MA/CSSE 473 Day 14

Permutations wrap-up

Subset generation

(Horner's method)

MA/CSSE 473 Day 14

- Student questions
- Monday will begin with "ask questions about exam material" time.
- Exam details are Day 16 of the schedule page.
- Today's topics:
  - Permutations wrap-up
  - Generating subsets of a set
  - (Horner's method)

Permutations and order

number	permutation	number	permutation
0	0123	12	2013
1	0132	13	2031
2	0213	14	2103
3	0231	15	2130
4	0312	16	2301
5	0321	17	2310
6	1023	18	3012
7	1032	19	3021
8	1203	20	3102
9	1230	21	3120
10	1302	22	3201
11	1320	23	3210

- Given a permutation of 0, 1, …, n-1, can we directly find the next permutation in the lexicographic sequence?
- Given a permutation of 0..n-1, can we determine its permutation sequence number?
- Given n and i, can we directly generate the i^{th} permutation of 0, …, n-1?

Yesterday's Discovery

- Which permutation follows each of these in lexicographic order?
  - 183647520 471638520
  - Try to write an algorithm for generating the next permutation, with only the current permutation as input.

Lexicographic Permutation class

```
class Permutation:
    "Set current to the unpermuted list."
    def __init__(self, n):
        self.current = list(range(0, n))
        self.n = n
        self.more = True # This is not the last permutation.

    def swap(self, i, j):
        self.current[i], self.current[j] = self.current[j], self.current[i]

    def reverse(self, i, j):
        while j > i:
            self.swap(i, j)
            i += 1
            j -= 1
```

- These are the basics of the class setup.
- The ***next()*** method provides an iterator over the permutations. How should it get from one permutation to the next?

Main permutation method

```
def next(self):
    "return current permutation and calculate next one"
    if not self.more:
        return False
    returnValue = list(self.current)
    i = self.n - 2
    while self.current[i] > self.current[i + 1]:
        i -= 1 # This avoids array-out-of-bounds because
    if i == - 1: # in Python, a[-1] means a[len(a)-1]
        self.more = False
    else:
        j = self.n - 1
        while self.current[i] > self.current[j]:
            j -= 1
        self.swap(i, j)
        self.reverse(i + 1, self.n - 1)
    return "".join([str(v) for v in returnValue])
```

More discoveries

- Which permutation follows each of these in lexicographic order?
  - 183647520 471638520
  - Try to write an algorithm for generating the next permutation, with only the current permutation as input.
- If the lexicographic permutations of the numbers [0, 1, 2, 3, 4] are numbered starting with 0, what is the number of the permutation 14032?
  - General algorithm? How to calculate efficiency?
- In the lexicographic ordering of permutations of [0, 1, 2, 3, 4, 5], which permutation is number 541?
  - General algorithm? How to calculate efficiently?
  - Application: Generate a random permutation

Memoized factorial function

```
class FactTable:  #memoized factorial function

    def __init__(self):
        self.table = [120, 24, 6, 2, 1, 1]
        self.max = 5

    def get(self, n):
        if n <= self.max:  # it's already in thr table
            return self.table[self.max - n]
        for i in range(self.max+1, n+1):  # put factorials in table
            self.table= [i*self.table[0]] + self.table
        self.max = n
        return self.table[0]

ft = FactTable()
```

Find a permutation's sequence

```python
def permNumber(p):
    """assumes that p is a permutation of 0..n-1.
    returns k such that p is the kth lexicographic
    permutation of those numbers."""
    p = list(p) # make a copy
    n = len(p)
    factList = [ft.get(i) for i in range (n-1,-1,-1)]
    sum = 0
    for i in range(n):
        sum += p[i] * factList[i]
        for j in range(i + 1, n):
            if p[j] > p[i]:
                p[j] -= 1
    return sum
```

Find permutation from sequence

```python
def kthPermutation(s, k):
    """return the kth lexocographic permutation of the
    distinct elements in list s.  Inverse of permNumber()"""
    s = list(s)
    result = []
    factTable = [ft.get(i) for i in range (len(s)-1,-1,-1)]
    for divisor in factTable:
        multiple = k // divisor
        k = k % divisor
        element = s[multiple]
        result.append(element)
        s.remove(element)
    return result
```

Bottom-up, “numeric order”, binary reflected Gray code

SUBSET GENERATION

Generate all Subsets of a Set

- Sample Application:
  - Solving the knapsack problem
  - In the brute force approach, we try all subsets
- If A is a set, the set of all subsets is called the **power set** of A, and often denoted 2^A
- If A is finite, then $\left|2^A\right| = 2^{|A|}$
- So we know how many subsets we need to generate.

