CS472 Module 4 Part A - Decrease and Conquer

Athens State University

February 9, 2017

Outline

Contents

1 What is "Decrease and Conquer"

Brute force vs. decrease and conquer

- *Brute-force algorithms*: A method of computation wherein all permutations of a problem are tried manually until one is found that provides a solution, in contrast to the implementation of a more intelligent algorithm.
- *Decrease and conquer*: A method of computation where a problem is successively broken down into single subprolems, whose solution may be extended to obtain the solution to the original problem.

Three divide and conquer approaches

- Decrease by a constant (usually by 1)
  - insert sort
  - toplogical sorting
  - algorithms for generating permutations and subsets
- Decrease by a constant factor (usually by half)
  - binary search and bisection
  - compute exponentiation by squaring
  - multiplication a la russe

- Decrease by a variable factor
  - Euclid's algorithm
  - selection by partition
  - Nim-like games

Compare with brute force

Consider exponentiation

- by brute force
- by decrease by one (same as brute force?)
- divide and conquer
- decrease by constant factor

Recall that recurrence relation for computing a^n has the form:

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive} \\ (a^{(n-1)/2})^2 * a & \text{if } n \text{ is odd} \\ 1 & \text{if } n = 0 \end{cases} \tag{1}$$

If we are measuring the performance of the algorithm by measuring the number of multiplications, then we would expect that the algorithm is in $\Theta(log(n))$. Can you explain why this is so?

2 A few examples: Decrease by constant

And let us reconsider insertion sort

To sort an array $A[0..n-1]$, recursively sort $A[0..n-2]$ and then insert $A[n-1]$ among the sorted A[0..n-2]

- But most people implement insertion sort as iterative algorithm
  - Processes the array in a bottom-up manner

And let us reconsider insertion sort

Algorithm 1: Insertion Sort

```
Input: An array A[0..(n-1)] of n orderable elements
Output: Array A sorted in nondecreasing order
for i ← [1..(n-1)] do
    v ← A[i];
    j ← i-1;
    while j ≥ 0 and A[j] > v do
        A[j+1] ← A[j];
        j ← j - 1;
    A[j+1] ← v;
```

And let us reconsider insertion sort

- Time efficiency
  - Worst case:
$$C_{worst}(n) = n(n-1)/2 \in \Theta(n^2)$$
  - Avg case:
$$C_{avg}(n) = n^2/4 \in \Theta(n^2)$$
  - Best case:
$$C_{best}(n) = n - 1 \in \Theta(n)$$
- Space efficiency: in-place
- Stable sort
- Best elementary sorting algorithm overall

2.1 Generating Permutations: Minimal Change

Generating permutations: minimal change

- A decrease by one algorithm
- If $n = 1$ return 1
- Otherwise, generate recursively the list of all permutations of 1,2, ..., $n-1$
  - Then insert n into each of those permutations by starting with inserting n into 1,2,...,$n-1$ from right to left
  - Switch direction for each new permutation

Generating permutations: minimal change

Algorithm 2: Permutation Generator

Input: An integer n and an array $A[0 \dots m]$ where $m \geq n$
Output: All permutations of the array
if $n = 1$ **then**
  write A
else
  for $i \leftarrow$ *[1 ... n]* **do**
    `permute(`*A, n - 1*`);`
    if *n is odd* **then**
      swap A[1] and A[n];
    else
      swap A[i] and A[n];

Generating permutations: minimal change: example

Suppose $n = 1$

- Start: 1
- Insert 2 into 1 right to left: 12 21
- Insert 3 into 12 right to left: 123 132 312
- Insert 3 into 21 left to right: 321 231 213

2.2 Generating Subsets:Remember Gray Code?

Generating Subsets: Gray Code

Decimal	Binary	Gray	Gray in decimal
0	000	000	0
1	001	001	1
2	010	011	3
3	011	010	4
4	100	110	6
5	101	111	7
6	110	101	5
7	111	100	4

Notice that the Gray code for decimal 7 rolls over to decimal 0 with only one switch change. This is called the "cyclic" property of a Gray code. In the standard Gray coding the least significant bit follows a repetitive pattern of 2 on, 2 off (... 11001100 ...); the next digit a pattern of 4 on, 4 off; and so forth.

Generating Subsets: Applications of Gray Code

- Position encoders are devices that convert the angular position of a shaft or axle to a digital signal
  - Gray codes are used in these devices to avoid having misreads occur when several bits change in the binary representation of an angle
- Solving the Towers of Hanoi program

  - Because of the math behind Gray Codes, one can use a Gray code as a solution guide to this problem

- Circuit minimization
  - A Karnaugh map (K-map) is a method used to simplify boolean expressions. The cells of a K-map are ordered using a Gray code
  - We will discuss K-maps in detail in an upcoming lecture

Generating subsets: Binary reflected Gray code generator

Minimal-change algorithm for generating 2^n bit strings corresponding to all subsets of an n element set where $n > 0$.

