recurssion- time complexity

Monday, 21 June 2021 12:09 PM

Decreasing Recurrence Relation — $T(n)=T(n-1)+c$

```
void Test (int n)      -> assume time taken is T(n)
{
  if (n>0)             - 1
  {
    cout << n;         - 1
    Test (n-1)         - T(n-1)
  }
}
```

∴ recurrence relation is

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

∴ $T(n-1)=T(n-2)+2 \Rightarrow T(n)=T(n-2)+2(2)$

∴ $T(n-2)=T(n-3)+2 \Rightarrow T(n)=T(n-3)+3(2)$

⋮

$\Rightarrow T(n)=T(n-k)+(k)(2)$

assume $n-k=0$ $\Rightarrow n=k$

$\Rightarrow T(n)=T(0)+n\cdot 2$

$=1+2n \rightarrow O(n)$ time complexity

Decreasing Recurrence Relation — $T(n)=2T(n-1)+c$

```
void Test (int n)      -> assume time taken is T(n)
{
  if (n>0)             - 1
  {
    cout << n;         - 1
    Test (n-1)         - T(n-1)
    Test (n-1)         - T(n-1)
  }
}
```

∴ recurrence relation is

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

∴ $T(n-1)=2T(n-2)+2 \Rightarrow T(n)=2(2\cdot T(n-2)+2)+2 = 2^2\cdot T(n-2)+2+2^2$

∴ $T(n-2)=2T(n-3)+2 \Rightarrow T(n)=2^2\cdot(2\cdot T(n-3)+2)+2+2^2 = 2^3\cdot T(n-3)+2+2^2+2^3$

⋮

$\Rightarrow T(n)=2^k\cdot T(n-k)+2+2^2+2^3+\dots+2^k$

assume $n-k=0$ $\Rightarrow n=k$

$\Rightarrow T(n)=2^n T(0)+2+2^2+\dots+2^n$

$=2^n+2+2^2+\dots+2^n$

$=2^n+2^{n+1}-2$

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

$=3\cdot 2^n-2 \rightarrow O(2^n)$ time complexity

Decreasing Recurrence Relation — $T(n)=T(n-1)+n$

```
void Test (int n)              -> assume T(n)
{
  if (n>0)                     - 1
  {
    for (i=0; i<n; i++)        - n+1
    {
      cout << n;               - n
    }
    Test(n-1)                  - T(n-1)
  }
}
```

∴ recurrance relation is

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

For ease of calculation consider following result will remain same.

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

∴ $T(n-1)=T(n-2)+(n-1) \Rightarrow T(n)=T(n-2)+(n)+(n-1)$

∴ $T(n-2)=T(n-3)+(n-2) \Rightarrow T(n)=T(n-3)+(n)+(n-1)+(n-2)$

⋮

$\Rightarrow T(n)=T(n-k)+(n)+(n-1)+(n-2)+\dots+(n-(k-1))$

$=T(n-k)+(n-(k-1))+(n-(k-2))+\dots+(n-2)+(n-1)+n$

assume $n-k=0$ $\Rightarrow k=n$

∴ $T(n)=T(0)+1+2+\dots+(n-1)+n$

$=T(0)+\frac{(n)(n+1)}{2}$

$=1+\frac{(n)(n+1)}{2} \rightarrow O(n^2)$ time complexity.

Decreasing Recurrence Relation — $T(n)=T(n-1)+\log_2 n$

```
void Test (int n)              -> assume T(n)
{
  if (n>0)                     - 1
  {
    for (i=0; i<n; i=i*2)      - log2 n + 1
    {
      cout << n;               - log2 n
    }
    Test(n-1)                  - T(n-1)
  }
}
```

∴ recurrance relation is

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

For ease of calculation consider following result will remain same.

