Chapter 15 Solusion

frc6.com

i@frc6.com

https://github.com/frc123/CLRS-code-solution

11/3/2021

15.1

15.1-1

Proof. We prove by substitution method. For $n = 0$, $T(0) = 2^0 = 1$. For $n > 0$,

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) = 1 + \sum_{j=0}^{n-1} 2^j = 1 + (2^n - 1) = 2^n \qquad \square$$

15.1-2

Consider the following case:

length i	1	2	3
price p_i	1	6	8
density p_i/i	1	3	2.67

If we use "greedy" strategy, our solution will be "2 1", and the total price will be 7. However, the optimal way is "3", and the total price is 8.

15.1-3

```
/**
* running time: O(n^2)
* p: table of prices (index start from 0)
* n: length of rod
* c: cost of each cut
* return maximum revenue
*/
int BottomUpCutRodWithCost(const std::vector<int>& p, int n, int c)
{
    int *r, q, i, j;
    r = new int[n + 1];
    r[0] = 0;
```

```
        for (j = 1; j <= n; ++j)
        {
            q = p[j - 1];
            for (i = 0; i < j - 1; ++i)
                q = std::max(q, p[i] + r[j - i - 1] - c);
            r[j] = q;
        }
        delete[] r;
        return q;
    }
```

15.1-4

```
/**
* p: table of prices (index start from 0)
* n: length of rod
* r: table of maximum revenue (index start from 1)
* s: table of optimal size i of the first piece to cut off (index start from 1)
* return maximum revenue
*/
int ExtendedMemoizedCutRodAux(const std::vector<int>& p, int n, int *r, int *s)
{
    int q, i, reminder_r;
    if (r[n] >= 0) return r[n];
    q = INT_MIN;
    for (i = 0; i < n; ++i)
    {
        reminder_r = ExtendedMemoizedCutRodAux(p, n - i - 1, r, s);
        if (q < p[i] + reminder_r)
        {
            q = p[i] + reminder_r;
            s[n] = i + 1;
        }
    }
    r[n] = q;
    return q;
}

/**
* running time: O(n^2)
* p: table of prices (index start from 0)
* n: length of rod
* return (r, s)
* r: table of maximum revenue (index start from 1)
* s: table of optimal size i of the first piece to cut off (index start from 1)
* caller is responsible to deallocate return value r and s
*/
std::pair<int*, int*> ExtendedMemoizedCutRod(const std::vector<int>& p, int n)
{
    int *r, *s, i;
    r = new int[n + 1];
    s = new int[n + 1];
    r[0] = 0;
    s[0] = 0;
    for (i = 1; i <= n; ++i) r[i] = INT_MIN;
    ExtendedMemoizedCutRodAux(p, n, r, s);
    return std::make_pair(r, s);
}
```

15.1-5

```
/**
* running time: O(n)
* n: n-th fibonacci number (must greater than 0)
*/
int FibonacciNumber(int n)
{
    int *f, i, result;
    f = new int[n + 1];
    f[0] = 0;
    f[1] = 1;
    for (i = 2; i <= n; ++i)
        f[i] = f[i - 1] + f[i - 2];
    result = f[n];
    delete[] f;
    return result;
}
```

15.2

15.2-1

Optimal parenthesization: $((1, 2), ((3, 4), (5, 6)))$

Minimum cost: 2010

15.2-2

```
/**
* s: table (2d) storing index of k achieved the optimal cost
*       (index start by 1)
* caller is responisble to deallocate the return value
*/
Matrix* MatrixChainMultiply
    (const std::vector<Matrix*>& matrices, const Table* s, int i, int j)
{
    Matrix *matrix_a, *matrix_b, *matrix_c;
    if (i == j)
    {
        return matrices[i - 1];
    }
    matrix_a = MatrixChainMultiply(matrices, s, i, (*s)[i][j]);
    matrix_b = MatrixChainMultiply(matrices, s, (*s)[i][j] + 1, j);
    matrix_c = MatrixMultiply(matrix_a, matrix_b);
    if (i != (*s)[i][j]) delete matrix_a;
    if ((*s)[i][j] + 1 != j) delete matrix_b;
    return matrix_c;
}
```

