Algorithmics: Fall 2012

Seminar 1: Reminder: Basic Combinatorics, Counting

Gabriel Istrate

1 A crash course in Combinatorics

First we reviewed some basic combinatorial notions, so that everyone is comfortable enough with them.

1.1 Permutations, factorial

Question 1 *How many ways are there to order n distinct items ?*

Example 1 *Let's order three elements A, B, C. There are six orderings:*

$$(A; B; C), (A; C; B), (B; A; C), (B, C; A), (C; A; B), (C, B, A).$$

The (simple) answer to Question 1 is $n!$, defined as the product

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1.$$

Let's reconstruct the reasoning behind this answer.

- We can chose the first element in n ways.
- Once we have chosen the first element we are left with ordering the remaining $n-1$ elements.

Example 2 *In Example 1, after chosing letter A as the first one, we are left ordering the two elements B and C.*

Suppose we didn't know the answer and denoted by $F(n)$ the desired aswer. The above argument gave

$$F(n) = n \cdot F(n-1)$$

Together with $F(1) = 1$ this implies the result we wanted.

Exercise for you 1 *Remember mathematical induction ? Sketch a proof by induction that $F(n) = n!$.*

The argument we gave above is not the only way to derive the answer. Suppose that we have ordered numbers $1, 2 \ldots, n-1$. We can extend the ordering above by inserting element n in the existing ordering.

Example 3 *Suppose $n = 4$ and the ordering of numbers 1 to 3 is the one described below. There are four positions to insert number 4 in the ordering, indicated by bullets:*

$$[\bullet\ 3\ \bullet\ \ 1\ \bullet\ 2\ \bullet]$$

One position is before all numbers from 1 to 3. Three of them following each of such number.

In the general case we get n positions.

Therefore we get the same result.

1.2 Six out of 49: Arrangements

Suppose now that we want to choose six number out of 49 elements. We assume that choice order makes a difference. That is

$$(21; 3; 45; 19; 7) \text{ and } (7; 19; 45; 3; 21)$$

are **not the same**.

Question 2 *How many ways are there to choose 6 distinct numbers out of 49? More generally, how many ways are to choose an ordered list of k numbers from $1, 2, \ldots, n$.?*

The reasoning is similar to the one in the previous case: the first element can be chosen in n ways, the second one in $n-1$, ..., the k'th element in $n-k+1$ ways.

Quiz 1 *Justify this last statement.*

The answer is therefore

$$n \cdot (n-1) \cdot (n-2) \cdot \ldots (n-k+1).$$

We can relate this value to the previous definition of factorial by writing

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots (n-k+1) \cdot (n-k) \cdot \ldots \cdot 1.$$

Our desired answer is the product of red terms, that is $n!$ divided by the product of blue terms, which is nothing but $(n-k)!$.

1.3 Combinations

Suppose now that element order doesn't matter, and we are simply interested to choose k elements out of n distinct ones.

Question 3 *How many ways are there to choose* **two deserts** *out of* **four choices** *? This was part of the question asked in the quiz from the lecture.*

More generally, how many ways are there to choose k **objects** *out of n* **choices***, if order doesn't matter ?*

If order doesn't matter anymore, the answer we gave in the "six out of 49" case is no longer valid: arrangements overcount the number of possibilities, since

$$(21; 3; 45; 19; 7) \text{ and } (7; 19; 45; 3; 21)$$

lead now to the same set $\{3, 7, 19, 21, 45\}$. However we use a key observation: there are $k!$ permutations that yield the same set. In the previous example, all the 5! possible reorderings of elements $\{3, 7, 19, 21, 45\}$ yield the same set.

To answer Question **??** we have to divide the number of arrangements by $k!$. That is, the answer is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

We read in English "n choose k". In Romanian you might have encountered an alternative notation C_n^k.

2 Multiple ways of counting objects

An identity you have encountered in high school, a special case of Newton's binomial formula was

$$\binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{n} = 2^n \quad (1)$$

We can re-derive this formula by counting combinatorial objects in several ways.

Definition 1 *A* **binary word** *of length n is a finite sequence $a_1a_2\ldots a_n$, where each a_i is either 0 or 1.*

Example 4 *000,010, 111 are binary words of length three.*

Question 4 *How many binary words of length n are there ?*

The answer is simple: 2^n: there are two choices for a_1, two choices for a_2, ..., etc, and all such combinations are possible.

Review Question 1 *We can define (general) words as finite sequences over a fixed alphabet. For instance, fragments of DNA can be encoded as words over the four-letter alphabet $\{A, C, G, T\}$.*

How many words of length n over an alphabeth with m letters are there ?

