Design and Analysis of Algorithms

L14: Divide and Conquer Advantages & Disadvantages Decrease & Conquer

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT/KSSEM
rprustagi@ksit.edu.in

Resources

- Text book 1: Sec 5.1-5.3 - Levitin

Divide and Conquer

- Advantages
  - Solution becomes easier as problem is divided into smaller size
  - Efficient compared to brute force approach
    - Binary search
    - Large number multiplication
    - Matrix Multiplication
    - Mergesort, quicksort
  - Smaller problems can be solved in parallel
    - Can improve algorithm running time
  - Can make effficient use of Caches
    - Small problem can be solved in cache itself

Divide and Conquer

- Dis-advantages
  - Makes use of recursion heavily, thus computation may slow down a bit
  - Usage of stacks (by recursion) requires more memory
  - Implementation of recursion requires clarity of thought. At times, simple iteration is good enough
    - e.g. print all N-digit decimal numbers
  - Even a minuscle error in recursion termination condition may result in infinite loop (invocation)
    - Program will run out of memory (stack)
  - Can not solve a problem where recursion depth is more than system allows.
  - When subproblems may repeat (e.g. same sub matrix)
    - Then it may do duplication of computation.

Decrease and Conquer

- Reduce the problem instance to a smaller instance problem of the same type
- Solve the smaller instance problem
- Use the solution of smaller instance problem to solve the original bigger instance problem
- Implementation choices
  - Top down (use recursion) or bottom up
  - Incremental approach /inductive solution

Differences with Divide and Conquer

- Divide and Conquer
  - Given problem instance divided into smaller instances
  - All smaller instances are solved (conquered)
  - Solutions of smaller instances are merged
  - Recursion : $T(n)=aT(n/b)+f(n)$
- Decrease and Conquer
  - Given problem instance reduced to a single smaller instance.
  - Only one smaller instance problem is to be solved
  - Use smaller instance problem to solve bigger instance problem
  - Recursion $T(n)=T(m)+f(n)$,where $m<n$

Types of Decrease and Conquer

- *A*:Decrease by a constant value `c` (`n → n-c`)
  - Usually decrease is by 1
  - Examples
    - Insertion sort
    - Graph traversal (DFS, BFS)
    - Topological sort
    - Generating permutations, subsets
- *B*:Decrease by a constant factor `c` (`n → n/c`)
    - Still only 1 sub-problem to solve
  - Usually decreases by half i.e. divide in equal half (`c=2`)
  - Examples:
    - Binary search
    - Exponentiation by squaring

Types of Decrease and Conquer

- *C:* Decrease by a variable size c_i ($n \rightarrow n-c_i$) at i^{th} step
  - The size decrease varies on each iteration
    - Depends upon input problem instance
  - Examples
    - Euclid's algorithm (greatest common divisor)

```
gcd(m,n) → gcd(n,m mod n)
```

Alternatively

```
if m>n
   gcd(m-n,n)
else
   gcd(n, m-n)
```

  - Selection by partition

Types of Decrease and Conquer

- `Nim-like` games (2 player)
  - A pile of `n` discs
  - Each player picks min `1`, max `m` discs
  - The person who picks last is winner.
  - Soln:
  - when will 1st person to pick loses?
    - when `n=`

Differences with Other Approaches

- **Problem instance: compute x^n**
- Decrease and Conquer approach

  $T(n) = T(n-1)+1 = n-1$
- Brute force approach
  - Multiply x by itself $n-1$ times

  $T(n) = n-1$
- Divide and Conquer approach
  - Multiply $x^{n/2}$ by $x^{n/2}$

  $T(n) = 2T(n/2)+1 = n-1$
- Decrease by a constant factor
  - Multiply k times $x^{n/k}$ by itself

  $T(n) = T(n/k)+k-1 = n-1$

D&C Appln:Celebrity Problem

- Q10 (Levitin):
  - A celebrity among a group of N people is defined as
    - a person who knows nobody but
    - is known to everybody else.
  - Identify the celebrity by only asking the questions to the people of the form:
    - "Do you know him/her?"
  - Design an efficient algorithm to identify a celebrity or determine that the group has no such person.
  - How many questions does your algorithm need to ask in the worst case?

Celebrity Problem

- Approach 1: Using Adjacency matrix
  - Build a graph with adjacency matrix A
  - Ask each person if he knows all other persons
    - Total num of Qs: $n(n-1) = O(n^2)$
    - $A[i,j]=1$ if i^{th} person knows person j
      - 0 otherwise
  - Find a column k, such that $\forall i$
    - $\Sigma A(i,k)=n-1$, and
    - $\Sigma A(k,i)=0$
- person k is celebrity

Celebrity Problem

- Approach 2 : Using Adjacency List
  - Build a graph with Adjacency List
  - Ask each person if he knows all other persons
    - Total num of Qs: $n(n-1) = O(n^2)$
    - Draw an edge (i, j) if person i knows person j.
  - Find a node k such that its
    - indegree is $(n-1)$, and
    - outdegree is 0.
- person k is celebrity

Celebrity Problem

- Approach 3: Using Decrease and conquer.
- Design function `celebrity(N)` which returns `k`
  - if `k` is non-zero, then `k` is celebrity
  - if `k` is zero, there there is no celebrity.
- `celebrity(N)` Using Decrease and conquer.
  - Invoke `k=celebrity(N-1)`
    - if `k=N`, and `N` does not know anyone, `N` is celebrity
      - Complexity: O(N)
    - if `k≠N`, and `N` knows `k`, `k` is celebrity, complexity `O(1)`
    - Else no celebrity
- Time Complexity:

  $T(n) = T(n-1)+O(n) = O(n^2)$

Celebrity Problem

- Approach 4: Using stacks
- Push all persons(elements) on the stack
  - stack size is `N`
- Repeat until stack size becomes `1`
  - pop two persons `A`, `B` from stacks
  - If `A` knows `B`, then `A` is not a celebrity
    - Push `B` on the stack
  - If `A` doesn't know `B`, then `B` is not a celebrity
    - Push `A` on the stack.
- The last person on the stack is celebrity (if does not know any one)
- Complexity: $3N-1 = O(N)$
  - `2N` pop operations, `N` push operations

Summary

- Advantages and disadvantages of Divide and Conquer
- Decrease and conquer approach