

初心者にもわかるナップザック問題

魔法の石と最適解の導き方



ミッション：リュックを壊さず、最大の価値を持ち帰る



リュックの容量: 5



石0

重量
2ブロック

価値
1コイン



石1

重量
2ブロック

価値
2コイン



石2

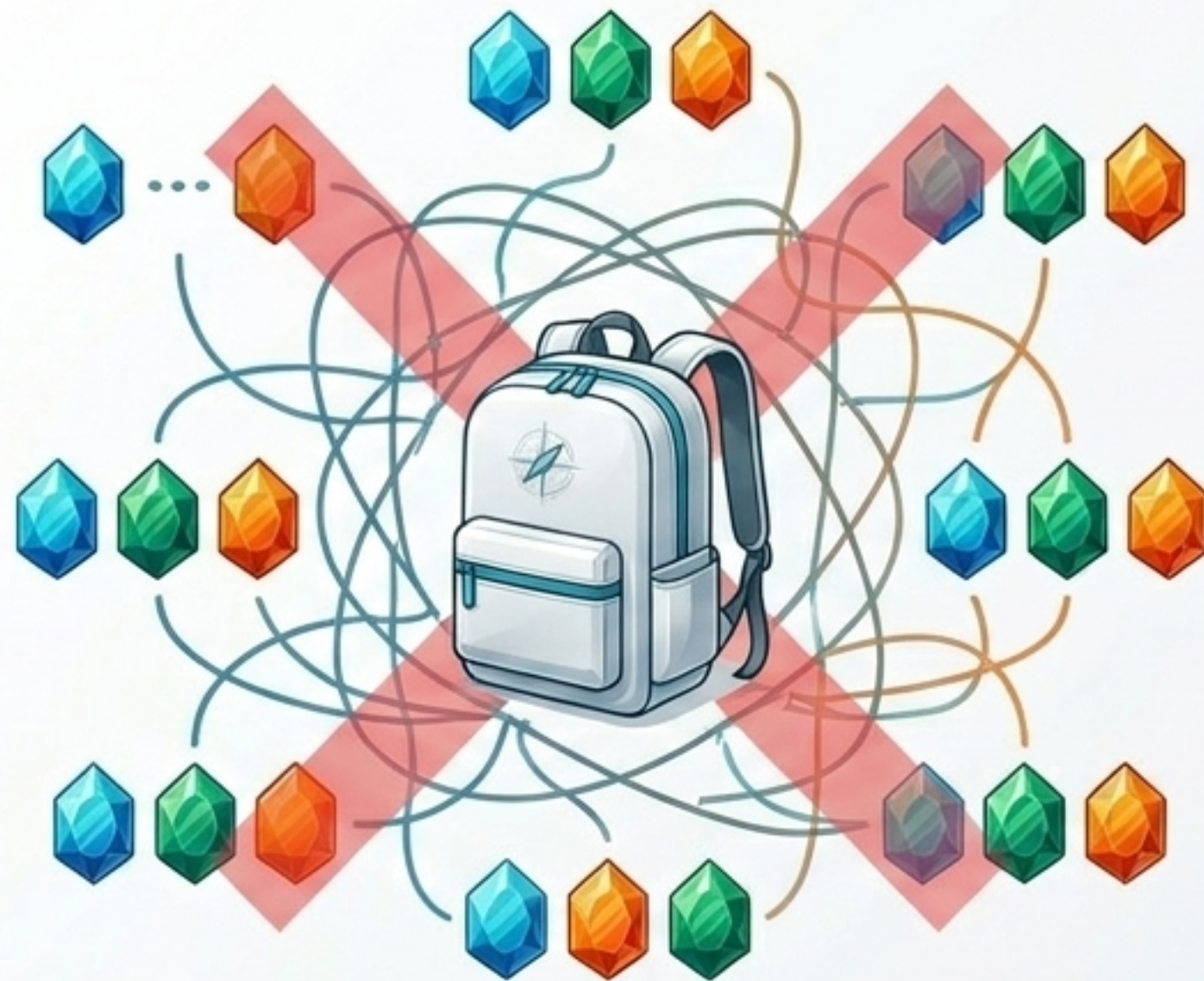
重量 3ブロック

価値 4コイン

各石は1個までしか選べない

どの組み合わせが一番価値が高くなるか？

闇雲に試すのではなく、「世界を少しずつ広げる」

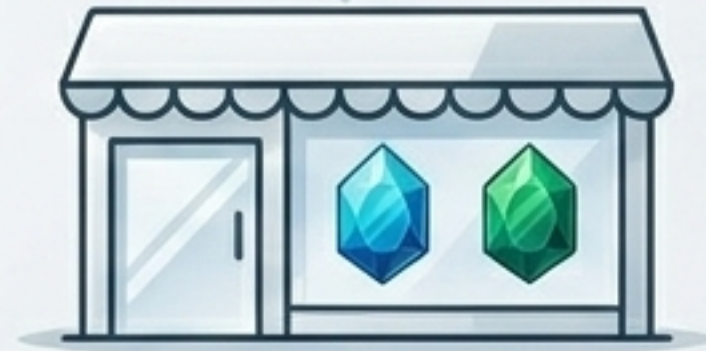


全パターンを試すのは非効率

ステップ1



ステップ2



ステップ3



魔法の言葉：「もしお店に、石が1種類しかなかったら？」から始める

すべての決断は、この「2択」の比較で決まる

新しい石を目の前にした時のルール

入れない場合

過去の実績を引き継ぐ



入れる場合



過去のベスト



今の石の価値

4コイン

+

(容量 - 今の石の重量)
の時の過去の最大価値

1コイン

この2つを計算し、**「大きい方」**を新しい正解とする。

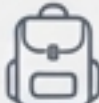
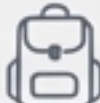
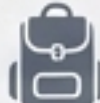
ステップ1：世界に「石0」しかない場合



	容量0	容量1	容量2	容量3	容量4	容量5
