École Polytechnique — CSE202 Design and Analysis of Algorithms

Divide-and-conquer for 2D-grid problems

1 Tiling by L-shapes

In this exercice we consider the problem of tiling a punctured $2^n \times 2^n$ square grid (*punctured* means that one square of the grid, called the *hole*, is missing) using what we call *L-shapes*, where an L-shape is any set of 3 unit squares that are inside a (necessarily unique) 2×2 square. Figure 1 shows an example of such a tiling of a punctured 4×4 grid. Each unit square is identified by its cartesian coordinates (x-coordinates are increasing from left to right, and y-coordinates are increasing from bottom to top). In python such a tiling will be recorded as a list of triples (the L-shapes) of integer pairs. For the example in Figure 1 the list would be (up to reordering the triples, and reordering within each triple) `[[(1, 1), (2, 2), (2, 1)], [(0, 0), (0, 1), (1, 0)], [(3, 1), (3, 0), (2, 0)], [(0, 2), (0, 3), (1, 2)], [(3, 2), (2, 3), (3, 3)]]`.

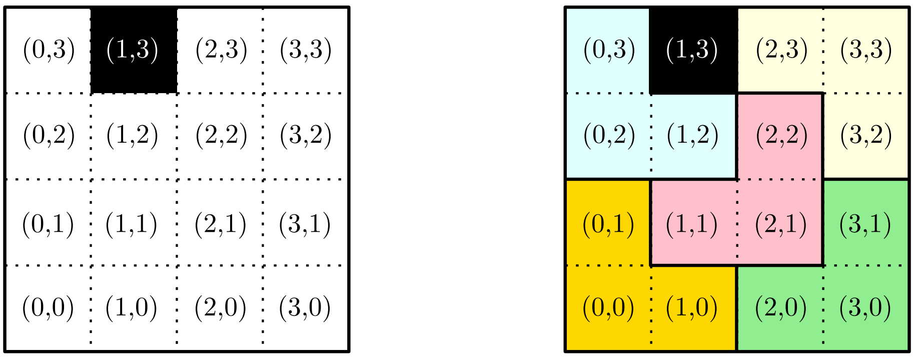

FIGURE 1 – Left : the punctured 4×4 grid G with the hole at position $(1, 3)$. Right : an L-tiling of G.

It will be convenient to allow the minimal x-coordinate (resp. y-coordinate) of a $2^n \times 2^n$ grid to be any fixed value $i \geq 0$ (resp. $j \geq 0$), so that the range of x-coordinates is $[i..i + 2^n - 1]$ and the range of y-coordinates is $[j..j + 2^n - 1]$). The integer n is called the *size* of the grid. We call *punctured grid of type* (n, i, j, a, b) the $2^n \times 2^n$ grid with x-coordinate range $[i..i + 2^n - 1]$, y-coordinate range $[j..j + 2^n - 1]$, and with the hole at coordinates (a, b).

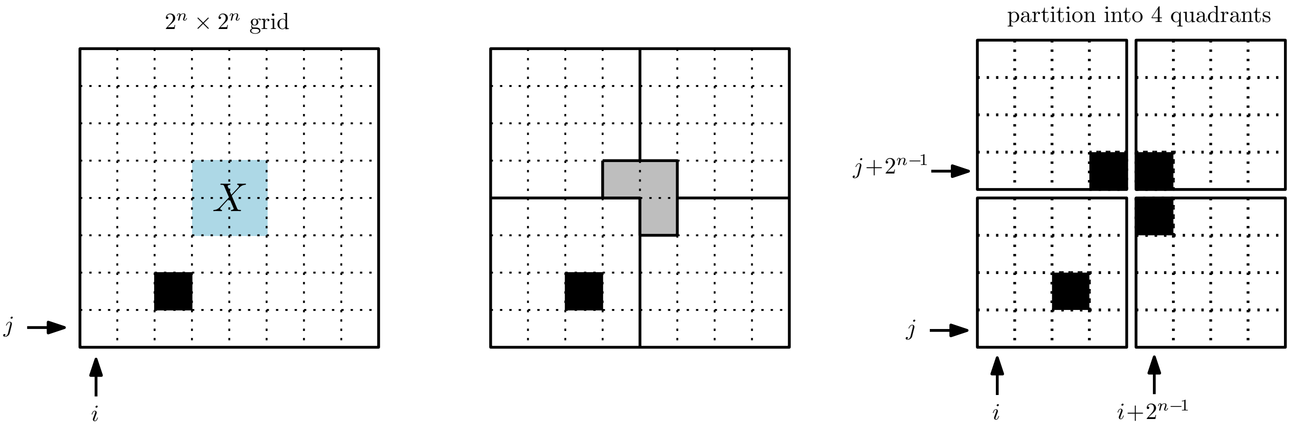

FIGURE 2 – The divide-and-conquer strategy to compute an L-tiling of a punctured $2^n \times 2^n$ grid.

