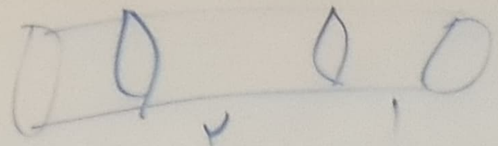
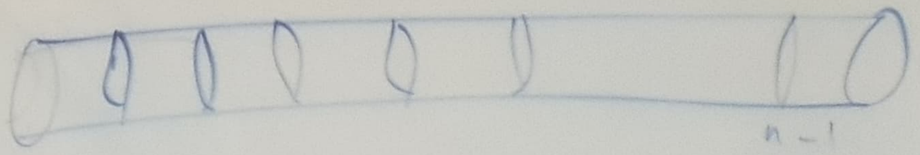




$r[n]$ = حداکثر مبلغ قابل برش میله به طول n

$$r[n] = \max \begin{cases} p_i \\ p_1 + r(i-1) \\ p_2 + r(i-2) \\ p_3 + r(i-3) \end{cases}$$

طول	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
قیمت p_i	۱	۵	۸	۹	۱۰	۱۷	۱۷	۲۰	۲۴	۳۰

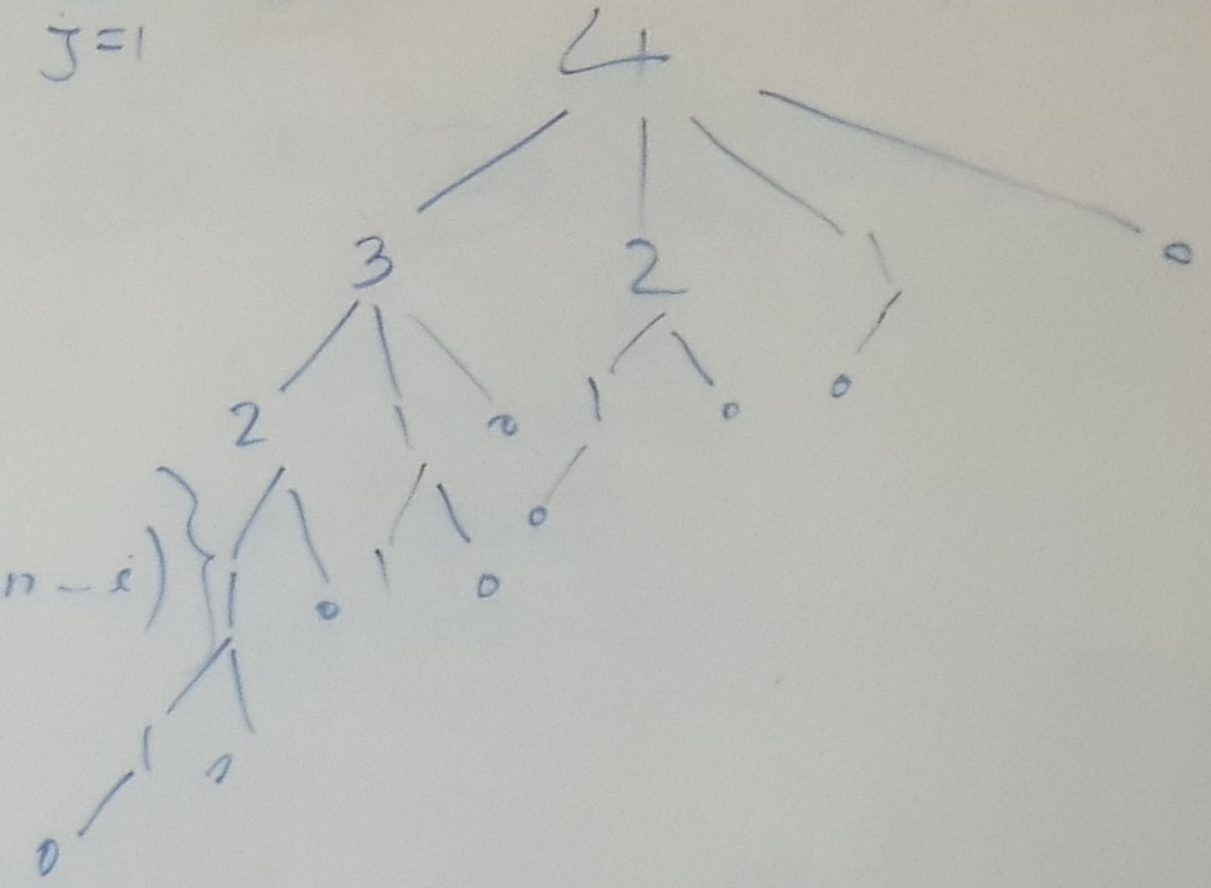
r_i	۱	۵	۸	۱۰

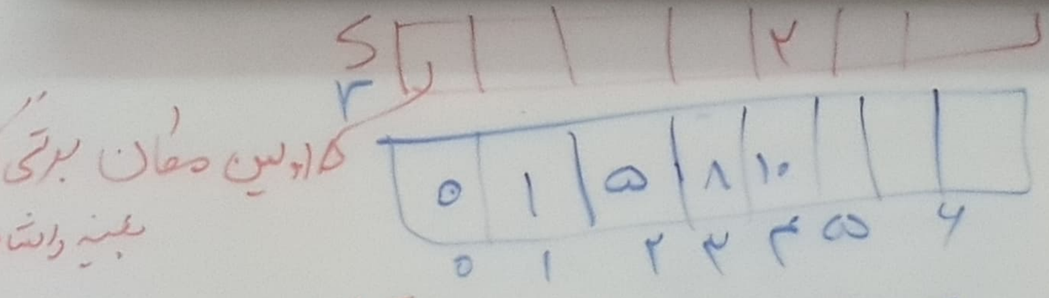
$$n = i_1 + i_2 + \cdots + i_k$$
$$r = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

بنام خدا

$$T[n] = 1 + \sum_{j=1}^{n-1} T[j]$$

```
f(int p, int n):
if n == 0
  return 0
q = -∞
for i = 1 to n
  q = max{q, p[i] + f(p, n-i)}
return q
```






```
int r[n]
r[0] = 0
for i = 1 to n {
    q = -∞
    for j = 1 to i {
        q = max{q, p[i] + r[i - j]}
    }
    r[i] = q
}
```

کاربرد این صفحه برای کد جواب بهینه را شامل می‌شود.

