Introduction to Algorithms HW4

109511207 蔡宗儒

Pseudo Code

```
MEMOIZED-CUT-ROD(p,n)
1.  let r[0…n] be a new array
2.  for i=0 to n
3.      r[i] = -∞
4.  return MEMOIZED-CUT-ROD-AUX(p,n,r)

MEMOIZED-CUT-ROD_AUX(p,n,r)
1.  if r[n] >= 0
2.      return r[n]
3.  if n==0
4.      q = 0
5.  else q = -∞
6.      for i = 1 to n
7.          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p,n-i,r))
8.  r[n] = q
9.  return q

EXTENDED-BOTTOM-UP-CUT-ROD(p,n)
1.  let r[0…n] and s[0…n] be new arrays
2.  r[0] = 0
3.  for j = 1 to n
4.      q = -∞
5.      for i = 1 to j
6.        if q < p[i] + r[j-i]
7.          q = p[i] + r[j-i]
8.          s[j] = i
9.      r[j] = q
10. return r and s

PRINT-CUT-ROD-SOLUTION(p,n)
1.  (r,s) = EXTENDED-BOTTOM-UP-CUT-ROD(p,n)
2.  while n > 0
3.      print s[n]
4.      n = n – s[n]
```

Introduction of Code

```
void top_down(const int *p, int n)
{
    cout << "\nTop Down:\n";
    cout << "total length: " << n << '\n';
    max_top_down_cut_rod(p,n);
    cout << '\n';
    min_top_down_cut_rod(p,n);
}
```

top_down 做為一個 interace 來分別執行 max_top_down_cut_rod 和 min_top_down_cut_rod 得出最高與最低的價格與其解

```
void max_top_down_cut_rod(const int *p, int n)
{
    int *r = new int[n+1];
    int *s = new int[n+1];
    for(int i=0;i<=n;i++)
        r[i] = -1;
    cout << "maximum price: " << max_top_down_cut_rod_aux(p,n,r,s) << '\n';
    int count = 0;
    while(n > 0)
    {
        cout << s[n] << ' ';
        n = n - s[n];
        count++;
    }
    cout << '\n';
    cout << "number of pieces: " << count << '\n';
}
```

max_top_down_cut_rod 參考了課本的 pseudo code。另外將 PRINT-CUT-ROD-SOLUTION 結合進了 function，在 function 內得出其解並輸出結果。

```
int max_top_down_cut_rod_aux(const int *p, int n, int *r, int *s)
{
    int q;
    if(r[n] >= 0)
        return r[n];
    if(n == 0)
        q = 0;
    else
    {
        q = -1;
        int size = (n > 10) ? 10 : n;
        for(int i=1;i<=size;i++)
        {
            if(q <= (p[i] + max_top_down_cut_rod_aux(p,n-i,r,s)))
            {
                q = p[i] + max_top_down_cut_rod_aux(p,n-i,r,s);
                s[n] = i;
            }
        }
    }
    r[n] = q;
    return q;
}
```

max_top_down_cut_rod_aux 參考了課本的 pseudo code，並結合了 EXTENDED-BOTTOM-UP-CUT-ROD 存解的方式，用 s[0…n]記錄過程。另外因為 price 僅有長度 1~10 的，所以可能會發生 overflow，因此我也用了一個變數 size 來讓 n > 10 時，切的最大長度被限制於 10。

```
void min_top_down_cut_rod(const int *p, int n)
{
    int *r = new int[n+1];
    int *s = new int[n+1];
    for(int i=0;i<=n;i++)
        r[i] = -1;
    cout << "minimum price: " << min_top_down_cut_rod_aux(p,n,r,s) << '\n';
    int count = 0;
    while(n > 0)
    {
        cout << s[n] << ' ';
        n = n - s[n];
        count++;
    }
    cout << '\n';
    cout << "number of pieces: " << count << '\n';
}
```

min_top_down_cut_rod 參考了課本的 pseudo code。一樣將 PRINT-CUT-ROD-SOLUTION 結合進了 function，在 function 內得出其解並輸出結果。

```
int min_top_down_cut_rod_aux(const int *p, int n, int *r, int *s)
{
    int q;
    if(r[n] >= 0)
        return r[n];
    if(n == 0)
        q = 0;
    else
    {
        q = INT_MAX;
        int size = (n > 10) ? 10 : n;
        for(int i=1;i<=size;i++)
        {
            if(q >= (p[i] + min_top_down_cut_rod_aux(p,n-i,r,s)))
            {
                q = p[i] + min_top_down_cut_rod_aux(p,n-i,r,s);
                s[n] = i;
            }
        }
    }
    r[n] = q;
    return q;
}
```

min_top_down_cut_rod_aux 參考了課本的 pseudo code，因為要求的東西變成最低價格，所以與 max_top_down_cut_rod_aux 不同的是，q = min(q, p[i] + min_top_down_cut_rod_aux(p,n-i,r,s))，且要將 q 的初值改為 INT_MAX。

```
void bot_up(const int *p, int n)
{
    cout << "\nBottom Up:\n";
    cout << "total length: " << n << '\n';
    max_bot_up_cut_rod(p,n);
    cout << '\n';
    min_bot_up_cut_rod(p,n);
}
```

