Tutorial 3: Divide-and-conquer

1 Comparing Rankings

Suppose two people rank a list of n items (say movies), denoted $M_1, \dots, M_n$. A *conflict* is a pair of movies $\{M_i, M_j\}$ such that $M_i > M_j$ in one ranking and $M_j > M_i$ in the other ranking. The number of conflicts between two rankings is a measure of how different they are.

For example, consider the following two rankings:

$$M_1 > M_2 > M_3 > M_4 \quad \text{and} \quad M_2 > M_4 > M_1 > M_3.$$

The number of conflicts is three; $\{M_1, M_2\}$, $\{M_1, M_4\}$, and $\{M_3, M_4\}$ are the conflicting pairs.

The purpose of this question is to find an efficient divide-and-conquer algorithm to compute the number of conflicts between two rankings of n items. You can assume n is a power of two, for simplicity.

Hint: Think of a MergeSort-like algorithm that solves two subproblems of size $n/2$. After solving the subproblems, you will also need to compute the number of conflicts between the two sublists during the "merge" step.

To simplify the notation in the algorithm, you can assume that the *first ranking* is $M_1 > M_2 > \cdots > M_n$.

1. Give a pseudocode description of your algorithm, briefly justify its correctness and analyze the complexity using a recurrence relation.

Solution

Algorithm **Conflicts**(M, n)
Post-condition: The algorithm returns (M, C), where M is sorted and C is the number of conflcts.

step 1 If $n = 1$ then **RETURN**$(M, 0)$ (this is the base case).

step 2 Divide M into a left half, M_L and a right half, M_R, each of size $n/2$.

step 3 $(M_L, C_L) \leftarrow$ **Conflicts**$(M_L, n/2)$ (Note: M_L is now sorted.)

step 3 $(M_R, C_R) \leftarrow$ **Conflicts**$(M_R, n/2)$ (Note: M_R is now sorted.)

step 4 Merge M_L and M_R, counting conflicts during the merge. Initialize C_0 to be 0, and initialize i and j to be 1. Whenever we compare $M_L[i]$ and $M_R[j]$ and $M_L[i] < M_R[j]$, increment C_0 by $n/2 - i + 1$. Copy the merged array back into M.

step 5 **RETURN**$(M, C_L + C_R + C_0)$.

Correctness: The only thing that is not immediately obvious is how to count conflicts during the "merge" step. Note that whenever $M_L[i] > M_R[j]$, the element $M_R[j]$ is in conflict with all the elements $M_L[i], \ldots, M_L[n/2]$. This creates $n/2 - i + 1$ new conflicts.

Complexity analysis: we need to solve two subproblems of "size" $n/2$. The additional work in the "merge" step is $\Theta(n)$. The recurrence is $T(n) = 2T(n/2) + \Theta(n)$. Using the Master Theorem, the solution is $\Theta(n \log n)$.

2. Illustrate the execution of your algorithm when the second ranking is

$$M_3 > M_1 > M_4 > M_6 > M_5 > M_2 > M_8 > M_7.$$

Solution

To compute **Conflicts**$((M_3, M_1, M_4, M_6, M_5, M_2, M_8, M_7), 8)$, we need to compute:

$$\textbf{Conflicts}((M_3, M_1, M_4, M_6), 4) \text{ and } \textbf{Conflicts}((M_5, M_2, M_8, M_7), 4).$$

Merging M_1, M_3, M_4, M_6 and M_2, M_5, M_7, M_8 yields the following:

i	j		C_0
1	1	$M_1 > M_2$	0
2	1	$M_3 < M_2$	3
2	2	$M_3 > M_5$	3
3	2	$M_4 > M_5$	3
4	2	$M_6 < M_5$	4
4	3	$M_6 > M_7$	4

So $C_0 = 4$.

To compute **Conflicts**$((M_3, M_1, M_4, M_6), 4)$, we need to compute

$$\textbf{Conflicts}((M_3, M_1), 2) \text{ and } \textbf{Conflicts}((M_4, M_6), 2).$$

Merging M_1, M_3 and M_4, M_6 yields

i	j		C_0
1	1	$M_1 > M_4$	0
2	1	$M_3 > M_4$	0

So $C_0 = 0$.

To compute **Conflicts**$((M_5, M_2, M_8, M_7), 4)$, we need to compute

$$\textbf{Conflicts}((M_5, M_2), 2) \text{ and } \textbf{Conflicts}((M_8, M_7), 2).$$

Merging M_2, M_5 and M_7, M_8 yields

i	j		C_0
1	1	$M_2 > M_7$	0
2	1	$M_5 > M_7$	0

So $C_0 = 0$.

To compute **Conflicts**$((M_3, M_1), 2)$, we need to merge M_3 and M_1, which results in $C_0 = 1$.

