**DP** for solving the **gap-subsequence** question **P :**

Given a global array **S** of size **n** & another corresponding array **Gap**, the function **return\_max\_score(x)** will return the maximum score of the gap-subsequence from **S[1…x].** We define a problem **Q** to solve **P.**

**1. Specify a problem Q (which will eventually solve P) :**

A sentinel of value=1 is added to the array **S,** so now **S’ = 1.S ;** & another sentinel of value=0 is added to the beginning of the array **Gap**, so now **Gap’ = 0.Gap.**

Previously indexing started from 1, so after adding the sentinels to both the arrays, the indexing starts from 0.

In problem **Q**, the function **max\_score(x)** will return the maximum score of the gap-subsequence from **S’[0…x].**

The sentinel value 1 in **S’** denotes that the **max\_score** function is called with that value i.e. the first element will always be included in the maximum score subsequence. Later we will discard this element once we have found our required subsequence.

We have assumed a **Memoization** table **M** of size **n+1**, & a Pointer table **P** of size **n+1.**

The **Memoization** table and **Pointer** table are filled right to left.

**2. Give a recurrence expression/formula or recursive algorithm for solving P :**

**max\_score(x)** will return the maximum score of the gap-subsequence from **S[1…x]**

**max\_score( x )** = 0 *for* **x > n**

Otherwise: **max\_score**( x ) = *argmax***(** **max\_score**(x+1), **max\_score**(x + g[x] +1) + S[i]) **)**

**3. Prove correctness of recurrence relation :**

In the given problem, we have to find the maximum gap-subsequence score, given **S** & its corresponding array **Gap**.

So any element, say **k** in the array, has two choices:

1. Exclude that element and then find the maximum score

i.e **max\_score**( k )= **max\_score**( k+1 ) for **k <= n**,

**max\_score**( k ) = **0** for **k > n**.

1. Include that element and then find the maximum score

**max\_score**( k ) = **max\_score**( k+g[k] +1) + S[ k ] for k <= n,

**max\_score**( k ) = 0 for k > n.

Since **max\_score** returns, a maximum score of **S’[k…n]**, **max\_score(0)** will return a **max\_score** of **S’[0…n].**

Building on the above intuition, an exhaustive search will be done w.r.t the arrays **S’** and **Gap’** if thought in terms of recursion.

Assuming we have found **max\_score( k )**, i.e. we have found the maximum gap-subsequence of **S’[k…n]**, then to find the maximum score of **S’[(k-1)...n]**, **max\_score( k - 1 )** has two choices :

* **(k-1)th** element is included, which means **S’[ k -1 ]** is added to the cumulative maximum score, so **S’[ k-1 ]** will be considered towards calculation of the maximum score of **S’[k…n].** Since no element in **S’** can be considered till the index **( k-1 + Gap’[k-1] )** can be taken.

Hence, **max\_score** will be called as **max\_score( (k-1) + g[ k-1 ] +1 )**. So finally **max\_score(k-1)** = **S[k-1] + max\_score((k-1)+g[k-1] +1)**

//OVERLAPPING SUBPROBLEMS

* Secondly, **(k-1)th** element is not included which implies maximum score until **max\_score( k )** equals **max\_score( k-1)**.

**4. Describe a memoization data structure :**

The memoization table **M** is an **1-D** array of size **n+1**, and any index of M, say kth index gives a maximum score of **S’[k…n]**.

The starting index 0 of the memoization table will give the final maximum score+1 of **S’[0..n].** *{ +1 is due to the sentinel added above, which we will subtract later. }*

The pointer table **P** is also an **1-D** array of size **n+1**, and any index of **P**, say **kth**index gives the index of the next element which should be considered in order to maximize the score.

**5. Give an algorithm/ordering for solving P for all values.**

The memoization table will be filled as follows:

**Assume M[ k ] =0 if k>n** i.e.for out of bound indices M[k]=0

for i in (n…0) :

a = M[ i +1 ] // excluding ith element

b = M[ i + Gap’[ i ] +1 ] + S’[ i ] // including ith element

if a > b:

M[ i ] = a // filling Memoization array if element is excluded

P[ i ] = i+1 // filling Pointer array if element is excluded

else:

M[ i ] = b // filling Memorization array if element is included

if (i + Gap’[ i ] +1) >n: // if max score is obtain by including itself

P[ i ] = NULL

else:

P[ i ] = i + Gap’[ i ] +1 // filling Pointer array if element is included

**6. How to solve problem Q from values**

M[0] also includes one additional sentinel score (of value 1) in addition to the maximum score. The actual maximum score is thus provided by **M[0]** -**1**.

Pointer table **P** contains the Sequence of indexes that are to be included in order to obtain the maximum score, the first value of the sequence is given by **P[ 0 ]**.

**7. What is space and time complexity for solving problems?**

Since the above problem has two 1-D arrays: one used for **Memoization** & the other for **Pointer**. Hence their total size = **2(n+1)**, so Space complexity - **O( n )**.

While filling the Memoization and Pointer tables, the function requires traversing over all elements of **S’** and corresponding **Gap’** which are each of size **(n+1)**.

So, the time complexity of filling memorization and Pointer tables is O(n).

To retrieve the sequence, traversal over elements of the memoization table is required which has (**n+1)** elements. So Time complexity of retrieving the sequence is **O(n).**

Total Time complexity = Time complexity of filling memorization and Pointer tables + retrieve the sequence = **O(n) + O(n)**

So, overall Time complexity = **O(n)**

**8. How to obtain the optimal structure?**

To compute the maximum score sequence itself, along with the maximum score, pointers are stored in the **Pointer** table. This can be implemented by storing indexes that point to other indexes of **S’**. We denote the pointer associated with **S’[j]** as **P[j]**. Let **Sj** be the sequence with the maximum score in **S’[j ... n]**.

Since **P[0]** corresponds to sentinel which is always included, **P[0]** will give the index of the next value that must be included in order to obtain the maximum score

i = P[ 0 ]

print( M[ 0 ] -1 ) //Sentinel was always included, so decrease the score by 1.

while (i <=n && i != NULL):

while M[ i ] == M[ i+1 ]:

i++ //since we want to retrieve that original element that propagated through copying to this new cumulative maximum

print(S[ i ] ) // printing the sequence index

i = P[ i ]

This prints all the elements of S which are included in the maximum sequence starting from **P[0]** until we hit a **NULL**.