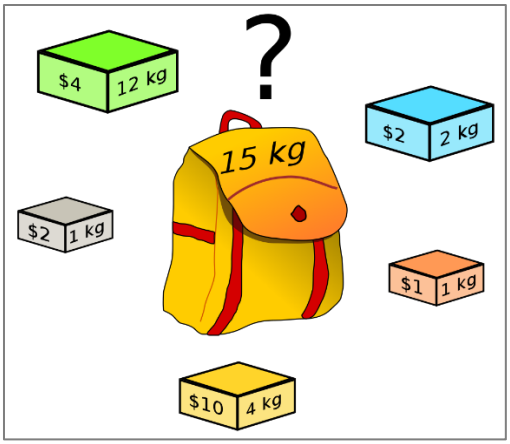
# Project 01
# Searching for the Knapsack

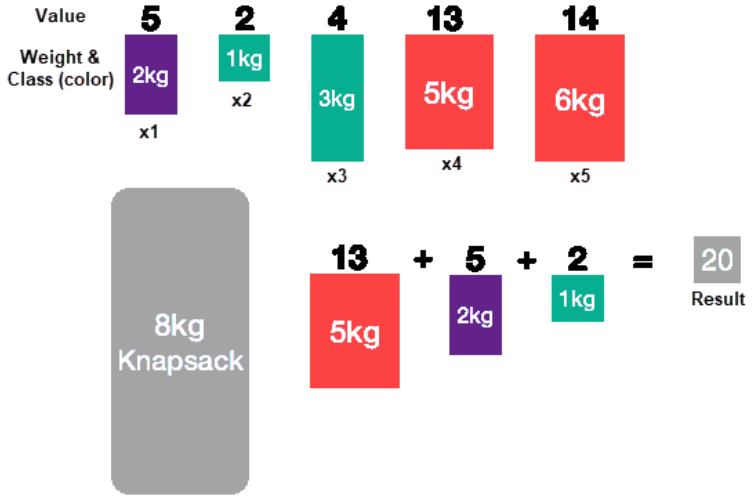
## 1. Description

"*The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible*." (Wikipedia)




*Source: en.wikipedia.org/wiki/Knapsack_problem*

The formal definition is as follows. A knapsack can hold no more than $W$ kgs. Suppose we have $n$ items, each with a value $v_i$, a class $c_i$, and weight $w_i$. Then, you are required to maximize the total value and select at least one item from each class while fitting a subset of the $n$ items into the knapsack without going over the weight limit.




*Source: kodebinary.com/0-1-knapsack-problem-example-with-recursion-and-dynamic-programming-solutions/*

Mathematically, the knapsack problem is represented as follows:

Maximize $\sum_{i=1}^{n} v_i x_i$ subject to satisfy constraints:

i. $\sum_{i=1}^{n} w_i x_i \leq W$, $x_i \in \{0,1\}$

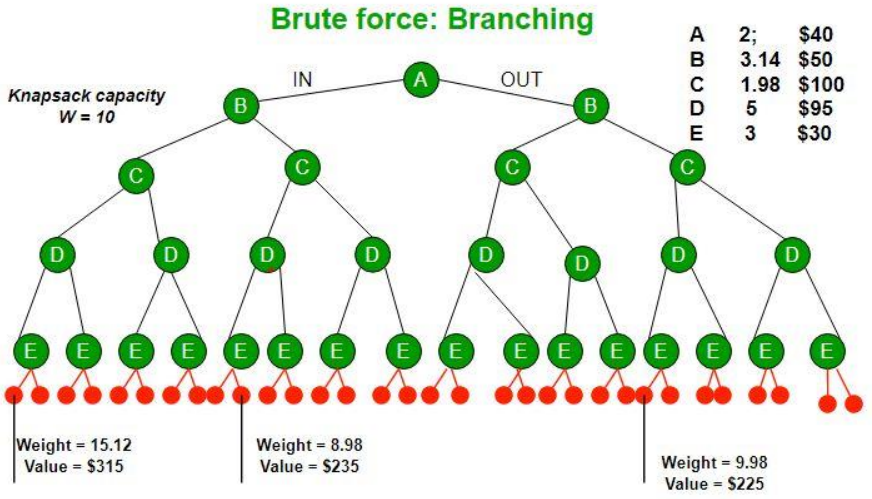
ii. At least one item from each class $c_j$, $j \in \{1, 2, \ldots, m\}$

## 2. Requirements

Implement the algorithms, give comments and comparisons.

