ECE374 Assignment 5

Due 03/20/2023

Group & netid

Chen Si *chensi3*

Jie Wang *jiew5*

Shitian Yang *sy39*

Problem 5

5. A certain string processing language allows the programmer to break a string into two pieces. It costs n units of time to break a string of n characters into two pieces, since this involves copying the old string. A programmer wants to break a string into many pieces, and the order in which the breaks are made can affect the total amount of time used. For example, suppose we wish to break a 20-character string after characters 3, 8, and 10. If the breaks are made in left-to-right order, then the first break costs 20 units of time, the second break costs 17 units of time, and the third break costs 12 units of time, for a total of 49 units. If the breaks are made in right-to-left order, the first break costs 20 units of time, the second break costs 10 units of time, and the third break costs 8 units of time, for a total of only 38 units.

   Give a dynamic programming algorithm that takes a list of character positions after which to break and determines the cheapest break cost in $O(n^3)$ time.

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 $cuts \leftarrow cuts + \{0, n\}$,

$$\forall i, j \in cuts, cost(cuts[i], cuts[j])$$

$$= \begin{cases} 0, j = i \text{ or } j = i + 1 \\ cuts[j] - cuts[i] + \min(cost(cuts[i[, cuts[p]), cost(cuts\big[p[, cuts[j])\big), & otherwise \end{cases}$$

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 (k + 2)^2 size
    // 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: