CS170–Spring 2019 — Homework 7 Solutions

Ran Liao, SID 3034504227

March 7, 2019

Collaborators:Jingyi Xu, Renee Pu

1 Study Group

Name	SID
Ran Liao	3034504227
Jingyi Xu	3032003885
Renee Pu	3032083302

2 Steel Beams

(a) (i) Denote the solution as (s_1, s_2, s_5) where s_i represents the number of i-foot-long steel beams. There're five possible situations for T.

$$\text{greedy solution} = \begin{cases} (0,0,k) & T \bmod 5 = 0 \\ (1,0,k) & T \bmod 5 = 1 \\ (0,1,k) & T \bmod 5 = 2 \\ (1,1,k) & T \bmod 5 = 3 \\ (0,2,k) & T \bmod 5 = 4 \end{cases}$$

k is some positive integer. Obviously, both of them are optimal solution.

(ii) Suppose the beam sizes are $1, 3, 4$, respectively. And the target T is 6. The optimal solution is $3 + 3$, whereas the greedy solution is $4 + 1 + 1$.

(b) (i) **Main Idea**

Define subproblem $L[i]$ representing the minimum number of beams that can be welded into a T-foot beam. And we have the following recursive relation. And let $T[0] = 0$.

$$L[i] = \min_{j=1}^{k}\{L[i - c_j] + 1\}$$

(ii) **Proof of Correctness**

The base case is obviously correct. Then, suppose we want to compute $L[i]$ and, $L[j]$ (where $j < i$) are optimal solutions. We use the minimum value of $L[j]$ for all possible j. This is actually greedy approach and thus guarantee the optimality of $L[i]$. Therefore, $L[T]$ will be optimal solution.

(iii) **Runtime Analysis**

T iterations is needed to solve T subproblems and each of them will cost $O(k)$ time. Therefore, the overall runtime is $O(kT)$ and $O(T)$ space is needed.

3 Egg Drop

(a) Drop the first egg from 14th story. If it doesn't break, drop it from $(14 + 13)$th story. If still it doesn't break, drop it from $(14 + 13 + 12)$th story. Repeat this process until the first egg breaks, and then use the second egg test remaining possible range one by one in increasing story order.

14 drops is needed for this strategy. Actually, it doesn't depend on when the first egg breaks,

(b) (i) **Main Idea**

Define subproblem $F[i][j]$ representing the number of drops needed for a i story building and have j eggs. The recursive relation is:

$$F[i][j] = \min_{x=1}^{i}\{\max\{F[i-x][j]+1, F[x-1][j-1]+1\}\}$$

(ii) **Proof of Correctness**

There're two possible outcomes when dropping egg from xth story. If it doesn't break, the solution is between $(x+1)$th story and ith story. It can be considered as a new building with $i-x$ stories and can be represented like this $F[i-x][j]+1$. If it breaks, the solution is between 1st story and $(x-1)$th story. So it can be represented by $F[x-1][j-1]+1$. The max operator will make sure we consider the worst case. The min operator will give the optimal strategy.

(iii) **Runtime Analysis**

There're nk subproblems and each of them will cost $O(n)$ time. Therefore, the overall runtime is $O(n^2k)$.

4 Non-Prefix Code

(a) **Main Idea**

Define subproblem as $N[i]$, which represents the number of ways we can interpret string $s[0 \cdots i]$. And the recursive relation is as follows. $N[0]$ is 0.

$$N[i] = \sum_{(k,v) \in d} N[i - \text{len}(v)] \, \text{Match}(v, i)$$

And

$$\text{Match}(v, i) = \begin{cases} 1 & \text{string } s[i - \text{len}(v) + 1 \cdots i] \text{ matches with } v \\ 0 & \text{Otherwise} \end{cases}$$

(b) **Proof of Correctness**

The basic case is trivial and correct. Then, suppose we want to compute $N[i]$, and $N[j]$ (where $j < i$) are all optimal solutions. If character v can be matched in $s[i - \text{len}(v) + 1 \cdots i]$, there're $N[i - \text{len}(v)]$ ways to interpret $s[1 \cdots i]$. Sum over all possible v will make $N[i]$ optimal.

