## Question 1

Le gain apporté par le responsable marketing de *BrainStuffing* pour ce carnet de commande sera de $g(7) + g(0) = 75 + 25 = 100$.

## Question 2

$\forall t, 0 \leq t \leq T, m(0,t) = 0$

## Question 3

$$\forall k \in [\![1,K]\!], \forall t \in [\![0,T]\!], m(k,t) = \begin{array}{lll} \max(m(k-1,t-d(k)) + g(k), m(k-1,t))) & \text{si} & d(k) \leq t \\ m(k-1,t) & \text{sinon} & \end{array}$$

```cpp
#include<iostream>
#include<iomanip>

#include "question4.hpp"
#include "question6.hpp"

std::ostream& operator<<(std::ostream& os, boost::numeric::ublas::matrix<int> mat) { // Fonction permettant l'impression d'une matrice
    int rows = mat.size1(),
        cols = mat.size2();

    for(int row = 0 ; row < rows ; ++row) {
        for(int col = 0 ; col < cols ; ++col) {
            if(col == 0) {
                if(row == 0) {
                    os << "⎡ ";
                } else if(row == rows - 1) {
                    os << "⎣ ";
                } else {
                    os << "⎢ ";
                }
            }

            os << std::setw(4) << mat(row, col) << " ";

            if(col == cols - 1) {
                if(row == 0) {
                    os << "⎤" << std::endl;
                } else if(row == rows - 1) {
                    os << "⎦" << std::endl;
                } else {
                    os << "⎥" << std::endl;
                }
            }
        }
    }

    return os;
}

template<typename T>
std::ostream& operator<<(std::ostream& os, std::vector<T> vec) {
    int vecSize = vec.size();
    for(int i = 0 ; i < vecSize ; ++i) {
        if(i == 0) {
            os << "⎛";
        } else if(i == vecSize - 1) {
            os << "⎝";
        } else {
            os << "⎜";
        }

        os << vec[i];

        if(i == 0) {
            os << "⎞" << std::endl;
        } else if(i == vecSize - 1) {
            os << "⎠" << std::endl;
        } else {
            os << "⎟" << std::endl;
        }
    }
    return os;
}

int main(int argc, char const* argv[]) {
    std::vector<std::tuple<int, int>> carnet(10);
    carnet[0] = std::tuple<int, int>{20, 25};
    carnet[1] = std::tuple<int, int>{20, 25};
    carnet[2] = std::tuple<int, int>{70, 65};
    carnet[3] = std::tuple<int, int>{10, 15};
    carnet[4] = std::tuple<int, int>{10, 5};
    carnet[5] = std::tuple<int, int>{40, 35};
    carnet[6] = std::tuple<int, int>{10, 15};
    carnet[7] = std::tuple<int, int>{80, 75};
    carnet[8] = std::tuple<int, int>{10, 15};
    carnet[9] = std::tuple<int, int>{40, 15};

    boost::numeric::ublas::matrix<int> dynaMat = question4::allonsY(carnet, 100);
    std::vector<int> spots = question6::computeSpots(dynaMat);

    //std::cout << dynaMat << std::endl
              //<< spots << std::endl << std::endl;

    // Question 7
    std::cout << "Sous-ensemble de spots de gain total maximum :" << std::endl
              << spots << std::endl
              << "Gain total maximum :" << std::endl
```

```cpp
#ifndef _QUESTION4_HPP_
#define _QUESTION4_HPP_

#include<boost/numeric/ublas/matrix.hpp>
#include<tuple>

using boost::numeric::ublas::matrix;

namespace question4 {

    matrix<int> allonsY(std::vector<std::tuple<int, int>> carnet, int slotLength);

}

#endif
```

```
#include "question4.hpp"

using boost::numeric::ublas::matrix;

namespace question4 {

    matrix<int> allonsY(std::vector<std::tuple<int, int>> carnet,
                        int slotLength) {
        int spotCount(carnet.size()); // K

        matrix<int> dynaMat(spotCount+1, slotLength+1);

        for(int i = 0 ; i <= slotLength ; ++i) { // M[0,t] = 0
            dynaMat(0, i) = 0;
        }

        for(int i = 1 ; i <= spotCount ; ++i) { // M[i,j]

            int spotDuration, spotProfit;
            std::tie(spotDuration, spotProfit) = carnet[i-1]; // On met le gain du spot et sa durée dans spotProfit et spotDuration

            for(int j = 0 ; j <= slotLength ; j++) {
                if(spotDuration <= j) { // La longeur du spot est ≤ à celle restante
                    dynaMat(i,j) = std::max(dynaMat(i-1,j-spotDuration) + spotProfit,
                            dynaMat(i-1,j)); // On prend le maximum entre M[k-1,t] et M[k-1,t-(durée du spot)] + (Gain du spot)
                } else { // La longeur du spot dépasse celle restante
                    dynaMat(i,j) = dynaMat(i-1,j);
                }
            }
        }

        return dynaMat;
    }

}
```

```
#ifndef _QUESTION5_HPP_
#define _QUESTION5_HPP_

#include<boost/numeric/ublas/matrix.hpp>
#include<vector>

using boost::numeric::ublas::matrix;

namespace question6 {

    std::vector<int> computeSpots(matrix<int> dynaMat);

}

#endif
```

```cpp
#include "question6.hpp"

using boost::numeric::ublas::matrix;

namespace question6 {

    std::vector<int> computeSpots(matrix<int> dynaMat) {
        std::vector<int> spots;
        int row(dynaMat.size1() - 1),
            col(dynaMat.size2() - 1);

        bool goUp = true;

        while(row > 0) {
            if(col == 0) {
                break;
            }

            int nextRow = (goUp) ? row-1 : row,
                nextCol = (!goUp)? col-1 : col;

            if(dynaMat(row, col) > dynaMat(nextRow, nextCol)) {
                if(goUp) {
                    spots.push_back(row - 1);
                }

                // Invert direction and re-establish the next cell
                goUp = not goUp;
                nextRow = (goUp) ? row-1 : row;
                nextCol = (!goUp)? col-1 : col;
            }

            row = nextRow;
            col = nextCol;
        }
        return spots;
    }

}
```