CSE204L Data structures and Algorithms

Analysing Recursive Algorithms

Analysis of recursive and non-recursive algorithms

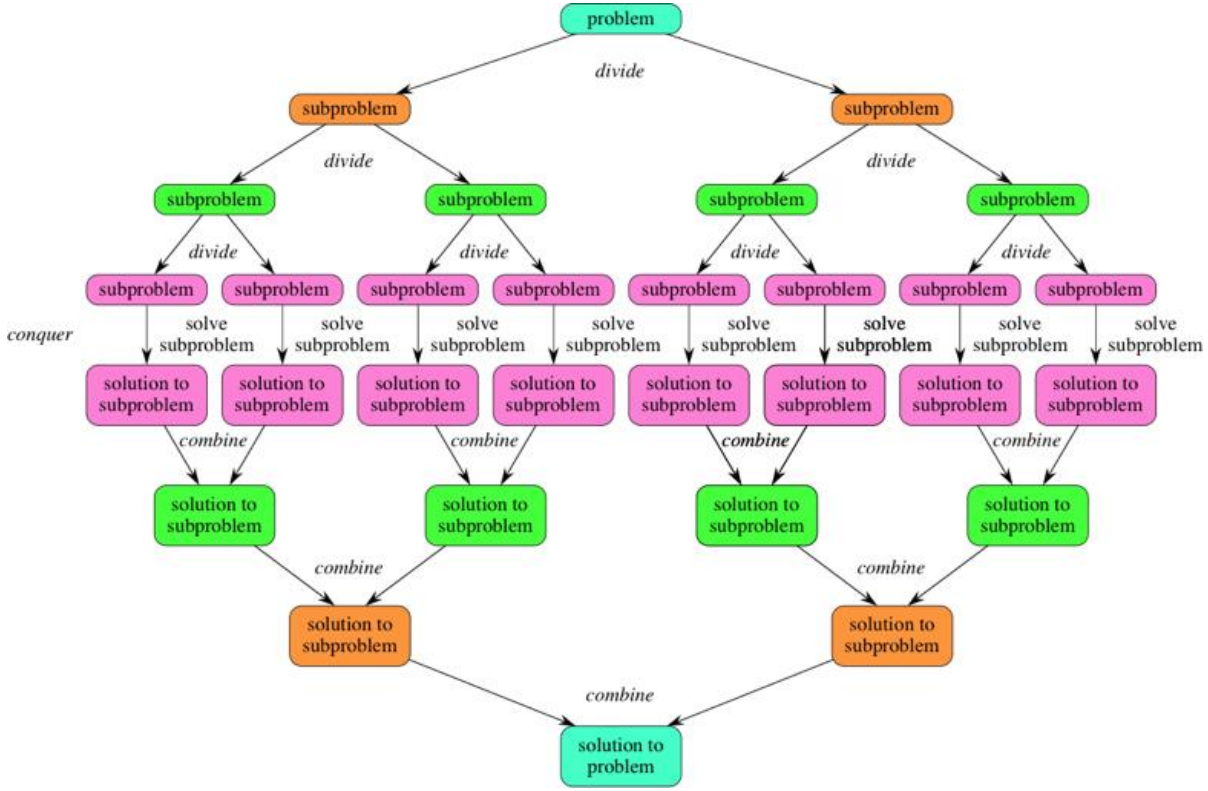
- Non-recursive → Frequency count method
  - Sequential instruction $\rightarrow$ Constant complexity $\rightarrow O(1)$
  - Loops $\rightarrow$ if depends on input size $\rightarrow O(n)$
  - Nested loops $\rightarrow$ if depends on input size $\rightarrow O(n^l)$ where l is the level of nesting
  - Branching $\rightarrow$ Maximum frequency count in any of the block under if/else-if/else-if/$\cdots$/else
- Recursive→ We should know recurrence relations!!