 石0 (重2, 価1)	0	0	1	1	1	1
 石0 (重2, 価1)	0	0	1	1	1	
 石0 (重2, 価1)	0	0	1	1	1	
 石0 (重2, 価1)	0	0	1	1	1	

選択肢が石0だけなら、どんなに大きなリュックを持っても価値は「1」が限界。

ステップ2：お店に「石1」が入荷された（容量2の決断）

Tactical UI Map データ		 容量0	 容量1	 容量2	 容量3	 容量4	 容量5
1	 石0 (重2, 価1)	0	0	1	1	1	1
2	 石1 (重2, 価2)			2			

1
↓

入れない場合
上のマスを見る
→ 価値1

入れる場合
石1を入れる
→ 価値2

$2 > 1$

古い石（価値1）を捨てて、新しい石（価値2）に入れ替えた瞬間。

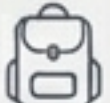
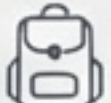
ステップ2：余った空間が「過去の知恵」を呼び覚ます（容量4の決断）



$[石1の価値: 2] + [過去の容量2の最大値: 1] = 3$

リュックの余り空間に、ステップ1で作った「最適解」をパズルのように嵌め込む。

ステップ3：最強の「石2」が登場。しかし重すぎる…？

Tactical UI Map データ		 容量0	 容量1	 容量2	 容量3	 容量4	 容量5
1	 石0 (重2, 価1)	0	0	1	1	1	1
2	 石1 (重2, 価2)	↓	↓	↓			
3	 石2 (重3, 価4)	0	0	2			



