

## Today's topics

- Counting
  - Generalized Pigeonhole Principle
  - Permutations
  - Combinations
  - Binomial Coefficients
  - Writing permutation algorithms
- Reading: Sections 4.2-4.3, 4.4, 4.6
- Upcoming
  - Probability

## Generalized Pigeonhole Principle

- If  $N$  objects are assigned to  $k$  places, then at least one place must be assigned at least  $\lceil N/k \rceil$  objects.
- *E.g.*, there are  $N=280$  students in this class. There are  $k=52$  weeks in the year.
  - Therefore, there must be at least 1 week during which at least  $\lceil 280/52 \rceil = \lceil 5.38 \rceil = 6$  students in the class have a birthday.

## Proof of G.P.P.

- By contradiction. Suppose every place has  $< \lceil N/k \rceil$  objects, thus  $\leq \lceil N/k \rceil - 1$ .
- Then the total number of objects is at most
$$k \left( \left\lceil \frac{N}{k} \right\rceil - 1 \right) < k \left( \left( \frac{N}{k} + 1 \right) - 1 \right) = k \left( \frac{N}{k} \right) = N$$
- So, there are less than  $N$  objects, which contradicts our assumption of  $N$  objects!  $\square$

## G.P.P. Example

- Given: There are 280 students in the class.
  - Without knowing anybody's birthday, what is the largest value of  $n$  for which we can prove using the G.P.P. that at least  $n$  students must have been born in the same month?
- Answer:

$$\lceil 280/12 \rceil = \lceil 23.3 \rceil = 24$$

## Permutations (§4.3)

- A *permutation* of a set  $S$  of objects is a sequence that contains each object in  $S$  exactly once.
  - An ordered arrangement of  $r$  distinct elements of  $S$  is called an  *$r$ -permutation of  $S$* .
- The number of  $r$ -permutations of a set with  $n=|S|$  elements is
$$P(n,r) = n(n-1)\dots(n-r+1) = n!/(n-r)!$$

## Permutation Example

- You are in a silly action movie where there is a bomb, and it is your job to disable it by cutting wires to the trigger device. There are 10 wires to the device. If you cut exactly the right three wires, in exactly the right order, you will disable the bomb, otherwise it will explode! If the wires all look the same, what are your chances of survival?

$P(10,3) = 10 \cdot 9 \cdot 8 = 720$ ,  
so there is a 1 in 720  
chance that you'll survive!

## Combinations (§4.3)

- An  $r$ -combination of elements of a set  $S$  is simply a subset  $T \subseteq S$  with  $r$  members,  $|T|=r$ .
- The number of  $r$ -combinations of a set with  $n=|S|$  elements is
$$C(n,r) = \binom{n}{r} = \frac{P(n,r)}{P(r,r)} = \frac{n!/(n-r)!}{r!} = \frac{n!}{r!(n-r)!}$$
- Note that  $C(n,r) = C(n, n-r)$ 
  - Because choosing the  $r$  members of  $T$  is the same thing as choosing the  $n-r$  non-members of  $T$ .

## Combination Example

- How many distinct 7-card hands can be drawn from a standard 52-card deck?
  - The order of cards in a hand doesn't matter.
- Answer  $C(52,7) = P(52,7)/P(7,7)$ 
$$= 52 \cdot \cancel{51} \cdot \cancel{50} \cdot \cancel{49} \cdot \cancel{48} \cdot \cancel{47} \cdot \cancel{46} / \cancel{7} \cdot \cancel{6} \cdot \cancel{5} \cdot \cancel{4} \cdot \cancel{3} \cdot \cancel{2} \cdot \cancel{1}$$
$$= 52 \cdot 17 \cdot 10 \cdot 7 \cdot 47 \cdot 46 = 133,784,560$$

## Binomial coefficients

- Binomial coefficient  $\binom{n}{r}$ 
  - Given  $(1+x)^r$
  - What are the coefficients of  $x^r$ ?

$$(1+x)^n = \sum_{r=0}^n \binom{n}{r} x^r \\ = \binom{n}{0} x^0 + \binom{n}{1} x^1 + \cdots + \binom{n}{n} x^n$$

- Show for  $(1+x)^4$

## Important Theorems on Binomials

- Binomial theorem
  - In the expansion of  $(1+x)^n$ , the coefficient of  $x^r$  equals

$$\binom{n}{r} = C(n,r) = \frac{n!}{r!}$$

- Pascal's Identity
$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$
- Proofs?

