Knapsack Backtracking Recursive

February 16, 2026

[1]:
```
from random import randint

capacity = 10
# items are (weight, value)
items = [(8,13),(3,7),(5,10),(5,10),(2,1),(2,1),(2,1)]

capacity = 23
items = [(randint(5,20),randint(5,20)) for _ in range(200)]
```

[4]:
```
# to help you write recursive functions, always plan out
#   SUPER explicitly what the inputs and outputs are
# input:
#   items_left: list of remaining items to choose from
#                  (at the start, all items are remaining)
#   capacity_left: remaining capacity
# output:
#   the best solution (as a list of 2-tuples) using just
#   "items_left" with capacity <= "capacity_left"

def solve(items_left, capacity_left, prefix=""):
    # return the set of items in the best solution
    print(f"{prefix}just got called with",(items_left, capacity_left))

    #if not items_left:
    if len(items_left) == 0:
        print(f"{prefix}|   about to return [] with total value 0")
        return []

    # item = (weight, value)
    first_item_weight = items_left[0][0]

    sol_without_item = solve(items_left[1:], capacity_left, prefix+"|   ")

    # if we have room for the first item, add it and recursively solve
    if first_item_weight <= capacity_left:
        # find the best solution that USES the first item
        sol_with_item = [items_left[0]] + solve(items_left[1:],␣
↪capacity_left-first_item_weight, prefix+"|   ")
```

```
    else:
        # if not, then only possible solution is exclusing the item
        # prune
        #print("about to return", sol_without_item)
        return sol_without_item

    # compare sol_with and sol_without, and return the best
    score_with = sum(item[1] for item in sol_with_item)
    score_without = sum(item[1] for item in sol_without_item)

    if score_with > score_without:
        print(f"{prefix}about to return {sol_with_item} with total value␣
↪{score_with}")
        return sol_with_item
    print(f"{prefix}about to return {sol_without_item} with total value␣
↪{score_without}")
    return sol_without_item
```

```
items = [(8,13),(3,7),(5,10)]

solve([(8,13),(3,7),(5,10)], 10)
 --> solve([(3,7),(5,10)], 10) # best solution without (8,13)
     --> solve([(5,10)], 10)
         solve([(5,10)], 7)
     vs
     solve([(3,7),(5,10)], 2) # best solution with (8,13)
```

[5]:
```
solve([(8,13),(3,7),(5,10)], 10)
```

```
just got called with ([(8, 13), (3, 7), (5, 10)], 10)
|   just got called with ([(3, 7), (5, 10)], 10)
|   |   just got called with ([(5, 10)], 10)
|   |   |   just got called with ([], 10)
|   |   |   |   about to return [] with total value 0
|   |   |   just got called with ([], 5)
|   |   |   |   about to return [] with total value 0
|   |   about to return [(5, 10)] with total value 10
|   |   just got called with ([(5, 10)], 7)
|   |   |   just got called with ([], 7)
|   |   |   |   about to return [] with total value 0
|   |   |   just got called with ([], 2)
|   |   |   |   about to return [] with total value 0
|   |   about to return [(5, 10)] with total value 10
|   about to return [(3, 7), (5, 10)] with total value 17
|   just got called with ([(3, 7), (5, 10)], 2)
|   |   just got called with ([(5, 10)], 2)
|   |   |   just got called with ([], 2)
|   |   |   |   about to return [] with total value 0
```

```
about to return [(3, 7), (5, 10)] with total value 17
```

```
[5]: [(3, 7), (5, 10)]
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```