Dynamic Algorithms Assignment

“A move service (driver + lorry with a load capacity of 15 mT) has been contracted by DYNAMIC Co. in order to carry goods from a given warehouse to another one. DC company will
pay for this service at a rate of 6€/kg transported regardless of the number of goods. As not all the goods weigh the same, the driver (and lorry’s owner) ask you to provide a dynamic programming algorithm to choose the goods to be carried so that the money received from DC will be maximum. This algorithm must ask the user for the weights of the goods and after executing should show the weights of the chosen goods and the amount of the bill for the transport service.”

Main ideas

The problem presented by this exercise is that we aim to transport goods from one warehouse to another, taking into account that the lorry to be used has a limited capacity and that the final company is willing to pay the same amount for each item regardless of its weight.

In order to resolve this exercise, we will be using dynamic programming, which consists in breaking down the problem into smaller parts so that we avoid doing unnecessary calculations while also fulfilling the main goal of maximizing the benefits.

All in all, we will be deciding in a recursive way for each good to either include it or not, we will store each of the decisions made, and afterwards we will get the best solution by choosing the results of each comparison/sub-problem previously made. We will do that by creating a table where the rows will be each good(first row: no goods, second row : first good, third row:first and second good…) and where each column will be the capacity left after inserting each object.

Formal description

- M[i,j] = Minimum amount of non-used space “j” using elements “i”.
- Ordering: Increasing.
- M[i,j] recurrent definition:

$$M[i,j] = min(M[i-1,j], M[i-1, j-w_t[i*] + w_t[i*]]$$

$$w_t = weight; i* = LastElement$$

- Brief description of how does Bellman Optimality Principle apply:

  The Bellman Optimality’s Principle will allow us to fulfill the main goal,obtaining the maximum profit by transporting objects, dividing the task into smaller pieces. It firstly divides into sub-problems the task, then it chooses the optimal solution for each of them and this is done by either deciding to include that object if it stills fits or not.

Pseudocode

```
int weights[MAX_CAPACITY], values[MAX_CAPACITY], ngoods

for i=0 to ngoods
    end_for
for i=0 to ngoods
    values[i] = weights[i]*6
    end_for

funcion(weights, values, ngoods)
int m[ngoods + 1][MAX_CAPACITY + 1]
m[i][j]=0

for i=0 to ngoods
    for j=0 to MAX_CAPACITY
    if(i==0||j==0) then
```

```
        m[i][j] = 0
    end_if
    else if(weights[i - 1] <= j) then
        new = values[i - 1] + m[i - 1][j - weights[i - 1]]
        if(new > m[i - 1][j]) then
            m[i][j] = new
        end_if
            else then
            m[i][j] = m[i - 1][j]
            end_else
    end_else_if
```

Implementation in C

```
#include <stdio.h>
#define MAX_CAPACITY 15000

void funcion(int weights[], int values[], int ngoods) {
    //The m[i][j] function used to calculte the price to be paic
    int m[ngoods + 1][MAX_CAPACITY + 1];
    for (int i = 0; i <=ngoods ; ++i) {
        for (int j = 0; j < MAX_CAPACITY; ++j) {
            m[i][j]=0;
        }
    }

    for (int i = 0; i <= ngoods; ++i) {
        for (int j = 0; j <= MAX_CAPACITY; ++j) {
            //Base case : If there's no object or no more capaci
            if (i == 0 || j == 0) {
                m[i][j] = 0;
            }// If the weight of the previous object is still lo
            else if (weights[i - 1] <= j) {
                //Calculates what the value would be if we intro
                //value of actual object + valor anterior
                int new = values[i - 1] + m[i - 1][j - weights[i
                //Checks if this new value is greater than the o
                //the object with the greatest value
                if(new > m[i - 1][j]) {
                    m[i][j] = new;
                }
                else {
                    m[i][j] = m[i - 1][j];
                }
            }
            else {
                m[i][j] = m[i - 1][j];
            }
        }
    }
    int remaining_capacity = MAX_CAPACITY;
    for (int i = ngoods, j = MAX_CAPACITY; i > 0; i--) {
        if (m[i][j] != m[i-1][j]) {
            printf("Object %d\twith weight %d\tgives us a value
            j -= weights[i-1];
            remaining_capacity -= weights[i-1];
        }
    }
    printf("Remaining capacity: %d\n", remaining_capacity);
    printf("The price to be paid is %d\n", m[ngoods][MAX_CAPACIT
}

int main() {
    int weights[MAX_CAPACITY], values[MAX_CAPACITY];
    int ngoods;
    printf("Welcome to GREEDY CO. services.\n");
    printf("Please, insert how many goods you want us to carry:
```

```
    scanf("%d", &ngoods);

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

    for (int i = 0; i < ngoods; i++) {
        values[i] = weights[i]*6; // Values are set same as wei
    }
    funcion(weights, values, ngoods);
}
```

Code explanation

Firstly we define the capacity that we can carry. After that the most important function is defined, with some arguments such as weight, value and number of goods.

