Solution 3: Dynamic Programming

Answer 1

a) The base case is: P(n)=0 if n=0

b) Since v_i is in the optimal solution, we need to select this coin and the remaining value becomes n-v_i. Thus, we have the following relationship:

$$P(n) = 1 + P(n - v_i)$$

c) Since we do not know what v_i is and we need to obtain the optimal solution of P(n), we should select the smallest optimal value of the sub-problems P(n-v_i) in order to combine the optimal value of problem P(n). We have:

$$P(n) = \begin{cases} 0 & if\ n = 0 \\ 1 + \min_{n \geq v_i} P(n - v_i) & if\ n > 0 \end{cases}$$

Answer 2

The table c is shown in Table 1. One LCS is $\langle 0, 1, 0, 1, 0, 1 \rangle$, which has the length 6.

Table 1: Table c

	i	0	1	2	3	4	5	6	7	8
j		$x[i]$	1	0	0	1	0	1	0	1
0	$y[j]$	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1$\nwarrow$	1	1	1	1	1
2	1	0	1	1	1	2$\nwarrow$	2	2	2	2
3	0	0	1	2	2	2	3$\nwarrow$	3	3	3
4	1	0	1	2	2	3	3$\uparrow$	4	4	4
5	1	0	1	2	2	3	3	4$\nwarrow$	4	5
6	0	0	1	2	3	3	4	4	5$\nwarrow$	5
7	1	0	1	2	3	4	4	5	5$\uparrow$	6
8	1	0	1	2	3	4	4	5	5	6$\nwarrow$
9	0	0	1	2	3	4	5	5	6	6$\uparrow$

Table 1: Table c

	i	0	1	2	3	4	5	6	7	8
j		$x[i]$	1	0	0	1	0	1	0	1
0	$y[j]$	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1 $\nwarrow$	1	1	1	1	1
2	1	0	1	1	1	2 $\nwarrow$	2	2	2	2
3	0	0	1	2	2	2	3 $\nwarrow$	3	3	3
4	1	0	1	2	2	3	3 $\uparrow$	4	4	4
5	1	0	1	2	2	3	3	4 $\nwarrow$	4	5
6	0	0	1	2	3	3	4	4	5 $\nwarrow$	5
7	1	0	1	2	3	4	4	5	5 $\uparrow$	6
8	1	0	1	2	3	4	4	5	5	6 $\nwarrow$
9	0	0	1	2	3	4	5	5	6	6 $\uparrow$

1 0 1 0 1 0

Answer 3

(a)

$$Base\ Case: \begin{cases} c[x][0] = 0 & , 0 \le x \le n+1 \\ c[x][n+1] = 0 & , 0 \le x \le n+1 \\ c[0][y] = 0 & , 0 \le y \le n+1 \\ c[n+1][y] = 0 & , 0 \le y \le n+1 \end{cases}$$

The base case is used to prevent the out-of-range cases, i.e., the checker cannot move to the diagonal right-above (left-above) square if it is already in the rightmost (leftmost) column.

Recursive Case:

$$c[x][y] = max\{c[x-1][y-1], c[x][y-1], c[x+1][y-1]\} + p[x][y], 1 \le x, y \le n$$

(b) From the recursive case of $c[x][y]$, it depends on $c[x-1][y-1]$, $c[x][y-1]$, $c[x+1][y-1]$, therefore, the algorithm should fill the table c from the bottom edge to the top edge.

Refer to Algorithm 1.

Algorithm 1: CheckerBoard($p[0..n+1]$)

```
create two new tables c[0..n+1][0..n+1] and s[1..n][1..n]
for x ← 0 to n+1 do
    c[x][0] ← 0
    c[x][n+1] ← 0
for y ← 0 to n+1 do
    c[0][y] ← 0
    c[n+1][y] ← 0
for y ← 1 to n do
    for x ← 1 to n do
        x* ← x
        if c[x-1][y-1] > c[x][y-1] and c[x-1][y-1] > c[x+1][y-1] then
        // c[x-1][y-1] is the largest
            x* ← x-1
        if c[x+1][y-1] > c[x][y-1] and c[x+1][y-1] > c[x-1][y-1] then
        // c[x+1][y-1] is the largest
            x* ← x+1
        c[x][y] ← c[x*][y-1] + p[x][y]
        s[x][y] ← x* - x
// scan the first row of the array c and get the maximum amount
maxAmount ← 0
for x ← 1 to n do
    if c[x][n] > maxAmount then
        maxAmount ← c[x][n]
return maxAmount
```

(c) Figure 2 shows the contents of the table c and s. Here we only show the squares with $x, y \geq 1$.

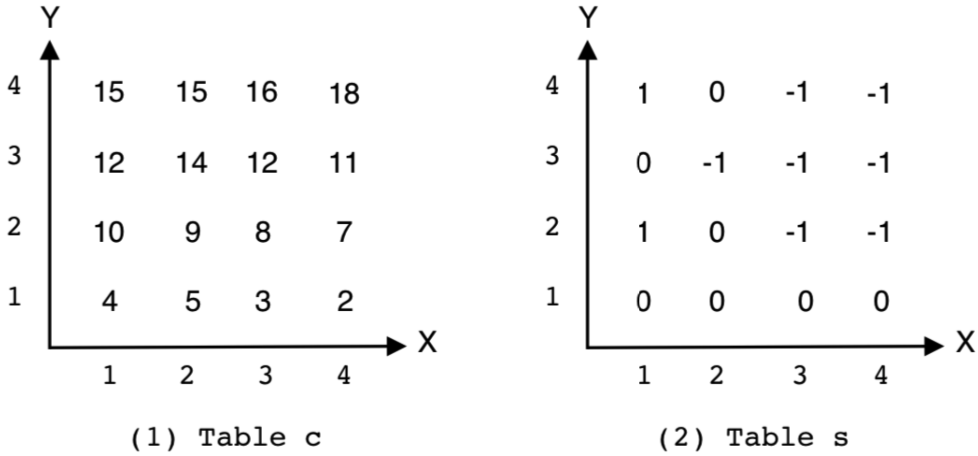

Figure 2: Checker Board Example

Following the array s, we can construct the path. If $s[x][y] = z$, then the previous square is $(x + z, y - 1)$. We start from the square with the maximum amount, i.e., $(4, 4)$. Because $s[4][4] = -1$, the previous square is $(3, 3)$. Then check $s[3][3] = -1$, then the previous square is $(2, 2)$. Similarly, we check $s[2][2] = 0$, then refer to $(2, 1)$. Thus, the path is $(2, 1) \rightarrow (2, 2) \rightarrow (3, 3) \rightarrow (4, 4)$.