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

∴ $T(n-1)=T(n-2)+\log(n-1) \Rightarrow T(n)=T(n-2)+\log(n-1)+\log n$

∴ $T(n-2)=T(n-3)+\log(n-2) \Rightarrow T(n)=T(n-3)+\log(n-2)+\log(n-1)+\log n$

⋮

$\Rightarrow T(n)=T(n-k)+\log(n-(k-1))+\dots+\log(n-2)+\log(n-1)+\log(n)$

assume $n-k=0$ $\Rightarrow n=k$

$\Rightarrow T(n)=T(0)+\log(1)+\log(2)+\dots+\log(n-1)+\log(n)$

$=1+\log((1)(2)(3)\cdots(n-1)(n))$

$=1+\log(n!) \rightarrow O(\log(n)^n)=O(n\log n)$ time complexity

---

$T(n)=T(n-1)+c \rightarrow O(n)$

$T(n)=T(n-1)+\log n \rightarrow O(n\log n)$

x2, x3

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

$T(n)=3T(n-1)+c \rightarrow O(3^n)$

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

2, 3, 2, 3

$T(n)=T(n-2)+c \rightarrow O(n/2)=O(n)$

$T(n)=T(n-3)+c \rightarrow O(n/3)=O(n)$

$T(n)=T(n-2)+\log n \rightarrow O(\frac{n}{2}\log\frac{n}{2})=O(n\log n)$

$T(n)=T(n-3)+\log n \rightarrow O(\frac{n}{3}\log\frac{n}{3})=O(n\log n)$

⇒ this causes no change.

xn, xn², xn

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

$T(n)=T(n-1)+n^2 \rightarrow O(n^3)$

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

step 1, step 2, step 1, step 2, step 1, step 2

$T(n)=2T(n-2)+c \rightarrow O(2^{n/2})=O(2^n)$

$T(n)=3T(n-2)+c \rightarrow O(3^{n/2})=O(3^n)$

$T(n)=3T(n-2)+\log n \rightarrow O((\log\frac{n}{2})(3^{n/2}))=O(\log n\cdot 3^n)$

hence we can ignore step 2

$T(n)=2T(n-1)+n \rightarrow O(n2^n)$

$T(n)=T(n-2)+n \rightarrow O((\frac{n}{2})^2)=O(\frac{n^2}{4})=O(n^2)$

$T(n)=T(n-2)+n\log n \rightarrow O((\frac{n}{2})^2\log(\frac{n}{2}))=O(n^2\log n)$

hence from now onwords ignore step 2

this case. this case. this case

$$T(n)=aT(n-b)+f(n)$$

where $a>0$, $b>0$

if $a=1 \longrightarrow O(n\times f(n))$

if $a>1 \longrightarrow O(f(n)\times a^n)$

if $(a<1) \longrightarrow O(f(n))$

Master's theorem for decreasing recurssive fn

Dividing Recurrence Relation — $T(n)=T(n/2)+c$

```
void Test (int n)      -> assume time taken is T(n)
{
  if (n>1)             - 1
  {
    cout << n;         - 1
    Test (n/2);        - T(n/2)
  }
}
```

∴ recurrence relation is

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

∴ $T(n/2)=T(n/4)+2 \Rightarrow T(n)=T(n/4)+2+2$

∴ $T(n/4)=T(n/8)+2 \Rightarrow T(n)=T(n/8)+2+2+2 = T(n/2^3)+3(2)$

⋮

$\Rightarrow T(n)=T(n/2^k)+k\cdot 2$

assume $\frac{n}{2^k}=1$ $\Rightarrow k=\log_2 n$

$\Rightarrow T(n)=T(1)+(\log_2 n)(2) = 1+(\log_2 n)(2)$

$O(\log_2 n)$ time complexity.

Dividing Recurrence Relation — $T(n)=2T(n/2)+c$

```
void Test (int n)      - assume T(n)
{
  if (n>1)             - 1
  {
    cout << n;         - 1
    Test(n/2);         - T(n/2)
    Test(n/2);         - T(n/2)
  }
}
```

∴ recurrence relation is

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

∴ $T(n/2)=2T(n/4)+2 \Rightarrow T(n)=(2)(2\cdot T(n/4)+2)+2$

∴ $T(n/4)=2T(n/8)+2 \Rightarrow T(n)=(2^2)(2\cdot T(n/8)+2)+2+2^2$

$=2^3\cdot T(n/2^3)+2+2^2+2^3$

⋮

$\Rightarrow T(n)=2^k\cdot T(n/2^k)+2+2^2+\dots+2^k$

$=2^k\cdot T(n/2^k)+2^{k+1}-2$

assume $n/2^k=1$ $\Rightarrow n=2^k \Rightarrow k=\log_2 n$

$\Rightarrow T(n)=2^{\log_2 n}\cdot T(1)+(2)(2^{\log_2 n}-1)$

$=n\cdot 1+(2)(n-1)$

$=3n-2 \rightarrow O(n)$ time complexity

Dividing Recurrence Relation — $T(n)=T(n/2)+n$

```
void Test (int n)              -> assume T(n)
{
  if (n>1)                     - 1
  {
    for (i=0; i<n; i++)        - n+1
    {
      cout << n;               - n
    }
    Test(n/2)                  - T(n/2)
  }
}
```

