Design and Analysis of Algorithms

Tutorial 2: Divide and Conquer Algorithms

许可 kexu@nlsde.buaa.edu.cn

童咏昕 yxtong@buaa.edu.cn

北京航空航天大学 计算机学院

问题1

- 使用归纳法证明下列问题。

(a) 递归函数为

$$\mathrm{T}(1) = 1$$

$$T(n) = T\left(\frac{n}{2}\right) + n \quad if\ n > 1$$

证明存在c使得$T(n) \leq c \cdot n$成立。

(b) 递归函数为

$$T(1) = 1$$

$$T(n) = 3T\left(\frac{n}{2}\right) + n^2 \qquad if\ n > 1$$

证明存在c使得$T(n) \leq c \cdot n$成立。

问题1

(c) 递归函数为

$$T(1) = 1$$

$$T(n) = T\left(\frac{n}{3}\right) + n \qquad if\ n > 1$$

证明存在c使得$T(n) \leq c \cdot n$成立。

(d) 递归函数为

$$T(1) = 1$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n \qquad if\ n > 1$$

证明存在c_1, c_2使得$T(n) \leq c_1 \cdot n^2 - c_2 \cdot n$成立。

问题1

(e) 递归函数为

$$T(1) = 1$$

$$T(n) = 3T\left(\frac{n}{2}\right) + n^2 \qquad if\ n > 1$$

证明存在c使得$T(n) \leq c \cdot n^2$成立。

(f) 递归函数为

$$T(1) = 0, T(2) = 1$$

$$T(n) = T\left(\frac{n}{2}\right) + \log_2 n \qquad if\ n > 2$$

证明存在c使得$T(n) \leq c \cdot \log^2 n$成立。

问题1 (a) –提示

初始化 $n = 1$: $T(1) = 1 \le c \cdot 1$对任意$c \ge 1$都成立.

归纳:

$$\begin{aligned} T(n) &= T(n/2) + n \\ &\le c \cdot n/2 + n \\ &= c \cdot n - c \cdot n/2 + n \\ &= c \cdot n - (c/2 - 1) \cdot n \\ &\le c \cdot n \qquad for\ c \ge 2 \end{aligned}$$

因此，对于$n \ge 1$和$c \ge 2$, 满足 $T(n) \le c \cdot n$.

问题1 (b) –提示

初始化 $n = 1 \colon T(1) = 1 \le c \cdot 1$对任意$c \ge 1$都成立.

$n = 2 \colon T(2) = 1 \le c \cdot 2$对任意$c \ge 1/2$都成立.

归纳:

$$
\begin{aligned}
T(n) &= T(n-2) + 1 \\
&\le c \cdot (n-2) + 1 \\
&= c \cdot n - 2c + 1 \\
&\le c \cdot n \qquad for\ c \ge 2
\end{aligned}
$$

因此, 对于$n \ge 1$和$c \ge 1$, 满足$T(n) \le c \cdot n$.

问题1 (c) –提示

初始化 $n = 1$: $T(1) = 1 \le c \cdot 1$对任意$c \ge 1$都成立.

归纳:

$$
\begin{aligned}
T(n) &= T(n/3) + n \\
&\le c \cdot (n/3) + n \\
&= c \cdot n - 2cn/3 + n \\
&= c \cdot n - (2c/3 - 1)n \\
&\le c \cdot n \qquad for\ c \ge 3/2
\end{aligned}
$$

因此, 对于$n \ge 1$和$c \ge 3/2$, 满足$T(n) \le c \cdot n$.

问题1 (d) –提示

初始化 $n = 1$: $T(1) = 1 \le c_1 \cdot 1 - c_2 \cdot 1$对任意$c_1 \ge c_2 + 1$都成立.

归纳:

$$
\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + n \\
&\le 4\left(c_1 \cdot \left(\frac{n}{2}\right)^2 - c_2 \cdot \frac{n}{2}\right) + n \\
&= c_1 \cdot n^2 - 2c_2 \cdot n + n \\
&= c_1 \cdot n^2 - c_2 \cdot n - (c_2 - 1) \cdot n \\
&\le c_1 \cdot n^2 - c_2 \cdot n \qquad for\ c_2 \ge 1
\end{aligned}
$$

因此，对于$n \ge 1, c_2 \ge 1$和 $c_1 \ge c_2 + 1$，满足$T(n) \le c_1 \cdot n^2 - c_2 \cdot n$.

