Backtracking Algorithms Assignment

“A move service (drivers + lorries with a maximum load capacity of 20 Tm) has been contracted by BACKTRACK Co. in order to carry goods from a given warehouse to another one. Those goods could have different weights. The whole service budget is 3141.59€. The CEO of the moving company thinks that 3 trucks should be enough for BC service. You are expected to provide a backtracking algorithm that first asks the user for the weights of the goods and then associates them to lorries in order to either find a solution by just 3 lorries or, otherwise, proving the CEO estimation is wrong.”

Main ideas

Our goal is to assign some goods to one of the three available trucks. We have to take also some restrictions into account: we have a maximum truck capacity that we cannot exceed, and also, the final cost of assignment cannot be greater than the budget given. In order to determine which goods should be included in each truck, we will be using a backtracking algorithm.

This algorithm consists of going through all possibilities in a recursive way and backtracking if we find a solution that does not fit with one of the restrictions.

For example, let’s say we introduce goods with weights 11, 12, 2, 4, 5. Firstly, we would draw as many nodes as goods given, and we will be creating a tree with all the options. Starting with the first one given (11), in the next level we should introduce the second one (12); however, introducing this one would exceed the maximum capacity, so we go back and try with the third node given until we find the best solution.

Formal Description

- Solution type of data: An array w that keeps the goods that has been included in the trucks. For example, w[i]=j, means the good i is carried in the truck j.For example, if w[1]=2, it means the first good in the list is carried by the second truck.
- Exhaustivity: From 0 to 4, 4 meaning a backtracking has been triggered, 0 no elements have been introduced/decided yet and 1,2,3 refering to the truck the good has been put on.

  j ∈ **{0,1,2,3,4}**
- Dead node condition (including backtracking condition): It is used to differ from where we are on the live condition, if the good does not fit in the truck we are on the dead node condition. if(loadTruck[solution[i]] + weights[i] > MAX_WEIGHT)
- Backtracking node condition: The process made when the backtracking is triggered.
  (solution[i] == NUM_TRUCKS) {
  solution[i] = -1;
  i--;
  if (i >= 0) {
  loadTruck[solution[i]] -= weights[i];
  }
- Live node condition: loadTruck[solution[i]] += weights[i];
- Solution node condition: (i == (n - 1) && (loadTruck[solution[i]] + weights[i]) <= MAX_WEIGHT)

Pseudocode

```
int MAX_WEIGHT, NUM_TRUCKS
int n, weights[n], loadTruck[NUM_TRUCKS], solution[n]
bool possible = true
for i = 0 to n
end_for
int i = 0
while(i >= 0 && i < n) then
    solution[i]++
    if(solution[i] == NUM_TRUCKS) then
        solution[i] =-1
```

```
        i--
        if(i>=0) then
            loadTruck[solution[i]] -= weights[i]
        end_if
    end_if
    else then
        if(loadTruck[solution[i]] + weights[i] > MAX_WEIGHT) the
            //Dead node, do nothing
        end_if
        else if(loadTruck[solution[i]] + weights[i] <= MAX_WEIGH
            //Live node
            loadTruck[solution[i]] += weights[i]
            i++
        end_else_if
        else if(i == (n - 1) && (loadTruck[solution[i]] + weight
            //Solution node
            break
    end_else
end_while

if(i==n) then
    for k = 0 to NUM_TRUCKS
        for j = 0 to n
            if(solution[j] == k) then
                print_weights
            end_if
        end_for
    end_for
end_if
else then
print_bossWrong
end_else
```

Computational cost

The computational cost will be divided into four main sections:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_WEIGHT 20000
#define NUM_TRUCKS 3

int main() {
    // Input
    int n;
    printf("Enter the number of items: ");
    scanf("%d", &n);

    int weights[n];
    printf("Enter the weights of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }
```

In this part of the code, the computationa cost is just “n”, due to the “for loop”.

```
int loadTruck[NUM_TRUCKS] = {0};
int solution[n];
for (int i = 0; i < n; i++) {
    solution[i] = -1;
}
bool possible = true;
int i = 0;
```

In the second part of the loop, similarlly to the first part, the cost is just “n”.

```
while (i >= 0 && i < n) {
    solution[i]++;
    if (solution[i] == NUM_TRUCKS) {
        solution[i] = -1;
        i--;
        if (i >= 0) {
            loadTruck[solution[i]] -= weights[i];
        }
    } else {
        if(loadTruck[solution[i]] + weights[i] > MAX_WEIGHT){
            // Do nothing
        } else if (loadTruck[solution[i]] + weights[i] <= MAX_WE
            loadTruck[solution[i]] += weights[i];
            i++;
        }
    }
}
```

The third part of the algorithm is the core of the code. As each item can be placed inside 3 trucks (#define NUM_TRUCKS 3), the potential number of combinations is:

$$3^n$$

