Review Lesson: Design and Analysis of Computer Algorithms

Yanhui Li
Department of Computer Science and Technology, Nanjing University
yanhuili@nju.edu.cn

December 14, 2018

Problem 1

给定n个物品，其大小分别为$s_1, s_2, \ldots, s_n$。是否可以将n个物品划分成两个子集S_1和S_2使得两个子集中物品大小之和相等:

$$\sum_{i \in S_1} s_i = \sum_{i \in S_2} s_i$$

Solution

回忆0-1背包问题:

$$\begin{aligned} & max \sum_{i=1}^{n} x_i v_i \\ sat. & \sum_{i=1}^{n} x_i w_i \leq C \\ & x_i \in \{0,1\}, i \leq i \leq n \end{aligned}$$

将以上问题转化为0-1背包问题,

$$\begin{aligned} C &= \frac{1}{2} \sum_{i \in S} s_i \\ v_i, w_i &= s_i, i \leq i \leq n \end{aligned}$$

替代v_i, w_i, C得到:

$$max \sum_{i=1}^{n} x_i s_i$$
$$sat. \sum_{i=1}^{n} x_i s_i \leq \frac{1}{2} \sum_{i \in S} s_i$$
$$x_i \in \{0, 1\}, i \leq i \leq n$$

等价性:该问题存在划分使得可将S分成相同大小的子集S_1和S_2,当且仅当对应的0-1背包问题的最优解为

$$max \sum_{i=1}^{n} x_i s_i = \frac{1}{2} \sum_{i \in S} s_i$$

Problem 2

假设$A_n = \{a_1, a_2, \ldots, a_n\}$是一组不同的硬币面值，其中$a_i$是正整数，满足$a_1 < a_2 < \cdots < a_n$。给定一个正整数$C$，设计一个动态规划算法，从$A_n$里寻找最少个数的硬币，使其面值总和为$C$(假设每种面值硬币可用的个数不受限制)。对你设计的算法的复杂度进行分析。

Solution

假设$coin[i, j]$表示用前i种面值$(a_1, a_2, \ldots, a_i)$的硬币来凑总数j所需要的最少硬币数，可以列出动态规划方程：

$$coin[i, j] = \begin{cases} 0 & j = 0 \\ +\infty & i = 1 \text{ and } j < a_1 \\ min(coin[i-1, j], 1 + coin[i, j - a_i]) & other \end{cases} \quad (1)$$

原来的问题就是要求$coin[n, C]$。可以用填二维表的方法求解如下：

function INT COINCHANGE(int C, int n, int[] $coin$)
  int $denomination[\,] = [a_1, a_2, \ldots, a_n]$;
  for $i = 1; i \leq n; i++$ **do**
    $coin[i, 0] = 0$;
  end for
  for $i = 1; i \leq n; i++$ **do**
    for $j = 1; j \leq C; j++$ **do**
      if $i = 1 \&\& j < denomination[i]$ **then**

```
            coin[i, j] = +∞;
        end if
        if i > 1&& j < denomination[i] then
            coin[i, j] = cost[i − 1, j];;
        end if
        if j ≥ denomination[i] then
            coin[i, j] = min(coin[i−1, j], 1+coin[i, j−denomination[i]);
        end if
      end for
    end for
  end function
```

复杂度：从上面程序段可以看出，是一个二重循环，时间复杂度和空间复杂度均为$O(nC)$。

Problem 3

对于任何整数$c > 1$，$A_n = \{c^0, c^1, \ldots, c^{n-1}\}$是一组不同的硬币面值对所有正整数$C$，使用贪婪算法总求出组成$C$面值的最小硬币数。要求写出具体的算法，并对算法的最优性作出分析和说明。

Solution

Greedy算法：每次尽可能选面值最高的硬币，具体策略如下：

```
function int coinChange(int C, int n, int[] coin )
    int denomination[ ] = [1, c, c^2, . . . , c^(n−1)];
    int count = 0;
    int i = n − 1;
    while C > 0 do
        count = count + C/denomination[i];
        C = C mod denomination[i];
        i − −;
    end while
    return count
end function
```