Problem solving strategies

- Approach/design to solve a computational problem
  - Greedy method
  - Divide and conquer
  - Dynamic programming
  - Backtracking
  - Branch and bound
- There is no thumb-rule, only some guidelines

Divide and Conquer

- Conditions:
  - Sub-problem must be same as the problem
  - Recursive in nature
  - Recurrence relations are to be familiarised



Recursive Functions and Recurrences

- Divide and conquer algorithms would generally be recursive in nature
- The time functions of recursive functions can be obtained as a recurrence relation
- To obtain the time complexity of recursive functions, the corresponding recurrence relations need to be solved

Solving Recurrences

- There are three methods
  1. Substitution method
  2. Tree Method
  3. Master’s method
- Explanation by examples

Recurrence Example -1

- $T(n) = \begin{cases} 1, & n = 0 \\ T(n-1) + 1, & n > 0 \end{cases}$

```
void foo(int n)
{
   if(n > 0)
   {
      printf("%d", n);
      foo(n-1);
   }
}
```

Recurrence Example -2

- $T(n) = \begin{cases} 1, & n = 0 \\ T(n-1) + n, & n > 0 \end{cases}$

```
void foo(int n)
{
   if(n > 0)
   {
      for(i=0; i<n; i++)
      {
         printf("%d", n);
      }
      foo(n-1);
   }
}
```

Recurrence Example -3

```
void foo(int n)
{
   if(n > 0)
   {
      for(i=0; i<n; i=i*2)
      {
         printf("%d", n);
      }
      foo(n-1);
   }
}
```

- The loop will run
  - 1 time when $n = 2$,
  - 2 times when $n = 4$,
  - 3 times when $n = 8$,
  - 4 times when $n = 16$,
  - $\log n$ times for any n
- $T(n) = \begin{cases} 1, & n = 0 \\ T(n-1) + \log n, & n > 0 \end{cases}$

Observe

- Wkt:
- $T(n) = T(n-1) + 1 = O(n)$
- $T(n) = T(n-1) + n = O(n^2)$
- $T(n) = T(n-1) + \log n = O(n \log n)$
- What if:
- $T(n) = T(n-2) + n$? It will be $O\left(\frac{n^2}{2}\right)$, but that is also $O(n^2)$
- Similarly $T(n) = T(n-p) + n = O(n^2)$ for any integer p
- But What if $T(n) = qT(n-p) + n$?

Recurrence Example -4

- $T(n) = \begin{cases} 1, & n = 0 \\ 2T(n-1) + 1, & n > 0 \end{cases}$

```
foo(int n)
{
   if(n > 0)
   {
     printf("%d", n);
     foo(n-1);
     foo(n-1);
     }
}
```

Dividing functions- Example-1

- The examples discussed so far were decreasing functions
- There is another category, dividing functions

```
Alg(int n)
{
      if (n > 1)
      {
            print(n)
            Alg(n/2)
      }
}
```

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

Dividing functions- Example-2

- $T(n) = \begin{cases} 1, \ n = 1 \\ T(\frac{n}{2}) + n, \ n > 1 \end{cases}$

Dividing functions- Example-3

- $T(n) = \begin{cases} 1, \ n = 1 \\ 2T(\frac{n}{2}) + n \ \ n > 1 \end{cases}$

Special Types of Recurrences

- If the RHS of the recurrence consists of summation of more than one different recursive terms
  - Substitution method→ Lengthy
  - Master's method→ Fails
  - Recurrence tree method is ideal

Recurrence Tree Method Advanced Exercises

1. $T(n) = \begin{cases} T(\frac{n}{2}) + T(\frac{n}{2}) + n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$ Answer $\Rightarrow O(n \log n)$

Already Solved by substitution method and tree method (Dividing example-3)

Solution Link:

https://www.youtube.com/watch?v=q2_GVA9nXXY&list=PLPzfPcir5uPREJTKg1bsDIOh2PB0VAVNs&index=18

Recurrence Tree Method Advanced Exercises

2. $T(n) = \begin{cases} T(\frac{n}{3}) + T(\frac{2n}{3}) + n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$

- Keep smaller term on left and larger term on right of tree
- Answer $\Rightarrow O(n \log_{\frac{3}{2}} n)$

