Knapsack Backtracking Recursive

February 24, 2023

[12]:
```python
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 = 22
items = [(randint(5,20),randint(5,20)) for _ in range(200)]
```

[13]:
```python
# 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):
    # return the set of items in the best solution
    # print("call with",(items_left, capacity_left))

    #if not items_left:
    if len(items_left) == 0:
        return []

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

    sol_without_item = solve(items_left[1:], capacity_left)

    # if we have room for the first item, add it and recursively solve
    if first_item_weight <= capacity_left:
        sol_with_item = [items_left[0]] + solve(items_left[1:],␣
↪capacity_left-first_item_weight)
    else:
        # if not, then only possible solution is exclusing the item
```

```python
        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:
        return sol_with_item
    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)
     --> solve([(5,10)], 10)
         solve([(5,10)], 7)
     vs
     solve([(3,7),(5,10)], 2)
```

[14]:
```python
solve(items, capacity)
```

[14]: [(5, 20), (5, 19), (5, 18), (7, 20)]

[ ]:
```python

```

[ ]:
```python

```