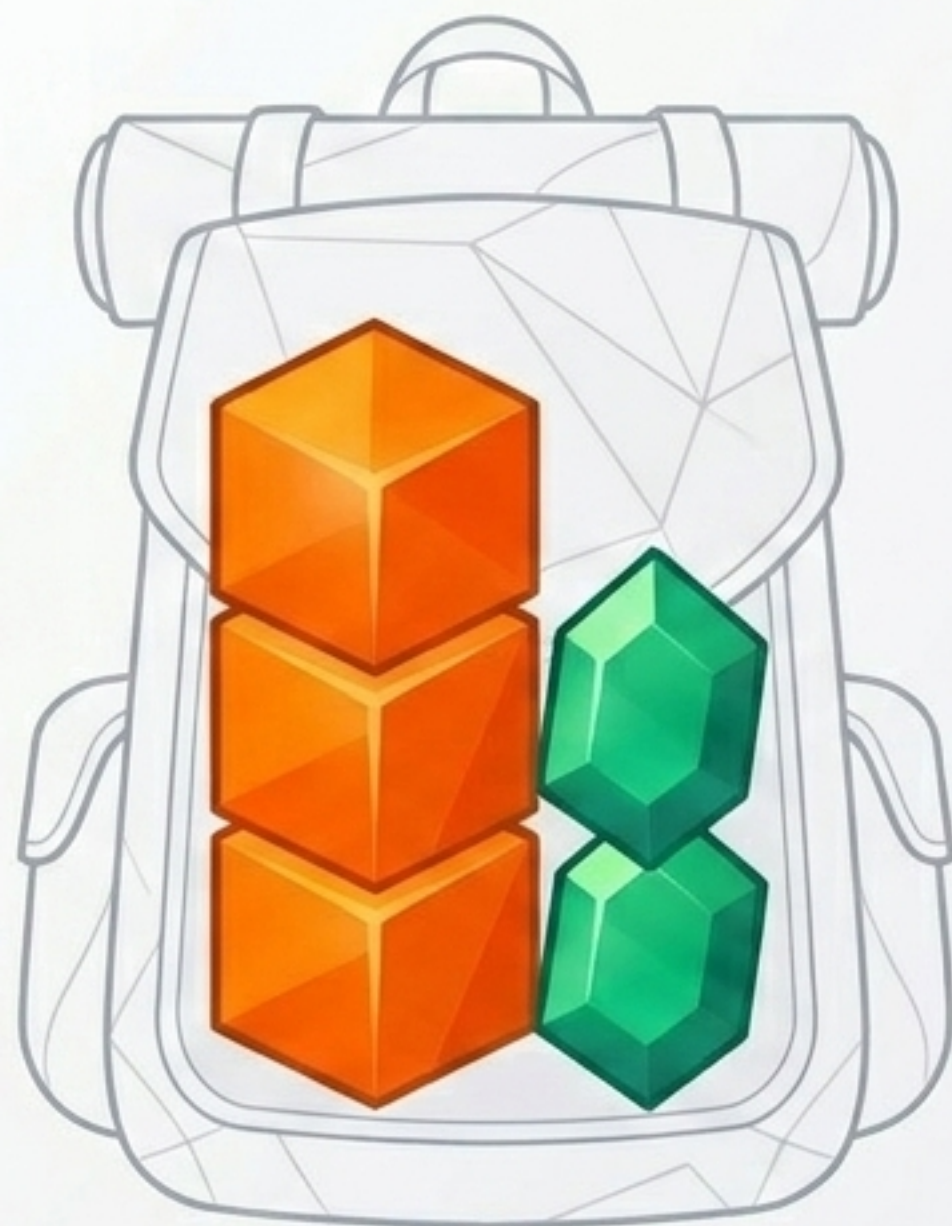
石2（重さ3）が入らない場合、
悩む必要はない。
「直前のステップの最高記録」
をそのまま引き継ぐ。

最終決断：容量5のリュックの限界に挑む



「今の石」と「過去の石」が完璧に組み合わせさった瞬間。

結論：これが最強の装備だ！



Scoreboard

合計価値:

6

選ばれた石:  石1 &  石2

複雑な組み合わせも、1歩ずつ「過去の最適解」を積み上げることで、確実に見つけ出せる。

完成したDPテーブル：可能性の地図

	容量0	容量1	容量2	容量3	容量4	容量5
初期状態	0	0	0	0	0	0
石0	0	0	1	1	1	1
石1	0	0	2	2	3	3
石2	0	0	2	4	4	6