Algorithm 3: Binary Reflected Gray Code

---

if $n = 1$ **then**
  Make list L of two bit strings 0 and 1;
else
  Generate recursively list L_1 of bit strings of length $n - 1$;
  Copy list L_1 in reverse order to get list L_2;
  Add 0 in front of each bit string in list L_1;
  Add 1 in front of each bit string in list L_2;
  Append L_2 to L_1 to get L;
return L;

---

3 A few examples: Decrease by constant factor

Recall our friend binary search

Algorithm 4: Binary search

Input: A sorted array A[0..(n-1)] and a search key K
Output: An index m indicating where K is located or -1 if not found

```
left ← 0;
right ← n-1;
while l ≤ r do
    m ← floor((left + right) / 2]);
    if K = A[m] then
        return m;
    else if K < A[m] then
        right ← m - 1;
    else
        left ← m + 1;
return -1;
```

The algorithm operates in three steps:

- If the item we are seeking is the middle item, stop.
- Otherwise
  1. *Divide* the container into two parts that are about half as large. If the search item is smaller than the middle item of the original list, choose the left-hand sub-container, else choose the right-hand sub-container.
  2. *Conquer* the smaller problem by checking to see if the search item is in the sub-container. Use recursion to do this unless the sub-container is sufficiently small.
  3. *Obtain* the solution to the larger container from the smaller container.

Recall our friend binary search

- Very fast:

$$C_w(n) = 1 + C_w(\lfloor(\rfloor n/2)), C_w(1) = 1$$

  - Closed form solution:

$$C_w(n) = \lceil(lg(n_1))\rceil$$

- Optimal algorithm for searching for sorted array
- In reality, is degenerate example of divide-and-conquer
- Has a counterpart in numerical analysis called bisection method for solving equations in one unknown

4 A few examples: Variable-size decrease algorithms

While we're on the topic of numerical analysis

A common problem in number theory is finding the greatest common divisor (GCD) between two integers m and n

- The Greek mathematican Euclid created one of the earliest instances of a recursive algorithm for finding the GCD of two numbers:
$$gcd(m, n) = gcd(n, m * mod(n))$$
- One can prove that the size, measured by the second number, decreases by half after two consecutive iterations.

Euclidean Algorithm

Algorithm 5: Euclidean Algorithm

Input: Integers a and b
Output: The GCD of a and b
if $b = 0$ **then**
  return a;
else
  return `Euclid` (b, a MOD b);

The Selection Problem

- Find the k-th smallest element in a list of n numbers
- Find the *median* $k = \lceil n/2 \rceil$ in a list of n numbers
- One solution is sorting: sort and return the k-th element

The Selection Problem: a faster algorithm?

Consider how one might *partition* an array A[0..(n-1)]

	s	
all are $\leq$ A[s]	A[s]	all are $\geq$ A[s]

Repeat until $s = k - 1$:

- If $s = k - 1$ then we have a solution.
- If $s > k - 1$ then look for the k-th smallest element to the left.
- If $s < k - 1$ then look for the $k - s$-th smallest element to the right

The Selection Problem: How do we partition?

One approach: Lomuto's Partitioning Algorithm

- Scan the array left to right, keeping the array in three sections
  - A set less than some value p, where p is the *pivot* of the partition. Put the pivot at the start of the array
  - A set greater than or equal to p
  - A set of values where we are uncertain if they are less than or greater than or equal to the pivot
- On each iteration, decrease the size of the unknown section by one element until it is empty. Achieve a partition by exchanging the pivot with the element in the split position s

The Selection Problem: Quickselect

Algorithm 6: Hoare's quickselect algorithm

```
Input: A list, an index left, an index right, and list length n
Output: The n-th smallest element within left and right range
if the list contains only one element then
    return that element;
Select a pivot between left and right;
pivot ← partition (list, left, right, pivot);
if pivot = n then
    return list;
else if n < pivot then
    return quickselect (list,list, pivot-1, n);
else
    return quickselect (list, pivotIndex + 1, right, n);
```

The Selection Problem: Quickselect: No Tail Recursion

Algorithm 7: Hoare's quickselect algorithm

```
Input: A list, an index left, an index right, and list length n
Output: The n-th smallest element within left and right range
if the list contains only one element then
    return that element;
Select a pivot between left and right;
pivot ← partition (list, left, right, pivot);
if pivot = n then
    return list;
else if n < pivot then
    right ← pivot - 1;
else
    left ← pivot + 1;
```

What do we mean by *tail recursion*? Note that there is no further executable statements in the recursive version of *Quickselect*. In those cases, it is very simple to replace the recursion (as shown in the second version of *Quickselect*).

5 Key Points

Key Points

- What is decrease and conquer?
- What are the three different cases for this meta-heuristic?
- Key examples