证明：用以上贪心算法得到就是最优解。

令$x_0^*, x_1^*, \ldots, x_{n-1}^*$表示最优解中各面值的硬币个数,即最优解将$C$化为:

$$C = x_0 c^0 + x_1 c^1 + \cdots + x_{n-1} c^{n-1}$$

显然

$$x_i^* < c, 0 \le i \le n \tag{2}$$

否则某个$x_j^* > c$, 可以把c个面值为a_i的硬币换成1个面值为a_{i+1}的硬币，使得硬币的总数减少，与最优解矛盾。

令$x_1, x_2, \ldots, x_n$是用Greedy算法求得的一个解中各种面值的硬币的个数，下面证明:

$$x_{n-1} = x_{n-1}^*, x_{n-2} = x_{n-2}^*, \ldots, x_1 = x_1^*$$

根据Greedy特性，必然$x_{n-1} \geq x_{n-1}^*$，只要证明$x_{n-1} > x_{n-1}^*$不可能。反证法：如果$x_{n-1} > x_{n-1}^*$,则$x_{n-1} - x_{n-1}^* \geq 1$，即与Greedy算法获得解相比,最优解需要将1个面值为c^{n-1}的硬币化为其他硬币.但下式显然满足:

$$c^{n-1} > (c-1)c^{n-2} + (c-1)c^{n-3} + \cdots + (c-1)c^0$$

也就是，1个面值为c^{n-1}的硬币换成其他面值的硬币时，必然有至少一种面值的硬币个数大于等于c，与(2)式矛盾。因此$x_{n-1} = x_{n-1}^*$。同理可证

$$x_{n-2} = x_{n-2}^*, \ldots, x_0 = x_0^*$$

Problem 4

Given a set $A = \{s_1, s_2, \ldots, s_n\}$, where s_i (for $i = 1, 2, \ldots, n$) is a natural number, and a natural number S, determine whether there is a subset of A totaling exactly S. Design a dynamic programming algorithm for this problem.

Solution

a) using a two-dimension boolean table T, in which $T[i, j] = true$ if and only if there is a subset of the first i items $(a_1, a_2, \ldots, a_i)$ of A totaling exactly j.

$$T[i,j] = \begin{cases} true & j = 0 \\ true & i = 1, j = s_1 \\ false & i = 1, j > 0, j \neq s_1 \\ T[i-1, j] & i \geq 2, j < s_i \\ T[i-1, j] \bigvee T[i-1, j-s_i] & i \geq 2, j \geq s_i \end{cases}$$

b) prove the equivalence between 0-1 Knapsack and the above problem, and using 0-1 Knapsack algorithm to solve this problem. Let $C = S$, and for any i, $1 < i < n$, $w_i, v_i = s_i$.

$$max \sum_{i=1}^{n} x_i v_i$$

$$sat. \sum_{i=1}^{n} x_i w_i \leq C$$

$$x_i \in \{0, 1\}, i \leq i \leq n$$

We can find that the above 0-1 Knapsack Problem has a perfect solution with $\sum_{i=1}^{n} x_i v_i = C$, iff A has a subset totaling exact S.

Problem 5

Given two strings $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_m$, we wish to find the length of their longest common substring, that is, the largest k for which there are indices i and j with $x_i, x_{i+1}, \ldots, x_{i+k-1} = y_j, y_{j+1}, \ldots, y_{j+k-1}$. Show how to do this in time $O(mn)$.

Solution

Design one int array $L[n][m]$, in which $L[i][j]$ denotes the length of the longest common substring

$$x_l, x_{l+1}, \ldots, x_i = y_s, y_{s+1}, \ldots, y_j$$

with the end point $x_i = y_j$.

$$L[i, j] = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ 0 & x_i \neq y_j \\ 1 + L[i-1, j-1] & x_i = y_j \end{cases}$$

Return the max of $L[i][j]$ as the length of the largest common substring.

Problem 6

A subsequence is palindromic if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is not palindromic). Devise an algorithm that takes a sequence $x[1 \ldots n]$ and returns the (length of the) longest palindromic subsequence. Its running time should be $O(n^2)$.