- **Algorithm 1: brute force searching**

  Brute force is a simple method to solve many search problems. However, when applied to the Knapsack problem, the algorithm can give a solution but encounters challenges when the data size is large. The idea of brute force is to try every possible combination to satisfy the constraints. The solution will be which combination gives the most optimal result at the end. Try implementing this algorithm to see the challenges it faces.

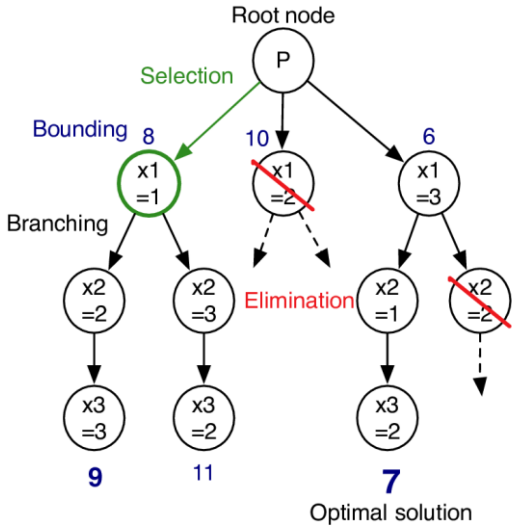



*Source: https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/*

- **Algorithm 2: branch and bound**

  The Branch and Bound method is used to solve the optimization problem. Basically, the algorithm is also based on recursive branching. However, the search tree is truncated based on the upper bound during branching. This bound utilize constraints shown in the Knapsack problem. By removing unnecessary branches, we don't have to go through unfeasible solutions and can arrive at the optimal solution faster. However, evaluating the pruned branch is a challenge that needs

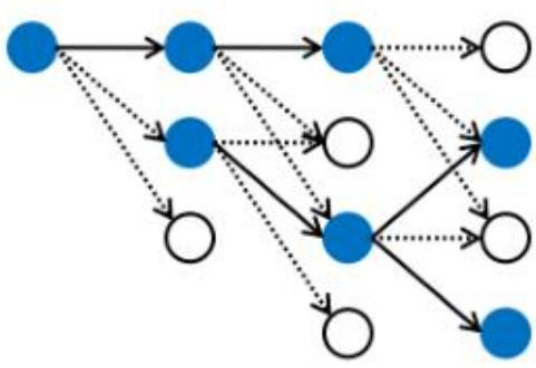
to be handled to optimize the search time. Therefore, let's design a way to prune branches reasonably and effectively.




*Source: https://www.researchgate.net/publication/281015427_PARALLEL_...*

- **Algorithm 3: local beam search**

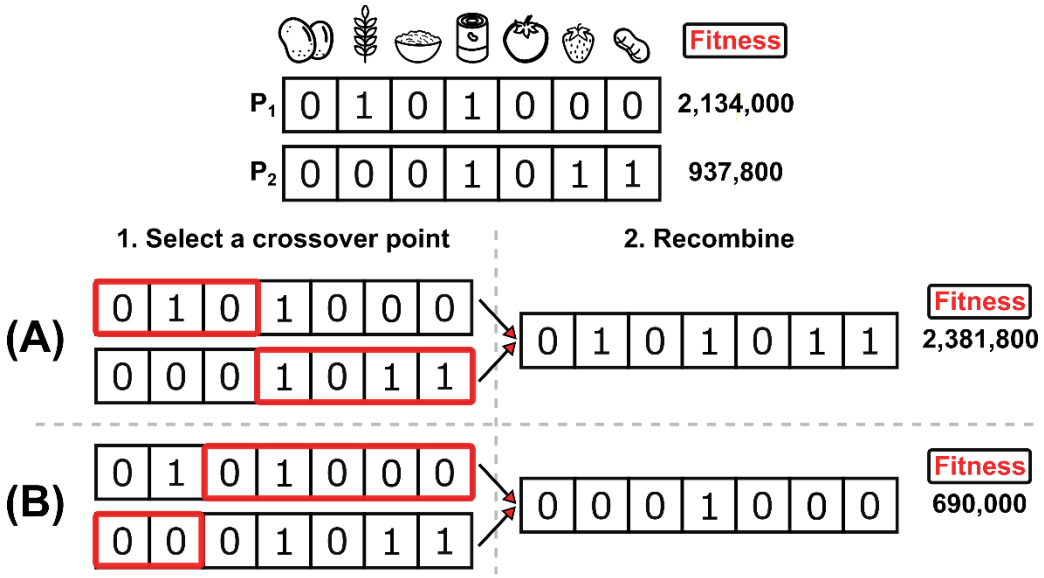
The Beam Search algorithm is a heuristic search algorithm that starts with a set of initial states. Then it creates descendants from these states. Next, the algorithm filters and retains suitable forms from them based on a specific metric. Gradually, the states will converge to an expected optimal point. However, the possible problem is that the convergence point can be locally optimal, or the states can be stuck into positions that cannot converge. Those are the challenges that we need to overcome to arrive at a well-designed algorithm that can be applied to solving the Knapsack problem. Let's implement this beam search algorithm and compare it with other algorithms mentioned.



