Environment

I used C++ version 14.0.0 to perform this program.
The below commands in a terminal can successfully run my program.

~ % g++ 110550091.cpp -o name
~ % ./name

Report

This homework took some time to think about the best approach, but it did eventually come to my mind, and the actual dynamic programming part only took around 10 lines of code.

First, we have to understand that each of the *n* projects may have up to *m* resources, we can only obtain *m* resources in total, and the profit in each project is different. My approach is to view each project one at a time and take another project into consideration each time.

Using the practice input as an example, *n=4,* and *m=6*. So we allocate three matrix memories *profit[4][7], result[7][7],* and *max[5][7]*. Matrix *profit* stores the original input where each row represents each project and each column represents the profit one will get if one gets that number of resources in that project:

profit	0	1	2	3	4	5	6
Project 1	0	4	6	7	7	7	7
Project 2	0	2	4	6	8	9	10
Project 3	0	6	8	8	8	8	8
Project 4	0	2	3	4	4	4	4

max (iteration 1)	0	1	2	3	4	5	6
Project()	0	0	0	0	0	0	0
Project(1)	0	0	0	0	0	0	0
Project(1+2)	0	0	0	0	0	0	0
Project (1+2+3)	0	0	0	0	0	0	0
Project (1+2+3+4)	0	0	0	0	0	0	0

Matrix *result* refreshes each time a new project is added into consideration. The column of *result 1* represents the number of resources allocated to row 1 of *max,* and the row of *result 1* represents the number of resources allocated to Project 1. Matrix *result* stores only numbers where i+j <= 6 (max number of resources = m). Matrix *max* stores the maximum profit of each iteration and would be updated along with the *result* matrix to choose the maximum number with the specific number of allocated resources. The highlighted numbers are the max number of profits with the specific number of allocated resources in Project() of *max* and Project 1, and they will be stored in row 2 of *max*.

result 1	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	4	4	4	4	4	4	
2	6	6	6	6	6		
3	7	7	7	7			
4	7	7	7				
5	7	7					
6	7						

max (iteration 2)	0	1	2	3	4	5	6
Project()	0	0	0	0	0	0	0
Project(1)	0	4	6	7	7	7	7
Project(1+2)	0	0	0	0	0	0	0
Project(1+2+3)	0	0	0	0	0	0	0
Project(1+2+3+4)	0	0	0	0	0	0	0

The column in *result 2* represents the number of resources allocated to row 2 of *max*, and the row in *result 2* represents the number of resources allocated to Project 2. The highlighted numbers are the max number of profits with the specific number of allocated resources in Project(1) of *max* and Project 2, and they will be stored in row 3 of *max*.

result 2	0	1	2	3	4	5	6
0	0	4	6	7	7	7	7
1	2	6	8	9	9	9	
2	4	8	10	11	11		
3	6	10	12	13			
4	8	12	14				
5	9	13					
6	10						

max (iteration 3)	0	1	2	3	4	5	6
Project()	0	0	0	0	0	0	0
Project(1)	0	4	6	7	7	7	7
Project(1+2)	0	4	6	8	10	12	14
Project(1+2+3)	0	0	0	0	0	0	0
Project(1+2+3+4)	0	0	0	0	0	0	0

The column in *result 3* represents the number of resources allocated to row 3 of *max*, and the row in *result 3* represents the number of resources allocated to Project 3. The highlighted numbers are the max number of profits with the specific number of allocated resources in Project(1+2) of *max* and Project 3, and they will be stored in row 4 of *max*.

result 3	0	1	2	3	4	5	6
0	0	4	6	8	10	12	14
1	6	10	12	14	16	18	
2	8	12	14	16	18		
3	8	12	14	16			
4	8	12	14				
5	8	12					
6	8						

max (iteration 3)	0	1	2	3	4	5	6
Project()	0	0	0	0	0	0	0
Project(1)	0	4	6	7	7	7	7
Project(1+2)	0	4	6	8	10	12	14
Project(1+2+3)	0	6	10	12	14	16	18
Project(1+2+3+4)	0	0	0	0	0	0	0

The column in *result 4* represents the number of resources allocated to row 4 of *max*, and the row in *result 4* represents the number of resources allocated to Project 4. The highlighted numbers are the max number of profits with the specific number of allocated resources in *max* of Project(1+2+3) and Project 4, and they will be stored in row 5 of *max*.

result 4	0	1	2	3	4	5	6
0	0	6	10	12	14	16	18
1	2	8	12	14	16	18	
2	3	9	13	15	17		
3	4	10	14	16			
4	4	10	14				
5	4	10					
6	4						

max (iteration 3)	0	1	2	3	4	5	6
Project()	0	0	0	0	0	0	0
Project(1)	0	4	6	7	7	7	7
Project(1+2)	0	4	6	8	10	12	14
Project(1+2+3)	0	6	10	12	14	16	18
Project(1+2+3+4)	0	6	10	12	14	16	18

Therefore, we get the final result of 18 on *max[4][6]* which means the maximum profit with resources allocated to each project and taking the maximum resources possible (6 in this case).

Excluding the time to take in the input and allocate space for the three matrices mentioned above, only a three-layer for loop has to be taken into account. On the first layer of the for loops, *i* iterates from 0 to n. On the second layer of the for loops, *j* iterates from 0 to m+1. On the third layer of the for loops, *k* iterates from 0 to m-j+1. Thus, the time complexity of this program would be $\boldsymbol{\emptyset((n)(m+1)(m-j+1)) = \emptyset(nm^2)}$.