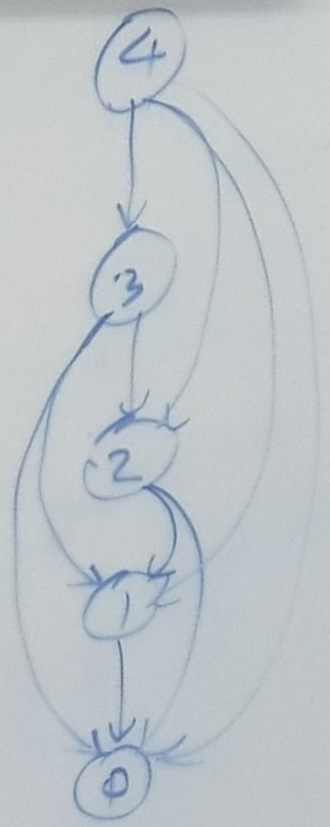
مقدار بهینه

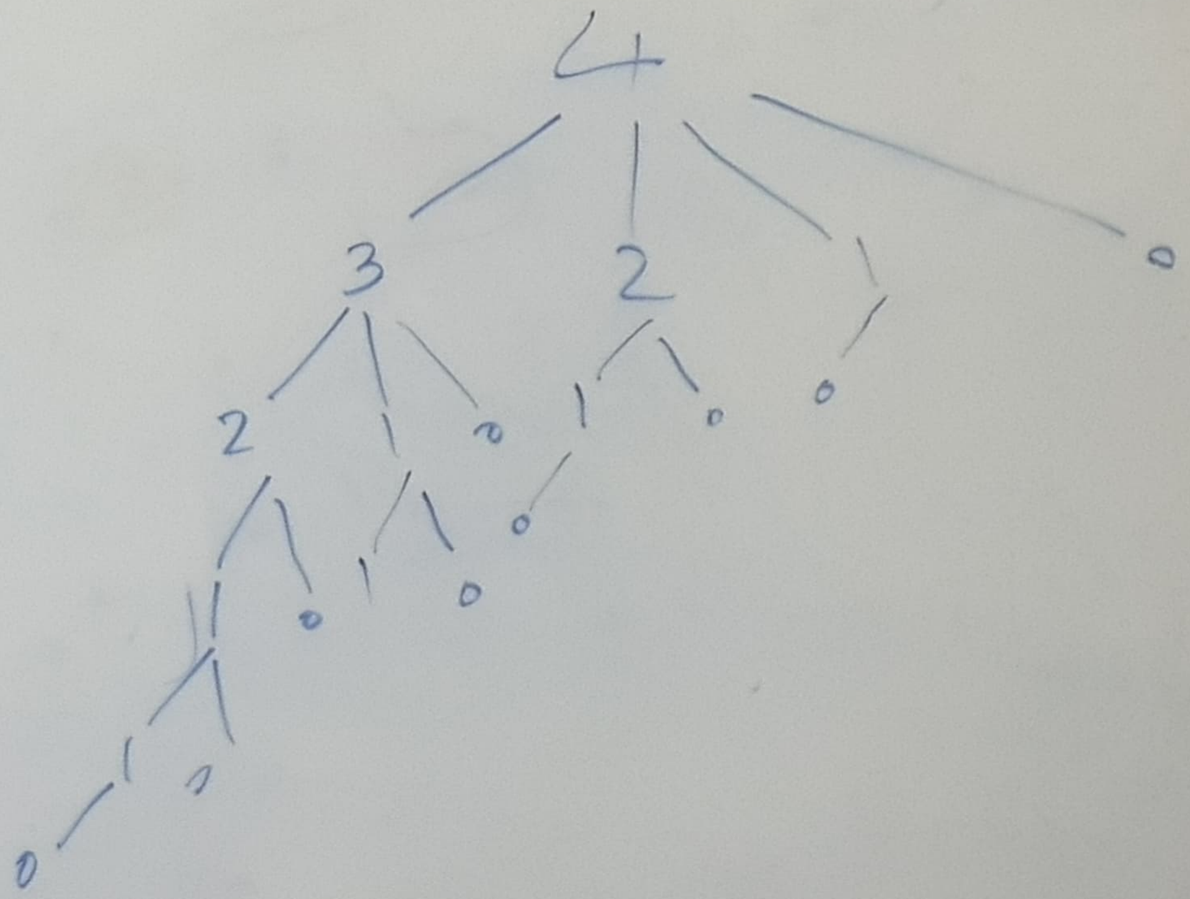
جواب بهینه

سود برش بهینه حاصله به طول i $= r[i]$

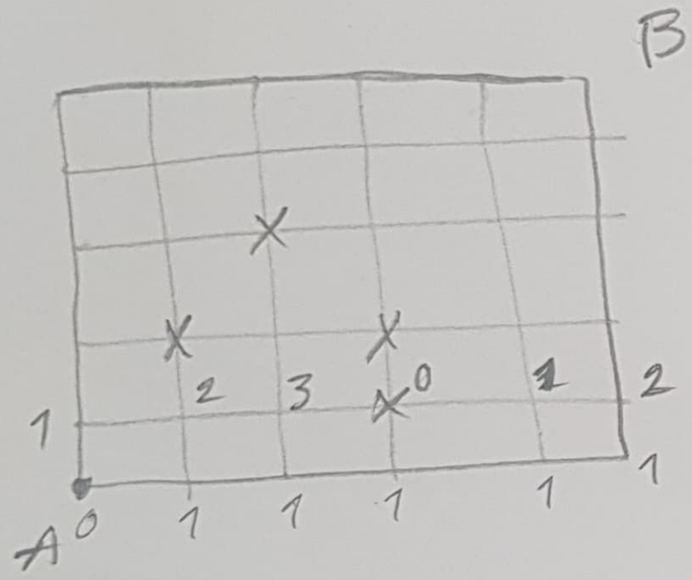
$O(n^2)$

به نام خدا





تعداد مسیرهایی که از A به B وجود دارد، با این محدودیت که نمی‌توان از نقاط دارای مانع عبور کرد و فقط راست R یا بالا U می‌توانیم برویم.



