**Problem: Optimal Resource Allocation**

**Scenario**

Imagine you are managing a set of projects for a large company. Each project has a certain value and requires a specific amount of resources. The company has a limited amount of resources available for allocation. Your goal is to maximize the total value of the projects that can be completed with the available resources.

**Problem Statement**

Given a list of projects, where each project has a value and a resource requirement, and a total available resource capacity, determine the maximum total value of projects that can be completed without exceeding the resource capacity.

**Input Format**

* The first line contains two integers N and R, representing the number of projects and the total available resource capacity, respectively.
* The next N lines each contain two integers value and resource, representing the value of the project and the resources it requires, respectively.

**Constraints**

* 1 <= N <= 1000
* 1 <= value <= 10000
* 1 <= resource <= 1000
* 1 <= R <= 10000

**Output Format**

* Print a single integer representing the maximum total value of projects that can be completed without exceeding the resource capacity.

**Sample Input**

5 10

60 2

100 3

120 4

80 5

50 1

**Sample Output**

330

**Explanation**

The optimal way to allocate resources is to select projects with values 100 (resources 3), 120 (resources 4), 50(resources 2) and 60 (resources 2), giving a total value of 330 and using 10 resources.

**Solution**

Here's a dynamic programming solution in Python:

python

def optimal\_resource\_allocation(N, R, projects):

dp = [0] \* (R + 1)

for value, resource in projects:

for r in range(R, resource - 1, -1):

dp[r] = max(dp[r], dp[r - resource] + value)

return dp[R]

# Input reading

N, R = map(int, input().split())

projects = []

for \_ in range(N):

value, resource = map(int, input().split())

projects.append((value, resource))

# Output

result = optimal\_resource\_allocation(N, R, projects)

print(result)

**Explanation of the Solution**

1. **Dynamic Programming Array**: The solution uses a DP array dp where dp[r] represents the maximum value that can be achieved with r resources.
2. **Iterating Over Projects**: For each project, the DP array is updated by iterating backwards from R to resource to ensure that each project is only considered once.
3. **Maximizing Value**: The DP array is updated to reflect the maximum value that can be achieved by either including or excluding the current project.
4. **Final Result**: The maximum value achievable with the total available resources is stored in dp[R].

**Additional Test Cases**

**Test Case 1**

**Input:**

4 7

50 3

60 4

70 5

30 2

**Output:**

110

**Test Case 2**

**Input:**

3 5

20 2

30 3

50 4

**Output:**

50

**Test Case 3**

**Input:**

6 15

100 3

200 4

150 5

120 2

180 6

130 7

**Output:**

600

**Test Case 4**

**Input**

5 10

60 2

100 3

120 4

80 5

50 1

**Sample Output**

330

**Test Case 5**

**Input**

5 9

60 2

100 3

120 4

80 5

50 1

**Sample Output**

280

These test cases ensure that the solution handles different scenarios and correctly computes the