Task1

Sub problem:

OPT(i, j ,e) = maximum of happiness that the quokka can achieve at the end of its journey from the top right to bottom left in matrix with width i and height j and e is the ending point.

Task2

Recurrence:

Case1: In i x j matrix, the left bottom position is reached by jumping.

OPT(i,j,e) = max[$OPT(i, j-k, w) + p(i, j, e)$] the quokka jumps k rows, k takes each value from 0≤k<j – iteration finding maximum

Case2: In i x j matrix, the left bottom position is reached by hopping left

Considering all the available path reaching right side of ending position from starting point, we can therefore make sure that it is a global maximum. We define H_m is happiness of each available path. H_m= max[OPT(i - m, j - k, w)] + p(i - m, j, w) + $\sum(p(i-q, j, w) - c)$ "m" represents how far away from right hand side of ending position (0<m<i). "k" means k rows to jump, k takes each value from 1≤k<j. "q" takes each value from m>q>0(decreasing order). "c" means the current consecutive cost corresponding to each position that will reach.

OPT(i,j,e) = $\max[H_m + p(i, j, e) - c]$

OPT(i,j,e) = max{$\max[OPT(i, j-k, w) + p(i, j, e)]$, $\max[H_m + p(i, j, e) - c]$ }

Notation: w is the previous end point in the previous matrix. "$p(i, j, e)$" is the value of current bottom left position e.

Base Case:

- OPT(1,1,e) = 0, which represents the starting point.
- OPT(i,0,e)= -∞, OPT(0,j,e)= -∞, OPT(0,0,e)= -∞ (impossible positions).
- OPT(i,1,e)= OPT(i-1,1,e)+p(i,j,e) – c.

Overall:

$$\text{OPT(i,j,e)}=\begin{cases} 0 & starting\ point \\ -\infty & impossible\ position \\ \text{OPT}(\text{i}-1,1,\text{e})+\text{p(i,j,e)}-\text{c} & OPT(i,1,e) \\ \max\{\max[OPT(i,j-k,w)+p(i,j,e)], \max[H_m+p(i,j,e)-c]\} & otherwise \end{cases}$$

Task3:

RECAP:

- OPT(i,j,e) = max{ max[$OPT(i, j-k, w) + p(i, j, e)$] , $\max[H_m + p(i, j, e) - c]$ }
- H_m= max[OPT(i- m, j-k, w)] + p(i - m, j, w) + $\sum(p(i-q, j, w) - c)$

Base case

- Starting point, the value of starting point is 0
- i x 1 matrix, because there is no way for the quokka jumping to reach those positions, according to the recurrence relation, there is only one path for the quokka to reach ending. In this case, H_m=p(i - m, j, w) + $\sum(p(i-q, j, w) - c)$ equals to OPT(i-1,1,e). Therefore, $\max[H_m + p(i, j, e) - c]$ transforms to OPT(i-1,1,e)+p(i,j,e) – c Also, because $OPT(i, 0, w)$ is impossible position, which equals to -∞, so algorithm always produces the only path to earn the maximum happiness based on OPT(i-1,1,e)+p(i,j,e) – c.

Inductive step

Assumption: OPT(i-m, j, e), OPT(i, j-k, e) and OPT(i-m, j-k, e) output the correct maximum happiness. (0<m<i ; 0<k<j). In other words, for any sub matrixes inside i x j matrix, my algorithm can output the correct maximum happiness.

Proving OPT(i, j, e) also output the correct maximum happiness.

Jumping:

By assumption, we have calculated OPT(i, j-k, w) correctly output maximum happiness, and jump does not reduce the happiness, which means previous optimal outputs can affect current maximum happiness, so choosing the maximum from the set outputs of OPT(i, j-k, w) plus current position value must be the max happiness.

Hopping:

By assumption, because we have calculated OPT(i-m, j-k, w) correctly output maximum happiness, and max[OPT(i- m, j-k, w)] + p(i - m, j, w) is jumping, which is already proven its correctness, H_m must outputs the correct maximum value for each path. As a result, $\max[H_m + p(i, j, e) - c]$ must output the correct maximum happiness ---Iteration finding maximum

Overall, OPT(i,j,e) still outputs the correct maximum happiness.

Conclusion:

Because algorithm works correctly for base case, and for 1…i x 2….j matrixes, it is sufficient to prove that my algorithm can output correct maximum happiness for any sizes of matrix.

Task4:

1. Time complexity

   For each 1 to heights(h)

   For each 1 to widths(w)

   If (1 x j matrix) OPT(1,j,e) →O(h)

   If (i x 1 matrix) OPT(i,1,e) →O(1)

   Else OPT(i,j,e) →O(wh) + O(h) = O(wh)

   By jumping [described in previous section] → O(h)

   By hopping left [described in previous section] → O(wh)

   Overall, $O(w^2h^2)$

2. Space complexity

   Using a 2D array (w x h matrix) to store optimal values for each cell.

   O(wh)