Article
Danny Wijnschenk · Nov 11, 2017 9m read

# Advent of Code 2016 Day11: Radioisotope Thermoelectric Generators

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 today is about microchips and generators. A microchip belongs to one particular generator, and the two can be on the same floor, or in the same elevator. But if a microchip is with another generator on the same floor or elevator, it will get toasted, except if his own generator is also on the same floor/elevator.
The elevator has only room for two items (can be mix of microchips/generators), the elevator stops on each floor for one second, enough for a microchip to get toasted if there is another type of generator on the same floor/elevator and its own generator is not there.

There are four floors, you have to find the quickest way to bring all microchips and generators to the fourth floor (without frying the microchips).

As an example, the initial state of each microchip/generator on the floors could be :

```
The first floor contains a hydrogen-compatible microchip and a lithium-
compatible microchip.
The second floor contains a hydrogen generator.
The third floor contains a lithium generator.
The fourth floor contains nothing relevant.
```

In this example, it would take minimum 11 steps. For the complete explanation of the challenge, go to http://adventofcode.com/2016/day/11, you can also find a step-by-step workout of this smaller example..

With your input (http://adventofcode.com/2016/day/11/input), what is the minimum number of steps to bring all objects to the fourth floor ?

This challenge is tough, much tougher than all the previous ones.

To solve it, I am going to use Brute Force : try all the steps and find out which was the minimum path to get to the result.

I store the initial state is a list variable like this:

```
Set F=$lb( $lb("E", "PG", "PM", "", "", "", "", "", "", "", ""),
           $lb("", "", "", "CG", "", "cG", "", "RG", "", "pG", ""),
           $lb("", "", "", "", "CM", "", "cM", "", "RM", "", "pM"),
           $lb("", "", "", "", "", "", "", "", "", "", ""))
```

I created methods to validate a floor, to calculate the amount of valid moves, to move a microchip or generator and I use recursion.

My solution works fine for the small example, but for the actual input, it takes forever. Seems that with every component you add in the input, the amount of iterations skyrocket (I am not even going to calculate how many). I decide to change the format, to have fewer instructions.

So I changed the storage in something lighter, that is easier to calculate with. The same state is now stored as follows :

```
 Set F="11123232323"
```

meaning : first position is for the elevator, second position for the first generator, third position for the first microchip etc. The value is the floor where it is located. Even with this simplification, it is still slow.

To get this challenge done before my PC runs out of steam, I have to find some shortcuts and stop at dead-ends or loops.

These were the optimizations I had to include :

- store the path while trying all steps, if the score does not go up after a few steps, this is a dead-end
- store all the intermediate steps (states) of all chips/generators on all floors. If the state was already obtained in an earlier path, with less steps, this is a dead-end
- when a path is bigger than the minimum path so far you are in a dead-end

Even with these optimizations, the speed is terrible, so back to the drawing board...

- Even if there are 5 different microchips/generator combinations, they are all the same : when you move one combination and you come to a dead-end, moving an other combination which were located on the same floors, you will also have the dead-end. So, you can eliminate them of the possible combinations to try.
- When saving the states of each floor, it does not matter which exact combinations is on which floor, but the type of the combination (microchip or generator).
  (microchip1+generator1 + microchip2+generator2 on floor1 and microchip3+generator3 on floor2 is exactly the same state as microchip1+generator1 + microchip3+generator3 on floor1 and microchip2+generator2 on floor2)

With the last two optimizations, I get the first part of the challenge solved in a timely manner.

```
Class AOC2016.Day11V2
{

ClassMethod Part1(%maxpath)
{
  #Dim F as %String
  Set F="11123232323"
//hold the floor of each item : elevator, generator1, microchip1, generator2, microch
ip2, ...
  kill %state
//will hold all the states of the floors we have examined, with the minimum path leng
th to get there from scratch
  Do ..Recursive(F,"")
  w !,"%maxpath : ",%maxpath
}
```

I know, never use %variables, but I did not want to add it to all my methods or make it public.

The method that is called recursively is as follows :

- For all move combinations:
  - If similar combination was already tried in elevator : false move
  - If move is valid :
    - expand the path
    - If path is longer than minimum so far : false move
    - If all items are on fourth floor : finished
    - else try new moves recursively

```
ClassMethod Recursive(F, path)
{
  #Dim done as %Boolean = 0
  #Dim copyF, copyPath, items as %String
  #Dim elevatorFloor, iItem, move, i, j as %Integer
  #Dim inElevator as %String = ""
//hold items that have been moved, to avoid similar moves

  Set copyF = F
  Set copyPath=path
  Set elevatorFloor=$Extract(F,1)
  Set items=""
  For iItem=2:1:$Length(F) {
    If $E(F,iItem)=elevatorFloor set items=items_$lb(iItem)
  }
  For move=1,-1 {
    For i=1:1:$LL(items) {
      For j=i:1:$LL(items) {
        Set F = copyF
        Set path = copyPath
        If ..inElevator($List(items,i),$List(items,j),elevatorFloor,move,F,.
inElevator) Continue
        If ..TryMove($List(items,i),$List(items,j),elevatorFloor,move,.F,path) {
          Set path=path_$lb($lb(F,..Score(F)))
          If ($ll(path)>%maxpath)&(%maxpath'="") set done=1 quit
          If ..Done(F) Set done = 2 Quit
          Do ..Recursive(F,path)
        }
      }
      If done Quit
    }
    If done Quit
  }
  If done,($ll(path)<%maxpath)!(%maxpath="") {
    set %maxpath=$ll(path)
    for i=1:1:$ll(path) {
      Do ..Output($list($list(path,i),1)) w !
    }
    w !,%maxpath,!
  }
  Quit
}
```