bot_up 做為一個 interace 來分別執行 max_bot_up_cut_rod 和 min_bot_up_cut_rod 得出最高與最低的價格與其解

```
void max_bot_up_cut_rod(const int *p, int n)
{
    int *r = new int[n+1];
    int *s = new int[n+1];
    r[0] = 0;
    int q;
    for(int j=1;j<=n;j++)
    {
        q = -1;
        int size = (j > 10) ? 10 : j;
        for(int i=1;i<=size;i++)
        {
            if(q <= p[i] + r[j-i])
            {
                q = p[i] + r[j-i];
                s[j] = i;
            }
        }
        r[j] = q;
    }
    cout << "maximum price: " << r[n] << '\n';
    int count = 0;
    while(n > 0)
    {
        cout << s[n] << ' ';
        n = n - s[n];
        count++;
    }
    cout << '\n';
    cout << "number of pieces: " << count << '\n';
}
```

max_bot_up_cut_rod 參考了課本的 pseudo code。另外一樣將 PRINT-CUT-ROD-SOLUTION 結合進了 function，在 function 內得出其解並輸出結果。且因為 price 僅有長度 1~10 的，所以可能會發生 overflow，因此也用了一個變數 size 來讓 n > 10 時，切的最大長度被限制於 10。

```
void min_bot_up_cut_rod(const int *p, int n)
{
    int *r = new int[n+1];
    int *s = new int[n+1];
    r[0] = 0;
    int q;
    for(int j=1;j<=n;j++)
    {
        q = INT_MAX;
        int size = (j > 10) ? 10 : j;
        for(int i=1;i<=size;i++)
        {
            if(q >= p[i] + r[j-i])
            {
                q = p[i] + r[j-i];
                s[j] = i;
            }
        }
        r[j] = q;
    }
    cout << "minimum price: " << r[n] << '\n';
    int count = 0;
    while(n > 0)
    {
        cout << s[n] << ' ';
        n = n - s[n];
        count++;
    }
    cout << '\n';
    cout << "number of pieces: " << count << '\n';
}
```

min_bot_up_cut_rod_aux 參考了課本的 pseudo code，一樣因為要求的東西變成最低價格，所以與 max_ bot_up _cut_rod_aux 不同的是，q = min(q, p[i] + min_ bot_up _cut_rod_aux(p,n-i,r,s))，且要將 q 的初值改為 INT_MAX。

執行結果(for the table in HW5.pdf)

```
Top Down:
total length: 1
maximum price: 1
1
number of pieces: 1

minimum price: 1
1
number of pieces: 1

Bottom Up:
total length: 1
maximum price: 1
1
number of pieces: 1

minimum price: 1
1
number of pieces: 1
```

```
Top Down:
total length: 2
maximum price: 5
2
number of pieces: 1

minimum price: 2
1 1
number of pieces: 2

Bottom Up:
total length: 2
maximum price: 5
2
number of pieces: 1

minimum price: 2
1 1
number of pieces: 2
```

```
Top Down:
total length: 3
maximum price: 8
3
number of pieces: 1

minimum price: 3
1 1 1
number of pieces: 3

Bottom Up:
total length: 3
maximum price: 8
3
number of pieces: 1

minimum price: 3
1 1 1
number of pieces: 3
```

```
Top Down:
total length: 4
maximum price: 10
2 2
number of pieces: 2

minimum price: 4
1 1 1 1
number of pieces: 4

Bottom Up:
total length: 4
maximum price: 10
2 2
number of pieces: 2

minimum price: 4
1 1 1 1
number of pieces: 4
```

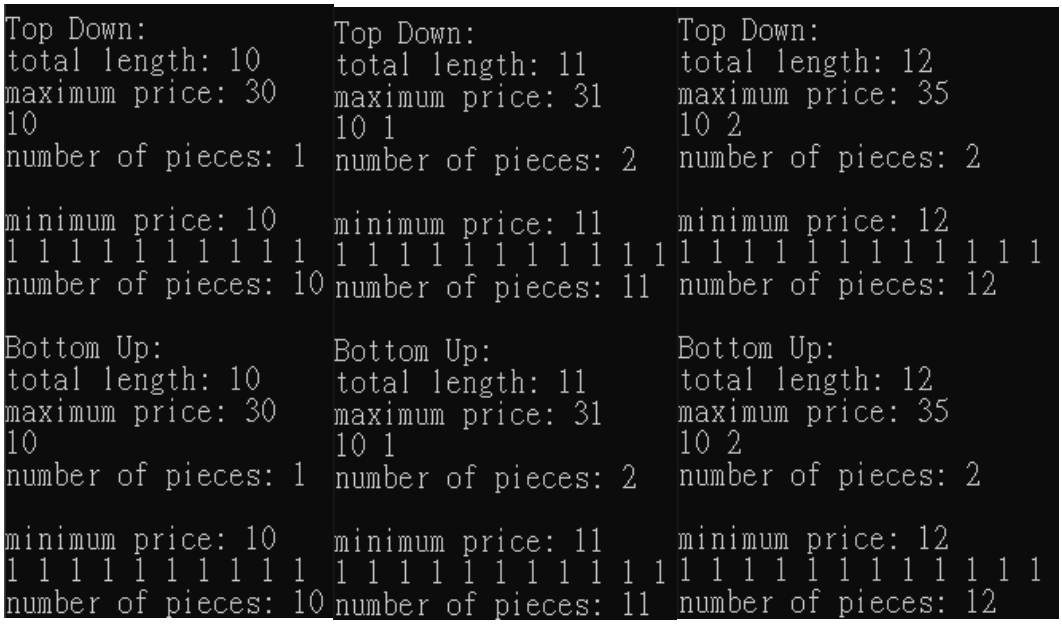
```
Top Down:
total length: 5
maximum price: 13
3 2
number of pieces: 2

minimum price: 5
1 1 1 1 1
number of pieces: 5

Bottom Up:
total length: 5
maximum price: 13
3 2
number of pieces: 2

minimum price: 5
1 1 1 1 1
number of pieces: 5
```

