Programming Assignment 3.

Dynamic Programming, *0-1 Knapsack*, *Coin Changes*, *Tree Tower*

Purpose of the exercise

This exercise will help you do the following:

1. Practice developing Dynamic Programming based algorithm to solve problems.
2. Practice implementing Dynamic Programming based algorithm for classical problems and apply it to solve variant problems.

Tasks

1. Charm Bracelet

Bessie has gone to the mall's jewelry store and spies a charm bracelet. Of course, she'd like to fill it with the best charms possible from the `N` (1 ≤ `N` ≤ 3,000) available charms. Each charm `i` in the supplied list has a weight `W_i` (1 ≤`W_i`≤ 300), a 'desirability' factor `D_i` (1 ≤ `D_i` ≤ 100), and can be used at most once. Bessie can only support a charm bracelet whose weight is no more than `M` (1 ≤ `M` ≤ 12,000).

Given that weight limit as a constraint and a list of the charms with their weights and desirability rating, deduce the maximum possible sum of ratings.

Implement an algorithm to solve the problem using **Dynamic Programming** strategy in function `int charm_bracelet(int M, std::vector<int> const& W, std::vector<int> const& D)`. This function takes in `M` the maximum weight the bracelet supports, `W` is the list of weights and `D` is the list of desirability ratings. It returns the maximum possible sum of desirability ratings.

Here are some samples arguments and returned result for validating your code.

> `M=5`, `W=[1,2,3]`, `D=[6,10,12]`, returned result is: 22
>
> `M=6`, `W=[1,2,3,4]`, `D=[4,6,12,7]`, returned result is: 23

More test cases 0-4 can be found in the driver source file - `qdriver.cpp`.

2. Coin changes

Finding the minimum number of coins for change can be a tricky problem. For example, if we require 90 cents as change, using the Singapore coin denominations, we can do it using 9 ten cent coins, or 4 twenty cent coins and 1 ten cent coins, or 1 fifty cent coins with 2 twenty cent coins. The answer is probably 3 coins is the minimum required. In this question, we try to solve the same general problem using dynamic programming.

Suppose we have coins of `k` denominations `0 < c1 < c2 ... < ck`, and a particular `X` cents that we need to make-up using coins, we try to figure out the minimum number of coins required to form `X` cents. Notice that:

1. Suppose we have unlimited number of coins and
2. `c1 == 1`

> Hint: It is a 0/1 knapsack-like problem. The difference is that, in 0/1 knapsack we just decide either choose an item or not; but here we should decide how many coins are needed, the number might be greater than 1.

Let `opt(i,j)` represent the minimum number of coins (with `i` possible values) used to make-up `j` amount. For example, `X`=18, `k`=5, `{c1 c2 c3 c4 c5}= {1 2 5 9 10}`, the algorithm calculate and return `opt(5,18)` - the minimum number of coins used to make-up 18, coins may have 5 different values.

First, consider the last value 10 in the denominations, we have 2 options:

1. not to use 10 valued coin: `opt(5,18) = opt(4,18)`
2. use `n` 10 valued coin: `opt(5,18)=opt(4,18-10*n) + n` (if `18-10*n>=0`)

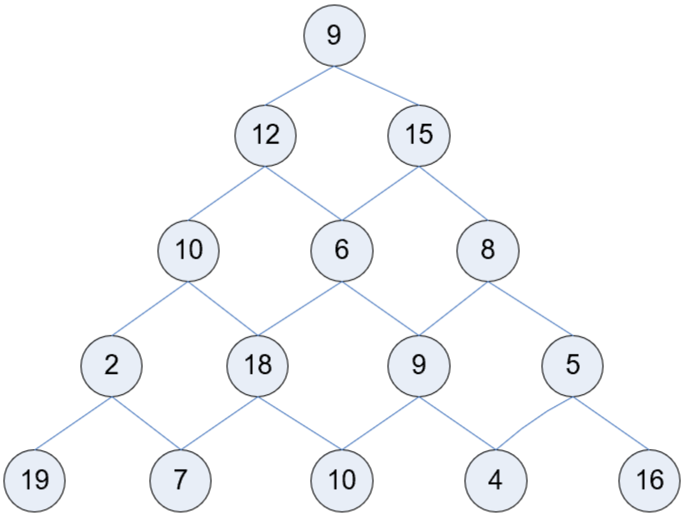
Try to figure out the recurrence of `opt(i,j)` and **initial values**, design an algorithm to solve the coin change problem using **Dynamic Programming** strategy. Implement your algorithm in the function:

```
int coin_changes(int change, std::vector<int> const& denominations);
```

Test cases 5-6 in the driver source file - `qdriver.cpp` are used to test the implementation.

3. Tree Tower

A tree tower is a tower can be represented by a tree. An example of a tree tower is shown below:



Each node in a tree tower has a weight. The goal of the algorithm is to find a path from root node to one of the leaf nods that has maximal total weight.

Data structure

A 2-d array can be used to represent the tower tree.

```
// [5][5]: [#levels][#leaves]
int weight[5][5] = {{9, 0, 0, 0, 0},
                    {12,15,0, 0, 0},
                    {10,6, 8, 0, 0},
                    {2,18, 9, 5, 0},
                    {19,7, 10,4,16}
                   };
```

optimal substructure

The optimal substructure recurrence could be:

$$opt[k,i] = max(opt[k-1, leftparent(i)], opt[k-1, rightparentp(i)]) + weight(k,i)$$

where `k` is the level number and `i` (0..5) is the node code. Take the third node at bottom level (its weight==10) of the tower,

$$opt[4,2] = max(opt[3,1], opt[3,2]) + 10$$

iterative implementation

Reverse the direction, a top-down optimal total weight can be generated from the tower:

9=9				
9+12=21	9+15=24			
10+21=31	max(21,24)+6=30	8+24=32		
2+31 = 33	max(31,30)+18=49	max(30,32)+9=41	5+32=37	
19+33=52	max(33,49)+7=56	max(49,41)+10=59	max(41,37)+4=45	16+37=53

Select the largest value from the last row, 59 is the optimal solution.

Implement your algorithm in the function:

```
int tree_tower(int rows, int cols, int *weights);
```

Test cases 7 in the driver source file - `qdriver.cpp` are used to test the implementation.

4. Compile, run and test code

Compile and link the completed source file *q.cpp* using the required g++ options:

```
g++ -std=c++17 -pedantic-errors -Wall -Wextra -Werror q.cpp qdriver.cpp -o q.out
```

Run the executable with input 0, ..., 7 respectively, and compare the output with the optimal solution given in the file `qdriver.cpp` to validate your code.

Submission

Once your implementation of *q.cpp* is complete, again ensure that the program works and that it contains updated **file-level** documentation comments.

```
/*!*****************************************************************************
\file    q.cpp
\author  ABC
\par     DP email: ABC@digipen.edu
\par     Course: CS330
\par     Section: A
\par     Programming Assignment #3
\date    20-07-2021

\brief

*******************************************************************************/
```

Then upload the file `q.cpp` to the submission page in Moodle and the system will auto-grade your submission.