Inside of it we defined a bidimensional array with one extra space, that will help us to calculate the following computation and check if it is better than the previous computation. This idea is the main one of the designing of a dynamical program.

After defining this idea the code has two fors, using the first one to check the number of goods to be computed and the second one to check the maximum storage. Continuing with the explanation, we found the base case where we check if we have any of the two indexes empty, if not, it checks if the empty space is higher than the object that is going to work with.

If the remaining space is higher than the new object weight's we have to introduce the weight in the new variable, that we will use later to compare which option is better, the new one or the previous. It will store the one with the highest weight.

Complexity of the algorithm

//no es la tabla

First, the nested loops will be calculated first (Initialization of the matrix):

```
for (int i = 0; i <= ngoods; ++i) {
    for (int j = 0; j <= MAX_CAPACITY; ++j) {
        m[i][j] = 0;
    }
}
```

The first “for” is executed ‘ngoods + 1’ times, and the second one ‘MAX_CAPACITY + 1’ times.

Therefore, the computational cost is:

(ngoods + 1) x (MAX_CAPACITY + 1)

Then, the second nested loop will be addressed. It is quite simillar to the first one

(matrix gets filled):

```
for (int i = 0; i <= ngoods; ++i) {
    for (int j = 0; j <= MAX_CAPACITY; ++j) {
        // if-else...
    }
}
```

Anew, the cost is:
(ngoods + 1) x (MAX_CAPACITY + 1)

After that the last for will be dicussed (Object recovery):

```
for (int i = ngoods, j = MAX_CAPACITY; i > 0; i--) {
        if (m[i][j] != m[i-1][j]) {
            printf("Object %d\twith weight %d\tgives us a value
            j -= weights[i-1];
            remaining_capacity -= weights[i-1];
        }
    }
```

If the worst case is taken into consideration, then this loop is executed ‘ngoods’ times, which means: $O(ngoods)$.

Finally, we have ‘main()’

The computational cost is $O(ngoods)$, due to storing a ‘ngoods’ amount of inputs.

To conclude, the total computational cost is:

- Initialization and fill of the matrix = 2 x ((ngoods + 1) x (MAX_CAPACITY + 1))
- Object recovery = $O(ngoods)$
- main() = $O(ngoods)$

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 cero.

```
Welcome to GREEDY CO. services.
Please, insert how many goods you want us to carry:
```

Then, you will need to introduce each product weight, again they need to be greater than 0. And our code will give you the final amount of money to be payed following dynamic algorithm for that service.

```
Welcome to GREEDY CO. services.
Please, insert how many goods you want us to carry:4
 Enter the weights of goods:
10000
3000
2001
```

```
2000
Object 4        with weight 2000        gives us a value of 120
Object 2        with weight 3000        gives us a value of 180
Object 1        with weight 10000       gives us a value of 600
Remaining capacity: 0
The price to be paid is 90000
//10.000 + 3.000 + 2.000 = 15.000 kilos
//15.000 is our limit.
//Each kilo is 6€, 15.000 * 6 = 90.000 euros
```

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.

```
Welcome to GREEDY CO. services.
Please, insert how many goods you want us to carry:5
 Enter the weights of goods:
5000
5000
10
2000
3001
Object 5        with weight 3001        gives us a value of 180
Object 3        with weight 10  gives us a value of 60
Object 2        with weight 5000        gives us a value of 300
Object 1        with weight 5000        gives us a value of 300
Remaining capacity: 1989
The price to be paid is 78066
//5.000 + 5000 + 10 + 3.001 = 13.011 kilos
//15.000 is our limit.
//Each kilo is 6€, 13.011  * 6 = 13.011 euros
```

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

```
Welcome to GREEDY CO. services.
Please, insert how many goods you want us to carry:3
 Enter the weights of goods:
0
-10
-3423
Remaining capacity: 15000
The price to be paid is 0
```

And lastly, here we have another output example where we introduce expected values so we get a expected output.

```
Welcome to GREEDY CO. services.
Please, insert how many goods you want us to carry:5
 Enter the weights of goods:
1
2
3
4
5
Object 5        with weight 5   gives us a value of 30
Object 4        with weight 4   gives us a value of 24
Object 3        with weight 3   gives us a value of 18
Object 2        with weight 2   gives us a value of 12
Object 1        with weight 1   gives us a value of 6
Remaining capacity: 14985
The price to be paid is 90

Process finished with exit code 0

// 1 + 2 + 3 + 4 + 5 = 15 kilos
//Each kilo is 6€, 15  * 6 = 90 euros
```