Solution

a) Design one int array $L[n][m]$, in which $L[i][j]$ denotes the length of the longest palindromic subsequence, which is selected from $x[i], x[i+1], \ldots, x[j]$.

$$T[i,j] = \begin{cases} 0 & i > j \\ 1 & i = j \\ \max\{T[i,j-1], T[i+1,j]\} & i < j, x_i \neq x_j \\ 2 + T[i+1,j-1] & i < j, x_i = x_j \end{cases}$$

b) For $S = x_1, x_2, \ldots, n$, define its reverse sequence $re(S)$ as:

$$re(S) = x_n, x_{n-1}, \ldots, x_1$$

Compute the length of the longest common sub sequence of $S = x[1 \ldots n]$ and its reverse sequence $re(S)$, which equals to the length of the longest palindromic sub sequence.

Hint 1 *Every palindromic subsequence of S is also a palindromic subsequence of $re(S)$.*

Hint 2 *The longest palindromic sub sequence of S is a common sub sequence of S and $re(S)$.*

Hint 3 *The length of the longest common subsequence of S and $re(S)$ is not less than the length of the longest palindromic subsequence of S.*

Now we will prove that the two lengths are the same. Assume the longest sub sequence is

$$s = x_{v_1}, x_{v_2}, \ldots, x_{v_l},$$

where $v_i < v_{i+1}$ for any i with $1 \leq i < l-1$. As s is also a sub sequence of $re(S)$, $re(S)$ has the corresponding sub sequence s':

$$s' = x_{w_1}, x_{w_2}, \ldots, x_{w_l} = s = x_{v_1}, x_{v_2}, \ldots, x_{v_l}$$

where for any i with $1 \leq i < l$, $w_i > w_{i+1}$. Here we get two sequences of integers from $\{1, \ldots, n\}$ in increasing order:

$$v_1, v_2, \ldots, v_{l-1}, v_l$$

$$w_l, w_{l-1}, \ldots, w_2, w_1$$

and these integers satisfy that for any i, $1 \leq i \leq l$, $x_{v_i} = x_{w_i}$. We compare the middle number $v_{l+1/2}, w_{l+1/2}$ of these two sequences:

- $v_{(l+1)/2} < w_{(l+1)/2}$. Now we focus on a new integer sequence constructed from the above two:

  $$v_1 < \cdots < v_{(l+1)/2-1} < v_{(l+1)/2} < w_{(l+1)/2} < w_{(l+1)/2-1} < \cdots < w_1$$

  And its corresponding character sequence

  $$s^* = x_{v_1}, x_{v_2}, \ldots, x_{v_{(l+1)/2}}, x_{w_{(l+1)/2}}, \ldots, x_{w_2}, x_{w_1}$$

  is a palindromic subsequence of S with length

  $$\lfloor \frac{l+1}{2} \rfloor \times 2 \geq l$$

  We get "=" iff l=even. So s^* is the longest palindromic subsequence with the length l.

- $v_{(l+1)/2} = w_{(l+1)/2}$. Now we focus on a new integer sequence constructed from the above two:

  $$w_l < \cdots < w_{(l+1)/2-1} < w_{(l+1)/2} = v_{(l+1)/2} < v_{(l+1)/2+1} < \cdots < v_l$$

  And its corresponding character sequence

  $$s^* = x_{w_l}, x_{w_{l-1}}, \ldots, x_{w_{(l+1)/2}}, x_{v_{(l+1)/2+1}}, \ldots, x_{v_{l-1}}, x_{v_l}$$

  is a palindromic subsequence of S with length

  $$(l - \lfloor \frac{l+1}{2} \rfloor) \times 2 + 1 \geq l$$

  We get "=" iff l=odd. So s^* is the longest palindromic subsequence with the length l.

- $v_{(l+1)/2} > w_{(l+1)/2}$. Now we focus on a new integer sequence constructed from the above two:

  $$w_l < \cdots < w_{(l+1)/2-1} < w_{(l+1)/2} < v_{(l+1)/2} < v_{(l+1)/2+1} < \cdots < v_l$$

  And its corresponding character sequence

  $$s^* = x_{w_l}, x_{w_{l-1}}, \ldots, x_{w_{(l+1)/2}}, x_{v_{(l+1)/2}}, x_{v_{(l+1)/2+1}}, \ldots, x_{v_{l-1}}, x_{v_l}$$

  is a palindromic subsequence of S with length

  $$(l - \lfloor \frac{l+1}{2} \rfloor) \times 2 + 2 > l$$

  A contradiction occurs when we can construct a palindromic subsequence with length larger than l.

The above proof has also showed that:

- if l is odd, the middle integers $v_{(l+1)/2}$ and $w_{(l+1)/2}$ of two sequences are the same.
- if l is even, $v_{(l+1)/2} < w_{(l+1)/2}$ holds.
- for any longest common sub sequence $s = x_{v_1}, x_{v_2}, \ldots, x_{v_l}$, we could construct a palindromic subsequence s^* with the same length l from it.
  - $l = 2k$, $s^* = x_{v_1}, x_{v_2}, \ldots, x_{v_{k-1}}, x_{v_k}, x_{v_k}, x_{v_{k-1}}, \ldots, x_{v_2}, x_{v_1}$
  - $l = 2k + 1$, $s^* = x_{v_1}, x_{v_2}, \ldots, x_{v_k}, x_{v_k+1}, x_{v_k}, \ldots, x_{v_2}, x_{v_1}$

Problem 7

Consider the following "3-PARTITION" problem. Given integers $a_1, \ldots, a_n$, we want to determine whether it is possible to partition $a_1, \ldots, a_n$ into three disjoint subsets I, J, K such that:

$$\sum_{a_i \in I} a_i = \sum_{a_i \in J} a_i = \sum_{a_i \in K} a_i = \frac{1}{3} \sum_{i=1}^{n}$$

For example, for input $(1, 2, 3, 4, 4, 5, 8)$ the answer is yes, because there is the partition $(1, 8), (4, 5), (2, 3, 4)$. On the other hand, for input $(2, 2, 3, 5)$ the answer is no. Devise and analyze a dynamic programming algorithm for "3-PARTITION" that runs in time polynomial in n and $\sum a_i$

Solution

Focus on the relationship between two problems "0-1 Knapsack" and "3-PARTITION". We extend "0-1 Knapsack" with considering two knapsacks, We can easily extends the solution of "0-1 Knapsack" to solve "3-PARTITION".

Let $C = \frac{1}{3} \sum_{i=1}^{n} a_i$. Design a boolean array $T[n][C][C]$, in which $T[i][l][k]$ denotes whether it is possible to partition $a_1, \ldots, a_i$ into three disjoint subsets I, J, K such that:

$$\sum_{a_i \in I} a_i = l, \sum_{a_i \in J} a_i = k$$

Now we give the recursive equation of $T[i][l][k]$:

$$T[i,l,k] = \begin{cases} false & l < 0 \\ false & k < 0 \\ true & l,k = 0 \\ false & i = 0, l > 0 \\ false & i = 0, k > 0 \\ T[i-1][l][k] \bigvee T[i-1][l-a_i][k] \bigvee T[i-1][l][k-a_i] & \text{others} \end{cases}$$

Obviously, there is a "3-PARTITION" of $a_1, \ldots, a_n$ iff $T[n][C][C] = true$.

Problem 8

- Given an unlimited supply of coins of denominations $x_1, x_2, \ldots, x_n$, we wish to make change for a value v; that is, we wish to find a set of coins whose total value is v. This might not be possible: for instance, if the denominations are 5 and 10 then we can make change for 15 but not for 12. Give an $O(nv)$ dynamic-programming algorithm for the problem.
- We are restricted to use each denomination at most once. Give an $O(nv)$ dynamic-programming algorithm for the problem.
- We wish to make change for a value v using **at most** k **coins**. Give an $O(nv)$ dynamic-programming algorithm for the problem.

Solution

Compare this Problem with Problem 3.

Problem 9

Sequence alignment. When a new gene is discovered, a standard approach to understanding its function is to look through a database of known genes and find close matches. The closeness of two genes is measured by the extent to which they are aligned. To formalize this, think of a gene as being a long string over an alphabet $\Sigma = A, C, G, T$. Consider two genes (strings) $x = ATGCC$ and $y = TACGCA$. An alignment of x and y is a way of matching up these two strings by writing them in columns, for instance:

$$\begin{array}{cccccccc} - & A & T & - & G & C & C \\ T & A & - & C & G & C & A \end{array}$$

Here the "−" indicates a "gap". The characters of each string must appear in order, and each column must contain a character from at least one of the strings. The score of an alignment is specified by a scoring matrix θ of size $(|\Sigma|+1)(|\Sigma|+1)$, where the extra row and column are to accommodate gaps. For instance the preceding alignment has the following score:

$$\theta(-,T)+\theta(A,A)+\theta(T,-)+\theta(-,C)+\theta(G,G)+\theta(C,C)+\theta(C,A).$$

Give a dynamic programming algorithm that takes as input two strings $x[1\ldots n]$ and $y[1\ldots m]$ and a scoring matrix θ, and returns the highest-scoring alignment. The running time should be $O(mn)$.

