110550136

- **Environment**

- Windows
- g++
- Dev C++ 5.11
- How to run my program: (in CMD)

```
C:\Users\user\Desktop\NYCU\Algorithm\Exercise#3>g++ -o test 110550136.cpp

C:\Users\user\Desktop\NYCU\Algorithm\Exercise#3>test
4 6
0 4 6 7 7 7 7
0 2 4 6 8 9 10
0 6 8 8 8 8 8
0 2 3 4 4 4 4

18
```

input

output

- **Solution**

- two tables:

Table	profit	max
Size	n*(m+1)	n*(m+1)
Meaning of (i,j) 0 <= i < n , 0 <= j <= m	profit of the i^{th} project with j resources	maximal profit of 0^{th}~i^{th} projects with j resources

- Button-up method:

start from length=1, that is, the <u>first project (0^{th})</u> with resource= 0, 1, ..., m

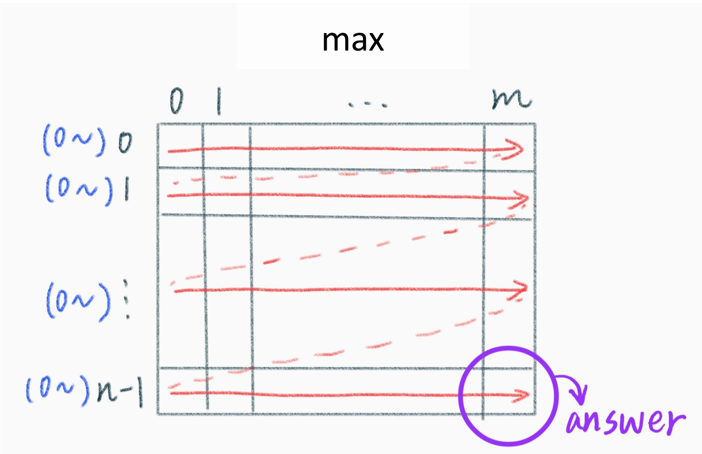
→ continue with length=2, the <u>first two projects (0^{th}~1^{th})</u> with resource= 0, 1, ..., m

→ length=3, the <u>first three projects (0^{th}~2^{th})</u> with resource= 0, 1, ..., m

→ ...

→ length=n, <u>all projects (0^{th}~$(n-1)^{th}$)</u> with resource= 0, 1, ..., m

➔ length=n and resource=m is the answer we look for



- **Algorithm**

create table *profit* with size n*(m+1);
create table *max* with size n*(m+1);

fill in *profit* from the input data;
fill in the first row of *max* since *max*[0][j]=*profit*[0][j];

```
// fill in max row by row, and column by column
for (int i=1 ; i<n ; i++)              // the second row to the last row
   for (int j=0 ; j<=m ; j++)          // the first column to the last column
   {
     int tmp=-1000;                    // initialize tmp as a very small number
     for (int k=0 ; k<=j ; k++)        // the new added project with 0~j resources
     {
        if (profit[i][k] + max[i-1][j-k] > tmp)
           tmp=profit[i][k] + max[i-1][j-k];
     }
     max[i][j]=tmp;    // tmp is the maximal profit of 0th~ith projects with j resources
   }

print max[n-1][m];
```

➔ have optimal structure:

$$max[i][j] = \begin{cases} profit[i][j] & \text{, if } i = 0 \\ \max(\, profit[i][k] + max[i-1][j-k] \,),\ 0 <= k <= j & \text{, if } i >= 1 \end{cases}$$

➔ have overlapping subproblems:
ex. *max*[2][2] and *max*[2][3] have to solve *max*[1][0], *max*[1][1], and *max*[1][2]

➔ dynamic programming

- **Time Complexity**

time complexity of a dynamic programming algorithm depends on the product of:

- number of subproblems overall → $\theta(\, n*(m+1) \,)$
- number of choices of each subproblem → $\theta(m+1)$

➔ time complexity = $\theta(nm^2)$