Counting Coins in the Dark

May 25, 2017

In[1]:=
```
Clear["Global`*"]
```

My friend Dario says this is a Google interview question.

Problem statement

You're in a dark room with n coins. You're told that there are m heads (and therefore $n - m$ tails). You're allowed to flip coins over if you want. The goal is to partition the n coins into 2 piles (not nec. w/ the same number of coins) such that each pile has the same number of heads.

Example

With $n = 10$ coins, $m = 6$ of which are heads, you might simply split the pile into two equal halves and hope that you get $m/2 = 3$ heads in each pile.

In[2]:=
```
SeedRandom[100];
coins = RandomSample[ConstantArray[H, 6] ~Join~ ConstantArray[T, 4]]
```

Out[3]=
```
{H, H, H, T, H, H, T, T, T, H}
```

In[4]:=
```
partitions = TakeDrop[coins, 5]
```

Out[4]=
```
{{H, H, H, T, H}, {H, T, T, T, H}}
```

You can see the partitions don't have an equal number of heads. You weren't feeling lucky.

So this scheme doesn't have the property that for all sets of coinflips, it partitions the set into two subsets which have the same number of heads.

Can we devise a better scheme?

Solution

There are 3 cases to consider:
Case 1. If $m == 0$, there are no heads, so we partition the coins into "the empty set" and "all the coins", which each has 0 heads, and you're done.
Case 2. If $m > n/2$, flip all the coins. Now you have a situation with n coins and $0 < m \leq n/2$ heads, which is Case 3.
Case 3. If $0 < m \leq n/2$, do this:

Partition the coins into an m-length list L_1 and an $(n - m)$-length list L_2. Let k denote the number of heads in list L_1, so the number of tails in L_1 is $(m - k)$. Since there are m heads total, if L_1 has k heads then L_2 has $m - k$ heads. Therefore, by flipping all L_1's coins, L_1's $m - k$ tails become $m - k$ heads, matching L_2's $m - k$ heads. So they have the same number of heads, so we are done.

Example

In[5]:=
```
coins = {H, T, T, H, H, T, T}
n = Length@coins
m = Count[coins, H]
```

Out[5]= {H, T, T, H, H, T, T}

Out[6]= 7

Out[7]= 3

In[8]:=
```
{p1, p2} = TakeDrop[coins, m]
newp1 = p1 /. {H → T, T → H}
```

Out[8]= {{H, T, T}, {H, H, T, T}}

Out[9]= {T, H, H}

In[10]:=
```
numH1 = Count[newp1, H]
numH2 = Count[p2, H]
numH1 == numH2
```

Out[10]= 2

Out[11]= 2

Out[12]= True

Proof by 1 example.

Numerical check of *n* = 1, ..., 100

First let's extend the definition of the Not[] function to work on the symbols H and T (heads & tails):

In[13]:=
```
Clear[H, T]
H /: Not[H] = T;
T /: Not[T] = H;
```

In[16]:=
```
Not[H]
```

Out[16]= T

In[17]:=
```
Not[T]
```

Out[17]= H

In[18]:=
```
Not /@ {H, T, T, H, H}
```

Out[18]= {T, H, H, T, T}

Here is a function that runs this algorithm:

In[19]:=
```
f[list_] := Module[{n = Length[list], m = Count[list, H]},
  Which[
   m == 0, {{}, list}, (* if m==0, the trivial partition works *)
   m > n/2, f[Not /@ list], (* if m>n/2, flip everything and try again *)
   True, {Not /@ Take[list, m], Take[list, -(n - m)]}
   (* make (1..m) and (m+1..n) and flip the (1..m) set *)
  ]]
```

It partitions the list into an m-length list and a (*n* – *m*)-length list, and flips the coins in the m-length list:

In[20]:=
```
f[{H, T, T, T, H, H, T}]
```

Out[20]=
```
{{T, H, H}, {T, H, H, T}}
```

If m==0, no heads. A boring partition will work:

In[21]:=
```
f[{T, T, T, T, T, T}]
```

Out[21]=
```
{{}, {T, T, T, T, T, T}}
```

If m>n/2, flip everything and run f[] again:

In[22]:=
```
f[{T, H, H, T, H, H, H}]
```

Out[22]=
```
{{T, H}, {T, H, T, T, T}}
```

Here is a function that checks if each set in the partition has the same number of heads:

In[23]:=
```
ok[partition_] := (partition // Map[Count[H]] // Apply[Equal])
```

Same number of Heads is OK:

In[24]:=
```
ok[{{H, H}, {H, H, T, T, T}}]
```

Out[24]=
```
True
```

Different number of Heads is not OK:

In[25]:=
```
ok[{{H, H}, {H, T, T, T}}]
```

Out[25]=
```
False
```

Here are all the possible sets of 4 coin flips (order doesn't matter):

In[26]:=
```
n = 4;
allCoins = Table[PadRight[ConstantArray[H, m], n, T], {m, 0, n}]
```

Out[27]=
```
{{T, T, T, T}, {H, T, T, T}, {H, H, T, T}, {H, H, H, T}, {H, H, H, H}}
```

Run f[] on all the flips and see if the resulting partition has equal numbers of heads:

In[28]:=
```
TableForm[Table[
  {c, f[c], ok[f[c]]}, {c, allCoins}],
 TableHeadings → {None, {"flips", "partition", "OK?"}},
 TableDepth → 2,
 TableAlignments → Center]
```

Out[28]//TableForm=

flips	partition	OK?
{T, T, T, T}	{{}, {T, T, T, T}}	True
{H, T, T, T}	{{T}, {T, T, T}}	True
{H, H, T, T}	{{T, T}, {T, T}}	True
{H, H, H, T}	{{H}, {T, T, H}}	True
{H, H, H, H}	{{}, {T, T, T, T}}	True

Here is a function that test whether f[] returns an OK partition for every possible set of *n* coinflips:

In[29]:=
```
test[n_] := Module[{allFlips},
  allFlips = Table[PadRight[ConstantArray[H, m], n, T], {m, 0, n}];
  AllTrue[allFlips, ok@*f]
 ]
```

We already saw that our algorithm works for *n* = 4 coins:

In[30]:=
```
test[4]
```

Out[30]=
```
True
```

Does it work for *n* coins, for n=1,...,100?

In[31]:=
```
AbsoluteTiming[
 AllTrue[Range[100], test]
]
```

Out[31]=
```
{0.304978, True}
```

Nice!