in this task: only identificator or identificator with a weight in parenthesis.

If a line has a format:
`get <index>`
your program has to call `get(index)` for current document and write on the screen field `ref` from returned `Link` object. If the index is incorrect, write in one line "error".

If a line has a format:
`clear`
your program has to call `clear()` for current document.

If a line has a format:
`index <str>`
your program has to call `index(str)` for current document and write on the screen returned value in one line.

If a line has a format:
`show`

your program has to write on the screen the result of caling `toString()` for the current document. The first line has to present text “Document: <name>”, the rest of lines – every link in separated line.

If a line has a format:
`reverse`
your program has to write on the screen the result of caling `toStringReverse()` for the current document. The first line has to present text “Document: <name>”, the rest of lines – every link in separated line in reverse order.

If a line has a format:
`remi <index>`
your program has to call `remove(index)` for current document and write on the screen returned `Link` object in the format `<name>(<weight>)`. If the index is incorrect, write in one line “error”.

If a line has a format:
`rem <idStr>`
your program has to call `remove(idStr)` for current document and write on the screen returned value in one line.

If a line has a format:
`size`
your program has to write on screen the result of `size()` for current document and write on the screen returned value in one line.

If a line has a format:
`remall <idStr>`
your program has to call `removeall(new Link(idStr))` for current document.

If a line has a format:
`ht`
your program has to call `toString()` for hash table with documents.

If a line has a format:
`ha`
your program has to end the execution, writing as the last line “END OF EXECUTION”. Every test ends with this line.

A simple test for this task:
INPUT:

```
#Test for Lab7
show
ld first
link=a(10)
eod
add b(11)
show
ld second
link=c(3)
eod
```

```
add d(7)
show
getdoc first
show
reverse
getdoc second
show
reverse
getdoc xxx
show
ld xxx
eod
ld qqq
eod
ld asd
eod
ht
ld zx
eod
ht
ha
```

OUTPUT:

```
START
!show
no current document
!ld first
!add b(11)
true
!show
Document: first
a(10) b(11)
!ld second
!add d(7)
true
!show
Document: second
c(3) d(7)
!getdoc first
!show
Document: first
a(10) b(11)
!reverse
Document: first
b(11) a(10)
!getdoc second
!show
Document: second
c(3) d(7)
!reverse
Document: second
d(7) c(3)
!getdoc xxx
!show
no current document
!ld xxx
!ld qqq
!ld asd
!ht
0: xxx
```

```
1: qqq
2: second, asd
3:
4:
5:
6: first
7:
!ld zx
!ht
0:
1:
2: second, asd
3:
4:
5:
6:
7:
8: xxx
9: qqq
10:
11:
12:
13:
14: first, zx
15:
!ha
END OF EXECUTION
```