**ECE374 Assignment 5**

Due 03/20/2023

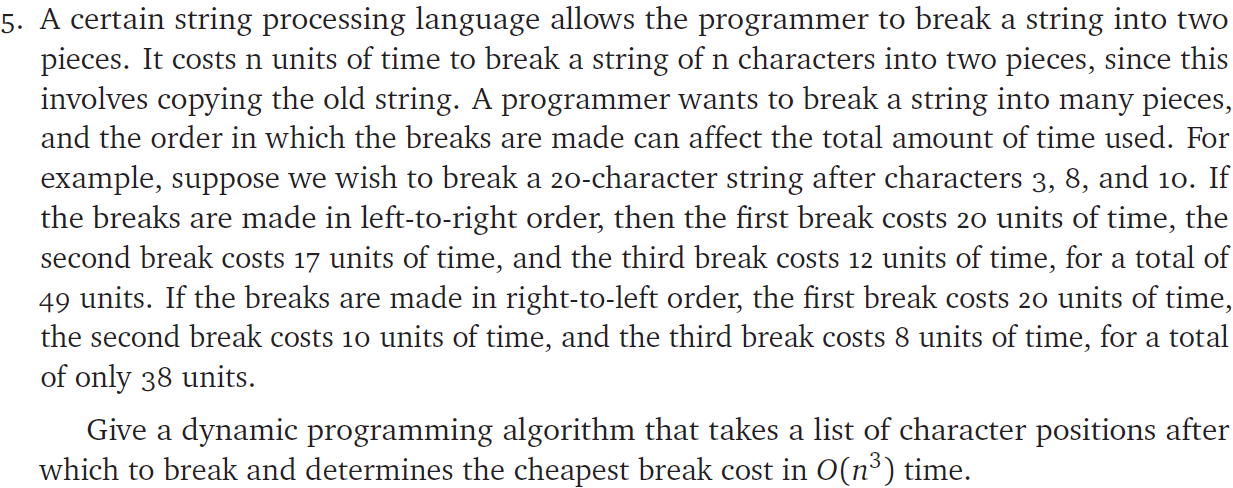
**Group & netid**

*Chen Si chensi3*

*Jie Wang jiew5*

*Shitian Yang sy39*

**Problem 5**

****

## Solution:

**Intuition:**

1. We should extend the array indexing [1, 2, …, n] to [0, 1, 2, …, n, n+1] by adding two “virtual nodes” to both ends of the array. Then, we could convert the problem into a recursive problem to determine the cost of “make k cuts with additional 2 cuts at index 0 and index n”.

**Cases**

* Base case:

When we have two cuts at index **cuts[i]** and **cuts[j]**, and i=j or i+1=j (they are same or adjacent)

🡪 the cost would be **0**, as we don’t need to perform any cuts on them.

* General Case:

If we are to make cuts at index **cuts[i]** and **cuts[j]** (j>i), with several other places to be cut in between. We could consider that the minimum cost is composed of copying everything between the **cuts[i]** and **cuts[j]**, since we couldn’t avoid this copy of elements, and the cheapest cost to finish up every cut in between, which is composed of cost of cutting from **cuts[i]** to **cuts[p]**, and from **cuts[p]** to **cuts[j]**, denoting the point to make this cheapest cost as p.

Therefore, the minimum cost would be **cuts[j] – cuts[i]** (copy the whole segment), plus **min(cost(i, p), cost(p, j)), for any i<p<j** (the minimum cost to cut through an index between them).

**Functional definition**

3. Therefore, we should have the following recurrence function:

After expanding the cuts set with ,

**Space Complexity**

Therefore, we could store all the necessary intermediate values in a |cuts|×|cuts| table M, in which each element M[i, j] represents the minimum cost of cutting everything between them. To obtain the final result value, we could iteratively calculate each element in the table, and the final result (global minimum cost to cut everything between index 0 and index n), would be stored in the element M[0, n]🡪M[cuts[1], cuts[k+2]] (Under index-1 world).

Therefore, the algorithm is:

**StringBreak**(n, cuts[1…k]):

cuts = [0] + cuts + [n] // expanding cuts

num\_cuts = k + 2 // update length after cuts

// Create table of

// Initialize everything to 0

// Including base cases

M[num\_cuts][num\_cuts] = table(0)

// Iteratively fill up table

for length = 2 to (num\_cuts – 1):

for i = 1 to (num\_cuts – length):

j = i + length

M[i, j] = inf

for p = (i + 1) to (j – 1):

M[i, j] = min(M[i, j], M[i, p] + M[p, j])

M[i, j] = M[i, j] + (cuts[j] – cuts[i])

return M[1, num\_cuts]

Analysis:

When we fill out each element in the table, we have to loop through each inter-cut **p** from (i + 1) to (j – 1), and obtain the minimum cost, which costs O(k) in time. As we are filling out the (k+2)×(k+2) table, we totally cost O(k^3) time.

Correction:

