CS 161 (Stanford, Fall 25) Section 6 Extra

1 Graph Algorithms

The Floyd–Warshall algorithm runs in $O(n^3)$ time on graphs with n vertices and m edges, whether or not the input graph contains a negative cycle. Modify the Floyd–Warshall algorithm so that it will detect negative cycles and stop faster. Specifically, design an algorithm with the following behavior:

- If the input graph contains *no* negative cycle, the algorithm should compute all-pairs shortest paths in the usual $O(n^3)$ time.
- If the input graph *does* contain a negative cycle, the algorithm should detect this in only $O(mn)$ time.

Hint: Can we try creating a new vertex and some associated edges, then use a different algorithm that runs in time O(mn)?

Solution

Modify the input graph $G = (V, E)$ by adding a new source vertex s and a new zero-length edge from s to each vertex $v \in V$. The new graph G' has a negative cycle reachable from s if and only if G has a negative cycle. Run the Bellman–Ford algorithm on G' with source vertex s to check whether G contains a negative cycle. If not, run the Floyd–Warshall algorithm on G.

2 Rod Cutting

Suppose we have a rod of length k, where k is a positive integer. We would like to cut the rod into integer-length segments such that we maximize the *product* of the resulting segments' lengths. Multiple cuts may be made. For example, if $k = 8$, the maximum product is 18 from cutting the rod into three pieces of length $3, 3$, and 2. Write an algorithm to determine the maximum product for a rod of length k.

Solution

To solve this problem we are going to exploit the following overlapping sub-problems. If we let $f(k)$, be the maximum product possible for a rod of length k, then we have

$$f(k) = \max_{c \in \{2, k-1\}} (k,\ c \cdot f(k-c))$$

Another way to think of this is that we are going to try cutting the rod of length k into two rods of length c and $k - c$ and try all possible values of c, taking the one which

produces the maximum product. Note that not cutting the rod at all is another option which we can take. Also notice that we do not need to consider cutting off a length of 1 since that will never yield the optimal product, and also do not need to try cuts any larger than $\lfloor k/2 \rfloor$ since those will already have been explored due to the symmetry of the cutting. The running time for this algorithm is $O(k^2)$ since for each value of k we loop through $O(k)$ values to get the answer for that k.

```
def max_rod_cut(k):

    # max_prods[i] := largest product for
    # cutting rod of length i
    max_prods = [0 for _ in range(k + 1)]
    max_prods[1] = 1 # length 1 cannot be cut more

    for i in range(2, k + 1):

        best_prod = i # compare against not cutting at all
        for cut in range(2, i // 2 + 1):
            remaining = i - cut # the length remaining
            p = max_prods[cut] * max_prods[remaining]
            best_prod = max(best_prod, p)

        max_prods[i] = best_prod

    return max_prods[k]
```

3 Matrix Chain Multiplication

Consider a scenario in which we would like to multiply a lot of matrices, with matrix A_i's dimensions given as $p_{i-1} \times p_i$. The goal is to determine the most efficient order in which to multiply the matrices, so that the total number of scalar multiplications is minimized. Note that matrix multiplication is associative, so the result does not depend on how the matrices are parenthesized, but the *number of operations* does.

Note that if A is a $p \times q$ matrix and B is a $q \times r$ matrix, then their product AB is a $p \times r$ matrix. Computing this product requires

$$p \cdot q \cdot r$$

scalar multiplications, since each of the pr entries in the result is obtained by taking a dot product of length q.

Example. Suppose we have only the first three matrices A_1 (10×30), A_2 (30×5), and A_3

(5×60), so $p = [10, 30, 5, 60]$. There are two possible parenthesizations:

$$(A_1A_2)A_3 \quad \text{and} \quad A_1(A_2A_3).$$

For the first, the cost is:

$$(10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500.$$

For the second, the cost is:

$$(30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000.$$

Thus, the optimal order is $(A_1A_2)A_3$.

Give the recurrence relation for performing dynamic programming on this problem, and also give pseudocode.

Solution

Let $m[i, j]$ denote the minimum number of scalar multiplications required to compute the product $A_iA_{i+1} \cdots A_j$.
The dynamic programming recurrence is:

$$m[i, j] = \begin{cases} 0, & \text{if } i = j, \\ \min\limits_{i \le k < j} \left(m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j \right), & \text{if } i < j. \end{cases}$$

The algorithm builds up solutions for increasing chain lengths:

```
MATRIX-CHAIN-ORDER(p)
    n = length(p) - 1
    m = n x n 2D table
    for i = 1 to n
        m[i,i] = 0
    for l = 2 to n // chain length
        for i = 1 to n - l + 1
            j = i + l - 1
            m[i,j] = +inf
            for k = i to j - 1
                q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j]
                if q < m[i,j] then
                    m[i,j] = q
    return m
```