Mathematics of Computer Science. - M.I.T. opencourseware

Abstract:
In this lecture, we learned about the definition of set and sequence. How to counting without redundency is important and there are tools for counting method. Permutation and combination are method of counting. Also product rule and sum rule make the calculation of sum to easy count.

For the English:
identical, be sharedm constructive, via

Lec 16. COUNTING RULES I:

- Set

DEF: a Set is <u>unordered</u> collection of <u>distinct</u> elements

DEF: the size or <u>Cardinality</u> of set is the number of elements in S. denoted |S|

- Sequence

DEF: a Sequence is an ordered collection of elemens(Components/terms) not necessarily distinct

- Permutation

DEF: a Permutation of a set S is a sequence contains every elements in S <u>exactly once</u>

- Function

DEF: a Function f: X -> Y is a reation between X and Y sets s/t every element of X is exactly related to one element of Y

- exactly one outgoing arrow from X to Y, however Y don't have to get equal arrow even none

DEF: <u>Surjective</u> if every single element of Y is mapped to <u>at least once</u> (every target got arrows) - 일대일 함수

DEF: Injective -> <u>at most once</u> 단사 함수

DEF: Bijective -> <u>exactly once</u> - 일대일 대응 함수

Bijective iff Surjective + Injective - check the origin example graph

ex) Let (a1 ~ an) be a permutation of S = {a, ~ an}

Define Π(ai) = i i.e. a ∈ S is mapped to i iff a is in the I-th term in the permutation => then Π is Bijective

- Mapping Rule

1. f: X -> Y Surjective => |X| >= |Y|
2. f: X -> Y Injective => |X| =< |Y|
3. f: X -> Y Bijective => |X| == |Y|

ex) X = all ways to select 12 donuts from 5 varieties (choco, lemmon, glaced, plain, sugar)
Y = set of all 16 bit of sequence with exactly 4 once
|X| == |Y| if Bijective

ex) Bijective from subset of X = {1~n} to n-bit sequences
S -f->(b1 ~ bn) via bi = {1 if i∈S, 0 if i!∈S} => 포함이냐 불포함이냐
n bits sequence = 2^n = # of subset of an e-element set
how we count subset number

- Generalized pigeon hole principle
If |X| > k|Y| then ∀f:X->Y ∃k+1 different elements of X mapped to the same element mY   (if k =1 this we call pigeon hole principle)

ex) > n pigeons(x) fly into n-holes(y) => at least 2 pigeon fly into same hole
ex) Boston has 500,000 non bald people, clasim ∃3 people in Boston with the same # of hairs // # hairs =< 200,000 : |X| > 2|Y| => at least 3 people have same hairs
ex) Pick 10 arbitrary double digit number
X = collection of subsets of #'s // |X| = 2^10 = 1024
Y + {0, 1, ~ 990 (10*99)} = all possible sums // |X| > |Y| : non constructive proof

- Division Rule
DEF: a k-to-1 function f:X->Y maps exactly k elements of X to every element of Y (generalizes Bijection rule : bijective iff 1-to-1)
if f is k-to-1 : then |X| = k|Y| => 당연하다

ex) How many ways to place 2 identical rooks on a chess board s/t no row/column is shared

Y =set of valid rook configuration
X = all the sequences (r1c1, r2c2) s/t r1 != r2 & c1 != c2
placement = {rook1, rook2} != {r2c2, r1c1} f is 2-to-1 // |Y| \ |X|/2
sequence (r1c1, r2c2)
chess board = 8*8 7*7 => (8*7)^2 / 2

- Generalized Product Rule
let S bw a set of length k sequences, if there are *n1 possible 1st entry, *n2 '' for each 1st entry *n3 '' 3rd entries for each combination of 1st/2nd entries ...
*nk ... // then |S| \ n1n2...nk => combination

ex) committee (x - leader, y - secretary, z - consultant) selected from n-members
*n-ways to choose x
*(n-1)-ways to choose y (except x)
*(n-2)-ways to choose z (except x and y)

n!/(n-r)!
ex) “Defective dollar” : some digit appears mare then once in the 8-bit serial #
Fraction non-defective = # non-defective serial #s / total # of serial #s
Y = 10^8, X = 10*9 … *3 => 10!/2!
DEF: a product of set A1*A2*…An = {a1 ~ an} a1∈A1, a2 ∈A2 .. an∈An
<u>Product rule</u> => |A1*A2…An| = |A1|*|A2| … |An|

- Sum rule (following counting mechanism)
If A1 ~ An are disjoint sets => |A1 U A2 … An| = |A1| + |A2| … |An|
ex) passwords 6~8 symbols
1^{st} symbol is a letter (upper or lower)
other symbol are letter or digits

F = {a,b,c … z, A, B, C … Z} => |F| = 52
S = { ‘’ …., 0, 1, …9} => |S| = 62
P(possible password) = (F*S*S*S*S*S)U(F*S^6)U(F*S^7)
|P| = |F*S^5| + |F*S^6| + |F*S^7| => |F|*|S^5| + |F|*|S^6| + |F|*|S^7|
Sum rule Product rule