Note 1 *In the case of permutations the independence property did not hold: choosing the first element constrained us not to use it anymore as a candidate for the second, third, etc .... In contrast, here we don't have such constraints.*

If we want to be fussy, we can formalize these questions as follows:

- *Denote by* $[n]$ *the set* $\{1, 2, \ldots, n\}$.
- *A permutation is a bijective function* $\pi : [n] \to [n]$.
- *A* word over an alphabet with m letters *corresponds to an arbitrary function* $f : [n] \to [m]$.

Review Question 2 *In the language of functions you learned in high-school, what kind of functions do arrangements correspond to ?*

Let's return to . The right-hand side of equation (1) counts binary words of length n.

Question 5 *Can we get the left-hand side of equation (1) by counting binary words some other (clever) way ?*

The answer is positive: a binary word of length n has k ones, $0 \leq k \leq n$. We can completely identify such a word with a subset of $[n]$, specifying positions in the bitstring where the corresponding symbol is a one.

Example 5 *Binary word* 01011 *corresponds to subset* $\{2, 4, 5\}$ *of* $[5]$. *Binary word* 11111 *corresponds to set* $[5]$ *itself.*

Review Question 3 *What does word 00000 correspond to ?*

Now, there are $\binom{n}{k}$ words of length n with exactly k ones. Summing these values over the range of k, from 0 to n counts the number of binary words of length n. Comparing this result to the answer to question (**??**) proves equation (1)

Review Question 4 *Suppose that instead of binary words of length* n *we counted words over an alphabet with* $a + b$ *letters of two types (e.g. roman vs greek)* a *letters "of the first type",* b *letters "of the second type". Redo the "double counting" argument above to prove Newton's binomial formula*

$$(a+b)^n = \binom{n}{0}a^n b^0 + \binom{n}{1}a^{n-1}b + \ldots + \binom{n}{k}a^{n-k}b^k + \ldots + \binom{n}{n}a^0 b^n \qquad (2)$$

<u>Hint:</u> Count letters of the two types together (left-hand side) and separately (right-hand side).

3 Reminder: geometric progressions

You have encountered the following sequence in high-school under the name *geometric progression.*

$$1, a, a^2, a^3, \ldots a^n.$$

Also, you might remember from high-school the identity

$$1 + a + a^2 + \ldots + a^n = \frac{a^{n+1} - 1}{a - 1} \tag{3}$$

Exercise 1 *Remember the algebraic proof for equation (3) by multiplying the left-hand side of equation (3) by $a - 1$ and canceling terms.*

Quiz 2 *For $a = 2$ identity (3) becomes:*

$$1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1 \tag{4}$$

I asked you in class: can you think of a "double counting" proof of equation (4) ?

An answer is below.

4 A coin-weighing problem

I started discussing in class the following

Quiz 3 *There are n coins. One of them is fake. All genuine coins have the same weight. The fake coin has a different weight from the genuine ones (can be lower or higher).*

We are given a scale. Using it we can compare the total weights of sets of coins placed on the left/right sides of the scale.

Our goal is to detect the fake coin using the smallest number of comparisons

I will return to this problem in the next seminar. For now I will recall the solution proposed by one of you (sorry, I don't know you by name yet):

- The solution works best if we assume that the number of coins is a power of two
- Also, for now I assume we know whether the fake coin is heavier/lighter than the rest
- Start by dividing the coins into two equal groups, and compare their weights.
- Using the information about the weight of the fake coin isolate the side that the fake coin belongs to
- Divide it into two equal halves and recursively apply the above procedure until you identify the fake coin.

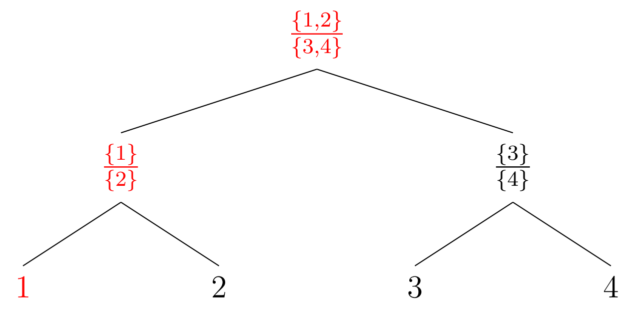

Figure 1: Tree representation of the coin-weighing algorithm. Leftmost branch (in red) corresponds to the following scenario: we compare coin sets $\{1, 2\}$ and $\{3, 4\}$. The first set is ligher. Then we compare coins 1 and 2, coin 1 being ligher; hence 1 is the counterfit coin in this situation.

5 Analysis of your coin-weighing algorithm

The algorithm is actually better than the one I thought you would find. I presented an informal analysis of the **complexity of the algorithm**, defined as the number of weighings until the bad piece is identified.

