Competitive programming
Divide and Conquer

Prof. Rhadamés Carmona

Last revision: 26 / 02 / 2020

Content

- Definition
- Complexity
- Exponentiation
- Binary search
- Merge sort
- Hanoi Towers
- Subtree of maximum sub
- Matrix multiplication: Strassen

Definition

- Divide and conquer is an algorithm design technique that involves breaking down complex cases into smaller, easier-to-solve cases, at the cost of combining the solutions obtained to build the solution to the original problem.
- Smaller problems are solved independently of each other. There is no overlapping between subproblems as it does occur in dynamic programming (DP).

Complexity (recursive case)

- As we've already seen in the topic of time complexity, the complexity of recursive problems that are usually solved with this technique have the following recurring function of time:

  $T(1) = 1$ {we take $T(1)=1$ for convenience)

  $T(n) = a.T(n/b)+f(n)$, if $n>1$

  $f(n)$ is the cost of merging “a” solutions.

Complexity (recursive case)

$T(1) = 1$

$T(n) = aT(n/b)+f(n)$, si $n>1$

$$a < f(b) \Rightarrow T(n) = \mathrm{O}(n^{\log_b^{f(n)}})$$

$$a = f(b) \Rightarrow T(n) = \mathrm{O}(n^{\log_b^a}.\log_b^n)$$

$$a > f(b) \Rightarrow T(n) = \mathrm{O}(n^{\log_b^a})$$

Exponentiation

- How to calculate X^n without using pow(x,n) ?
- The classic solution is n-1 products x*x*x***x, O(n).
- The solution divide and conquer divides the problem into two problems of size n/2, and then combine them.
- Since both n/2 problems are "identical", it is reduced to a single n/2 problem.

Exponentiation

$X^n = X^{n/2} * X^{n/2}$ if n>0 is even

$X^n = X * X^{n/2} * X^{n/2}$ if n>0 is odd

$X^n = 1$ if n=0

Algorithmically $X^{n/2}$ can be computed once

Merging cost = 1 or 2 multiplications ➔ f(n)=1

a=1, f(n)=1, b=2 ➔ $a = f(b) \Rightarrow T(n) = \mathrm{O}(n^{\log_b^a} . \log_b^n)$

=O(log n).

Binary search

- Used to search for an "comparable" x element in an ordered array.
- The central element of the array is compared with x, and if it is not there, the problem is reduced to searching in one of the left or right subarrays from the center element, until it is found or until there is only one item left to review.
- Reduces an n-size problem, to a n/2 size problem, with an additional cost f(n)=1.
- T(n) = 1*T(n/2)+1 ➔ a=1, $f(n)$=1, b=2
- $a = f(b) \Rightarrow T(n) = \mathrm{O}(n^{\log_b^a} . \log_b^n) = \mathrm{O}(\log_2^n)$

Merge sort

- It consists of dividing the array into 2 parts
- Each part is divided recursively until it reaches a trivial case (for example, 1 element)
- The core of the algorithm is merging two sorted sub arrays
- T(n)=2T(n/2)+n, a=b=2, f(n)=n

$$a = f(b) \Rightarrow T(n) = \mathrm{O}(n^{\log_b^a} . \log_b^n) \equiv \mathrm{O}(n . \log_2^n)$$

- O(nlogn)

Hanoi Towers

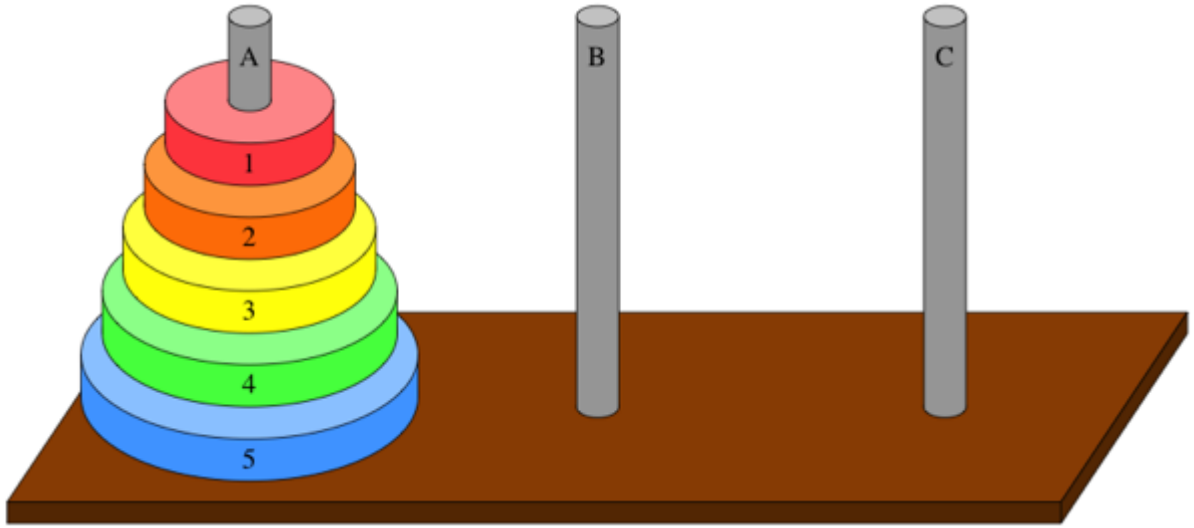
- 1883. Game invented by a French mathematician named Odouard Lucas.
- There are 3 towers. The idea is to move the disk stack from one tower to another tower with the following conditions
  - Only one disc is moved at a time
  - One disc cannot be placed on a smaller disc

