Demo: knapsack problem

Load skid data

In [1]:
```
def load_skids(filename):

    skid_dict = dict()

    f = open(filename, 'r')

    for line in f:
        line_data = line.split(',')
        skid_dict[line_data[0]] = int(line_data[1])

    return skid_dict
```

In [2]:
```
skids = load_skids("demo_data.txt")
skids
```

Out[2]:
```
{'Statics': 3,
 'Dynamics': 7,
 'Mechanics of Materials': 9,
 'Thermodynamics': 6,
 'Heat Transfer': 3,
 'Machine Design': 2,
 'Fluid Mechanics': 5,
 'Linear Algebra': 2,
 'Control Systems': 2,
 'Calculus': 9}
```

Freight Consolidation

Partitioning items (skids)

In [3]:
```
def partitions(set_):
    """ This function is adapted from codereview.stackexchange.com. """
    if not set_:
        yield []
        return
    for i in range(2**len(set_)//2):
        parts = [set(), set()]
        for item in set_:
            parts[i&1].add(item)
            i >>= 1
        for b in partitions(parts[1]):
            yield [parts[0]]+b
```

In [4]:
```
def get_partitions(set_):
    """ This helper function fetches all of the available partitions. """
    for partition in partitions(set_):
        yield [list(elt) for elt in partition]
```

In [5]:
```
for item in (get_partitions(['a','b','c','d'])):
   print(item)
```

```
[['c', 'd', 'a', 'b']]
[['c', 'd', 'b'], ['a']]
[['c', 'a', 'd'], ['b']]
[['c', 'd'], ['a', 'b']]
[['c', 'd'], ['b'], ['a']]
[['d', 'a', 'b'], ['c']]
[['d', 'b'], ['c', 'a']]
[['d', 'b'], ['a'], ['c']]
[['a', 'd'], ['c', 'b']]
[['a', 'd'], ['b'], ['c']]
[['d'], ['c', 'a', 'b']]
[['d'], ['a', 'b'], ['c']]
[['d'], ['c', 'b'], ['a']]
[['d'], ['b'], ['c', 'a']]
[['d'], ['b'], ['a'], ['c']]
```

Using Brute-Force Algorithm

In [6]:
```
def bruteforce_transport(skids, limit=10):
    """
    Finds the allocation of skids that minimizes the number of trips or trailers
    via brute force.  The algorithm should follow the following method:

    1. Enumerate all possible ways that skids can be divided for separate trips.
    2. Select the allocation that minimizes the number of trips without making
        any trip that does not obey the weight limitation.

    Parameters:
    skids - a dictionary of name (string), weight (int) pairs
    limit - weight limit of the fixed-size trailer (one int)

    Returns:
    A list of lists, with each inner list containing the names of skids transported
    on a particular trailer and the overall list containing all the trailers
    """

    skids_list = list(skids.copy()) # not mutate the given dictionary of skids

    min_no_trip = len(skids_list) # set max number of trips = number of skids

    for skids_divided in get_partitions(skids_list):

        feasible = True
        for trip in skids_divided:
            if sum(skids[skid] for skid in trip) > limit:
                feasible = False
                break

        if feasible and len(skids_divided) < min_no_trip:
            min_no_trip = len(skids_divided)
            out = skids_divided
            break

    return out
```

In [7]:
```
bruteforce_transport(load_skids("demo_data.txt"), 10)
```

Out[7]:
```
[['Calculus'],
 ['Statics', 'Dynamics'],
 ['Mechanics of Materials'],
 ['Fluid Mechanics', 'Machine Design', 'Control Systems'],
 ['Linear Algebra', 'Thermodynamics'],
 ['Heat Transfer']]
```

Using Greedy Heuristics

In [8]:
```
def greedy_transport(skids, limit=10):
    """
    Uses a greedy heuristic to determine an allocation of skids that attempts to
    minimize the number of trips needed to transport all the skids.
    The greedy heuristic should follow the following method:

    1. As long as the trip can fit another skid, add the largest skid that fits
    2. Once the trip is full, begin a new trip to transport the remaining skids

    Parameters:
    skids - a dictionary of name (string), weight (int) pairs
    limit - weight limit of the fixed-size trailer (one int)

    Returns:
    A list of lists, with each inner list containing the names of skids transported
    on a particular trailer and the overall list containing all the trailers
    """

    trips = []
    skids_left = skids.copy() # not mutate the given dictionary of skids

    while len(skids_left) != 0:
        trip = []
        capacity_left = limit
        try:
            smallest = min(skids_left.values())
        except ValueError:
            smallest = 0

        while capacity_left >= smallest and len(skids_left) != 0:
            weight_sorted = sorted(skids_left.values(), reverse=True) # largest

            for w in weight_sorted: # search for next weight allowed on the trip
                if w <= capacity_left:
                    next_weight = w
                    break

            for skid in skids_left.keys(): # search for corresponding skid
                if skids_left[skid] == next_weight:
                    next_skid = skid
                    break

            if next_skid in trip:
                break
            else:
                trip.append(next_skid)
                capacity_left -= skids_left[next_skid]
                del(skids_left[next_skid])

        trips.append(trip)
    return trips
```

In [9]:
```
greedy_transport(load_skids("demo_data.txt"), 10)
```

Out[9]:
```
[['Mechanics of Materials'],
 ['Calculus'],
 ['Dynamics', 'Statics'],
 ['Thermodynamics', 'Heat Transfer'],
 ['Fluid Mechanics', 'Machine Design', 'Linear Algebra'],
 ['Control Systems']]
```

Comparing Solution Techniques

In [10]:
```
import time

limit = 10

print("Brute force")
start = time.time()
print("Number of trips: " + str(len(bruteforce_transport(skids, limit))) )
end = time.time()
print("Computing time: " + str(end - start) + " seconds")

print("Greedy heuristics")
start = time.time()
print("Number of trips: " + str(len(greedy_transport(skids, limit))) )
end = time.time()
print("Computing time: " + str(end - start) + " seconds")
```

```
Brute force
Number of trips: 6
Computing time: 0.5431070327758789 seconds
Greedy heuristics
Number of trips: 6
Computing time: 0.00011086463928222656 seconds
```