Applying the resolution of recurrences: Divide and Conquer

Algorithms

Alberto Valderruten, Elena Hernández Pereira,
José M. Casanova

Computer Science and Information Technologies Department
Faculty of Computer Science

UNIVERSIDADE DA CORUÑA

1. Resolution of recurrences theorem: Divide and Conquer
   - Theorem
   - O rules
2. Maximum subsequence sum
3. Binary search

- **Given the recurrence**

$$T(n) = \ell T(n/b) + cn^k, n > n_0 \qquad (1)$$

with $\ell \geq 1, b \geq 2, k \geq 0, n_0 \geq 1 \in \mathbb{N}$ y $c > 0 \in \mathbb{R}$,
when n/n_0 is an exact power of b ($n \in \{bn_0, b^2n_0, b^3n_0 \dots\}$).

- **Divide and Conquer theorem**:
If a recurrence is of the form (1), we apply:

$$T(n) = \begin{cases} \theta(n^k) & si \quad \ell < b^k \\ \theta(n^k logn) & si \quad \ell = b^k \\ \theta(n^{log_b \ell}) & si \quad \ell > b^k \end{cases} \qquad (2)$$

In analysis of algorithms, inequalities are commonly used:
$T(n) \leq \ell T(n/b) + cn^k, n > n_0$ with n/n_0 an exact power of b

$$\Rightarrow T(n) = \begin{cases} O(n^k) & si \quad \ell < b^k \\ O(n^k logn) & si \quad \ell = b^k \\ O(n^{log_b \ell}) & si \quad \ell > b^k \end{cases}$$

1. Elemental operation $= 1 \leftrightarrow$ Computational model
2. **sequence**: $S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow \boxed{S_1; S_2} = O(f_1(n) + f_2(n)) = O(max(f_1(n), f_2(n)))$
   - With Θ too
3. **condition**: $B = O(f_B(n)) \wedge S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow \boxed{\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2} = O(max(f_B(n), f_1(n), f_2(n)))$
   - Si $f_1(n) \neq f_2(n)$ y $max(f_1(n), f_2(n)) > f_B(n) \leftrightarrow$ **Worst case**
   - ¿Middle case? $\rightarrow f(n)$: average of f_1 y f_2 weighted with the frequencies of each branch $\rightarrow O(max(f_B(n), f(n)))$
4. **iteration**: $B; S = O(f_{B,S}(n)) \wedge$ num. iter$= O(f_{iter}(n))$
   $\Rightarrow \boxed{\textbf{while } B \textbf{ do } S} = O(f_{B,S}(n) * f_{iter}(n))$
   iff each iteration cost does not change, else: $\sum$ individual costs.
   $\Rightarrow \boxed{\textbf{for } i \leftarrow x \textbf{ to } y \textbf{ do } S} = O(f_S(n) *$num. iter$)$
   iff each iteration cost does not change, else: $\sum$ individual costs.
   - B is a two integer comparison $= O(1)$; num. iter $= y - x + 1$

- $a_1, \ldots, a_n \rightarrow \sum_{k=i}^{j} a_k$ to be maximum
  - Example: $MSS(-2, 11, -4, 13, -5, -2) = 20[2..4]$
- MSS recursive: Divide and Conquer strategy
  - Divide the input in halves $\rightarrow$ Two recursive solutions
  - Conquer using the two solutions $\rightarrow$ Solution for the original input
  - The MSS can be
    - in the first half
    - in the second half
    - between the two halves
  - The first two solutions are the ones obtained recursively
  - The third solution is obtained summing
    - the MSS of the first half which includes the right end, and
    - the MSS of the second half which includes the left end

MSS recursive

```
function MSS (a[1..n]): value      /* interface function */
  return MSSRecursive(a,1,n)
end function

function MSSRecursive(var a[1..n],left,right): value
{1}   if left = right then
{2}     if a[left] > 0 then
{3}       return a[left]          /*base case: if >0 is MSS */
        else
{4}       return 0
        end if
      else
{5}     middle := (left + right)/2;
{6}     FirstSolution := MSSRecursive(a,left,middle);
{7}     SecondSolution := MSSRecursive(a,middle+1,right);
```