To compute a valid L-tiling of a punctured grid G we use a divide-and-conquer strategy as shown in Figure 2. That is, if the size n is non-zero we decompose the grid G into 4 quadrants (each one a grid of size $n-1$) and call *marked quadrant* the one containing the hole of G. We let the *middle L* of G be the L-shape obtained from the central 2×2 square X by removing the unique square of X that belongs to the marked quadrant (see the left and middle drawing of Figure 2). Then we puncture the marked quadrant to have the same hole as G, whereas the other 3 quadrants are punctured at their unique unit square belonging to X (see the right drawing of Figure 2). After that it just remains to apply recursively the tiling procedure to each of the 4 quadrants.

Question 1. Complete the function `middleL(n,i,j,a,b)` that has to return a triple of the form `[(x1,y1),(x2,y2),(x3,y3)]` giving the coordinates of the three unit squares of the middle L (any order of the 3 squares is accepted), for the punctured grid of type (n,i,j,a,b).

Question 2. Complete the function `lower_left_hole(n,i,j,a,b)` that has to return the coordinates `k,l` of the hole in the lower left quadrant of the punctured grid of type (n,i,j,a,b). Complete similarly the functions for the other 3 quadrants `lower_right_hole(n,i,j,a,b)`, `upper_left_hole(n,i,j,a,b)`, `upper_right_hole(n,i,j,a,b)`.

Question 3. Complete the recursive function `tile(n,i,j,a,b)` such that after calling it, the global variable `Llist` contains a list of triples of integer-pairs, these triples corresponding to the L-shapes forming a valid tiling of the punctured grid of type (n,i,j,a,b).

Once it passes the test you can run the function `display_tiling_with_random_hole(n)` that displays the L-tiling of the $2^n \times 2^n$ grid punctured at a unit square chosen at random.

Exercise (ungraded). By applying the master theorem you should check that the complexity of computing the tiling is $O(N^2)$ with $N = 2^n$, i.e., the complexity is of the order of the area (which is also the number of unit squares) of the grid to be tiled.

2 Finding peaks in arrays and matrices

Let $N \geq 1$ and $L = [a_0, \dots, a_{N-1}]$ be a list of numbers in $\mathbb{R}$. For $i \in [0..N-1]$ we say that there is a peak at position i if $a[i]$ is at least as large as any of its neighbours. For instance for $L = [2,2,1,3,3,4,1]$ there are peaks at indices $0,1,3,5$.

Question 4. Write a simple iterative function `peak_naive(L)` that returns the leftmost peak position in L, in time $O(N)$. (Be careful to return the position, not the peak-value)

We now discuss a divide and conquer approach, shown in Figure 3. For $0 \leq p < q \leq N$ we let $L[p:q]$ be the list $[a_p, \dots, a_{q-1}]$. If $q - p \geq 2$ we let $\ell = \lfloor (p+q)/2 \rfloor$, and we call *middle-entries* of $L[p:q]$ the two entries $a_{\ell-1}$ and a_ℓ. If $a_{\ell-1} \geq a_\ell$ but $a_{\ell-1}$ is not a peak of $L[p:q]$, then one can easily see that any peak of $L[p:\ell]$ is also a peak of $L[p:q]$. On the other hand, if $a_{\ell-1} < a_\ell$ but a_ℓ is not a peak of $L[p:q]$, then any peak of $L[\ell:q]$ is also a peak of $L[p:q]$.

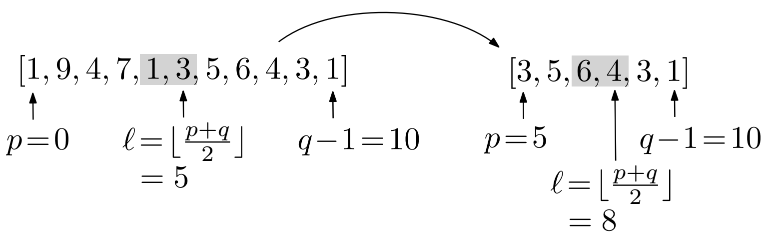

FIGURE 3 – The DAC approach to find a peak, illustrated on a list of length 11. The two middle entries are at positions $\{4,5\}$, the larger one is at position 5. It is not a peak, so we recurse in the right part of L, i.e., the array $L[5:11]$. Then the two middle entries are at positions $\{7,8\}$. The larger one is at position 7, and it is a peak, so we end there and return 7 as a peak position.