Generating Subsets of $\{a_1, \ldots, a_n\}$

- Decrease by one (bottom up):
- Generate S_{n-1}, the collection of the 2^{n-1} subsets of $\{a_1, \ldots, a_{n-1}\}$
- Then $S_n = S_{n-1} \cup \{ S_{n-1} \cup \{a_n\} : s \in S_{n-1}\}$
- Numeric approach:
  - Each subset of $\{a_1, \ldots, a_n\}$ corresponds to an bit string of length n, where the i^{th} bit is 1 iff a_i is in the subset

Details of numeric approach:

- Each subset of $\{a_1, \ldots, a_n\}$ corresponds to a bit string of length n, where the J^{th} bit is 1 if and only if a_J is in the subset

```
def allSubsets(a):
  n = len(a)
  subsets=[]
  for i in range(2**n):
    subset = []
    current = i
    for j in range (n):
      if current % 2 == 1:
        subset += [a[j]]
      current /= 2
    subsets += [subset]
  return subsets
```

Output for a=[1, 2, 3]:

```
[[], [1],
[2], [1, 2],
[3], [1, 3],
[2, 3],
[1, 2, 3]]
```

Gray Codes

- Named for Frank Gray
- An ordering of the 2^n n-bit binary codes such that any two consecutive codes differ in only one bit
- Example:
  000, 001, 011, 010, 110, 111, 101, 100
- Note also that only one bit changes between the last code and the first code.
- A Gray code can be represented by its **transition sequence:** indicates which bit changes each time
  In above example: 0, 1, 0, 2, 0, 1, 0
- Traversal of the edges of a (hyper)cube.
- In terms of subsets, the transition sequence tells which element to add or remove from one subset to get the next subset

Recursively Generating a Gray Code

- Binary Reflected Gray Code
- $T_1 = 0$
- $T_{n+1} = T_n$, n, $T_n^{reversed}$
- Show by induction that $T_n^{reversed} = T_n$
- Thus $T_{n+1} = T_n$, n, T_n

Iteratively Generating a Gray Code

- We add a parity bit, p.
- Set all bits (including p) to 0.

```
while True:
    printSet(a)
    p = 1 - p #flip the parity bit
    if p == 1:
        j = 0
    else:
        j = 1
        while a[j-1]==0:  # find position to the
            j += 1        # left of the rightmost 1
        if j == n:
            break
    a[j] = 1 - a[j]   # flip this bit.
```

* Based on Knuth, Volume 4, Fascicle 2, page 6.

Side road: Polynomial Evaluation

- Given a polynomial
  $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$
- How can we efficiently evaluate p(c) for some number c?
- Apply this to evaluation of "31427894" or any other string that represents a positive integer.
- Write and analyze (pseudo)code

Horner's method code

```
def polyEval(coefficientList, val):
    "coefficientList[i] is the coefficient of x^i"
    "Uses Horner's method to evaluate polynomial at val"

    result = 0
    for power in range(len(coefficientList)-1, -1, -1):
        result = result * val + coefficientList[power]
    return result

print (polyEval([4, 0,-7, 0, 3, 6], 3))
```

Decrease by a constant factor

Decrease by a variable amount

OTHER DECREASE-AND-CONQUER ALGORITHMS

Fake Coin Problem

- We have n coins
- All but one have the same weight
- One is lighter
- We have a balance scale with two pans.
- All it will tell us is whether the two sides have equal weight, or which side is heavier
- What is the minimum number of weighings that will guarantee that we find the fake coin?
- Decrease by factor of two.

Decrease by a Constant Factor

- **Examples that we have already seen:**
  - Binary Search
  - Exponentiation (ordinary and modular) by repeated squaring
  - Recap: Multiplication à la Russe (The Dasgupta book that I followed for the first part of the course called it "European" instead of "Russian")
    - Example

11	13
5	26
~~2~~	~~52~~
1	104
	143

Then strike out any rows whose first number is even, and add up the remaining numbers in the second column.

Decrease by a variable amount

- Search in a Binary Search Tree
- Interpolation Search
  - See Levitin, pp190-191
  - Also Weiss, Section 5.6.3
- Median Finding
  - Find the k^{th} element of an (unordered) list of n elements
  - Start with quicksort's partition method
  - Best case analysis