پیاده‌سازی پروژه کوله پشتی 0-1 با دو روش برنامه نویسی پویا و راهبرد عقبگرد

نام و نام‌خانوادگی:

نام استاد:

1. شرح مسئله

- هدف، قرار دادن این اشیا در کوله‌پشتی با ظرفیت W به صورتی است که مقدار ارزش بیشینه حاصل شود. به بیان دیگر، دو آرایه صحیح val[0..n-1] و wt[0..n-1] وجود دارند که به ترتیب مقادیر و وزن‌های تخصیص داده شده به n عنصر هستند.
- همچنین، یک عدد صحیح W نیز داده شده است که ظرفیت کوله پشتی را نشان می‌دهد. هدف، پیدا کردن زیرمجموعه‌ای با مقدار بیشینه val[] است که در آن، مجموع وزن‌ها کوچک‌تر یا مساوی W باشد.
- امکان خورد کردن اشیا وجود ندارد و باید یک شی را به طور کامل انتخاب کرد و یا اصلا انتخاب نکرد. این گونه از مساله کوله پشتی را، «مساله کوله پشتی ۱-۰» می‌گویند.

2. کد مربوط به هر دو روش (اسکرین از کدها)

- **روش برنامه سازی پویا: (پایین به بالا)**

```python
def knapSackDP(W, wt, val, n):
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]
```

- **الگوریتم عقبگرد:**

در حل مسائل با رویکرد عقبگرد، تا زمانی که جستجوی تمام گره‌ها تمام نشود اطمینان نداریم که آیا گره‌ای در برگیرنده راه‌حل است یا خیر. پس اگر با رسیدن به گره‌ای سود بیشتری از سود فعلی به ما برسد، مقدار بیشترین سود را بروزرسانی می‌کنیم.

```
n = len(p)-1
numbest = 0
maxprofit = 0
bound = 0
totweight = 0

def knapsack ( i, profit, weight):
    global maxprofit
    if (weight <= W and profit > maxprofit):
        maxprofit = profit
    if (promising(i,weight,profit)):
        knapsack(i + 1, profit + p[i + 1], weight + w[i + 1])
        knapsack (i + 1, profit, weight)

def promising (i,weight,profit):
    if (weight >= W):
        return False
    else:
        j = i + 1
        bound = profit
        totweight = weight
        while (j <= n and totweight + w[j] <= W):
            totweight = totweight + w[j]
            bound = bound + p[j]
            j = j+1
    k = j;
    if (k <= n):
        bound = bound + (W - totweight)*p[k]/w[k]
    return bound > maxprofit

start = default_timer()
sort(p,w)
knapsack(0, p[0], w[0])
print(maxprofit)
end = default_timer()
print(end-start)
```

3. تصویر خروجی برنامه در هردو روش (حداقل 5 ورودی مختلف، هرکدام شامل 5 کالا با ارزش و وزن مشخص)

- روش برنامه‌نویسی پویا

	وزن ها	ارزش ها	ظرفیت کوله پشتی	سود	زمان روش DP
مثال 1	[4, 2, 1, 4, 1]	[12, 2, 2, 10, 1]	15	27	0.000401
مثال 2	[20, 10, 40, 30, 5]	[40, 100, 50, 60, 10]	60	200	0.000496
مثال 3	[23, 26, 20, 18, 32]	[505, 352, 458, 220, 354]	67	1183	0.000760
مثال 4	[27, 29, 26, 30, 27]	[414, 498, 545, 473, 543]	67	1088	0.000908
مثال 5	[1, 3, 8, 7, 4]	[20, 10, 40, 15, 25]	10	60	0.000426

مثال 1:

```
27
0.00040129999979399145
```

مثال 2:

```
200
0.0004965999978594482
```

مثال 3:

```
1183
0.0007609999956912361
```

مثال 4:

```
1088
0.0009083000040845945
```

مثال 5:

```
60
0.0004268000047886744
```

- **روش عقب گرد**

	وزن ها	ارزش ها	ظرفیت کوله پشتی	سود	زمان روش عقبگرد
مثال 1	[4, 2, 1, 4, 1]	[12, 2, 2, 10, 1]	15	27	0.000566
مثال 2	[20, 10, 40, 30, 5]	[40, 100, 50, 60, 10]	60	200	0.000169
مثال 3	[23, 26, 20, 18, 32]	[505, 352, 458, 220, 354]	67	1183	0.000258
مثال 4	[27, 29, 26, 30, 27]	[414, 498, 545, 473, 543]	67	1088	0.000408
مثال 5	[1, 3, 8, 7, 4]	[20, 10, 40, 15, 25]	10	60	0.000257

مثال 1:

```
27
0.00056669999321457
```

مثال 2:

```
200
0.00016939999477472156
```

مثال 3:

```
1183
0.00025880000612232834
```

مثال 4:

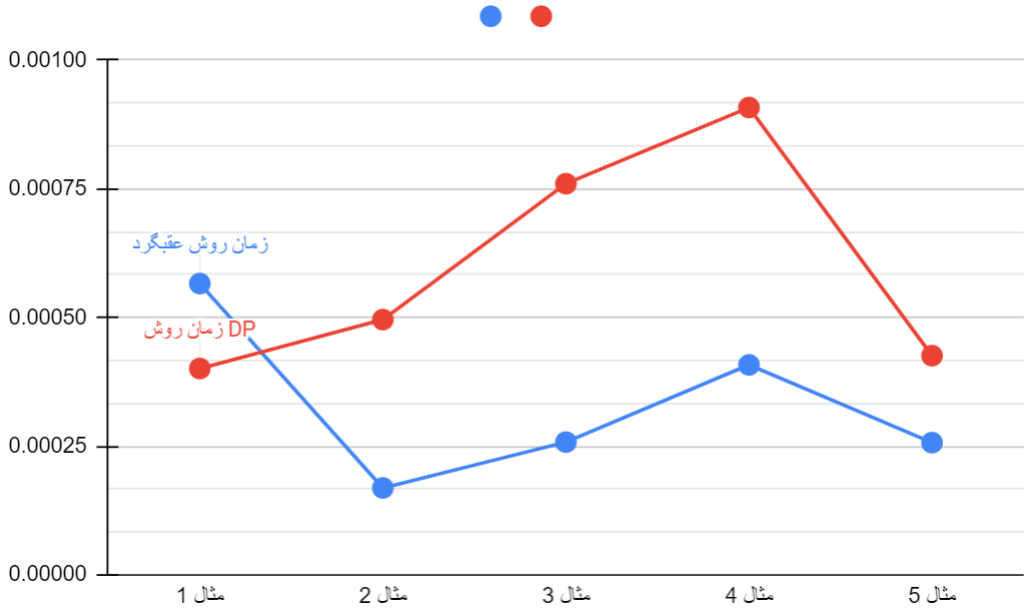
```
1088
0.0004084000029251911
```

مثال 5:

```
60
0.00025720000121509656
```

4. مقایسه دو روش به صورت نمودار بر اساس زمان اجرا

	وزن ها	ارزش ها	ظرفیت کوله پشتی	سود	زمان روش عقبگرد	زمان روش DP
مثال 1	[4, 2, 1, 4, 1]	[12, 2, 2, 10, 1]	15	27	0.000566	0.000401
مثال 2	[20, 10, 40, 30, 5]	[40, 100, 50, 60, 10]	60	200	0.000169	0.000496
مثال 3	[23, 26, 20, 18, 32]	[505, 352, 458, 220, 354]	67	1183	0.000258	0.00076
مثال 4	[27, 29, 26, 30, 27]	[414, 498, 545, 473, 543]	67	1088	0.000408	0.000908
مثال 5	[1, 3, 8, 7, 4]	[20, 10, 40, 15, 25]	10	60	0.000257	0.000426



5. پیچیدگی زمانی دو روش

- **روش برنامه سازی پویا:**

در صورت استفاده از روش برنامه سازی پویا، پیچیدگی زمانی برنامه در بدترین حالت برابر است:

$$O(\min(2^n, nW))$$

- **روش عقبگرد:**

در صورت استفاده از روش عقبگرد، پیچیدگی زمانی برنامه در بدترین حالت برابر است:

$$\theta(2^n)$$

6. **تحلیل و مقایسه نتایج در چند سطر**

در 5 مثال داده شده در 4 مورد اختلاف قابل توجهی با هم داشتند. که این موضوع خود نیز قابل توجه است. ولی به طور قطع، هردو روش از روش عادی بازگشتی بهینه‌تر و سریع‌تر می‌باشند.

اما این که دقیق بگوییم کدام مورد بهتر است به آزمایش‌های متعددتری نیاز داریم. ولی با توجه به آمار فعلی روش عقبگرد عملکرد بهتری را داشته است.

7. **نتیجه گیری (بر اساس تحلیل و مقایسه و پیچیدگی زمانی)، کدام روش بهتر است و چرا؟**

اما به یاد بیاورید که بدترین تعداد ورودی هایی که توسط الگوریتم برنامه نویسی پویا برای مسئله کوله پشتی 0-1 محاسبه می شود، $O(\min(2^n, nW))$ در بدترین حالت، الگوریتم عقبگرد گره ها $\theta(2^n)$ را بررسی می کند. با توجه به کران اضافی nW، ممکن است به نظر برسد که الگوریتم برنامه نویسی پویا برتر است اما به طوری کلی نیاز به برسی نظری است و نمیشه خیلی دقیق نظری داد. همانطور که در نمودار دیدیم در یک مورد برنامه نویسی پویا بهتر عمل کرد و شاید اگر الگوریتم را برای نمونه‌های بزرگتر اجرا کنیم نتایج دقیق‌تری بدست آوریم.

با تشکر