(c) **Runtime Analysis**

This algorithm will solve n subproblems and each of them will check m possible values in d. Each check will cost at most $O(l)$ time. Therefore, the overall runtime is $O(nml)$.

5 Non-Prefix Code

(a) **Main Idea**

Define subproblem as $N[i]$, which represents the number of ways we can interpret string $s[0 \cdots i]$. And the recursive relation is as follows. $N[0]$ is 0.

$$N[i] = \sum_{(k,v) \in d} N[i - \text{len}(v)] \, \text{Match}(v, i)$$

And

$$\text{Match}(v, i) = \begin{cases} 1 & \text{string } s[i - \text{len}(v) + 1 \cdots i] \text{ matches with } v \\ 0 & \text{Otherwise} \end{cases}$$

(b) **Proof of Correctness**

The basic case is trivial and correct. Then, suppose we want to compute $N[i]$, and $N[j]$ (where $j < i$) are all optimal solutions. If character v can be matched in $s[i - \text{len}(v) + 1 \cdots i]$, there're $N[i - \text{len}(v)]$ ways to interpret $s[1 \cdots i]$. Sum over all possible v will make $N[i]$ optimal.

(c) **Runtime Analysis**

This algorithm will solve n subproblems and each of them will check m possible values in d. Each check will cost at most $O(l)$ time. Therefore, the overall runtime is $O(nml)$.

6 Breaking Chocolate

(a) **Subproblem**

Define subproblem as $P[a,b][c,d]$, which represents the minimum number of breaks needed to separate the raisins out for chocolate with upper left corner position as (a,b) and lower right corner position as (c,d).

(b) **Recurrence Relation**

$$P[a,b][c,d] = \min(\min_{k=a}^{c}\{P[a,b][k,d] + P[k,b][c,d] + 1\}, \min_{k=b}^{d}\{P[a,b][c,k] + P[a,k][c,d] + 1\})$$

(c) **Runtime Analysis**

There're m^2n^2 subproblems and each of them will cost $O(m+n)$ time to check each possibility. Therefore, the overall runtime is $O(m^2n^2(m+n))$

7 Propositional Parentheses

(a) (i) **Main Idea**

Define subproblem $T[i][j]$ representing the number of different ways there are to correctly parenthesize the formula from ith to jth position, so that it evaluates to true. And define subproblem $F[i][j]$ representing the number of different ways there are to correctly parenthesize the formula from ith to jth position, so that it evaluates to false.

Denote $S[i]$ represents the ith symbol in input. The recurrence relation is :

$$T[i][j] = \sum_{k=i+1}^{j-1} \begin{cases} 0 & S[k] = T \text{ or } F \\ (T[i][k] + F[i][k]) \cdot T[k][j] + T[i][k] \cdot (T[k][j] + F[k][j]) & S[k] = \vee \\ T[i][k] \cdot T[k][j] & S[k] = \wedge \end{cases}$$

$$F[i][j] = \sum_{k=i+1}^{j-1} \begin{cases} 0 & S[k] = T \text{ or } F \\ F[i][k] \cdot F[k][j] & S[k] = \vee \\ (T[i][k] + F[i][k]) \cdot F[k][j] + F[i][k] \cdot (T[k][j] + F[k][j]) & S[k] = \wedge \end{cases}$$

(ii) **Proof of Correctness**

If there're a possible ways to correctly parenthesize formula from i to k, and b possible ways to correctly parenthesize formula from k to j. There're $a \cdot b$ ways to correctly parenthesize formula from i to j. The remaining part in this formula is just some basic facts in logic operation.

(iii) **Runtime Analysis**

Suppose the length of input formula is n. There're n^2 subproblems, and each of them will cost $O(n)$ times. Therefore, the overall runtime is $O(n^3)$.

(b) The probability is $P = \frac{T[1][n]}{T[1][n]+F[1][n]}$.