问题1 (e)-提示

初始化 $n = 1$: $T(1) = 1 \le c \cdot 1$对任意$c \ge 1$都成立.

归纳:

$$
\begin{aligned}
T(n) &= 3T\left(\frac{n}{2}\right) + n^2 \\
&\le 3c \cdot \left(\frac{n}{2}\right)^2 + n^2 \\
&= 3c \cdot \frac{n^2}{4} + n^2 \\
&= c \cdot n^2 - \left(\frac{c}{4} - 1\right)n^2 \\
&\le c \cdot n^2 \qquad for\ c \ge 4
\end{aligned}
$$

因此, 对于$n \ge 1$和$c \ge 4$, 满足$T(n) \le c \cdot n^2$.

问题1(f)-提示

初始化 $n = 1 : T(1) = 0 \leq c \cdot 0$对任意$c$都成立.

$n = 2 : T(2) = 1 \leq c \cdot 1$对任意$c \geq 1$都成立.

Induction:

$$
\begin{aligned}
T(n) &= T(n/2) + \log_2 n \\
&\leq c \cdot \log_2^2\left(\frac{n}{2}\right) + \log_2 n \\
&= c \cdot (\log_2 n - 1)^2 + \log_2 n \\
&= c \cdot (\log_2^2 n - 2 \cdot \log_2 n + 1) + \log_2 n \\
&= c \cdot \log_2^2 n - 2c \cdot \log_2 n + c + \log_2 n \\
&= c \cdot \log_2^2 n - (c - 1) \cdot \log_2 n - c(\log_2 n - 1) \\
&\leq c \cdot \log_2^2 n \qquad for\ c \geq 1\ and\ n \geq 2
\end{aligned}
$$

因此, 对于$n \geq 1$和$c \geq 1$, 满足$T(n) \leq c \cdot \log_2^2 n$.

问题2

- 令A[1..n]表示一列由正整数组成的数组，设计一个分治算法计算数组中A[j]-A[i]的最大值，其中，j≥i.同时分析所给算法的时间复杂度。

问题2-提示

与最大子数组问题类似，如果将A分为两个长度相等的子数组（每个子数组长度为n/2），则所求$A[j]-A[i](1\le i\le j\le n)$的最大值必定是下面三种情况之一。

- 数组A[1..$\left\lfloor\frac{n}{2}\right\rfloor$]中$A[j]-A[i](1\le i\le j\le \left\lfloor\frac{n}{2}\right\rfloor)$的最大值。
- 数组A[$\left\lfloor\frac{n}{2}\right\rfloor+1..n$]中$A[j]-A[i](\left\lfloor\frac{n}{2}\right\rfloor+1\le i\le j\le n)$的最大值。
- 值$A[j]-A[i]$，其中$A[i]$为数组A[1..$\left\lfloor\frac{n}{2}\right\rfloor$]中的最小值，$A[\mathrm{j}]$为数组A[$\left\lfloor\frac{n}{2}\right\rfloor+1..n$]中的最大值。

问题2-算法细节

- Input
  - 一列由正整数组成的数组**A**
  - A中起始下标**p**
  - A中结束下标**r**
- Output
  - **MaxAll,MinAll**:数组中起始、结束下标范围内的最大值和最小值
  - **ValAll**:数组中起始、结束下标范围内最大差值A[j]-A[i](j≥i)

问题2–算法细节

Find-Max-Diff(A, p, r)

Input: $\boldsymbol{A}$ is an array of positive integers, $\boldsymbol{p}$ is the start index of A, $\boldsymbol{r}$ is the end index of A.