15.2-3

Proof. We prove by substitution method. For $n = 1$, $P(1) = 1 \geq 2^k$ for $k \leq 0$. For $n \geq 2$,

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k) \geq \sum_{k=1}^{n-1} (c \cdot 2^k)(c \cdot 2^{n-k}) = \sum_{k=1}^{n-1} (c^2 \cdot 2^n) = (n-1)(c^2 \cdot 2^n) \geq c^2 \cdot 2^n$$

for some constant c. □

15.2-4

For all vertices $v_{i,j}$ in the graph, it contains edge $(v_{i,j}, v_{i,k})$ and $(v_{i,j}, v_{k+1,j})$ for all $i \leq k < j$.

Vertices:

$$\binom{n}{2} + n = \frac{n(n-1)}{2} + n = \frac{n(n+1)}{2}$$

Edges:

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(j-i) = \sum_{i=1}^{n}(\sum_{j=i}^{n}(j) - \sum_{j=i}^{n}(i)) = \sum_{i=1}^{n}(\sum_{j=1}^{n}(j) - \sum_{j=1}^{i-1}(j) - (n-i+1)i)$$
$$= \sum_{i=1}^{n}(\tfrac{n(n+1)}{2} - \tfrac{(i-1)i}{2} - (n-i+1)i) = \tfrac{n^2(n+1)}{2} - \sum_{i=1}^{n}(\tfrac{i-i^2+2ni}{2})$$
$$= \tfrac{n^2(n+1)}{2} + \tfrac{1}{2}\sum_{i=1}^{n}(i^2) - \tfrac{1}{2}(1+2n)\sum_{i=1}^{n}(i) = \tfrac{n^2(n+1)}{2} + \tfrac{n(n+1)(2n+1)}{12} - \tfrac{n(n+1)(2n+1)}{4}$$
$$= \tfrac{n^2(n+1)}{2} - \tfrac{n(n+1)(2n+1)}{6} = \tfrac{(n-1)n(n+1)}{6}$$

15.2-5

Proof. Notice that $\sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j)$ is equal to the total times of any entries are referenced during the entire call of MATRIX-CHAIN-ORDER. In other words, it is equal to twice the times of line 10 was executed during the entire call.
Hence, we have

$$\sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j) = \sum_{l=2}^{n}\sum_{i=1}^{n-l+1}\sum_{k=i}^{i+l-2} 2 = 2\sum_{l=2}^{n}(n-l+1)(l-1) = 2((n+2)\sum_{l=2}^{n} l - \sum_{l=2}^{n} l^2 - (n-1)(n+1))$$
$$= 2((n+2)(\tfrac{n(n+1)}{2} - 1) - (\tfrac{n(n+1)(2n+1)}{6} - 1)) = \tfrac{n^3-n}{3}$$

□

15.2-6

Proof. We prove by induction. Let $P(n)$ be the claim: A full parenthesization of an n-element expression has exactly $n-1$ pairs of parentheses.
(*Base Case*) A 2-element full parenthesization (A_1, A_2) has only one pair of parentheses clearly. Hence, we have proved $P(2)$ is true.
(*Induction Step*) Suppose that $P(n)$ is true. Let C be a sequence with $n+1$ elements: $A_1A_2...A_nA_{n+1}$. Delete one arbitrary element from C, we have a sequence with n elements. By induction hypothesis, C (n-element) has exactly $n-1$ pairs of parentheses now. Add the deleted element back, we can add one pair of parentheses to surround the deleted element and one of the element's neighbor element or one of the element's neighbor parenthesization. This says, C ($n+1$-element) has exactly n pairs of parentheses now. We have proved $P(n+1)$ is true. □