Question 5. Write a DAC function `peak(L)` that returns a peak position of a list L (assuming L is not empty) following the approach shown in Figure 3. It is convenient to write also a function `peak_aux(L,p,q)` that returns (assuming $p < q$) a peak position of $L[p:q]$.

Exercise (ungraded). Using the master theorem you can check that the complexity of this algorithm is $O(\log(n))$, a significant improvement over the naive iterative method, which runs in time $O(n)$.

We now consider the 2D version of the problem (finding a peak position in a matrix). Let $M = [[m_{0,0}, \ldots, m_{0,J-1}], [m_{1,0}, \ldots, m_{1,J-1}], \ldots, [m_{I-1,0}, \ldots, m_{I-1,J-1}]]$ be an $I \times J$-matrix. (Beware that, in contrast to the first exercise, the rows are ordered from top to bottom, the usual convention for matrices.) Recall that in Python, the coefficient $m_{i,j}$ is obtained as `M[i][j]`. We say that there is a peak at position (i, j) if $M[i][j]$ is at least as large as any of its neighbours ($M[i][j]$ has at most 4 neighbours, which are those of $M[i-1][j]$, $M[i+1][j]$, $M[i][j-1]$, $M[i][j+1]$ whose indices are within the bounds of M).

peak at position $(0, 1)$

$$\begin{bmatrix} 1 & 3 & 3 & 2 \\ 2 & 2 & 6 & 4 \\ 2 & 5 & 3 & 4 \end{bmatrix}$$

FIGURE 4 – A 3×4-matrix (the peaks are circled).

Question 6. Write a function `is_peak(M,i,j)` that returns `true` if there is a peak of M at position (i, j), and returns `false` otherwise (we assume that (i, j) is within the bounds of M).

Question 7. Write a simple iterative function `peak2d_naive(M)` that returns the first encountered position `(i,j)` where there is a peak. We assume that the entries of M are visited in the following order : $m_{0,0}, \ldots, m_{0,J-1}$, then $m_{1,0}, \ldots, m_{1,J-1}, \ldots, m_{I-1,0}, \ldots, m_{I-1,J-1}$, The method should run in time $O(I \times J)$ in the worst case.

We are now going to describe a DAC algorithm for finding a peak in a matrix (the approach is illustrated in Figure 5). For $p < q$ and $r < s$ we let $M[p : q][r : s]$ be the submatrix of entries $M[i][j]$ where $p \leq i < q$ and $r \leq j < s$. Given two indices $c < d$ we define the index-set $A(c, d)$ as

$$A(c, d) = \{c\} \text{ if } d - c = 1, \quad A(c, d) = \{c, \ell - 1, \ell, d - 1\} \text{ if } d - c \geq 2, \text{ where } \ell = \lfloor (c + d)/2 \rfloor.$$

For $p < q$ and $r < s$ we define $F(p, q, r, s)$ as the set of index pairs (i, j) such that either $i \in A(p, q)$ and $j \in [r..s - 1]$, or $i \in [p..q - 1]$ and $j \in A(r, s)$, see Figure 5 for two examples where each time $F(p, q, r, s)$ is shown in gray. We let x be the maximal value of $M[i][j]$ over all $(i, j) \in F(p, q, r, s)$ and we define a *pivot* for $M[p : q][r : s]$ as a pair $(i, j) \in F(p, q, r, s)$ such that $M[i][j] = x$.

Question 8. (Do not use a DAC approach for this question) Write a function `pivot(M,p,q,r,s)` that receives a matrix M and 4 indices p, q, r, s (with $p < q$ and $r < s$) and returns a pivot `(i,j)` for $M[p : q][r : s]$.

