cse210 ass01

pa5795

2 April 2019

1 Master's Theorem

Extended master's theorem: $T(n) = aT(\frac{n}{b}) + \Theta(n^k log^p n)$

1.1 Case 1: $a > b^k \Rightarrow T(n) = \Theta(n^{log_b a})$

1.2 Case 2: $a = b^k$

1.2.1 $p > -1 \Rightarrow T(n) = \Theta(n^k log^{p+1} n)$

1.2.2 $p = -1 \Rightarrow T(n) = \Theta(n^k log(logn))$

1.2.3 $p < -1 \Rightarrow T(n) = \Theta(n^k)$

1.3 Case 3: $a < b^k$

1.3.1 $p \geq 0 \Rightarrow T(n) = \Theta(n^k log^p n)$

1.3.2 $p < 0 \Rightarrow T(n) = O(n^k)$

2 Minimum Spanning Trees

2.1 Prim's Algorithm

Time Complexity: $O((V + E)log(V))$

$V \leftarrow$ (number of vertices)
$U \leftarrow 1$
$T \leftarrow \emptyset$
while $V \neq U$ **do**
  $(u, v) \leftarrow$ Lowest cost edge; $u \in U$ and $v \in V - U$
  $T \leftarrow T \cup (u, v)$
  $U \leftarrow U \cup v$
end while

2.2 Kruskal's Algorithm

Time Complexity: $O(Elog(V))$

$V \leftarrow$ (number of vertices)
$T \leftarrow \emptyset$
for e in G.E (Ordered by weight in ascending order) **do**
  if $find(u) \neq find(v)$ **then**
    $T = T \cup (u, v)$
    Union(u,v)
  end if
end for

---

3 Activity Selection

Require: lists sorted in ascending order by end times
$i \leftarrow 0$
$j \leftarrow 1$
$activities = []$
while $j < N$ **do**
  if $start[j] > end[i]$ **then**
    $activities[i] \leftarrow (start[j], end[i])$
    $i = j$
  end if
  $j \leftarrow j + 1$
end while

Time Complexity: As sorted array is required by the algorithm, the time complexity is just $O(n)$.

4 Time Table Scheduler

Code[Python]:

```
def time_table(start,end,k):
    n = len(start)
    arr = sorted(list(zip(start,finish)),key=lambda x: x[1])
    rooms = [[] for _ in range(k)]
    for i in range(n):
        for j in rooms:
            if not j or j[-1]<=arr[i][0]:
                j.append(arr[i][j])
                break
    return rooms
```

5 Asymptotic Notations

5.1 Big-O: Upper Bound

This gives the upper bound of a function. If $f(n) = O(g(n)$, the time complexity of the function ($f(n)$) is at least $g(n)$.
Rule: $f(n) = O(g(n))$ iff $\exists c, x_0$, such that, $f(x) \leq cg(x), x \geq x_0$
Implication: The growth rate of $f(n) \leq$ the growth rate of $g(n)$
Example: $f(2n + 3) = O(g(10n)) = O(g(n)) = O(g(n^2)) = O(g(n^3)) = ...$

5.2 Omega Ω: Lower Bound

This gives the lower bound of a function. If $f(n) = \Omega(g(n)$, the time complexity of the function ($f(n)$) is at most $g(n)$.
Rule: $f(n) = \Omega(g(n))$ iff $\exists c, x_0$, such that, $f(x) \geq cg(x), x \geq x_0$
Implication: The growth rate of $f(n) \geq$ the growth rate of $g(n)$
Example: $f(n^3) = \Omega(g(n^2)) = \Omega(g(n)) = \Omega(g(logn)) = ...$

5.3 Theta Θ: Range Bound

This gives the range bound of a function. If $f(n) = \Theta(g(n)$, the time complexity of the function ($f(n)$) lies between $g(n)$.
Rule: $f(n) = \Theta(g(n))$ iff $\exists c_1, c_2, x_0$, such that, $c_1 g(x) \leq f(x) \leq c_2 g(x), x \geq x_0$
Another way to write the rule: $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
Implication: The growth rate of $f(n) ==$ the growth rate of $g(n)$
Example: $f(2n^3) = \Theta(g(n^3))$

6 Recursive Fibonacci

Code [Python]:

```python
def fib(n):
  if n==0 or n==1: return 1
  return fib(n-1)+fib(n-2)
```

Time Complexity:
$T(n) = T(n-2) + T(n-1) + O(1)$
Let $T(n) = r^n$
$r^n = r^{n-1} + r^{n-2} + k$ [k:constant]
$r^2 = r + 1 + \frac{k}{r^{n-2}}$
$r^2 - r - c = 0$ $[c = 1 + \frac{k}{r^{n-2}}]$
$r = \frac{1 \pm \sqrt{1+4c}}{2}$
$r^n = (\frac{1 \pm \sqrt{1+4c}}{2})^n$
$T(n) = r^n = (\frac{1 \pm \sqrt{1+4c}}{2})^n$
$T(n) = O(k^n)$