Article
Danny Wijnschenk · Nov 16, 2017 6m read

Advent of Code 2016 Day16: Dragon Checksum

This is a series of programming challenges for beginners and experienced Caché programmers.

For an introduction : go to article https://community.intersystems.com/post/advent-code-2016-day1-no-time-ta...

The challenge of day 16 is about generating random data using a modified version of a dragon curve (you can find more info on fractal curves like Dragon here : https://en.wikipedia.org/wiki/Dragoncurve).

First you need to generate data in a loop as follows :

```
Create an initial state with your puzzle input

Call the data you have at this point "a".
Make a copy of "a"; call this copy "b".
Reverse the order of the characters in "b".
In "b", replace all instances of 0 with 1 and all 1s with 0.
The resulting data is "a", then a single 0, then "b"

Repeat until your generated data is equal or longer than 272 digits.
```

After this proces, you need to calculate a checksum, this is the string that is the answer to the puzzle.
The checkum is made by replacing all pairs of 1 and 0's by a single digit : 1 if the pair is the same digit, 0 if the pair are two different digits. (start with the first 272 digits)
Repeat this calculation until your checksum has an odd length.

The complete instructions are to be found at http://adventofcode.com/2016/day/16, it will also contain your puzzle input.

The code we wrote for this part of the challenge was not too difficult, here is Bert's code :

```
Start() PUBLIC {
  s input = 11100010111110100
  s minLength=272
  while $LENGTH(input)<minLength {
    s input=$$dragonCurve(input)
  }
  w !,$$calculateChecksum($EXTRACT(input,1,minLength))
}

dragonCurve(Input) {
  s result=Input_"0"
  for i=$LENGTH(Input):-1:1 {
    s result = result_'$EXTRACT(Input,i)
  }
  q result
```

```
}

calculateChecksum(Input) {
  s checksum=""
  for i=1:2:$LENGTH(Input)-1 {
    if $EXTRACT(Input,i)=$EXTRACT(Input,i+1) {
      s checksum=checksum_"1"
    } else {
      s checksum=checksum_"0"
    }
  }
  if '($LENGTH(checksum)#2) {
    s checksum=$$calculateChecksum(checksum)
  }
  q checksum

}
```

The second part of the challenge made a rewrite necessary : the length of the dragon changed to 35.651.584. As you might know, the limit of strings in Cache is 32.767 if long strings are not enabled, and 3.641.144 if long strings are enabled.

So instead of using one string, I used an array of strings. To make the reverse operations easy, each loop in creating the dragon doubles the indices in the array. Bert used some optimizations to make his code faster : he makes the array smaller before calculating the checksum, and calculate the first 16K using the code of part1.

Here is my code (which took a lot longer to calculate) :

```
Class AOC2016.Day16
{

ClassMethod Part2()
{
  #Dim state as %String = "01110110101001000"
  #Dim disk as %Integer = 35651584 ;272
  #Dim dragon as Array of %String
  #Dim iDragon as %Integer

  set dragon($i(dragon)) = state
  For {
    Do ..Dragon(.dragon)
    If ..Length(.dragon)>disk Quit
  }
  Do ..Cut(.dragon,disk)
  For {
    Do ..Checksum(.dragon)
    If ..Length(.dragon)#2'=0 Quit
  }
  Set iDragon="" For {
    set iDragon=$O(dragon(iDragon)) If iDragon="" Quit
    Write dragon(iDragon)
  }
  Quit
```

```
}

ClassMethod Dragon(ByRef dragon)
{
  #Dim middle as %Integer
  #Dim iDragon as %Integer

  set middle = $order(dragon(""),-1)
  set dragon($i(dragon))="0"  //first append 0
  //than append reverse of previous dragon (and negate digits)
  For iDragon=middle:-1:1 {
    Set dragon($i(dragon)) = $TR($reverse(dragon(iDragon)),"10","01")
  }
  Quit
}

//Cut length of array to maxlen
ClassMethod Cut(ByRef dragon, maxlen)
{
  #Dim iDragon, length as %Integer = 0
  For iDragon=1:1:dragon {
    Set length=length+$Length(dragon(iDragon))
    If length > maxlen set dragon(iDragon)=$E(dragon(iDragon),1,*-(length-
maxlen)) Quit
  }
  For iDragon=iDragon+1:1:dragon Kill dragon(iDragon)
  set dragon=$order(dragon(""),-1)
  Quit
}

ClassMethod Length(dragon)
{
  #Dim length as %Integer = 0
  #Dim iDragon as %Integer = ""
  For {
    Set iDragon = $Order(dragon(iDragon)) if iDragon = "" Quit
    Set length=length+$Length(dragon(iDragon))
  }
  Quit length
}

ClassMethod Checksum(str)
{
  #Dim rest as %String = ""
  #Dim iStr as %Integer = ""
  #Dim oldString as %String
  #Dim pair as %Integer

  For {
    Set iStr=$o(str(iStr)) if iStr="" Quit
    Set oldString = str(iStr)
    Set str(iStr)=""
      For pair=1:2:$length(oldString) {
        If (pair+1)>$length(oldString) set str(iStr+1)=$e(oldString,*)_$Get(str(
iStr+1)) Continue
        If $E(oldString,pair)=$E(oldString,pair+1) {
          set str(iStr)=str(iStr)_"1"
        } else {
          set str(iStr)=str(iStr)_"0"
```

```
        }
      }
      If str(iStr)="" Kill str(iStr)
    }
    Quit
  }

}
```