Hanoi Towers

```
void move(n, source, target, auxiliary)
{
   if (n > 0) {
            move(n - 1, source, auxiliary, target);
            target.push(source.pop());
            move(n - 1, auxiliary, target, source);
   }
}
```

… main …

Move(5, A, C, B)



Torres de Hanoi

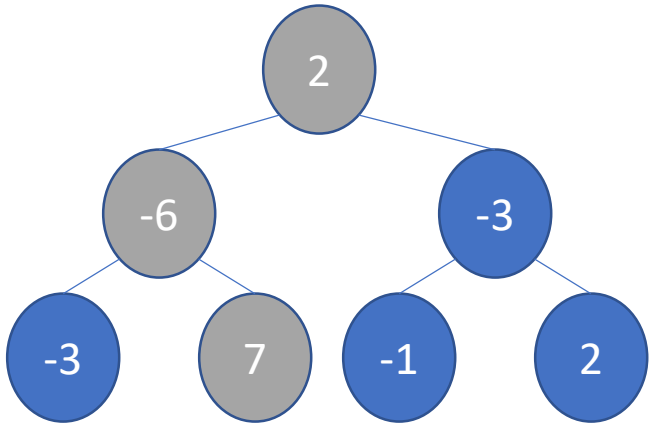
```
void move(n, source, target, auxiliary)
{
   if (n > 0) {
          move(n - 1, source, auxiliary, target);
          target.push(source.pop());
          move(n - 1, auxiliary, target, source);
   }
}
```

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

Subtree of maximum sum

- Given a binary tree, determine the subtree whose sum of the elements is maximum, but only print that sum
- If all elements are positive, the solution is trivial (the addition of all the elements)
- The problem is the existence of negatives
- Can be solved with a post-order tree traversing algorithm O(n)

Subtree of maximum sum




```
int f(Node* root, int& maxsum)
{
        if (root == NULL)
                return 0;
        int sum = root->value+  f(root->left, max) + f(root->right, max);
        if (sum > maxsum)
                maxsum = sum;
        return max(0,sum);
}
```

Matrix multiplication

- C = A*B, $O(n^3)$

```
for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
        {
                c[i][j] = 0;
                for (int k=0; k<n; k++)
                        c[i][j] += a[i][k] * b[k][j];

        }
```

Matrix multiplication

- Naïve divide and conquer: still requires $n^3 = 2^3 = 8$ multiplications. We can apply the save recursive approach for each multiplication $A_{i,j} * B_{j,k}$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

Matrix multiplication

- Using Strassen (end of 60's), it is $O(n^{2.8074})$
- There are 7 "cubic" multiplications but many "quadratic" sums… For large values of n it is posible to observe the time reduction

M1 = (A12 – A22).(B21 + B22)
M2 = (A11 + A22).(B11 + B22)
M3 = (A11 – A21).(B11 + B21)
M4 = (A11 + A12).B22
M5 = A11 .(B12 –B22)
M6 = A22.(B21 – B11)
M7 = (A21 + A22).B11

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$
$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

- Another method, Coppersmith-Winograd (2008), $O(n^{2.376})$.

Bibliography

- Robert Sedgewick. “Algorithms in C++”, Third edition, parts 1-4. Addison-Wesley, 1998.

Problems

- https://www.spoj.com/problems/NPC2014B/ (also DP)
- https://www.spoj.com/problems/DIVCON/
- https://www.spoj.com/problems/BURGLARY/
- https://www.spoj.com/problems/CPP/