```
if (i == n) {
    for (int k = 0; k < NUM_TRUCKS; k++) {
        printf("Truck %d weights:", k + 1);
        for (int j = 0; j < n; j++) {
            if (solution[j] == k) {
                printf(" %d", weights[j]);
            }
        }
        printf("\n");
    }
} else {
```

```
    printf("The boss was wrong, it's not possible with 3 trucks
}
```

In the last part, there is a nested loop so the computational cost is NUM_TRUCKS * n.

Which trasnlates to 3*n.

The total cost of this code is the most expensive one, which is O(3^n).

Implementation in C

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_WEIGHT 20000
#define NUM_TRUCKS 3

int main() {
    // Input
    int n;
    printf("Enter the number of items:\n");
    scanf("%d", &n);
        while(n<=0){
            printf("Please enter a number of items greater than
            scanf("%d", &n);

        }

    int weights[n];
    printf("Enter the weights of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
        while(weights[i]<=0){
            printf("Please enter a weight greater than 0:\n");
            scanf("%d", &weights[i]);
```

```
    }
}

// Problem variables
int loadTruck[NUM_TRUCKS] = {0}; // Array to store the load
int solution[n]; // Array to store the solution
for (int i = 0; i < n; i++) {
    solution[i] = -1; // Initialize solution array
}
bool possible = true;
int i = 0;

// Backtracking logic
while (i >= 0 && i < n) {
    solution[i]++;
    if (solution[i] == NUM_TRUCKS) {
        solution[i] = -1;
        i--;
        if (i >= 0) {
            loadTruck[solution[i]] -= weights[i];
        }
    } else {
        //Dead node
        if(loadTruck[solution[i]] + weights[i] > MAX_WEIGHT
            // Do nothing
        }
            //Live node
        else if (loadTruck[solution[i]] + weights[i] <= MAX_
            loadTruck[solution[i]] += weights[i];
            i++;
            //Solution node
        }else if (i == (n - 1) && (loadTruck[solution[i]] +
            break;
        }
```

```
        }
    }

    // Check if a solution was found
    if (i == n) {
        // Output selected weights for each truck
        for (int k = 0; k < NUM_TRUCKS; k++) {
            printf("Truck %d weights:", k + 1);
            for (int j = 0; j < n; j++) {
                if (solution[j] == k) {
                    printf(" %d", weights[j]);
                }
            }
            printf("\n");
        }
        printf("Your ceo was right\n");

    } else {
        printf("The boss was wrong, it's not possible with 3 tru
    }
}
```

Code explanation

Firstly, we create some global variables and also initialize all the variables we will be using while also asking the user to introduce the number of goods and their weights.

Secondly, we will start applying the backtracking logic. The code will be repeating as long as the i is not negative, i becomes negative when all possible conditions have been explored, and as long that same i is smaller than the number of goods we have.

In each iteration of the main loop, the value of solution is incremented, representing the truck to which the items will be assigned. If the value of solution reaches the maximum number of trucks , it backtracks to try a different option(decrementing i).

When the current weight being carried by the chosen truck plus the weight of item i exceeds the maximum weight capacity , it's considered a dead node.

If the current load of the selected truck plus the weight of item i does not exceed the maximum weight capacity, it's considered a Live Node. In a live node, the good i is assigned to the corresponding truck , and the algorithm moves on to the next item to continue exploring.

When the last item is reached, and the load of the last truck plus the weight of item i doesn't exceed the maximum weight capacity, it's considered a Solution Node.

The algorithm can exit the main loop and terminate execution.

How to execute the .exe file

Once you have opened the .c file in any compiler (we will be using “Clion”) you should click on the run option.

Afterwards, our code will ask you how many goods you want the company to carry. Then you should write any value greater than zero.

```
Enter the number of items:
```

Then, you will need to introduce each product weight, again they need to be greater than 0. And our code will give you the items chosen following the backtracking algorithm.

```
Enter the number of items:4
 Enter the weights of the items:
10000
15000
5000
5000
Truck 1 weights: 10000 5000 5000
Truck 2 weights: 15000
Truck 3 weights:
Your ceo was right
```

And if you chose too many objects, the sum of all the goods is greater than the capacity of the trucks, or if one object for example is greater than 20 000 kg, this is the output you will be getting.

```
Enter the number of items:4
 Enter the weights of the items:
12000
12000
12000
12000
The boss was wrong, it's not possible with 3 trucks.
```

Examples

Now that we have showed how to execute the code, we will show you an output where we introduce some articles without any apparent problem.

```
Enter the number of items:4
 Enter the weights of the items:
15000
5000
2500
15000
Truck 1 weights: 15000 5000
Truck 2 weights: 2500 15000
Truck 3 weights:
Your ceo was right
```

Now, we are showing what the outputs would be if we introduced an incorrect weight or number of goods.

```
Enter the number of items:
0
Please enter a number of items greater than 0:
0
Please enter a number of items greater than 0:
-1
Please enter a number of items greater than 0:
```

```
4
Enter the weights of the items:
0
Please enter a weight greater than 0:
```