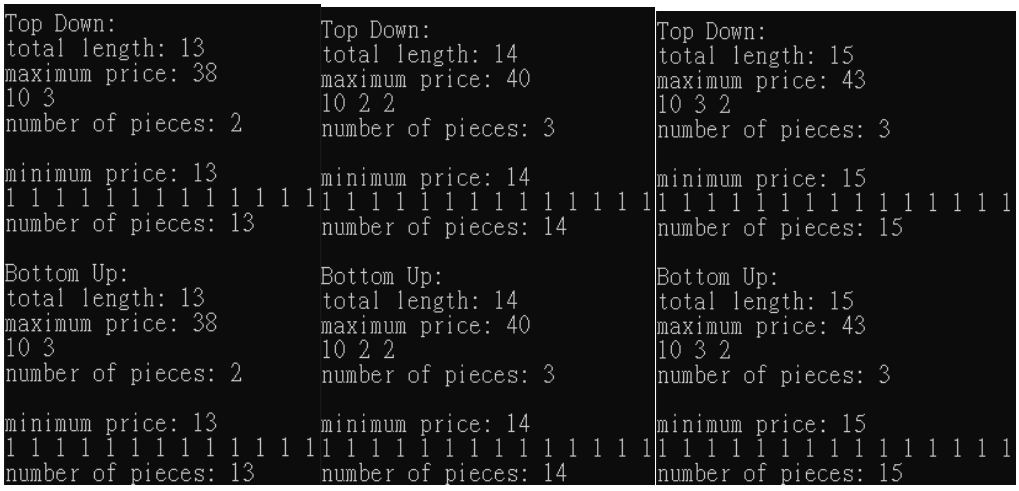
```
Top Down:
total length: 6
maximum price: 17
6
number of pieces: 1

minimum price: 6
1 1 1 1 1 1
number of pieces: 6

Bottom Up:
total length: 6
maximum price: 17
6
number of pieces: 1

minimum price: 6
1 1 1 1 1 1
number of pieces: 6
```

```
Top Down:
total length: 7
maximum price: 18
6 1
number of pieces: 2

minimum price: 7
1 1 1 1 1 1 1
number of pieces: 7

Bottom Up:
total length: 7
maximum price: 18
6 1
number of pieces: 2

minimum price: 7
1 1 1 1 1 1 1
number of pieces: 7
```

```
Top Down:
total length: 8
maximum price: 22
6 2
number of pieces: 2

minimum price: 8
1 1 1 1 1 1 1 1
number of pieces: 8

Bottom Up:
total length: 8
maximum price: 22
6 2
number of pieces: 2

minimum price: 8
1 1 1 1 1 1 1 1
number of pieces: 8
```

```
Top Down:
total length: 9
maximum price: 25
6 3
number of pieces: 2

minimum price: 9
1 1 1 1 1 1 1 1 1
number of pieces: 9

Bottom Up:
total length: 9
maximum price: 25
6 3
number of pieces: 2

minimum price: 9
1 1 1 1 1 1 1 1 1
number of pieces: 9
```





Analysis

Top-Down 是利用 recursive 的方式去將大問題拆解成小問題，好理解也好實踐，但是一直重複地將大問題拆解成一樣的小問題非常沒有效率，會重複做一樣的事，所以 Dynamic Programming 的方式會利用一個 r[0…n]的 array 紀錄 optimal 的 result，這樣當執行時會先檢查這個小問題是否解過了，如果解過了就可以直接把解拿來用，也就可以避免一直解一樣參數的小問題。

Bottom-Up 則是先從最小問題開始解，再一層一層往上解，解的過程中一樣利用一個 r[0…n]的 array 紀錄 optimal 的 result，這樣無論解到多大的問題，我們都有所有比這個問題還要小的問題的解了，如此一來一樣每個小問題最多只會解一次而已，並不會重複解小問題。

而雖然兩者的 running time 都是$\Theta(n^2)$，但因為 Bottom-Up 不用 recursively call function，所以 Bottom-Up 通常有較好一點的效率。