MSS recursive (II)

```
{8}      MaxLeftSum := 0;  LeftSum := 0;
{9}      for i := middle to left step −1 do
{10}       LeftSum := LeftSum + a[i];
{11}       if LeftSum > MaxLeftSum then
{12}         MaxLeftSum := LeftSum
         end for;

{13}     MaxSumRight := 0; RightSum := 0;
{14}     for i := middle+1 to right step 1 do
{15}       RightSum := RightSum + a[i];
{16}       if RightSum > MaxRightSum then
{17}         MaxRightSum := RightSum
         end for;

{18}     return max(FirstSolution,SecondSolution,
                    MaxLeftSum+MaxRightSum)
   end if
end function
```

- Understand and execute the MSS recursive algorithm with an example and write the recursive tree
- Analyse the MSS algorithm setting out the recurrence relation and applying the resolution recurrence theorem divide and conquer

- Example of a *logarithmic algorithm*
- Given x and an *ordered* array a_1, a_2, ... a_n of integer,

$$\text{return: } \begin{cases} i \text{ if } \exists a_i = x \\ \text{“element not found”} \end{cases}$$

$\rightarrow$ Compare x and a_{middle}, with $middle = (i + j)div2$, being $a_i..a_j$ the *search space*:

1. $x = a_{middle}$: finish (interruption)
2. $x > a_{middle}$: continue searching in $a_{middle+1}..a_j$
3. $x < a_{middle}$: continue searching in $a_i..a_{middle-1}$

- ¿number of iterations? $\leftrightarrow$ size d evolution in the search space

Invariant: $d = j - i + 1$

$$\text{¿How is } d \text{ decreasing? } \begin{cases} i \leftarrow middle + 1 \\ j \leftarrow middle - 1 \end{cases}$$

- *Worst case*: the normal termination of the loop is reached $\equiv i > j$

```
    function Binary Search(x, a[1..n]): position
      {a: ordered array with no decreasing ordered }

{1}   i := 1 ; j := n ;              {search space: i..j}
{2}   while i <= j do
{3}       middle := (i + j) div 2 ;
{4}       if a[middle] < x then
{5}           i := middle + 1
{6}       else if  a[middle] > x then
{7}           j := middle − 1
{8}       else return  middle       {the loop stops}
      end while;
{9}   return "element not found" {normal loop end}
    end function
```

... worst case

- Being $< d, i, j >$ iteration $< d', i', j' >$:
  1. $i \leftarrow middle + 1$:
     $$\begin{aligned} i' &= (i+j)div2 + 1 \\ j' &= j \\ d' = j' - i' + 1 \quad &= j - (i+j)div2 - 1 + 1 \\ &\leq j - (i+j-1)/2 \\ &= (j-i+1)/2 \\ &= d/2 \end{aligned}$$
     $$\boxed{\rightarrow d' \leq d/2}$$
  2. $j \leftarrow middle - 1$:
     $$\begin{aligned} i' &= i \\ j' &= (i+j)div2 - 1 \\ d' = j' - i' + 1 \quad &= (i+j)div2 - i - 1 + 1 \\ &\leq (i+j)/2 - i \\ &< (j-i+1)/2 \\ &= d/2 \end{aligned}$$
     $\boxed{\rightarrow d' < d/2}$ *(decreases faster)*

... worst case

- ¿$T(n)$? Being d_l: d *after* the l-th iteration

  $$\begin{cases} d_0 = n \\ d_l \leq d_{l-1}/2 \ \forall l \geq 1 \end{cases} \quad \text{(induction)} \rightarrow d_l \leq n/2^l$$

  until $d < 1 \rightarrow l = \lceil log_2 n \rceil + 1 = O(logn)$ iterations
  Each iteration is $\Theta(1)$ (rules) $\Rightarrow T(n) = O(logn)$
- **Alternative reasoning**: *thinking in a recursive version*

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

  Theorem Divide and Conquer: $l = 1, b = 2, c = 1, k = 0, n_0 = 1$
  Case $l = b^k \Rightarrow T(n) = \Theta(n^k logn) \rightarrow T(n) = \Theta(logn)$
- **Conclusions**:
  - Think about a recursive version would be useful
  - is Divide and Conquer? $\rightarrow$ $\boxed{\text{Reduction algorithms}}$ $(l = 1)$
  - $T(n) = \Theta(logn) \leftrightarrow$ data is in memory
    (Computational model)

Exercise:

- Design a recursive version of the binary search algorithm
- Analyse the resulting recursive algorithm applying the recurrence resolution theorem Divide and Conquer

⋆ Brassard, G. and Bratley, P. Fundamentals of algorithmics. Prentice Hall, 1996.