石2を追加し、残り容量2のベストを参照

この表は単なる数字の羅列ではなく、「未来を過去の最適解から構築した」足跡です。

コードは「地図の描き方」を指示しているだけ

```
def solve_knapsack():
```

```
    items = [
```

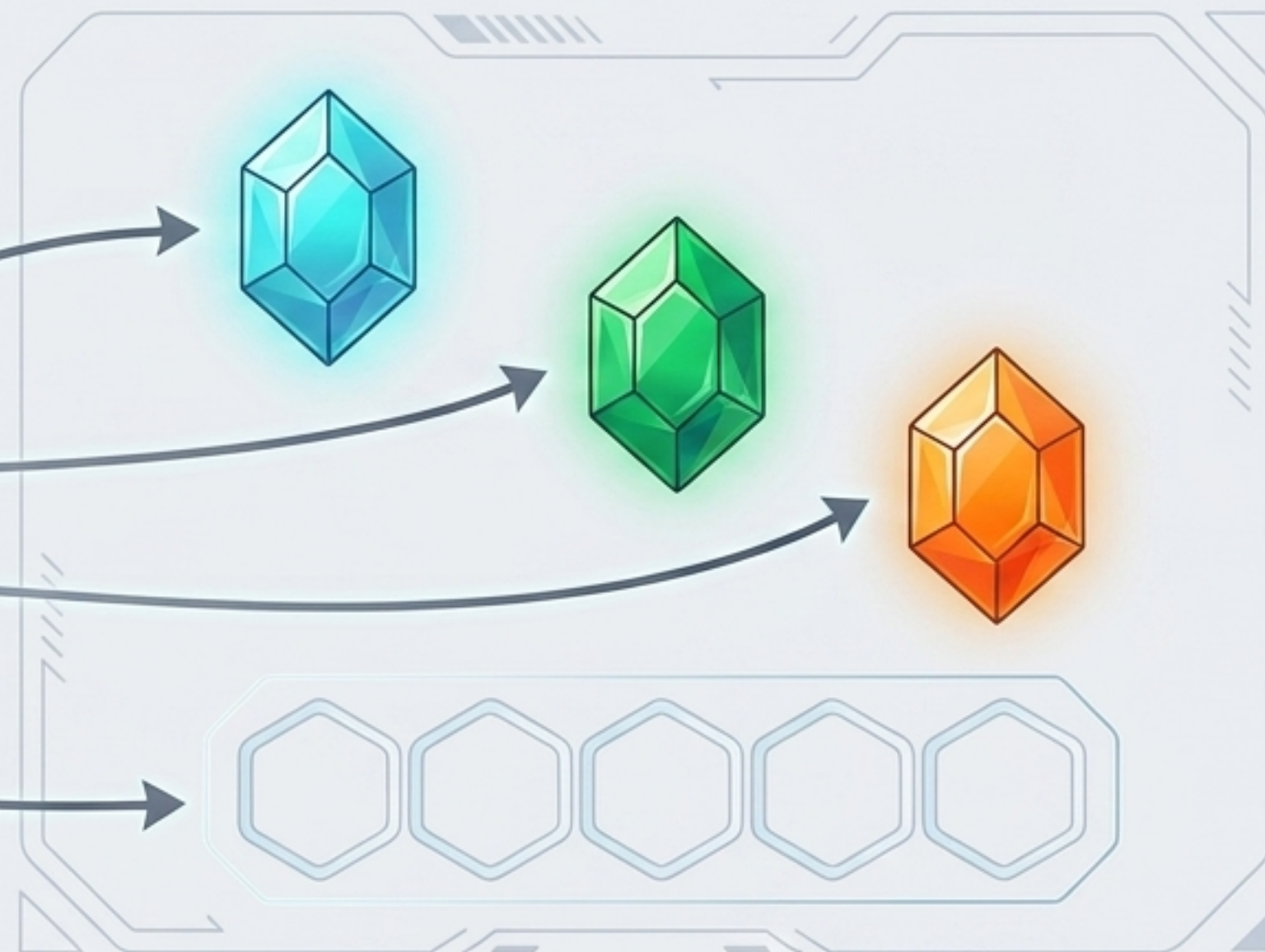
```
        (1, 2), # 石 0
```

```
        (2, 2), # 石 1
```

```
        (4, 3) # 石 2
```

```
    ]
```

```
    capacity = 5
```



プログラムも全く同じ思考回路。「入れる・入れない」の比較をコンピュータが超高速で表に書き込んでいるだけです。

まとめ：ナップザック問題が教えてくれること



1. 大きな問題は、小さく切り刻む

全部の組み合わせを考えるのではなく、「石が1つ増えたら？」という小さなステップに分解する。



2. 過去の成功を再利用する

余ったリュックの空間には、すでに計算済みの「過去のベスト」を当てはめる。



3. 複雑さは、シンプルな2択の積み重ね

毎回の決断は「今のままが良いか、入れ替えた方が良いか」を比較するだけ。

動的計画法（DP）は、魔法ではなく「賢い整理術」です。