هر خانه جمع خانه چپ و پایین خود مسیر دارد.

به تعداد

مانع ها صفر مسیر دارند

ok تدریس شد، اما عکس گرفته نشد!

به نام خدا

0-1 knapsack

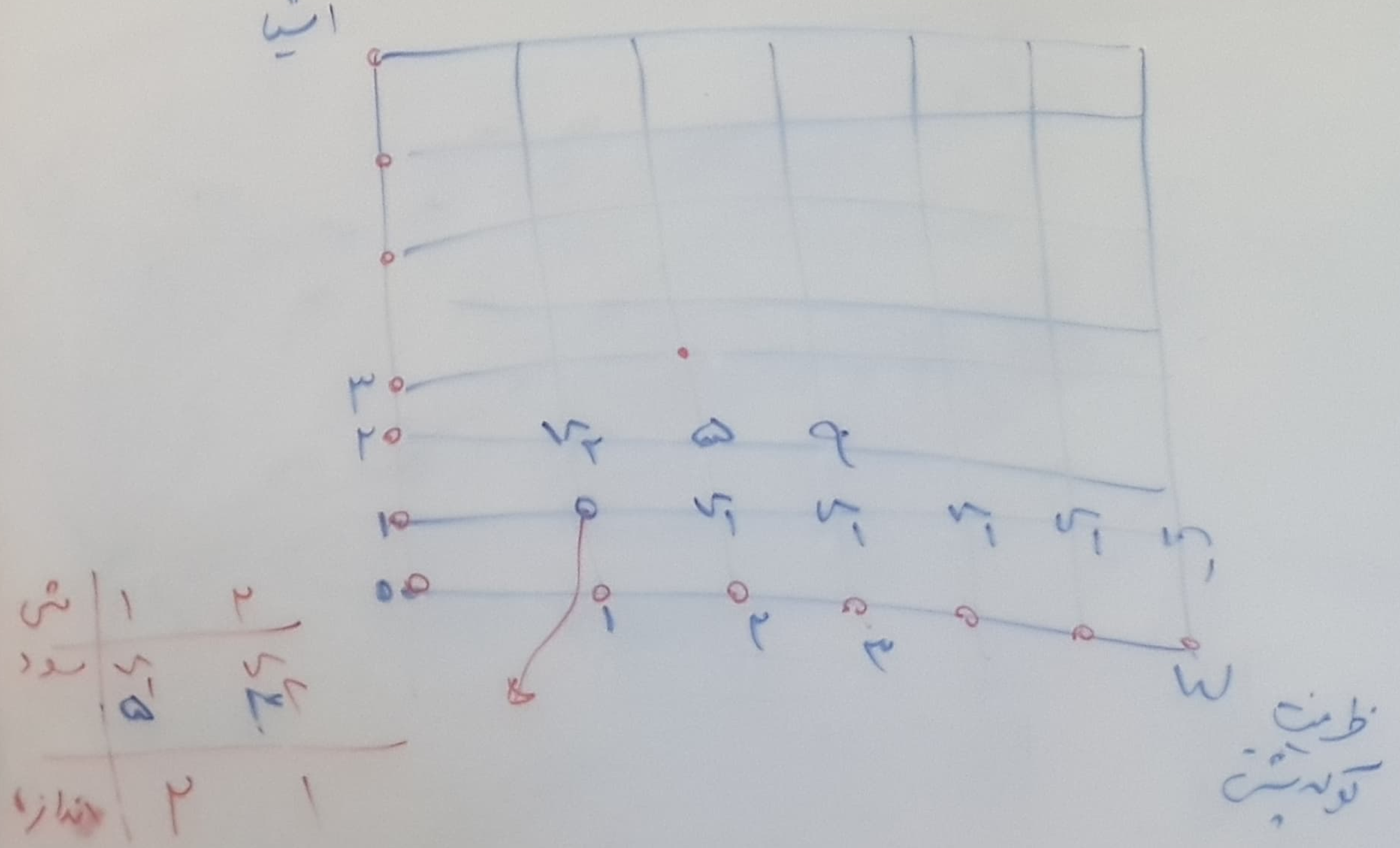
کوله پشتی 0-1

داشتن کیفی که ظرفیت وزنی W و اندازه n را دارد و ظرفیت کل کوله پشتی W است.