and the helper methods for  detecting similar elevator moves :

item#2 is odd or even, meaning generator or microchip, we only want to test on type not on actual chip number

```
ClassMethod inElevator(item1, item2, floor, move, F, ByRef inElevator) As %Boolean
```

```
{
  #Dim floor1, floor2 as %Integer
  #Dim found as %Boolean = 0
  Set floor1=$S(item1#2:$E(F,item1-1),1:$E(F,item1+1))
  Set floor2=$S(item2#2:$E(F,item2-1),1:$E(F,item2+1))
  If item1=item2 { set item2="", floor2="" }
  If $ListFind(inElevator,$lb(item1#2,item2#2,floor,move,floor1,floor2)) {
    set found=1
  } else {
    set inElevator=inElevator_$lb($lb(item1#2,item2#2,floor,move,floor1,floor2))
  }
  Quit found
}
```

And the method for valid moves and valid floors :

Before testing if all floors are valid, I test if we had this state before. I 'compress' the state before doing that : in stead of comparing the item numbers in order,

i change it into a structure where i just store how many microchips and generators there are on what floor (a state like 1222233 is identical to 1223322)

```
ClassMethod TryMove(item1, item2, floor, move, ByRef F, path) As %Boolean
{
  #Dim tryF as %String
  If ((floor+move)>4)!((floor+move)<1) Quit 0 //If floor not within 1..4 range
  If (item1#2)'=(item2#2),item1'=$S(item2#2:item2-1,1:item2+1) Quit 0
//if not same items (both micochip or both generator) and not same type of microchip
and generator : invalid
  Set tryF=F
  Set $E(tryF,1)=floor+move  //move elevator
  Set $E(tryF,item1)=floor+move  //move first item
  If item1'=item2 Set $E(tryF,item2)=floor+move  //move second item if any
  If ..ValidF(tryF,path,floor,floor+move) Set F=tryF Quit 1
//if rest of the start and end floor is valid : ok
  Quit 0
}

ClassMethod ValidF(F, path, floor1, floor2) As %Boolean
{
  #Dim F2 as %String
  #Dim error as %Boolean = 0
  #Dim floor, iItem, itemG, itemM as %Integer

  Set F2=..CompressState(F)
  If $Data(%state(F2)),%state(F2)<$ll(path) Quit 0
  For floor=1:1:4 set floor(floor)=""
  For iItem=2:2:$Length(F) {
    Set itemG=$E(F,iItem)
    Set itemM=$E(F,iItem+1)
    If floor(itemG)["M",floor(itemG)'["G" set error=1 quit
;there is already an unprotected microchip, cannot stand G
    If itemG'=itemM,floor(itemM)["G" set error=1 quit
;this unprotected microchip can not be put on a floor with a G
    Set floor(itemG)=floor(itemG)_"G"
    Set floor(itemM)=floor(itemM)_"M"
  }
  If 'error If $LL(path)>8 If ..Score(F)<$List($List(path,$ll(path)-8),2) set error=1
```

```
  If 'error {
    Set F2=..CompressState(F)
    set %state(F2)=$ll(path)
  }
  Quit 'error
}

ClassMethod CompressState(F) As %String
{
  #Dim iItem, iItem1, iItem2 as %Integer
  #Dim F2, states as %String
  For iItem=2:2:$Length(F) {
    set states($E(F,iItem),$E(F,iItem+1))=$Get(states($E(F,iItem),$E(F,iItem+1)))+1
  }
  Set F2=$E(F,1)
  Set iItem1="" For {
    Set iItem1=$Order(states(iItem1)) If iItem1="" Quit
    Set iItem2="" For {
      Set iItem2=$Order(states(iItem1,iItem2)) If iItem2="" Quit
      Set F2=F2_iItem1_","_iItem2_":"_states(iItem1,iItem2)_"."
    }
  }
  Quit F2
}
```

Finally some more methods to do simple tasks :

```
ClassMethod Score(F) As %Integer
{
  #Dim score as %Integer = 0
  #Dim iItem as %Integer
  For iItem=2:1:$length(F) {
    set score=score+($E(F,iItem)*10)
  }
  Quit score
}

ClassMethod Done(F) As %Boolean
{
  Quit ($Translate(F,4)="")
}

ClassMethod Output(F)
{
  #Dim floor, iItem as %Integer
  For floor=4:-1:1 {
    Write "F",floor
    For iItem=1:1:$length(F) {
      Write $Select($E(F,iItem)=floor:"#",1:" ")
    }
    Write !
  }
  Write !
}
```

This runs fine and gets me the result that unlocks the second challenge in time.

The second challenge however, adding  2 pairs of microchips/generators to the input, is enough to slow down my solution to an unbearable waiting time (and my son want to use my machine for his gaming), so I have to buy a bigger PC to get an answer ...

So I guess I need your help to come up with a faster solution (or you might consider to start up some crowd funding to get me a better machine).

(Since it is a holiday today, I did not want to disturb Bert to get his story told, we will put it here as a comment later)

Look here for all our solutions so far, be warned that the code for day 11 might disturb your power consumption : 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

**Source URL:**https://community.intersystems.com/post/advent-code-2016-day11-radioisotope-thermoelectric-generators