**Conditional probability**

P(“Some event” | “Given something we know/assume is true” )

<https://en.wikipedia.org/wiki/Conditional_probability>

**Law of Total Probability**

If we don’t know the second part, we can decompose: Assume it’s true. Then correct for the incertitude. And integrate that over all possible values.

<https://en.wikipedia.org/wiki/Law_of_total_probability>

**At least One**

P( “at least one ” ) = 1 - P(“exactly zero”)

P(“exactly zero”)

= P(“exactly 0 at step 1” AND “exactly 0 at step 2” AND … 0 at step k)

= P(“exactly 0 at step 1” ) \* P(“exactly 0 at step 2” ) \* … P(0 at step k)

(And relation is multiplications of P. Or relation is additions of P)

**The idea:**

* Instead of dealing with sequences of possible output we deal with probabilities after the nth step.
* So instead of going deep first search, we do breathe first search.
* Unfortunately, the pulls are not independent. They depend on previous pulls.
* Fortunately, that dependence exists only via a compact state.
  + Mostly via the remaining budget.
  + (Maybe also the availability of the 4\* pool yes/no)
* Not all budgets are as likely, we keep track of the probability of each.
  + In the above total probability example. All possibilities will be the list of all budgets.
  + And P(X) is the likelihood of seeing that budget.
* The base case is easy. There’s a valid set of sleep styles, given a budget at step N.  
  The probability is 1/ sizeOf(valid set) if the target is in the valid set. Or 0 if the target is not.
* For the following steps, we’ll call the base case again and again for different possible states(budgets) at the start of the step. Then we use the Law of Total Probability to correctly assemble those probabilities togethers.
* At each step we find the probability that we pick target at this step. (we allow repeat) Then we can flip to not pick it, combine (multiply) the “not pick” for a global “exactly zero” then flip again for “at least once.”

**Repeated State (Budget)**

If two branch get us to the same state (repeated budget, #of remaining pull) then we can merge them and add their associated probabilities.

**Repeated Cost**

Most SPO cost are repeated. We can pre-process that fact once at the start and deal with

`(cost, count)` items.

**Rank Query**

The base case (count all element below budget) can be repeated millions of times. Fortunately, it’s a standard operation and there’s data structures for it.   
**Rank x in S: the number of elements in S that are no greater than x.**

There’s specialized structure for that but for now we’ll keep it simple with binary search over our (cost,count) items. Actually we’ll store (cost,count,rank) where rank is cumulative sum of count after being sorted.

**Easy case, step 1**

The budget at the start of the step is perfectly known. So, probability of selection is 1/count of items below budget.

**Medium Case, step 2**

P( “select target” | budget after step1) =

* Loop all the valid sleep style at step 1.
* Compute the budgets after that pool. (child budget = parent – pull cost)
* Group budgets that are the same. Keep (value, count)
* P(bx) is count / total

**Hard Case, step 3+**

* For each of the budgets at the end of the previous step.
* We repeat making all possible pull.
* The child budget probability is also count / total but multiplied by parent probability.   
  (AND is multiplication)
* It’s likely that multiple parents will results in the same children budget, we can merge two paths to get the same state by adding their probabilities. (OR is addition)
* The total probability law is the same as with step 2, we just have exponentially more starting state to consider.
* Once a budget get below the cost of target, we know that budget and all it’s children will get multiplied by zero in the P(“select target”), so we can prune that budget line.

**Last step**

* Mostly the same as the other but P(“select target”) has a different law.
* It’s now just a question whether target is the largest element remaining.

**Fillers**

* We know once budget reach zero, we’ll only select filler. If target is above the filler line, we can 100% ignore this. The game can select 1000 fillers or 0 fillers and that does not change the probability of seeing target. (when target value >filler value)
* If target is a filler, the easy fix is to set a floor on the budget same as filler value.   
  Child budget = Max(parent-cost, filler value)