## 1661. Рюкзак Алладина

Попав в пещеру с сокровищами, Алладин не стал брать старую почерневшую лампу. Вместо этого он принялся заполнять свой рюкзак золотыми монетами и драгоценными камнями. Конечно, ему хотелось забрать всё, но чудес не бывает – рюкзак имеет ограниченную грузоподъёмность и может не выдержать слишком большой вес.

Много раз он выкладывал одни предметы и заменял их другими, стремясь максимально увеличить общую стоимость содержимого рюкзака.

Требуется определить максимальную стоимость груза, который Алладин сможет унести.

Будем считать, что в пещере есть предметы *n* различных типов, причём количество предметов каждого типа не ограничено. Рюкзак может выдержать вес не более *s*. Каждый предмет типа *i* имеет вес *wi* и стоимость *vi* ​(*i* = 1, 2, ..., *n*).

**Вход.** В первой строке заданы два натуральных числа *s* и *n* (1 ≤ *s* ≤ 250, 1 ≤ *n* ≤ 35) – максимальный вес предметов в рюкзаке и количество типов предметов.

Далее следуют *n* строк, каждая из которых содержит два числа *wi*  и *vi* (1 ≤ *wi* ≤ 250, 1 ≤ *vi* ≤ 250) – вес и стоимость предмета типа *i*.

**Выход.** Выведите максимальную стоимость груза, который можно унести, не превышая предельный вес *s*.

|  |  |
| --- | --- |
| **Пример входа** | **Пример выхода** |
| 10 25 106 19 | 20 |

## РЕШЕНИЕ

**рюкзак**

**Анализ алгоритма**

Пусть dp*k*[*s*] – максимальная стоимость груза весом не более *s*, если разрешено использовать только предметы первых *k* типов.

Рассмотрим два возможных варианта при составлении груза весом *i*:

* **Не использовать предмет *k*-го типа**: в этом случае оптимальное значение не изменится, то есть

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

* **Использовать предмет *k*-го типа**: тогда его вес *wk* займёт часть вместимости рюкзака, а стоимость увеличится на *vk*, то есть

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

Так как требуется максимизировать стоимость груза, получаем рекуррентное соотношение:

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

Базовые случаи:

dp0[*i*] = 0 и dp*k*[0] = 0

Пусть функция f(*k*, *s*) вычисляет максимальную стоимость груза, который можно собрать в рюкзак весом не более *s*, используя первые *k* типов предметов. Тогда имеет место рекуррентное соотношение:

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

Осталось вычислить значение функции f(*k*, *s*) используя мемоизацию.

**Реализация алгоритма**

Объявим рабочие массивы:

* w[*i*] – вес предмета *i*-го типа;
* p[*i*] – стоимость предмета *i*-го типа;
* dp[*i*] – максимальная стоимость груза весом не более *i*;

#define MAXW 252

#define MAXN 37

int w[MAXN], p[MAXN];

int dp[MAXW];

Читаем входные данные.

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

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

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

Последовательно обрабатываем *n* типов предметов.

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

{

Пересчитываем значения массива *dp*, учитывая возможность использования предметов типа *k*. Поскольку количество предметов каждого типа не ограничено, каждый предмет можно выбирать несколько раз.

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

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

}

Выводим ответ.

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

**Реализация алгоритма – рекурсия**

Объявим рабочие массивы:

* w[*i*] – вес предмета *i*-го типа;
* p[*i*] – стоимость предмета *i*-го типа;
* dp[*i*] – максимальная стоимость груза весом не более *i*;

#define MAXW 252

#define MAXN 37

#define INF 2000000000

int w[MAXN], v[MAXN];

int dp[MAXN][MAXW];

Функция f(*k*, *s*) вычисляет максимальную стоимость груза, который можно собрать в рюкзак весом не более *s*, используя первые *k* типов предметов.

int f(int k, int s)

{

Если *k* = 0 или *s* = 0, то рюкзак пуст, и его стоимость равна 0.

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

Если *s* < 0, то мы превысили допустимый вес, и этот вариант не имеет смысла. Поэтому возвращаем -INF (INF – это некоторое большое число).

 if (s < 0) return -INF;

Если значение f(*k*, *s*) уже вычислено, то возвращаем его.

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

Вычисляем значение f(*k*, *s*) и запоминаем его в dp[*k*][*s*].

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

}

Основная часть программы. Читаем входные данные.

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

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

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

Вычисляем и выводим ответ.

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

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