## 1661. Aladdin’s knapsack

Upon entering the treasure cave, Aladdin chose not to take the old, blackened lamp. Instead, he began filling his backpack with gold coins and precious gems. Of course, he wanted to take everything, but miracles don’t happen – his backpack had a limited carrying capacity and might not withstand excessive weight.

Many times, he removed some items and replaced them with others, striving to maximize the total value of his backpack’s contents.

The task is to determine the maximum value of the cargo that Aladdin can carry.

We assume that the cave contains *n* different types of items, with an unlimited number of each type available. The backpack can hold a maximum weight of *s*. Each item of type *i* has a weight of *wi* and a value of *vi* ​(*i* = 1, 2, ..., *n*).

**Input.** The first line contains two positive integers *s* and *n* (1 ≤ *s* ≤ 250, 1 ≤ *n* ≤ 35) – the maximum weight the backpack can carry and the number of item types.

The next *n* lines each contain two numbers *wi* and *vi* (1 ≤ *wi* ≤ 250, 1 ≤ *vi* ≤ 250) –the weight and value of an item of type *i*.

**Output.** Print the maximum total value of the cargo that can be carried without exceeding the weight limit *s*.

|  |  |
| --- | --- |
| **Sample input** | **Sample output** |
| 10 25 106 19 | 20 |

## SOLUTION

**knapsack**

# Algorithm analysis

Let dp*k*[*s*] be the maximum value of cargo with a weight of at most **s**, considering only the first *k* types of items.

Now, consider two possible options when forming cargo of weight *i*:

* **Do not use an item of type *k***: In this case, the optimal value remains unchanged, meaning:

dp*k*[*i*] = dp*k*-1[*i*].

* **Use an item of type *k***: Then its weight *wk* will occupy part of the knapsack’s capacity, and the value will increase by *vk*, meaning:

dp*k*[*i*] = dp*k*[*i* – *wk*] + *vk*.

Since the goal is to maximize the cargo value, we obtain the following recurrence relation:

dp*k*[*i*] = max(dp*k*-1[*i*], dp*k*[*i* – *wk*] + *vk*), *wk* ≤ *i* ≤ *s*

Base cases:

dp0[*i*] = 0 and dp*k*[0] = 0

Let the function f(*k*, *s*) compute the maximum value of cargo that can be packed into a knapsack with a weight limit of *s*, using the first *k* types of items. Then, the next recurrence relation holds:

f(*k*, *s*) = max(f(*k* – 1, *s*), f(*k*, *s* – w[*k*]) + v[*k*])

Finally, we need to compute the value of f(*k*, *s*) using memoization.

# Algorithm implementation

Declare the arrays:

* w[*i*] – the weight of an item of type *i*;
* p[*i*] – the value of an item of type *i*;
* dp[*i*] – the maximum value of cargo with a weight of at most *i*;

#define MAXW 252

#define MAXN 37

int w[MAXN], p[MAXN];

int dp[MAXW];

Read the input data.

scanf("%d %d", &s, &n);

for (i = 0; i < n; i++)

 scanf("%d %d", &w[i], &p[i]);

Process *n* types of items sequentially.

for (k = 0; k < n; k++)

{

Update the values in the *dp* array, considering the possibility of using items of type *k*. Since the number of items of each type is unlimited, each item can be chosen multiple times.

 for (i = w[k]; i <= s; i++)

 dp[i] = max(dp[i], dp[i - w[k]] + p[k]);

}

Print the answer.

printf("%d\n", dp[s]);

# Algorithm implementation – recursion

Declare the arrays:

* w[*i*] – the weight of an item of type *i*;
* p[*i*] – the value of an item of type *i*;
* dp[*i*] – the maximum value of cargo with a weight of at most *i*;

#define MAXW 252

#define MAXN 37

#define INF 2000000000

int w[MAXN], v[MAXN];

int dp[MAXN][MAXW];

The function f(*k*, *s*) computes the maximum value of cargo that can be packed into a knapsack with a weight limit of *s*, using the first *k* types of items.

int f(int k, int s)

{

If *k* = 0 or *s* = 0, the knapsack is empty, and its value is 0.

 if (k == 0 || s == 0) return 0;

If *s* < 0, we have exceeded the allowed weight, making this option invalid. Therefore, we return **-**INF (where INF is a sufficiently large number).

 if (s < 0) return -INF;

If the value of f(*k*, *s*) is already computed, return it.

 if (dp[k][s] != -1) return dp[k][s];

Compute f(*k*, *s*) and store it in dp[*k*][*s*].

 return dp[k][s] = max(f(k - 1, s), f(k, s - w[k]) + v[k]);

}

The main part of the program. Read the input data.

scanf("%d %d", &s, &n);

for (i = 1; i <= n; i++)

 scanf("%d %d", &w[i], &v[i]);

Compute and print the answer.

memset(dp, -1, sizeof(dp));

printf("%d\n", f(n, s));