1) We need a data structure to organizes *websites*, where a website is defined by a structured collection of webpages that reside at the same host: e.g., www.unisa.it/index.html and www.unisa.it/diem/daa.html are pages of the same website, since they are on the same host (www.unisa.it), while www.unisa.it/index.html and www.unisal.it/diem/daa.html do not share the same host and thus they are on different websites.

The structure of the website is given by the directory structure as defined by the *URL*: e.g., www.unisa.it/ is the *home directory*, www.unisa.it/diem/daa.html is a web page contained in the directory diem that, in turn, is contained in the home page.

A special webpage is the page index.html contained in the home directory, that is called *homepage*.

You need to implement a class **Element** modeling either directories or webpages and a class **WebSite** that saves a website and it has the following interface:

- **WebSite(host)**: it creates a new WebSite object for saving the website hosted at **host**, where **host** is a string;
- **getHomePage()**: it returns the home page of the website at which the current object refers or it throws an exception if an home page does not exist;
- **getSiteString()**: it returns a string showing the structure of the website. The string must be formatted as follows: in the first line there must be the hostname at which the site is hosted; each page or directory contained in the root directory will appear in a new line preceded by "--- " (three dashes and a space) in alphabetical order (with numbers preceding lowercase letters preceding uppercase letters). For each directory the files and directories contained therein will appear just after the name of the directory in a new line preceded by a number of dashes = 3 x the number of parent directories, and a space. For example, the following is a valid output:

  ```
  www.unisa.it
  --- 1zz.html
  --- aaa.html
  --- diem
  ------ daa.html
  ------ profs
  --------- auletta.html
  --------- ferraioli.html
  --------- vinci.html
  --- index.html
  --- AAAA.html
  ```

- **insertPage(url, content)**: it saves and returns a new page of the website, where **url** is a string representing the URL of the page, and **content** is a string representing the text contained in the page.
- **getSiteFromPage(page)**: that given an **Element page** returns the **WebSite** object at which that page belongs.

Moreover, the WebSite class must contain the following private methods:

- **__hasDir(ndir, cdir)**: if in the current directory **cdir** there is a directory whose name is **ndir**, then it returns a reference to this directory, otherwise it throws an exception. An exception must be thrown even if the **cdir** is not a directory. Here **ndir** is a string, while **cdir** and the return value are objects of the class **Element**;

- **__newDir(ndir, cdir)**: if in the current directory **cdir** there is a directory whose name is **ndir**, then it returns a reference to this directory, otherwise it creates such a directory and returns a reference to it. An exception must be thrown even if the **cdir** is not a directory. Here **ndir** is a string, while **cdir** and the return value are objects of the class **Element**;
- **__hasPage(npag, cdir)**: if in the current directory **cdir** there is a webpage whose name is **npage**, then it returns a reference to this page, otherwise it throws an exception. An exception must be thrown even if the **cdir** is not a directory. Here **npage** is a string, while **cdir** and the return value are objects of the class **Element**;
- **__newPage(npag, cdir)**: if in the current directory **cdir** there is a webpage whose name is **npage**, then it returns a reference to this page, otherwise it creates such a page and returns a reference to it. An exception must be thrown even if the **cdir** is not a directory. Here **ndir** is a string, while **cdir** and the return value are objects of the class **Element**;
- **__isDir(elem)**: if the object **Element** referenced by **elem** is a directory returns **True**, otherwise it returns **False**. The format of **elem** is not constrained.
- **__isPage(elem)**: if the object **Element** referenced by **elem** is a web page returns **True**, otherwise it returns **False**. The format of **elem** is not constrained.

Note that public methods defined above may use these private methods, and private methods may re-use other private methods.

Provide an implementation of the class above so that:

- **WebSite(host)**, **getHomePage()**, **getSiteFromPage(page)**, **__isDir(elem)**, and **isPage(elem)** take time **O(1)**;

- **getSiteString()** takes time **O(n)**, where **n** is the number of pages and directories of the website;

- **insertPage(url, content)** takes time **O(l*k)**, where **l** is the number of parent directories of the page and **k** is the number of pages or directories contained in the last already existing directory;

- **__hasDir(ndir, cdir)** and **__hasPage(npag, cdir)** take time **O(log k),** where **k** is the number of pages or directories contained in **cdir**;

- **__newDir(ndir, cdir)** and **__newPage(npag, cdir)** take time **O(k),** where **k** is the number of pages or directories contained in **cdir**.

2) We have to implement a simplified model of a search engine, that allows users to retrieve relevant information from the collected Web Pages, by identifying pages containing a given keyword.

The core information stored by a search engine is a dictionary, called an *inverted index* or inverted file, storing key-value pairs **(w, L)**, where **w** is a word and **L** is a collection of pages containing word **w**. The keys (words) in this dictionary are called *index terms* and should be a set of vocabulary entries and proper nouns as large as possible. The elements in this dictionary are called *occurrence lists* and should cover as many Web pages as possible.

Actually, we would like to keep the size of the inverted index sufficiently small to fit in internal memory, even if the occurrence lists can have large total size, and thus they need to be stored on a disk. In other words, we can represent the inverted index through two separate data structures: the actual inverted index that is small to fit in the internal memory and that contains only a reference to occurrence lists, and the occurrence lists stored as separate objects.

The first data structure (the inverted index) must allow for efficient word-matching queries (i.e., to find the given keyword, and to return the corresponding occurrence list).

Specifically, consider the class **InvertedIndex** with the following interface:

- **InvertedIndex()**: it creates a new empty **InvertedIndex**;
- **addWord(keyword)**: it adds the string **keyword** into the **InvertedIndex**;
- **addPage(page)**: it processes the **Element page**, and for each word in its content, this word is inserted in the inverted index if it is not present, and the page is inserted in the occurrence list of this word. The occurrence lists also saves the number of occurrences of the word in the page.
- **getList(keyword)**: it takes in input the string **keyword**, and it returns the corresponding occurrence list. It throws an Exception if there is no occurrence list associated with the string **keyword**.

Provide an implementation of this class so that:

- **InvertedIndex()** takes time **O(1)**;
- **addWord(keyword)** and **getList(keyword)** have amortized and expected complexity **O(len(keyword))**;
- **addPage(page)** has amortized and expected complexity **O(len(word) + log list(word))** for each word in the content of the page, where **list(word)** is the number of pages in the occurrence list of word.

3) Implement a class **SearchEngine** with the following methods:

- **SearchEngine(namedir)**: that initializes the **SearchEngine**, by taking in input a directory in which there are multiple files each representing a different webpage. Each file contains in the first line the URL (including the hostname) and in the next lines the content of the webpage. This function populates the database of the search engine, by initializing and inserting values in all the necessary data structures.
- **search(keyword, k)**: it searches the **k** web pages with the maximum number of occurrences of the searched **keyword**. It returns a string **s** built as follows: for each of these **k** pages sorted in descending order of occurrences, the site strings (as defined above) of the site hosting that page is added to **s**, unless this site has been already inserted.

You will be provided a test dataset for testing your code. Notice that in grading your projects they will be run against a dataset different from the test one. The 15 projects that will return the correct result against this new dataset in the shortest time will receive a bonus point.