## §4.6, Generating Perms. & Combs.

- We will go over algorithms for:
  - Generating the next largest permutation, in lexicographic order.
  - Generating the next largest bit string.
    - Remember, a bit string can represent a combination.
  - Generating the next  $r$ -combination in lexicographic order.
- Also we'll give recursive algorithms for generating permutations, combinations, and  $r$ -combinations.

## Generating All Permutations

```
procedure genAllPerms( $n > 0$ : integer)
  { output all permutations of the integers
  1,..., $n$ , in order from smallest to largest }
  for  $i := 1$  to  $n$  begin used $_i = \mathbf{F}$  end
  { none of the integers have been used yet }
  recursiveGenPerms( $i, n$ )
```

The *recursiveGenPerms* procedure is on the next slide...

## Recursive Permutation Generator

```
procedure recursiveGenPerms(i,n)
  if i>n then begin {We're done, print the answer.}
    for k := 1 to n print permk; print newline
  end else
    for j := 1 to n {Consider all poss. next items.}
      if ¬usedj then begin {Choose item j.}
        usedj := T; permi := j
        recursiveGenPerms(i+1,n)
        usedj := F {Now back up}
      end
    end
  end
```

## Next Permutation in Order

- Given an existing permutation  $a_1, \dots, a_n$  of  $\{1, \dots, n\}$ , how do we find the *next* one?
- Outline of procedure:
  - Find largest  $j$  such that  $a_j < a_{j+1}$ .
  - Find the smallest integer in  $a_{j+1}, \dots, a_n$  that is greater than  $a_j$ . Put it in position  $j$ .
  - Sort the remaining integers in  $a_j, \dots, a_n$  from smallest to largest. Put them at  $j+1$  through  $n$ .

## Next-Permutation Procedure

```
procedure nextPerm(a1, ..., an: perm. {1, ..., n})
  j:=n-1; while aj>aj+1 and j>0 do j:=j-1
  if j=0 then return {no more permutations}
  k:=n; while aj > ak do k:=k-1
  swap(aj, ak); r:=n; s:=j+1
  while r>s do begin
    swap(ar, as); r:=r-1; s:=s+1 end
```

## Combination Generator

- Suppose we want to generate all combinations of the elements of the set  $\{1, \dots, n\}$ .
  - Or any other set with  $n$  elements.
- A combination is just a subset.
  - And, a subset of  $n$  items can be specified using a bit-string of length  $n$ .
    - Each bit says whether the item is in the subset.
- Therefore, we can enumerate all combinations by enumerating all bit-strings of length  $n$ .

## Recursive Bit-String Enumerator

```
procedure recEnumBitStrings(soFar,n)
  {enumerate all strings consisting of soFar
  concatenated with a bit-string of length n}
  if n=0 then begin print soFar; return end
  enumBitStrings(soFar·'0', n-1)
  enumBitStrings(soFar·'1', n-1)
procedure enumBitStrings(n∈N)
  recEnumBitStrings( $\epsilon$ , n) {soFar=empty str}
```

## Generating the Next Bit String

- Note that this is essentially just a binary increment (“add 1”) operation.
- ```
procedure nextBitString(bn-1...b0: bit string)
  i:=0; {start at right end of string}
  while bi = 1 and i<n begin {trailing 1s}
    bi := 0; i:=i+1 end {change to 0s}
  if i=n then return “no more” else begin
    bi = 1; return bn-1...b0 end {chg. 0→1}
```

## Generating *r*-combinations

- How do we list all *r*-combinations of the set  $\{1, \dots, n\}$ ?
- Since the order of elements in a combination doesn't matter, we can always list them from smallest to largest.
- We can thus do it by enumerating the possible smallest items, then for each, enumerating the possible next-smallest items, etc.

## A Recursive *r*-comb. Generator

```
procedure recEnumRCombs(min,j,r,n)
  if j>n then print comb1,...,combn; return
  for i:=min to n-r+1 begin
    combj = i
    recEnumRCombs(i+1, j+1, r-1, n)
  end
```

## Generating Next $r$ -combination

```
procedure nextRComb ( $\{a_1, \dots, a_r\} \subset \{1, \dots, n\}$   
                    with  $a_i < a_{i+1}$ )  
  { Find the last item that can be inc'd }  
   $i := r$ ; while  $a_i = n - r + i$  do  $i := i - 1$   
   $a_i := a_i + 1$       { Increment it }  
  for  $j := i + 1$  to  $r$     { Set remaining items }  
     $a_j := a_j + j - i$     { to the subsequent #'s }  
  return  $\{a_1, \dots, a_r\}$ 
```