Subproblems

Define $OPT(i,j,b)$ to be the maximum total score that can be obtained by a feasible climbing path in the bottom-leftmost $(i+1)\times(j+1)$ grid with an energy budget of b. We define $i+1$ to be the number of columns and $j+1$ to be the number of rows. When $i=0$ and $j=0$, this corresponds to a problem with a 1×1 grid.

We also define $C(i,j,b)$ to be the cost of a path to the top-right element in the $(i+1)\times(j+1)$ grid with an energy budget of b. This path does not have to be feasible, as the algorithm will later check if a path is feasible.

Recurrences

$$OPT(i,j,b) = \max\begin{cases} \max\{OPT(i-1,j,b), OPT(i,j-1,b)\} & \text{if } i>0, j>0 \\ OPT(i,j-1,b) & \text{if } i=0, j>0 \\ OPT(i-1,j,b) & \text{if } j=0, i>0 \\ S[i][j] + \max\{OPT(i-1,j,b-c_r), OPT(i,j-1,b-c_u)\} & \text{if } b\geq c_r(*), b\geq c_u(**), i>0, j>0 \\ S[i][j] + OPT(i,j-1,b-c_u) & \text{if } b< c_r(***), b\geq c_u(**), j>0 \\ S[i][j] + OPT(i-1,j,b-c_r) & \text{if } b\geq c_r(*), b< c_u(****), i>0 \end{cases}$$

(*) and $b \geq C(i-1,j,b-c_r)+c_r$
(**) and $b \geq C(i,j-1,b-c_u)+c_u$
(***) and $b < C(i-1,j,b-c_r)+c_r$
(****) and $b < C(i,j-1,b-c_u)+c_u$

Base Cases

The base cases are when $i=0, j=0$. In this case, $OPT(0,0,b)=S[0,0]$ and $C(0,0,b)=0$.

Maximum Total Score

The maximum total score that can be obtained is in the subproblem $OPT(n-1,m-1,B)$. We define n to be the number of columns and m to be the number of rows.

Proof of Correctness

The base cases are sufficient. If $i>0, j>0$, each $OPT(i,j,b)$ depends on some $OPT(i-1,j,b)$ and $OPT(i,j-1,b)$. Otherwise, if $i=0, OPT(i,j,b)$ depends on some $OPT(i,j-1,b)$ and if $j=0, OPT(i,j,b)$ depends on some $OPT(i-1,j,b)$. Thus, the recurrence will always end up at $OPT(0,0,b)$ for some b. We have $OPT(0,0,b)=S[0,0]$ since the maximum total score that can be achieved in a 1×1 grid is the score of the bottom-left element. As we start from this position, the total cost is 0, as described by $C(0,0,b)=0$.

The recurrence is correct. First we observe that the conditions of each recurrence ensure that any parameter would have a value of at least 0. The first three recurrences represent the case where grid position (i,j) is not chosen for the optimal climbing path. In this case, the maximum score of the grid is given by the highest maximum score of the grid position to the left or below (i,j) since we are only allowed to climb rightwards or upwards. If the maximum score is equal we choose either one. We need three separate conditions, as some grid positions may not have a grid position to the left or a grid position below. As (i, j) is not chosen, b does not change. Since $C(i, j, b)$ represents the cost of a path to (i, j), regardless of whether (i, j) is actually chosen, we use the previous grid position with the highest maximum score and set $C(i, j, b)$ (in this case) to be the sum of the cost of the previous path and the cost of movement to (i, j). For example, if the grid position below had the highest maximum score, then $C(i, j, b) = C(i, j - 1, b) + c_u$. We also observe that even if there were multiple paths to a grid position, the total cost is the same due to equivalent numbers of up and/or right movements to reach a certain grid position.

The last three recurrences represent the case where grid position (i, j) is chosen for the optimal climbing path. As (i, j) is chosen, there is a cost of movement c, which can be c_u or c_r, associated with this choice. Hence, we need to compare the maximum scores of previous grid positions (left or below) with an updated budget $b - c$. In order to determine whether the current climbing path is feasible, we also need to check if the cost of current path is less than or equal to the current budget. To calculate the cost of the current path, we use the sum of c and the value from C of the previous grid position. In this case, $C(i, j, b)$ is set to this new sum, and this does not have to be feasible as it is always checked by the procedure outlined previously.

By taking the maximum of these two cases, we ensure that $OPT(i, j, b)$ returns the maximum total score of a feasible climbing path in this grid. $C(i, j, b)$ is set by the case which contains the maximum score.

The optimal score for the original problem is $OPT(n - 1, m - 1, B)$ as by definition of OPT, this is the maximum total score of a feasible climbing path that can be achieved by a $n \times m$ grid (as $n - 1$ and $m - 1$ denotes the top-right grid position) with an energy budget of B.

Time and Space Complexity

Time Complexity. The base cases take $O(B)$ time as there are $B+1$ of them and setting them to $S[0, 0]$ for OPT takes constant time. For the recursive cases in OPT, there are $n \times m$ iterations to traverse every grid position. Each iteration has B iterations to traverse every possible budget up to B. These final iterations take $O(1)$ time to compute since it only requires a constant number of comparisons, arithmetic operations and array lookups. Returning the optimal value takes $O(1)$ time for an array lookup. Thus the total time complexity for OPT is $O(nmB)$. Similarly, the total time complexity for C is $O(nmB)$. Hence, the total time complexity for this algorithm is $O(nmB) + O(nmB) = O(nmB)$

Space Complexity. The space required by the algorithm is dominated by the number of subproblems, which is $O(nmB)$ as we have a subproblem $OPT(i, j, b)$ and $C(i, j, b)$ for each $0 \leq i \leq n - 1, 0 \leq j \leq m - 1$, and $0 \leq b \leq B$.