Solution

Design one int array $L[n][m]$, in which $L[i][j]$ denotes the the highest-scoring of all alignment for two sub-strings $x[1,\ldots,i]$ and $y[1,\ldots,j]$

$$T[i,j]=\begin{cases}0 & i=0,j=0\\ \sum_{s=1}^{j}\theta(-,y[s]) & i=0,j>0\\ \sum_{s=1}^{i}\theta(x[s],-) & i>0,j=0\\ \max\{T[i,j-1]+\theta(-,y[j]), & i>0,j>0\\ \quad T[i-1,j]+\theta(x(i),-), & \\ \quad T[i-1,j-1]+\theta(x[i],y[j])\} & \end{cases}$$

Problem 10

Give an $O(n)$-time dynamic-programming algorithm to compute the n-th Fibonacci number.

Solution

```
function INT FIBONACCI(int n)
    int[ ] Fib= new int[n + 1];
    Fib[0] = 1;
    Fib[1] = 1;
    for int i = 2; i + +; i ≤ n do
        Fib[i] = Fib[i − 1] + Fib[i − 2];
    end for
    return Fib[n];
end function
```

Problem 11

Given a directed acyclic graph $G = (V, E)$ with real valued edge weights and two distinguished vertices s and e. Design a dynamic programming algorithm to find a longest weighted path from s to e.

Solution

DFS+DP: $L(s) = \max\{L(w)+Weight(s,w)|(s,w) \in E\}$, where $L(w)$ means the length of longest path from w to e. Do DFS to determine the value $L(w)$ of every node w in G by compute the value of its children nodes.

Problem 12

Consider the following game. A "dealer" produces a sequence $S = s_1 \dots s_n$ of "cards," face up, where each card s_i has a value v_i. Then two players take turns picking a card from the sequence, but can only pick the first or the last card of the (remaining) sequence. The goal is to collect cards of largest total value. (For example, you can think of the cards as bills of different denominations.) Assume n is even.

(a) Show a sequence of cards such that it is not optimal for the first player to start by picking up the available card of larger value. That is, the natural greedy strategy is suboptimal.

(b) Give an $O(n^2)$ algorithm to compute an optimal strategy for the first player. Given the initial sequence, your algorithm should precompute in $O(n^2)$ time some information, and then the first player should be able to make each move optimally in $O(1)$ time by looking up the precomputed information.

Solution

a) Assume $S = 3, 7, 2, 1$. Assume we apply the greedy strategy: in the first round, the first player picks up 3, and the second players picks up 7; and in the second round, the first player picks up 2 and the second player picks up 1. The total value of the first player's cards is $3 + 1 = 4$. The optimal strategy for the first plays is picking up 1 in the first round, and 7 in the second round. The optimal value is $1 + 7 = 8$. That means the natural greedy strategy is suboptimal.

b) Design one int array $L[n][n]$, in which $L[i][j]$ denotes the largest total value for the first player choosing cards from $s_i, \dots, s_j$.

$$L[i,j] = \begin{cases} 0 & i >= j \\ 0 & j - i \text{ is even} \\ \max\{s_i, s_j\} & j - i = 1 \\ \max\{s_i + \min\{L[i+1][j-1], L[i+2][j]\} & \text{other} \\ \qquad s_j + \min\{L[i][j-2], L[i+1][j-1]\}\ \} & \end{cases}$$

Problem 13

Rod Cutting Problem. Consider a rod of length n units, made out of relatively valuable metal. The rod needs to be cut into a number of pieces to be sold separately. Selling a piece of rod of length i units earns $P[i]$ dollars. The Rod Cutting problem is formulated as follows: given n and the $P[i]$ table for $i = 1 \ldots n$, find the set of cuts of the rod, i.e., the set/sequence of lengths $L = (l_1, \ldots, l_k)$, such that:

$$\sum_{j=1}^{k} l_j = n$$

$$\sum_{j=1}^{k} P(l_j) = \max\{\sum_{j=1}^{s} P(l_j^*) | \sum_{j=1}^{s} l_j^* = n\}$$

i.e., find the set of rod cuts that lead to the largest generated amount of money.