And the code Bert wrote that was faster :

```
Start2() PUBLIC {
  s strings=1
  s input=11100010111110100
  s minLength=35651584

  //first create big string with dragonCurve1 then loop with dragon curve 2
  while $LENGTH(input)<16000 {
    s input=$$dragonCurve(input)
  }
  s strings(1)=input

  s length=$$totalLength(.strings)
  while length<minLength {
    w !,"current length: "_length
    d dragonCurve2(.strings)
    s length=$$totalLength(.strings)
  }
  w !,"starting making it shorter"
  d makeShorter(.strings,minLength)
  w !,$$totalLength(.strings)
  w !,"starting checksum: "
  w !
  d calculateChecksum2(.strings)
}

dragonCurve(Input) {
  s result=Input_"0"
  for i=$LENGTH(Input):-1:1 {
    s result = result_'$EXTRACT(Input,i)
  }
  q result
}

makeShorter(Strings,maxLength) {
  s totLength=$$totalLength(.Strings)

  while (totLength>maxLength) {
    s struct=$ORDER(Strings(""),-1,value)
    if $LENGTH(value)>(totLength-maxLength) {
      s value=$EXTRACT(value,1,*-(totLength-maxLength))
      s Strings(struct)=value
    } else {
      k Strings(struct)
    }
```

```
    s totLength=$$totalLength(.Strings)
  }
}

totalLength(Strings) {
  s result=0
  s struct=$ORDER(Strings(""),1,string)
  while struct'="" {
    s result=result+$LENGTH(string)
    s struct=$ORDER(Strings(struct),1,string)
  }
  q result
}

dragonCurve2(Strings) {
  s Strings($INCREMENT(Strings))="0"
  s struct=$ORDER(Strings($ORDER(Strings(""),-1)),-1,value)
  while struct'="" {
    s Strings($INCREMENT(Strings))=$TR($REVERSE(value),"10","01")
    s struct=$ORDER(Strings(struct),-1,value)
  }
}

calculateChecksum2(Strings) {
  s pair=""
  s result=""
  s struct=$ORDER(Strings(""),1,string)
  while struct'="" {
    f i=0:1:$LENGTH(string) {
      s letter=$EXTRACT(string,i)
      s pair=pair_letter
      if $LENGTH(pair)=2 {
        if $EXTRACT(pair)=letter {
          s result=result_1
        } else {
          s result=result_0
        }
        if $LENGTH(result)>30000 {
          s newStrings($INCREMENT(newStrings))=result
          s result=""
        }
        s pair=""
      }
    }
    s struct=$ORDER(Strings(struct),1,string)
  }
  s newStrings($INCREMENT(newStrings))=result

  if '($$totalLength(.newStrings)#2) {
    d calculateChecksum2(.newStrings)
  } else {
    zw newStrings
  }
}
```

Look here for all our solutions so far : https://bitbucket.org/bertsarens/advent2016 and https://github.com/DannyWijnschenk/AdventOfCode2016

Here is the list of all Advent of Code 2016 articles :

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

#Caché #Code Snippet #Contest #ObjectScript