Modul:Exponential search

This module provides a generic exponential search algorithm. This kind of search can be useful when you want to find a key in some kind of sorted array, and you want to do it by checking as few array elements as possible. This could include situations like:

- Finding the highest archive number in a set of archives without checking whether they all exist.
- Finding the number of positional arguments in frame.args without having to expand the wikitext for each of them.

You shouldn't use this module if any of the following apply:

1. You can use the Lua length operator to find what you need.
2. Your array has any gaps in it. (In other words, any of the items before the final item is `nil`, e.g. `{'foo', 'bar', nil, 'baz'}`.) If you try and use this module on a sparse array, you might get an erroneous value.
3. Your array has less then about 10 items in it. It's possible to use this module for those arrays, but you will access most of the array elements anyway (perhaps some of them twice), and your code will be more complicated than if you just used a for loop.

Inhaltsverzeichnis

1 Usage 1
2 Examples 2
2.1 Jimbo's talk archives 2
2.2 Village pump archives 2

Usage

First, load the module.

```
local expSearch = require('Module:Exponential search')
```

You can then use the expSearch function with the following syntax:

```
expSearch(testFunc, init)
```

Parameters:

- *testFunc* - a test function for your array. This function should take a positive integer *i* as its first parameter. If the element corresponding to *i* is not in the array, then the function should return false or nil; and if it *is* in the array, then the function should return a truthy value (anything other than false or nil). (required)
- *init* - the initial value of *i* to check. For advanced users. (optional)

expSearch will return the highest value of *i* for which testFunc was truthy. If no values were truthy, the function will return nil.

Examples

Jimbo's talk archives

User talk:Jimbo Wales has archives at User talk:Jimbo Wales/Archive 1, User talk:Jimbo Wales /Archive 2, ... To find the highest archive number, you would use code like this:

```
local expSearch = require('Module:Exponential search')
local highestArchive = expSearch(function (i)
        local archive = 'User talk:Jimbo Wales/Archive ' .. i
        return mw.title.new(archive).exists
end)
```

Village pump archives

Wikipedia:Village pump (proposals) has old archives at Wikipedia:Village pump (proposals) /Archive A, Wikipedia:Village pump (proposals)/Archive B, etc. After they go through to Archive Z, the next archive is Archive AA. Although these archives aren't being updated anymore, as a demonstration we can find the highest one using this module; all we need is a function that converts from an integer to the corresponding archive name.

```
local expSearch = require('Module:Exponential search')

local function integerToAlpha(i)
        -- This function converts 1 to A, 2 to B, ... 26 to Z, 27 to AA, ...
        local ret = ''
        while i > 0 do
                local rem = i % 26
                if rem == 0 then
                        rem = 26
                end
                local char = string.char(rem + 64) -- the "rem"th letter of the a
                ret = char .. ret
                i = (i - rem) / 26
        end
        return ret
end

local function integerToArchive(i)
        return 'Wikipedia:Village pump (proposals)/Archive ' .. integerToAlpha(i)
end

local highestInteger = expSearch(function (i)
        local archive = integerToArchive(i)
        return mw.title.new(archive).exists
end)
local highestArchive = integerToArchive(highestInteger)
```

```
-- This module provides a generic exponential search algorithm.

local checkType = require('libraryUtil').checkType
local floor = math.floor

local function midPoint(lower, upper)
	return floor(lower + (upper - lower) / 2)
end

local function search(testFunc, i, lower, upper)
	if testFunc(i) then
		if i + 1 == upper then
			return i
		end
		lower = i
		if upper then
			i = midPoint(lower, upper)
		else
			i = i * 2
		end
		return search(testFunc, i, lower, upper)
	else
		upper = i
		i = midPoint(lower, upper)
		return search(testFunc, i, lower, upper)
	end
end

return function (testFunc, init)
	checkType('Exponential search', 1, testFunc, 'function')
	checkType('Exponential search', 2, init, 'number', true)
	if init and (init < 1 or init ~= floor(init) or init == math.huge) then
		error(string.format(
			"invalid init value '%s' detected in argument #2 to " ..
			"'Exponential search' (init value must be a positive inte
			tostring(init)
		), 2)
	end
	init = init or 2
	if not testFunc(1) then
		return nil
	end
	return search(testFunc, init, 1, nil)
end
```