Solution Link:
https://www.youtube.com/watch?v=tCDFjasdHGo&list=PLPzfPcir5uPREJTKg1bsDIOh2PB0VAVNs&index=19

Recurrence Tree Method Advanced Exercises

3. $T(n) = \begin{cases} T(\frac{n}{5}) + T(\frac{4n}{5}), n > 1 \\ 1, n = 1 \end{cases}$

- Answer $\Rightarrow O(n \log_{\frac{5}{4}} n)$

Solution Link:
https://www.youtube.com/watch?v=CmIetFGijco&list=PLPzfPcir5uPREJTKg1bsDIOh2PB0VAVNs&index=20

4. $T(n) = \begin{cases} 2T(\frac{n}{2}) + n^2, \text{ if } n > 1 \\ 1, \text{ if } n = 1 \end{cases}$

- Answer $\Rightarrow O(n^2)$

Solution link:
https://www.youtube.com/watch?v=_fvvsnB_mw4&list=PLPzfPcir5uPREJTKg1bsDIOh2PB0VAVNs&index=21

Masters Theorem for Decreasing functions

- $T(n) = aT(n-b) + f(n), a > 0, b > 0$
- Case1:
  - If $a < 1, T(n) = O\big(f(n)\big)$
- Case2:
  - If $a = 1, T(n) = O(nf(n))$
- Case3:
  - If $a > 1, T(n) = O(a^{\frac{n}{b}} f(n))$

Masters Theorem (Decreasing functions) Examples

- These examples are already solved using tree method and substitution methods
- Here we will verify them using Masters method

1. $T(n) = T(n-1) + 1$ → Ans: $O(n)$
2. $T(n) = T(n-1) + n$ →Ans: $O(n^2)$
3. $T(n) = T(n-1) + \log n$ → Ans: $O(n \log n)$
4. $T(n) = 2T(n-1) + 1$ → Ans: $O(2^n)$
5. $T(n) = 2T(n-1) + n$ → Ans: $O(n2^n)$

Masters theorem for dividing functions

$$T(n) = aT(n/b) + f(n),$$

Let $a \geqslant 1, b > 1, f(n) = \Theta(n^k \log^p n)$

Case 1: if $\log_b a > k$, then $\Theta(n^{\log_b a})$

Case 2: if $\log_b a = k$,

if $p > -1$, then $\Theta(n^k \log^{p+1} n)$

if $p = -1$, then $\Theta(n^k \log \log n)$

if $p < -1$, then $\Theta(n^k)$

Case 3: if $\log_b a < k$

if $p \geqslant 0$, then $\Theta(n^k \log^p n)$

if $p < 0$, then $O(n^k)$

Masters theorem examples

(1) $T(n) = 2T(n/2) + 1$

(2) $T(n) = 4T(n/2) + n$

(3) $T(n) = 8T(n/2) + n \log n$

(4) $T(n) = 2T(n/2) + n$

(5) $T(n) = 2T(n/2) + \dfrac{n}{\log n}$

(6) $T(n) = 2T(n/2) + \dfrac{n}{\log^2 n}$

(7) $T(n) = T(n/2) + n^2 \log^2 n$

(8) $T(n) = 4T(n/2) + \dfrac{n^3}{\log^2 n}$

Answers

1. $\Theta(n)$
2. $\Theta(n^2)$
3. $\Theta(\mathrm{n}^3)$
4. $\Theta(n \log n)$
5. $\Theta(n \log \log n)$
6. $\Theta(n)$
7. $\Theta(n^2 \log^2 n)$
8. $O(n^3)$

Best case, worst case and average case

- The running time of an algorithm increases with input size (n)
- Sometimes for the input is same, the running time varies depending on the input instances.
- In such cases, we need to perform best, worst and average case analyses
- Best case → the minimum time
- Worst case → the maximum time
- Average case → the time required on average to execute the algorithm.

Best case, worst case and average case- Illustration- Linear search

Summary

- Growth functions
- Asymptotic notations
- Problem solving strategies
- Divide and conquer
- Recurrences
- Best, worst and average case analysis
- Masters Theorem