This paragraph is there just to define a type of submatrices that will have good properties to carry on a DAC recursive approach. For a submatrix $M[p : q][r : s]$, the *outer frame* of $M[p : q][r : s]$ is the set of entries $M[i][j]$ such that either $i = p$ or $i = q - 1$ or $j = r$ or $j = s - 1$. An entry (i, j) of M is called an *exterior neighbour* of $M[p : q][r : s]$ if (i, j) is not in $M[p : q][r : s]$ but is adjacent to an entry in $M[p : q][r : s]$ (such a neighbour being necessarily in the outer frame of $M[p : q][r : s]$). We let z be the maximal value over all entries of the outer frame of $M[p : q][r : s]$. Then the submatrix $M[p : q][r : s]$ is called *good* if it has no exterior neighbour that it strictly larger than z. Note that in particular $M = M[0 : I][0 : J]$ is a good submatrix of M (in this case we have no exterior neighbour).

Let $M[p : q][r : s]$ be a good submatrix of M, and let (i, j) be a pivot of $M[p : q][r : s]$. Note that for $q - p \leq 4$ or $s - r \leq 4$, the set $F(p, q, r, s)$ exactly covers all entries of $M[p : q][r : s]$. Hence (since we consider a good submatrix) the entry (i, j) is a peak of the whole matrix M.

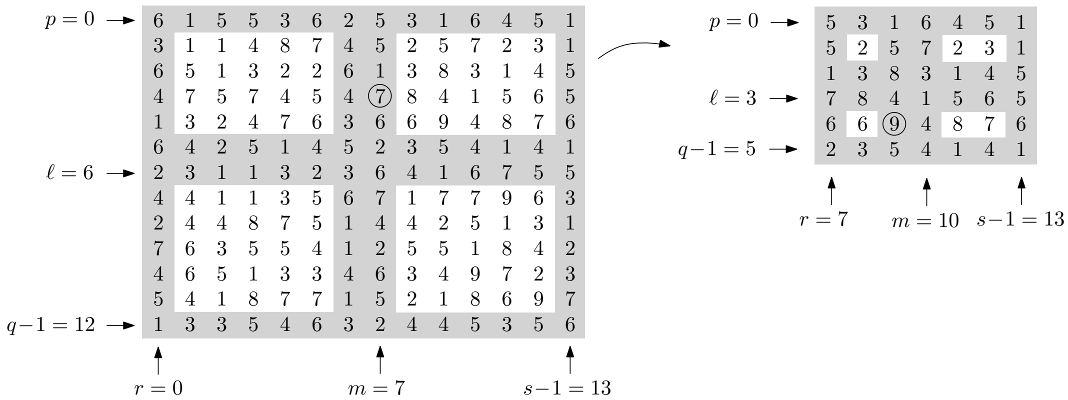

FIGURE 5 – The DAC approach to find a peak, illustrated on a 13×14-matrix M. The first step is to look for a global maximum in the 'frame' $F(0, 13, 0, 14)$. The maximal value is 7 and is reached 5 times in the frame, we choose the one at position $(3, 7)$ as the pivot. Since it is not a peak of the matrix (the right neighbour is larger), we have to recurse in the quadrant containing the pivot, i.e., the submatrix $M[0:6][7:14]$. The maximal value in $F(0, 6, 7, 14)$ is uniquely attained at position $(4, 9)$, which is a peak of M. We return `(4,9)` as a peak position of M.

If $q - p > 4$ and $s - r > 4$, we let $\ell = \lfloor (p + q)/2 \rfloor$ and $m = \lfloor (r + s)/2 \rfloor$, and let (i, j) be a pivot of $M[p : q][r : s]$. Note that $M[p : q][r : s]$ is partitioned into the 4 submatrices $M[p : \ell][r : m]$, $M[p : \ell][m : s]$, $M[\ell : q][r : m]$, $M[\ell : q][m : s]$, which correspond respectively to the upper-left, upper-right, lower-left and lower-right quadrant within $M[p : q][r : s]$. Let Q be the one of the four quadrants that contains (i, j) (in the first drawing of Figure 5 it is the upper-right quadrant, while in the second drawing it is the lower-left quadrant). The crucial point is that, by the properties of the pivot, Q is also a good submatrix. Hence if (i, j) is not already a peak of M, then we can recurse to the quadrant Q.

Question 9. Write a DAC method `peak2d(M)` that returns a peak position of a matrix M, following the approach shown in Figure 5. As in the 1D case, it is convenient to write an auxiliary method `peak2d_aux(M,p,q,r,s)` that returns a peak of M within a submatrix $M[p : q][r : s]$ that is assumed to be a good submatrix.

Exercise (ungraded). Using the master theorem you can check that the complexity of this algorithm is $O(n)$, again a significant improvement over the naive iterative method, which runs in time $O(n^2)$.