*Source: https://www.slideserve.com/wyatt-berry/beyond-classical-search*

- **Algorithm 4: genetic algorithms**

  Genetic algorithms start with a set of initial states known as the population. These states are then combined to form descendant states. A fitness function evaluates the instances and retains the best matches. We can use a bit vector to represent a potential outcome when applied to the Knapsack problem. After that, generate a series of these vectors to form a population. Next, based on probability, determine random mutation locations, and perform hybridization to create new individuals. The process repeats until a given evolutionary cycle threshold or constraints are satisfied. Let's experiment with a genetic algorithm with the Knapsack problem to see the process in more detail.




*Source: https://freecontent.manning.com/genetic-algorithms-biologically-inspired-fast-converging-optimization/*

## 3. Input and Output

Input data is stored in the file $INPUT\_x.txt$, where $x$ is the sequence number of the test cases. The structure of the input file includes:

- The first line represents the Knapsack's storage capacity ($W$).
- The following line is the number of classes ($m$).
- The third line is the **weights** of $n$ items in floating point numbers ($w_i$).
- The fourth line is the **values** (benefits) of $n$ items in integers, respectively ($v_i$).
- The last line is the class label of each item in the order ($c_i \in \{1, \dots, m\}$).

For example:

```
101
2
85, 26, 48, 21, 22, 95, 43, 45, 55, 52
79, 32, 47, 18, 26, 85, 33, 40, 45, 59
1, 1, 2, 1, 2, 1, 1, 2, 2, 2
```

Output data is stored in the file $OUTPUT\_x.txt$, where $x$ is the sequence number of the test cases. The structure of the output file includes:

- The first line is the highest total value of the filled Knapsack.
- The following line is a sequence of 0s and 1s representing which items in n items are placed in the Knapsack.

The output for the above example is:

```
117
0, 1, 0, 0, 1, 0, 0, 0, 0, 1
```

## 4. Report

The report must also give the following information:

- Your detailed information (Student Id, Full Name)
- Assignment Plan
- Self-assessment for completion level for each requirement.
- Your work (algorithms' description, pseudo code, explanation, visualization, test cases, evaluation, comments, conclusions)
- References (if any)

## 5. Assessment

| No. | Criteria | Scores |
|---|---|---|
| 1 | Algorithm 1 | 10% |
| 2 | Algorithm 2 | 15% |
| 3 | Algorithm 3 | 15% |
| 4 | Algorithm 4 | 10% |

| | | |
|---|---|---|
| 5 | Generate at least 5 small datasets of sizes 10-40. Compare with performance in the experiment section. Create videos to show your implementation. | 10% |
| 6 | Generate at least 5 large datasets of sizes 50-1000 with number of classes of 5-10. Compare with performance in the experiment section. Create videos to show your implementation. | 15% |
| 7 | Report your algorithms, experiments with some reflection or comments. | 25% |
| **Total** | | 100% |

## 6. Notices

- The project is a **GROUP** assignment.
- Submission link and the deadline are set on Moodle.
- Any plagiarism, any tricks, or any lie will have a 0 point for the COURSE grade.