Design and Analysis of Algorithms

L09: Algo Design Ideas

Dr. Ram P Rustagi
Sem IV (2020-Even)
Dept of CSE, KSIT/KSSEM
rprustagi@ksit.edu.in

Resources

- Martin Richards
  - Notes on Data Structures and Algorithms
- Text book 2: Horowitz
- Text book 1: Levitin
- https://visualgo.net/en

Ideas on Algorithm Design

- Recognize a variant on known problem
- Reduce to a simpler problem
- Divide and Conquer
- Estimation of cost by recurrence relation
- Dynamic Programming
- Greedy Algorithm
- Back tracking
- Hill Climbing
- Identify wasted work in a simple method
- Find a mathematical lower bound
- Million Monkey method

Variant on a Known Problem

- Try to identify a related problem whose solution is known
- Use solution of one problem to solve essentially tricky part of other problem.
- Solve the problem in totality
- Example 1: multiply two numbers
  - Known variant: add two numbers
  - How to use addition of two numbers to achieve multiplication
- Example 2: Class assignment submission
  - variant: Identify classmate who has done similar
  - Use the solution with different input params

Reduction to a Simpler Problem

- Typically works where induction methods are used to show correctness proof
- Recursive functions take this appproach
- Example: Compute n^2

  $n^2 = (n-1)^2 + ??$

  $\quad = (n-1)^2 + 2n - 1$
- Example: factorial n

  $n! = n*(n-1)!$
- Example: GCD computation

  `gcd(y,x)= gcd(x,r)` where `r=y%x`

Divide and Conquer Algo

- Divide (break) the problem (size `n`) into similar sub problems
  - Size of sub problems should be some factor of original e.g. `n/c`
    - When small enough, solve by brute force
- Conquer (Solve) the sub-problem
  - Use recursion to solve small problem
- Combine (Merge) the solution of sub-parts
- The cost is
  - cost of breaking
  - cost of solving subproblem
  - cost of combining

Cost Estimation by Recurrence

- Use recurrence relation to find the cost of operation
  - Once the cost is known, then algorithm framework is worked out
  - Design the algorithm on how to break the given problem
- Example: Mergesort

  $f(n) = 2f(n/2)+n$

  $\rightarrow f(n) = O(n\log_2 n)$
- Example: Word search in English dictionary

  $f(n) = f(n/k) + c$

  $\rightarrow f(n) = \log_k n$

Dynamic Programming

- Build a table of solutions to smaller versions of the problem
- Work towards the solution by using this table
- Example: Binomial Coefficient

```
    1         (a+b)^0
   1 1        (a+b)^1
  1 2 1       (a+b)^2
 1 3 3 1      (a+b)^3
1 4 6 4 1     (a+b)^4
```

- Example: Computing n^2

```
        1            1^2
      1+2+1           2^2
    1+2+3+2+1          3^2
  1+2+3+4+3+2+1         4^2
1+2+3+4+5+4+3+2+1        5^2
```

Greedy Algorithms

- Useful when some sort of optimizations is involved.
- Basic idea:
  - Perform whatever operation contributes most towards the final goal
  - Next step will be same approach after the previous step
  - Note: Final solution may not always be optimal
- Example:
  - Eating in restaurant: get whatever available now
    - No waiting is to be done
  - Taking exam (you know all answers) but time is less
    - Which questions to start writing

Backtracking

- When algorithm involves search (explore)
  – backtracking is useful
- Split the search procedure in multiple paths(parts)
  – Start with one path
  – If solution not found, and path ends,
  – Backtrack to previous starting point and
    - search on next path
- Example:
  – Finding a solution to the maze (childhood comicbooks)
  – Chessboard: queens placement
  – Chessboard: Knights traversal

Hill Climbing

- Generally, used in optimization problems
- Start with some feasible (non-optimal) solution
- Incrementally work towards improving the solution
- May not always find the optimal solution
- Analogy: Climbing to hilltop in hilly region
- Example:
  - Getting a job(placement)
    - Start with some placement (job)
    - Work towards a better placement (better company)

Identify Wasted Work

- Design a simple solution to the problem
- Analyze the solution and
  – identify critical costly part of the solution
- Attack the weakness of critical part to improve the solution
- Example (real life):
  – Phone converstation, and you are put on hold
  – can do some useful work while on call hold
- Example (CS): CPU Scheduling
  – when a processing is waiting for I/O
  – other process is assigned CPU

Seek Mathematical Lower Bound

- Establish a proof that some task must take at least certain minimum time
- Use this insight to design the algorithm
- A properly proved lower bound can prevent wasted time seeking improvement
- Example: Sorting `n` numbers:
  - min time: `ceil(log`$_2$ `n!)` comparisons
  - Consider `n=4`, min time = `log`$_2$`(24)=5`
  - Consider `n=5`, min time = `log`$_2$`(120)=7`
  - Design the algorithm

Million Monkey (MM) Method

- Problem: Give one typewriter each to million monkeys (random character press) and a milliion years, and one of the monkey will write the Shakespear play.
  - Idea: give a problem to a group of researchers and sufficient time, they will be able to find the solution of the problem.
- Example: expected output '`hello`'.
  - Assume typewriter has 50 keys
  - Probability of not typing '`hello`' = $1-(1/50)^5$
  - Probability of not typing '`hello`' by million monkeys in 10000 tries is
    - $(1-(1/50)^5)^{10} = 1.2*10^{-14}$
  - Probability of typing '`hello`' = $1-1.2*10^{-14} \approx 1$

Exercises-A

- Given 4 numbers: `a, b, c, d`
  - Sort them using $\log_2 4!$

    $=\log_2 24$

    $=5$ comparisons

Sort 4 numbers

abcd	~~bacd~~	cabd	~~dabc~~
~~abdc~~	~~badc~~	cadb	~~dacb~~
acbd	~~bcad~~	~~cbad~~	~~dbac~~
acdb	~~bcda~~	~~cbda~~	~~dbca~~
~~adbc~~	~~bdac~~	cdab	~~dcab~~
~~adcb~~	~~bdca~~	~~cdba~~	~~dcba~~

Consider two numbers `a, b`

if `a < b`

    `12` combinations goes out

else

    other `12` combinations goes out

if `c < d`

    6 combinations goes out

if `b < c`

    ?? what happens. Can we use it?

Exercise-B

- Given 5 numbers: `a, b, c, d, e`
  - Sort them using $\log_2 5!$

    $= \log_2 120$

    $= 7$ comparisons

Summary

- Ideas for algorithm design
  - Recognize a variant on known problem
  - Reduce to a simpler problem
  - Divide and Conquer
  - Estimation of cost by recurrence relation
  - Dynamic Programming
  - Greedy Algorithm
  - Back tracking
  - Hill Climbing
  - Identify wasted work in a simple method
  - Find a mathematical lower bound
  - Million Monkey method