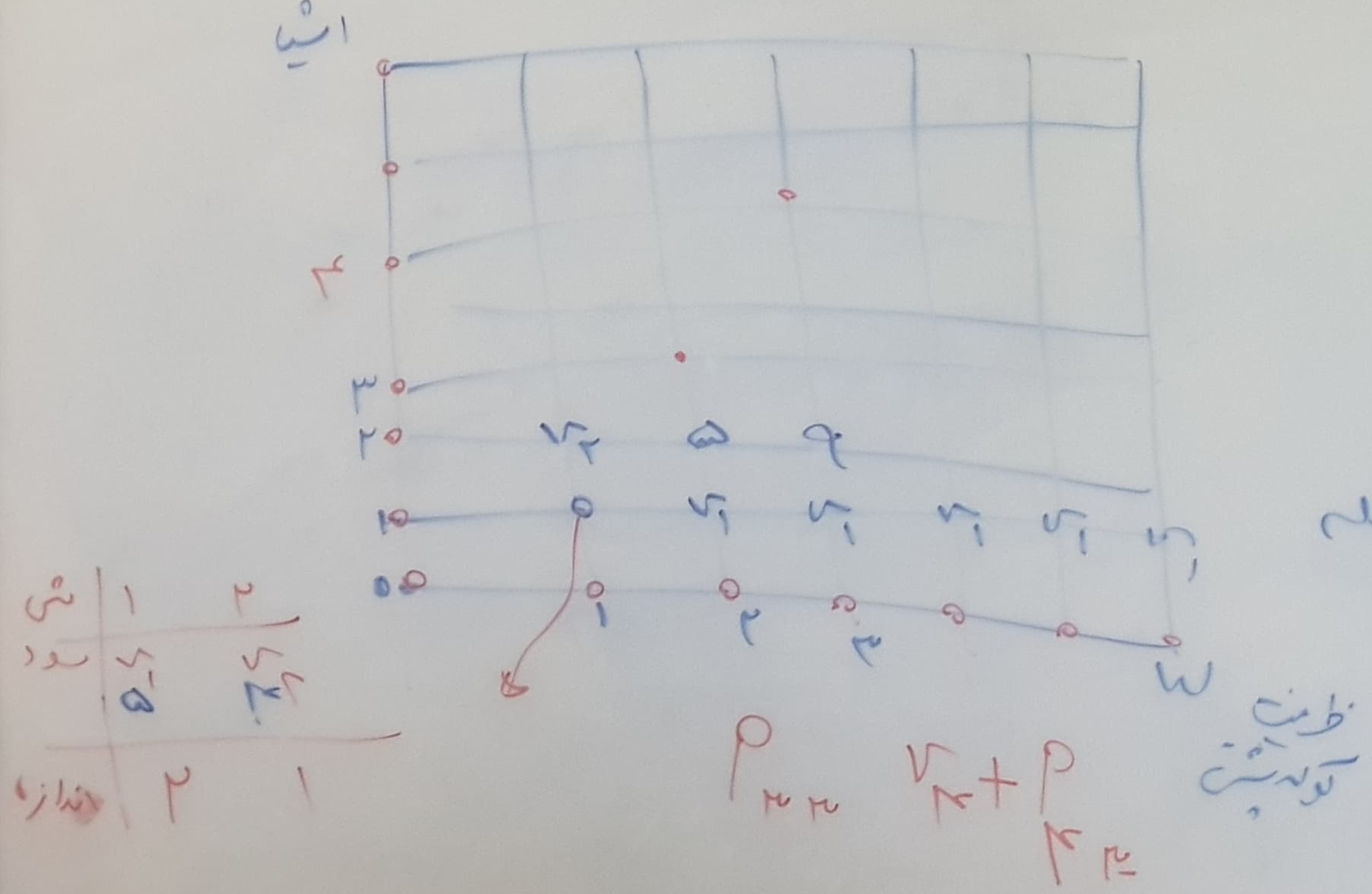
(فرض کنیم وزن‌ها و ارزش‌ها عدد صحیح باشند).

