
Instructions:
Solutions to this assignment must be submitted through Gradescope by **Tuesday, August 9th 2022 at 10 PM**— no late submissions will be accepted.
PLEASE USE THE STARTER CODE PROVIDED.

---

1. **(Assignment: Branch and Bound)** Recall the *Assignment Problem*: Given n agents $a_1, a_2, \ldots, a_n$, n tasks $t_1, t_2, \ldots, t_n$, and each agent a_i takes time c_{ij} to complete task t_j, assign every agent to exactly one task so that the sum of the time to complete the tasks is minimized. Mathematically, our problem is to minimize

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

such that

$$\begin{aligned}
\sum_{j=1}^{n} x_{ij} &= 1 \quad \text{for } i \in \{1, \ldots, n\},\\
\sum_{i=1}^{n} x_{ij} &= 1 \quad \text{for } j \in \{1, \ldots, n\},\\
x_{ij} &\in \{0, 1\} \quad \text{for } i, j \in \{1, \ldots, n\}.
\end{aligned}$$

One way of solving this problem is through *Branch-and-Bound*. Write a procedure called `myBranchBound`(C), where

(a) $C = (c_{ij})_{i,j=1}^{n}$: an $n \times n$ array of positive integers

and returns a tuple containing

(a) X (an $n \times n$ array of 0's and 1's) that solves the Assignment problem as formulated above.

(b) A list of upper bounds when they were updated (the first item in the list should be the first upper bound calculated by SNH)

(c) The total number of nodes constructed

For example, consider the following input:

$$C = \begin{pmatrix} 6 & 4 & 2 & 5 \\ 2 & 1 & 5 & 4 \\ 4 & 2 & 1 & 3 \\ 5 & 3 & 3 & 2 \end{pmatrix}$$

Branch-and-Bound would run as: (This is a sample evaluation that shows the process of *Branch-and-Bound* and the returns format. Your program is not required to print this and we will not be grading you on what you print.)

```
UB1 = SNH(C) =  10
ub_list = [10]
// your initial upper bound should be from SNH
// and needs to be added to your list of upper bounds
Node  1 :
    - LB =  10
    - LB( 10 ) >= UB1( 10 )
Node  2 :
    - LB =  9
    - UB =  9
    - new upper bound UB 2 ( 9 )
    - LB( 9 ) >= UB2( 9 )
    - ub_list = [10, 9]
Node  3 :
    - LB =  7
    - UB =  9
Node  4 :
    - LB =  10
    - LB( 10 ) >= UB2( 9 )
Next pass:
BRANCH: Node  3 with LB =  7
Node  5 :
    - LB =  8
    - UB =  8
    - new upper bound UB 3 ( 8 )
    - LB( 8 ) >= UB3( 8 )
    - ub_list = [10, 9, 8]
Node  6 :
    - LB =  8
    - LB( 8 ) >= UB3( 8 )
Node  7 :
    - LB =  11
    - LB( 11 ) >= UB3( 8 )
Next pass:
All nodes are exhausted!

You should return a tuple containing:
the array X encoding the optimal assignment:
X =
[
[0, 0, 1, 0],
[1, 0, 0, 0],
[0, 1, 0, 0],
[0, 0, 0, 1]
]
the list of upper bounds updated:
ub_list = [10, 9, 8]
the number of nodes constructed:
node_count = 7
```

2. **(Knapsack: Dynamic Programming)** Recall the *Knapsack Problem*: Given a set of n items each with weight w_i and value v_i, for $i = 1$ to n, choose a subset of items (i.e. to carry in a knapsack) so that the total value carried is maximized, and the total weight carried is less than or equal to a given carrying capacity, c. Mathematically, our problem is to maximize

$$\sum_{i=1}^{n} v_i x_i$$

such that

$$\sum_{x_i} w_i x_i \leq c$$
$$x_{ij} \in \{0,1\} \quad \text{for } i,j \in \{1,\ldots,n\}$$

One way of solving this problem is through *Dynamic Programming* — break a problem into smaller subproblems, solve each sub-problem, and then combine the results to get the answer to the initial problem.

In Homework 1 (Question 12), you are asked to develop a dynamic programming formula for the Knapsack Problem.

Implement this formula in a procedure called `myDynamicProgramming`(n, c, V, W), where

(a) n (a positive integer): the number of items

(b) c (a positive integer): the capacity of the knapsack

(c) $V = (v_i)_{i=1}^n$ (an array of positive integers of length n): the values of each item

(d) $W = (w_i)_{i=1}^n$ (an array of positive integers of length n): the weights of each item,

and returns

(a) an array DP the 2D-matrix containing the intermediate results generated while calculating the optimal solution

(b) an array Z the optimal choice of items for the given constraints. [If you are implementing using C++, you would need to calculate and return this in a procedure called `myBitmask`(n, c, V, W)]

For example, consider the following input:

$$n = 3$$
$$V = [5, 8, 12]$$
$$W = [4, 5, 10]$$
$$c = 11$$

The return should be:

$$Z = [1, 1, 0]$$

$$DP = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 0 & 0 & 0 & 0 & 5 & 8 & 8 & 8 & 8 & 13 & 13 & 13 \\ 0 & 0 & 0 & 0 & 5 & 8 & 8 & 8 & 8 & 13 & 13 & 13 \end{pmatrix}$$