Solution

$Cut[n] = \max\{Cut[n-i] + P(i) | 1 \leq i \leq n\}$. We can compute $Cut[i$ from 1 to n:

- $Cut[0] = 0$
- $Cut[1] = P(1)$
- $Cut[2] = \max\{P(1) + P(1), P(2)\}$
- $\cdots$

Problem 14

A mission-critical production system has n stages that have to be performed sequentially; stage i is performed by machine M_i. Each machine M_i has a probability r_i of functioning reliably and a probability $1 - r_i$ of failing (and the failures are independent). Therefore, if we implement each stage with a single machine, the probability that the whole system works is

$$r_1 \times r_2 \times \cdots \times r_n$$

To improve this probability we add redundancy, by having m_i copies of the machine M_i that performs stage i. The probability that all m_i copies fail simultaneously is only $(1-r_i)^{m_i}$, so the probability that stage i is completed correctly is $1-(1-r_i)^{m_i}$ and the probability that the whole system works is

$$\prod_{i=1}^{n}(1-(1-r_i)^{m_i})$$

Each machine M_i has a cost c_i, and there is a total budget B to buy machines. (Assume that B and c_i are positive integers.) Given the probabilities $r_1, \ldots, r_n$, the costs $c_1, \ldots, c_n$, and the budget B, find the redundancies $m_1, \ldots, m_n$ that are within the available budget and that maximize the probability that the system works correctly.

Solution

Since $m_i \geq 1$, let $B = B - \sum_{i=1}^{n} c_i$. Design one array $L[n][B]$, in which $L[i][j]$ denotes the max probability that the whole system works complete correctly from stage 1 to stage i within the budget $j + \sum_{i=1}^{n} c_i$.

$$L[i][j] = \begin{cases} f(1,j) & i = 1 \\ \max_{s \leq j}\{L[i-1][j-s] \cdot f(i,s)\} & i > 1 \end{cases}$$

where $f(i,j)$ is defined as:

$$m = \lfloor \frac{j}{c_i} \rfloor$$

$$f(i,j) = (1-(1-r_i)^{m+1})$$

Problem 15

Counting heads. Given integers n and k, along with $p_1, \ldots, p_n \in [0, 1]$, you want to determine the probability of obtaining exactly k heads when n biased coins are tossed independently at random, where p_i is the probability that the i-th coin comes up heads. Give an $O(n^2)$ algorithm for this task. Assume you can multiply and add two numbers in[0,1] in $O(1)$ time.

Solution

Design a table $P[n][k]$, in which $P[i][j]$ shows the probability of obtaining exactly j heads from the first coin to the i-th coin.

$$P[i][j] = \begin{cases} 0 & i < j \\ \prod_{1 \le s \le i} p_s & i = j \\ p_i P[i-1][j-1] + (1-p_i)P[i-1][j] & i > j \end{cases}$$

Problem 16

Let us define a multiplication operation on three symbols a, b, c according to the following table; thus $ab = b$, $ba = c$, and so on. Notice that the multiplication operation defined by the table is neither associative nor commutative.

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

Find an efficient algorithm that examines a string $S = s_1, \ldots, s_n$ of these symbols $s_i \in \{a, b, c\}$, and decides whether or not it is possible to parenthesize the string in such a way that the value of the resulting expression is a. For example, on input *bbbbac* your algorithm should return yes because $((b(bb))(ba))c = a$.

Solution

Design a table $T[n][n]$, in which $T[i][j]$ is a set including all values of the resulting expressions of $s_i, s_{i+1}, \ldots, s_j$ with different parenthesized methods. For example, *abc* can be parenthesized as

$$(ab)c = bc = a \text{ or } a(bc) = aa = b$$

$$T[i][j] = \begin{cases} \emptyset & i > j \\ \{s_i\} & i = j \\ \cup_{i \le k < j} T[i][k] \otimes T[k+1][j] & i < j \end{cases}$$

where for two subsets A and B of $\{a, b, c\}$, $A \otimes B$ is defined as:

$$A \otimes B = \{xy | x \in A, y \in B\}$$