هدف انتخاب اشیایی با بیشترین ارزش است که مجموع اندازه آنها از ظرفیت کوله پشتی بیشتر نشود.

بیشترین سودی که از انتخاب اشیاء $P = \{w_1, w_2, \dots, w_n\}$ تا n می‌توان به دست آورد به شرطی که ظرفیت کوله پشتی W باشد.

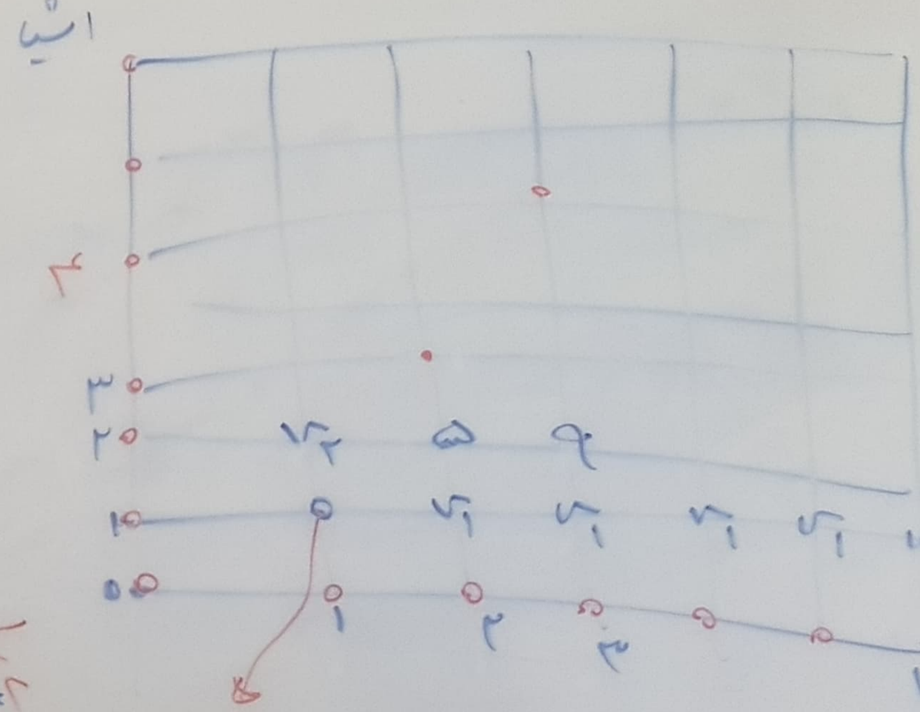


$$P_{i,j} = \max\{P_{i-1,j} \,,\, P_{i-1,j-w_i} + v_i\}$$



$$P_{i,j} = \max\{P_{i-1,j} \, , \, P_{i-1,j-w_i} + v_i\}$$

شبه چند جمله ای $O(nW)$



$$P_{i,j} = \max \left\{ P_{i-1,j} \;,\; P_{i-1,j-w_i} + v_i \right\}$$

پیچیدگی: $O(nW)$

(اثبات: استقرا)

```python
def dp_knapsack(items, weight_limit):
    weight_limit += 1
    max_value = [[0] * weight_limit for _ in range(len(items))]

    for weight in range(weight_limit):
        if weight >= items[0].weight:
            max_value[0][weight] = items[0].value

    for item_number in range(1, len(items)):
        for weight in range(weight_limit):
            option1 = max_value[item_number - 1][weight]
            option2 = 0
            if items[item_number].weight <= weight:
                option2 = max_value[item_number][weight - items[item_number].weight]\
                    + items[item_number].value
            max_value[item_number][weight] = max(option1, option2)
    return max_value[-1][-1]
```