To compute **Conflicts**$((M_4, M_6), 2)$, we need to merge M_4 and M_6, which results in $C_0 = 0$.

To compute **Conflicts**$((M_5, M_2), 2)$, we need to merge M_5 and M_2, which results in $C_0 = 1$.

To compute **Conflicts**$((M_8, M_7), 2)$, we need to merge M_8 and M_7, which results in $C_0 = 1$.

The final answer is $1 + 1 + 0 + 1 + 0 + 0 + 4 = 7$.

2 Tiling a Square

An *L-tile* consists of three square 1×1 cells that form a letter L. There a four possible orientations of an L-tile:

```
XX    XX    X      X
X      X    XX    XX
```

The purpose of this question is to find a divide-and-conquer algorithm to tile an $n \times n$ square grid with $(n^2 - 1)/3$ L-tiles in such a way that only *one corner cell* is not covered by an L-tile. This can be done whenever $n \geq 2$ *is a power of two.*

Hint: The basic idea is to split the $n \times n$ grid into four $n/2 \times n/2$ subgrids. This defines four subproblems that can be solved recursively. Then you have to combine the solutions to the four subproblems to solve the original problem instance.

1. (Give a pseudocode description of a divide-and-conquer algorithm to solve this problem, briefly justify its correctness and analyze the complexity using a recurrence relation. Remember that we are assuming n is a power of two.

Solution

Algorithm **Tile**$(n, posn)$ ($posn$ specifies the position of the uncovered corner cell, which is one of the four possible values $LLcorner$ $LRcorner$, $ULcorner$ or $URcorner$, where LL denotes "lower left", LR denotes "lower right", UL denotes "upper left" and UR denotes "upper right").

step 1 If $n = 2$ then we can take a single tile to leave any desired corner cell uncovered. This is the base case.

step 2 Otherwise (when $n > 2$), there are four cases, depending on the value of $posn$. For purposes of illustration, assume $posn = LLcorner$ (the other cases are similar. Call **Tile**$(n/2, LR)$, **Tile**$(n/2, LL)$, and **Tile**$(n/2, UL)$. These three recursive calls return three tilings, denoted T_1, T_2 and T_3 respectively, that cover all cells except for the specified corner cells in an $n/2 \times n/2$ array of cells.

step 4 Form the $n \times n$ array

T_1	T_2^1
T_2^2	T_3

using two copies of T_2, denoted T_2^1 and T_2^2.

step 5 Add one more L-tile, filling in the the three contiguous empty cells in the centre of this array.

Correctness: This is pretty obvious. We really only need to note that the empty cells in the tilings T_1, T_2^1 and T_3 form an L-tile which can then be added to the tilings so only one corner cell remains unfilled.

Complexity analysis: We solve three subproblems of "size" $n/2$. The additional work to create the tiling of the $n \times n$ array, given the tilings of the $n/2 \times n/2$ arrays, is $\Theta(n^2)$ (since we have to make a copy of the solution to one of the subproblems). The recurrence is $T(n) = 3T(n/2) + \Theta(n^2)$. Using the Master Theorem, the solution is $\Theta(n^2)$.

Alternative analysis: We can solve four subproblems of "size" $n/2$ (i.e., solve one of the subproblems twice). Now the additional work to create the tiling of the $n \times n$ array, given the tilings of the $n/2 \times n/2$ arrays, is $\Theta(1)$ (since no copying of a solution to a subproblem is required). The recurrence is now $T(n) = 4T(n/2) + \Theta(1)$. Using the Master Theorem, the solution is again $\Theta(n^2)$.

2. Suppose we specify *any single cell* in the $n \times n$ grid (this is not necessarily a corner cell). *Modify your first algorithm* so the remaining $n^2 - 1$ cells are exactly covered by $(n^2 - 1)/3$ L-tiles. You just need to describe the modifications.

Solution

We modify the solution from part 1.

Algorithm **Tile**$(n, posn)$ (here *posn* is any cell in the array).

step 1 If $n = 2$ then we can take a single tile to leave any desired cell uncovered (which of course will be a corner cell in this case). This is the base case.

step 2 Otherwise (when $n > 2$), determine which of the four quadrants contains the *posn* cell. For purposes of illustration, suppose *posn* is in the LL quadrant (the other cases are similar).

step 3 Call

$$\textbf{Tile}(n/2, LRcorner),\ \textbf{Tile}(n/2, LLcorner),$$
$$\textbf{Tile}(n/2, posn) \text{ and } \textbf{Tile}(n/2, ULcorner).$$

These four recursive calls return four tilings, denoted T_1, T_2, T_3, T_4

step 4 Form the $n \times n$ array

T_1	T_2
T_3	T_4

All cells are covered except for the *posn* cell and three centre cells in an $n/2 \times n/2$ array of cells.

step 5 Add one more L-tile, filling in the the three contiguous empty cells in the centre of this array.