We can use a Knapsack type algorithm to do this. Each trip to a city is like an item we can pick. The cost of travel and budget acts as the weight and capacity of the knapsack, and the max value of the knapsack is the same as the maximum number of cities visited.

Max_Visits(A[1…n],B)-> A vector/array of costs for each city, but value is not needed since each city added to travel is just +1 to the maximum number of cities visited.

```
      D[0…n, 0…w] <- initialized to all zeroes ;Array where D[i,C], where i is the number of visitable cities with a budget ≤ C.
      For i = 1 to n do:
            For C = 1 to B do:
                  If A[i]>C then:
                        D[i,C]<- D[i-1,C];cant afford within budget, skip
                  Else:
                        D[i,C]<-max(D[i-1,C],D[i-1, C-A[i]]+1);choose max of either skipping city, or visiting. (+1 because each city is +1 to the max number of cities visited.)
      Return: D[n,B]
```

Runtime analysis: lines 4-7 take O(1) time, just simple comparisons. The inner loop loops B times, then the outer loop loops n times. This means that the total runtime for the algorithm is O(1)*nB equals O(nB).

Space Analysis: Each entry in the stored array takes up O(1) space. The array is of size (n+1)x(B+1) (Since the arrays start at 0) this means that if each each entry takes up O(1) space and there are (n+1)(B+1) entries, the total space of the algorithm is O(1)*(n+1)(B+1) = O(nB).