The basic idea is that one can map the evolution of the algorithm onto a binary tree The root of the tree corresponds to the initial test. Edges correspond to possible answers. Subsequent nodes correspond to tests.

Example 6 *Consider case of four objects, with the fake coin being lighter than the others. The binary tree corresponding to the algorithm is depicted in the Figure 1. Each node is marked with a label of type $\frac{A}{B}$, with A, B the sets of coins to be compared. Left branch from such a node corresponds to situation $weight(A) < weight(B)$ (right branch corresponds to the opposite situation).*

Suppose that we start with $n = 2^k$ elements. At the root (level 1) the counterfit coin is part of a set of 2^{k-1} elements. At level 2 the size gets halved, hence it is 2^{k-2} elements. The answer is reached at level l such that $2^{k-l} = 1$, i.e.

$$k = l$$

The number of tests the algorithm makes is therefore

$$k - 1 = \log_2(n) - 1 \text{ tests.}$$

This is better than expected, and, as we will see, optimal up to a constant factor.

Note 2 *We don't know yet enough to analyze the case of a general n (not a power of two). We will be able to do this fairly soon.*

Note 3 *One thing I mentioned in relation to this example is that for us constant factors won't matter. That is we will regard complexities of say*

$$2\log_2(n) \text{ and } 3\log_2(n) \text{ operations.}$$

as entirely equivalent. More about this later.

6 (Complete binary) trees and geometric progressions

How many nodes does a binary tree have ? Let's count them

- **one node (the root) on level 1.**
- **two nodes on level 2.**
- ...
- 2^{k-1} **nodes on level** k **(if the tree extends to that level)**

If T extends to n levels (and no more), $T(n)$ that we define as the total number of nodes is

$$T(n) = 1 + 2 + \ldots + 2^{n-1}$$

Let's count $T(n)$ in a different way: a tree with n levels is composed of the root and two trees with $n-1$ levels. Hence

$$T(n) = 2 \cdot T(n-1) + 1$$

Let's add 1 to both sides. We get

$$T(n) + 1 = 2[T(n-1) + 1]$$

Hence $T(n) + 1 = 2^{n-1}[T(1) + 1] = 2^{n-1} \cdot 2 = 2^n$.
We get

$$T(n) = 2^n - 1, \text{ and}$$

$$1 + 2 + \ldots + 2^{n-1} = 2^n - 1.$$

that is the formula in Quiz 2 !

Review Question 5 *Use a similar argument with a tree that branches out in $a \geq 2$ directions at each node to prove identity (3).*

7 [HARDER, OPTIONAL] The binary tree algorithm is optimal, up to constant factors

I didn't touch much of this section in the lab session. It's here for reference, for those of you that are curious. It's also not required material, as it is more difficult

Quiz 4 *Can you come up with a algorithm that seems even better algorithm* **if constant factors matter ?** *I am NOT asking you to* **prove** *that it's better, just to think about algorithm.*

Hint:YES. Consider the case of three coins, and try to extend it.

I stated informally in class that the binary algorithm is optimal up to constant factors. Proving this result is messy, as it requires precisely defining the computation model[1] and complexity, but I can give you the basic intuition even at this early stage.

Suppose the coins are $(a_1, a_2, \ldots, a_n)$. By identifying the counterfeit coin we mean identifying the index i of the counterfeit coin, $1 \leq i \leq n$.

Assume there would be an algorithm that found the counterfeit coin with $f(n)$ comparisons. At each node the algorithm compares two sets of coins A and B (having the same number of coins).

Sets A and B may *not necessarily contain* the counterfeit coin, in which case the weights of A and B can be equal.

Example 7 *Consider the case of three coins 1,2,3. One cannot divide the three coins into two equal sets, so we have to let one coin out. Say we let coin 3 out and compare the weights of coins 1 and 2.*

What result do we get if coin 3 is the counterfeit one ? What can we conclude about coin 3 if the weight of coins 1 and 2 is the same ?

Each decision point corresponds to a "branching out" in a decision tree.

Example 8 *Consider the moment when we compare coins 1 and 2 in Example 7. We can get three answers:*

- *same weight.*
- *coin 1 is lighter.*
- *coin 2 is lighter.*

In any case the number of branches at each node is upper bounded by a constant K (at most three in the previous example).

Suppose that our algorithm never made more than $f(n)$ comparisons. Then the number of leaves would be at most $K^{f(n)}$.

If $f(n)$ grows much slower than $log(n)$ then this number will become smaller than n for large n.

But we need at least n leaves, as the bad coin could be in any position $1 \leq i \leq n$

The conclusion is that $K^{f(n)} \geq n$, so $f(n) \geq log_K(n)$, which is the same as $\log_2(n)$ up to constant factors.

[1]called decision tree for this particular example