COMP550 Midterm1 Take Home Part

Siyang Jing

730042022

Definitions:

- $n \in \mathbb{N}^+$, the numer of potions;
- $m \in \mathbb{N}$, the total stench, m is initially 0;
- sequence $v[1, 2, ..., n]$, the (initial) stench factors, where $\forall i \in \{1, 2, ..., n\}, v[i] \in \mathbb{Z}_{50} (i.e. \{0, 1, ..., 49\})$;
- function $(,): \mathbb{Z}_{50} \times \mathbb{Z}_{50} \to \mathbb{Z}_{50}$, the action of pouring potion, defined as $(a, b) = a + b$, here $+$ is the addition in $\mathbb{Z}_{50}$;
- sequence of evaluation should be done sequentially, i.e. we can only pour potions from adjacent one, i.e. (a, b) should only be done if a and b are numbers (in other words expressions that have already been evaluated), e.g. $(1, (2, 3))$ should be done in the order of $(2, 3)$ and then $(1, 5)$;
- each evaluation (a, b) increments m by $a \times b$, here $\times$ is the regular multiplication, i.e. the effect of pouring potion on the total stench;
- sequence of function evaluations $F_\sigma(i, k)$ that sequentially evaluates $(,)$ on the elements of sequence $v[i, ..., k]$ in a certain order, i.e. the order of pouring potion, where $\sigma \in \mathfrak{F}(i, k)$ is some index denoting one possible way of evaluating $v[i, ..., k]$, and $\mathfrak{F}(i, k)$ is the index set for $F_\sigma(i, k)$, for example $F_\mu(1, 3) = (1, (2, 3))$, and $F_\nu(1, 3) = ((1, 2), 3)$, and etc;
- we notice that the the value obtained from the evaluation $F_\sigma(i, k)$ does not depend on σ, therefore we define $F(i, k) =$ value of $F_\sigma(i, k) = \sum_{l=i}^{l=k} v[l]$;

  $F(i, i) := v[i]$, which is a simple value, and no evaluation is needed;
- the minimum total stench:

  $\hat{m} = \min_{\sigma \in \mathfrak{F}(1,n)} \{m \text{ after sequence of evaluations } F_\sigma(1, n)\}$;
- the minimum total stench after pouring potions i to k:

  $\hat{m}(i, k) = \min_{\sigma \in \mathfrak{F}(i,k)} \{m \text{ created by } F_\sigma(i, k)\}$, note that $\hat{m}(1, n) = \hat{m}$;

  $\hat{m}(i, i) := 0$;
- the order of pouring potions i to k, that creates the minimum stench:

  $\hat{F}(i, k) = \underset{\sigma \in \mathfrak{F}(i,k)}{\text{argmin}} \{m \text{ created by } F_\sigma(i, k)\}$

Statement of the problem:

The aim is to develop an algorithm that,

given $v[1, ..., n]$, finds $\hat{m} = \min_{\sigma \in \mathfrak{F}(1,n)} \{m \text{ after sequence of evaluations } F_\sigma(1, n)\}$,

i.e. finds the order of pouring that creates the minimum total stench.

Statement of the algorithm:

First we notice that, there is a one-one correspondence between

$\sigma \in \mathfrak{F}(i,k)$ and $(j \geqslant i, \mu \in \mathfrak{F}(i,j), \nu \in \mathfrak{F}(j+1,k))$, and, in particular,

$\forall k > i+1, \sigma \in \mathfrak{F}(i,k), \exists! j \geqslant i, \mu \in \mathfrak{F}(i,j), \nu \in \mathfrak{F}(j+1,k), s.t. F_\sigma(i,k) = (F_\mu(i,j), F_\nu(j+1,k))$,

and therefore we have:

$\hat{m}(i,k) = \min_{\sigma \in \mathfrak{F}(i,k)} \{m \text{ created by } F_\sigma(i,k)\}$

$= \min_{j,\mu,\nu} \{m \text{ created by } (F_\mu(i,j), F_\nu(j+1,k))\}$

$= \min_{j,\mu,\nu} \{m \text{ created by } F_\mu(i,j) + m \text{ created by } F_\nu(j+1,k) + F(i,j) \times F(j+1,k)\}$

$= \min_j \{\min_{\mu,\nu} \{m \text{ created by } F_\mu(i,j) + m \text{ created by } F_\nu(j+1,k) + F(i,j) \times F(j+1,k)\}\}$

$= \min_j \{\min_\mu \{m \text{ created by } F_\mu(i,j)\} + \min_\nu \{m \text{ created by } F_\nu(j+1,k)\} + F(i,j) \times F(j+1,k)\}$

$= \min_j \{\hat{m}(i,j) + \hat{m}(j+1,k) + F(i,j) \times F(j+1,k)\}$

which is the optimal substructure of the problem.

To develop an algorithm from this optimal substructure, we either

- first recursively call the function to compute $\hat{m}(i,j)$ and $\hat{m}(j+1,k)$, and then iterate over j to find $\hat{m}(i,k)$, which describes a top-down recursive algorithm,
- or we can first compute all the $\hat{m}(i,i)$, then $\hat{m}(i,i+1)$, and then $\hat{m}(i,i+2)$, till $\hat{m}(1,n)$ to get the desired solution, which describes a bottom-up algorithm.

Either way, we need to memoize/store the values of $\hat{m}(i,j)$ that have been calculated to avoid repetitive calculation of the same value, and we also need calculate and store the values of $F(i,j)$ beforehand to avoid repetition.

The time cost for bottom-up solution is $\Theta(n^3)$, and the space cost is $\Theta(n^2)$, there is no worst case, best case, or average case, since in any situation, we calculate everything.

In the worst case, no matter what method is employed, we need to find $\hat{m}(i,k)$ for every i,k, and in that process, we need to iterate through all possible j values, which ends up with a $\Theta(n^3)$ time cost, and we also need to store all the values computed to avoid repetition, therefore $\Theta(n^2)$ is required. Therefore, our algorithm is the optimal one.

Bottom-up solution is sometimes better than recursive solution up to constant, since procedural calls are reduced.

Below is the pseudo code for the bottom-up algorithm:

The input v is a sequnce (1D array), which stores the sequence v described above.

The m 2D array stores $\hat{m}(i,j)$ described above in $m[i,j]$, and $m[i,i] = 0$.

The f 2D array stores the value of the function evaluation $F(i,j)$ at $f[i,j]$, and $f[i,i] = v[i]$.

The output is $m[1,n] = \hat{m}(1,n) = \hat{m}$, the minimum total stench.

Algorithm Potion-Adding-Order(v)

```
n = v.length
let m[1...n, 1...n] and f[1...n, 1...n] be new tables
for i = 1 to n
    m[i, i] = 0
    f[i, i] = v[i]
    for j = i + 1 to n
        f[i, j] = f[i, j − 1] + v[j] mod 50
    end
end
for l = 2 to n
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + f[i, k] × f[k + 1, j]
            if q < m[i, j]
                m[i, j] = q
            end
        end
    end
end
return m[1, n]
```