∴ recurrence relation is

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

For ease of calculation

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

∴ $T(n/2)=T(n/4)+n/2 \Rightarrow T(n)=T(n/4)+n/2+n$

∴ $T(n/4)=T(n/8)+n/4 \Rightarrow T(n)=T(n/8)+n/4+n/2+n$

⋮

$\Rightarrow T(n)=T(\frac{n}{2^k})+\frac{n}{2^{k-1}}+\frac{n}{2^{k-2}}+\dots+\frac{n}{2}+n$

assume $\frac{n}{2^k}=1$ $\Rightarrow n=2^k \Rightarrow \log_2 n=k$

$\Rightarrow T(n)=T(1)+(n)(1+1)=1+2n \rightarrow O(n)$ time complexity.

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

where $a\geq 1$, $b>1$ and $f(n)=n^k\cdot\log_2^p n$

if $\log_b a>k \longrightarrow O((n)^{\log_b a})$

if $\log_b a=k$:
- if $p>-1$ (i.e. log not present or present in numerator of f(n)) $\longrightarrow O(f(n)\times\log_2 n)$ $=O(n^k\log^{p+1}n)$ but don't use this form
- if $p=-1$ (i.e. one log present in denominator of f(n)) $\longrightarrow O(n^k\log_2\log_2 n)$
- if $p<-1$ (i.e. multiple logs present in denominator of f(n)) $\longrightarrow O(n^k)$

if $\log_b a<k$:
- if $p\geq 0$ (i.e. log not present or present in numerator of f(n)) $\longrightarrow O(f(n))$
- if $p<0$ (i.e. log present (single or multiple log) in denominator of f(n)) $\longrightarrow O(n^k)$

Master's Theorem for Dividing Recursive Function

examples for using master's theorem

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

$\Rightarrow a=2,\ b=2,\ f(n)=1=n^0\log^0 n \Rightarrow k=0$ & $p=0$

$\Rightarrow \log_b a=\log_2 2=1$

$\Rightarrow \log_b a>k$

$\Rightarrow$ time complexity is $O(n^{\log_b a})=O(n^1)=O(n)$

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

$\Rightarrow a=4,\ b=2,\ f(n)=n=n^1\log^0 n \Rightarrow k=1, p=0$

$\Rightarrow \log_b a=\log_2 4=2 \Rightarrow k<\log_b a$

$\Rightarrow$ time complexity is $O(n^{\log_b a})=O(n^2)$

$T(n)=8T(n/2)+n$

$\Rightarrow a=8,\ b=2,\ f(n)=n=n^1\cdot\log^0 n \Rightarrow k=1, p=0$

$\Rightarrow \log_b a=\log_2 8=3 \Rightarrow \log_b a>k$

$\Rightarrow$ time complexity is $=O(n^{\log_b a})=O(n^3)$

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

$\Rightarrow a=2,\ b=2,\ k=1,\ p=0$

$\Rightarrow \log_b a=1=k$

$\Rightarrow O(f(n)\cdot\log n)=O(n\cdot\log_2 n)$

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

$\Rightarrow a=1, b=2, f(n)=n \Rightarrow k=1, p=0$

$\Rightarrow \log_b a=0 \Rightarrow k>\log_b a$

$\Rightarrow O(f(n))=O(n)$

---

root Recurrence Relation — $T(n)=T(\sqrt{n})+c$

```
void Test (int n)      -> assume time taken is T(n)
{
  if (n>2)             - 1
  {
    cout << n;         - 1
    Test(√n);          - T(√n)
  }
}
```

∴ recurrence relation is

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

$T(\sqrt{n})=T(n^{1/4})+2 \Rightarrow T(n)=T(n^{1/4})+2$

$T(n^{1/4})=T(n^{1/8})+2 \Rightarrow T(n)=T(n^{1/8})+2+2$

$=T(n^{1/2^3})+2+2$

⋮

$\Rightarrow T(n)=T(n^{1/2^k})+(k-1)(2)$

assume $n^{\frac{1}{2^k}}=2 \Rightarrow \frac{1}{2^k}=\frac{\log_2 2}{\log_2 n} \Rightarrow \log_2 n=2^k \Rightarrow k=\log_2\log_2 n$

$\Rightarrow T(n)=T(2)+(k-1)(2)$

$=1+((\log_2\log_2 n)-1)(2) \rightarrow O(\log_2\log_2 n)$ time complexity