Output: $\boldsymbol{MaxAll}$ and $\boldsymbol{MinAll}$ are the maximum and minimum value of the input array $A[p..r]$ respectively, $\boldsymbol{ValAll}$ is the maximum value of $A[j] - A[i]$ with $j \geq i$ in $A[p..r]$.

if *p is equal to r* **then**

  $MaxAll \leftarrow A[p], MinAll \leftarrow A[p], ValAll \leftarrow 0;$ $//O(1)$

end

else

  $m \leftarrow \lfloor \frac{p+r}{2} \rfloor;$

  $[MaxL, MinL, ValL] \leftarrow \text{FMD}(A, p, m);$ $//T(\lfloor n/2 \rfloor)$

  $[MaxR, MinR, ValR] \leftarrow \text{FMD}(A, m+1, r);$ $//T(\lceil n/2 \rceil)$

  $ValM = MaxR - MinL;$ $//O(1)$

  $ValAll \leftarrow \max(ValL, ValR, ValM);$ $//O(1)$

  $MaxAll \leftarrow \max(MaxL, MaxR);$ $//O(1)$

  $MinAll \leftarrow \max(MinL, MinR);$ $//O(1)$

end

return $[MaxAll, MinAll, ValAll];$

问题2-复杂度

令$T(n)$表示最坏情况下算法使用的操作次数。

当n = 1时，FMD执行需要O(1)时间，T(1)=O(1).

当n>1时，FMD对平均划分的两个子数组递归调用FMD函数，规模减小一半，同时使用O(1)时间对两部分的结果进行组合.

因此有如下递归函数关系

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

为了简化分析过程，假设n为2的幂.

问题2-复杂度

需要解决下述递归函数关系:

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

令$h=\log_2 n$

$$\begin{aligned} T(n) &= 2\cdot T\left(\frac{n}{2}\right)+c \\ &= 2\cdot\left(2\cdot T\left(\frac{n}{4}\right)+c\right)+c \\ &= 4\cdot T\left(\frac{n}{4}\right)+2c+c \\ &\dots \\ &= 2^h\cdot T\left(\frac{n}{2^h}\right)+2^{h-1}c+2^{h-2}c+\cdots+c \\ &= 2^h\cdot T\left(\frac{n}{2^h}\right)+\frac{c\left(2^h-1\right)}{2-1} \end{aligned}$$

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

问题3

- 设计一个线性时间复杂度的算法解决最大子数组问题(Maximum Contiguous Subarray Problem)。

  提示：若已知A[1..j]的最大子数组(1≤j<n),则A[1..j+1]的最大子数组要么是A[1..j]的最大子数组，要么是某个子数组A[i..j+1](1≤i≤j+1)。

问题3-提示

提示：若已知A[1..j]的最大子数组(1≤j<n)，则A[1..j+1]的最大子数组要么是A[1..j]的最大子数组，要么是某个子数组[i..j+1](1≤i≤j+1)。

- 使用变量low,high,max-sum分别记录已扫描过的数组(即A[1..j])中的最大子数组的起点、终点和值。
- 使用变量iter-low,iter-high,iter-sum分别记录以j为终点的最大子数组的起点、终点和值。每次对该值进行如下更新：
  - 若iter-sum≤0，则A[i..j+1](1≤i≤j+1)中，最大值为A[j+1]，对应的子数组为A[j+1..j+1]；
  - 若iter-sum>0，则A[i..j+1](1≤i≤j+1)中，最大值为iter-sum + A[j+1]，对应的子数组为A[iter-low..j+1].

问题3-伪代码

Max-Subarray-Linear(A)

```
Input: an array A
Output: Bound and sum of the maximum subarray
        low,high,max-sum
n ← size of A;
max-sum ← −∞;
iter-sum ← −∞;
for j ← 1 to n do
    //Update the variables iter-low,iter-high,iter-sum.
    iter-high ← j;
    if iter-sum > 0 then
        iter-sum ← iter-sum + A[j];
    end
    else
        iter-low ← j;
        iter-sum ← A[j];
    end
    //Record the maximum subarray while iterating.
    if iter-sum > max-sum then
        max-sum ← iter-sum;
        low ← iter-low;
        high ← iter-high;
    end
    return low, high, max-sum;
end
```

问题3-分析

进入循环首先判断以j为终点的最大子数组是否仅包含A[j](注意此处终点总为j，因此直接对iter-high赋值为j).

- 若iter-sum + A[j] ＞ A[j](即iter-sum＞0),则将以j-1为终点的最大子数组添加A[j]扩展为以j为终点的最大子数组(此时iter-low值不变).
- 否则，以j为终点的最大子数组仅包含A[j]一个元素，更新iter-low和iter-sum的值。

每一轮循环结束，判断并记录当前已找到的最大子数组。

每一轮循环中仅包含O(1)次操作，因此该算法的时间复杂度为O(n).