Optiver

Hackers Rank - 2018 Nov30 *(3 questions in 3 hours, I finished in 2 hours)*

Question 1 : Complete a function that returns the product of two very large unsigned numbers, both input numbers and output number are represented as strings.

```
int ctoi(char c) { return c-'0'; }

std::string add(const std::string& x, const std::string& y) // it can handle zero-x / zero-y / both-zero
{
	std::string z;
	int n = x.size()-1;
	int m = y.size()-1;
	int carry = 0;

	while(n>=0 && m>=0)
	{
		int tmp = ctoi(x[n]) + ctoi(y[m]) + carry;
		z.insert(0, std::to_string(tmp%10)); carry = tmp/10;
		--n; --m;
	}
	while(n>=0)
	{
		int tmp = ctoi(x[n]) + carry;
		z.insert(0, std::to_string(tmp%10)); carry = tmp/10;
		--n;
	}
	while(m>=0)
	{
		int tmp = ctoi(y[n]) + carry;
		z.insert(0, std::to_string(tmp%10)); carry = tmp/10;
		--m;
	}
	if (carry > 0) z.insert(0, std::to_string(carry));
	return z;
}

std::string scale(const std::string& x, int scale, int order) // it can handle zero-x, but special care for zero-scale
{
	if (x == "0" || scale == 0) return "0";

	std::string z;
	int n = x.size()-1;
	int carry = 0;

	while(n>=0)
	{
		int tmp = ctoi(x[n]) * scale + carry;
		z.insert(0, std::to_string(tmp%10)); carry = tmp/10;
		--n;
	}
	if (carry > 0) z.insert(0, std::to_string(carry));
	z.append(order, "0");
	return z;
}

std::string multiply(const std::string& x, const std::string& y)
{
	std::string z = "0";
	for(int n=0; n!=y.size(); ++n) z = add(z, scale(x, ctoi(y[n]), y.size()-n-1));
	return z;
}
```

Question 2 : Given the Morse code of 7 alphabets as the following :

```
O    =    ---                V    =    ...-
P    =    .--.               E    =    .
T    =    -                  R    =    .-.
I    =    ..
```

Write a function that reads a Morse code and returns all possible translations as vector of strings. Please verify solution.

```
void decode_morse(const std::string& code, std::vector<std::string>& out)
{
    out.clear();
    std::vector<std::string> tmp;

    // *** Game tree (hard code cases) *** //
    if       (code.size()>=1 && code.substr(0,1)==".")        { decode_morse(code.substr(1), tmp); reduce('E', tmp, out); }
    else if  (code.size()>=2 && code.substr(0,2)="..")        { decode_morse(code.substr(2), tmp); reduce('I', tmp, out); }
    else if  (code.size()>=3 && code.substr(0,3)=="---")      { decode_morse(code.substr(3), tmp); reduce('O', tmp, out); }
    else if  (code.size()>=4 && code.substr(0,4)==".--.")     { decode_morse(code.substr(4), tmp); reduce('P', tmp, out); }
    else if  (code.size()>=3 && code.substr(0,3)==".-.")      { decode_morse(code.substr(3), tmp); reduce('R', tmp, out); }
    else if  (code.size()>=1 && code.substr(0,1)=="-")        { decode_morse(code.substr(1), tmp); reduce('T', tmp, out); }
    else if  (code.size()>=4 && code.substr(0,4)="...-")      { decode_morse(code.substr(4), tmp); reduce('V', tmp, out); }
    else return;                                              // boundary case [problem solved, return empty output]
}

void reduce(char c, const std::vector<std::string>& tmp, std::vector<std::string>& out)
{
    if (tmp.empty())
    {
        std::string str(1,c);
        out.push_back(str);
    }
    else
    {
        for(const auto& x:tmp)
        {
            std::string str(1,c); str.append(x);
            out.push_back(str);
        }
    }
}
```

Apply to Sudoku

This solution can be applied to finding all Sudoku solutions as well. Please pay attention to :

- the pattern of returning sub-solutions, there is a minor difference in both solutions on ***handling boundary case***
- the pattern of decoupling target string checking, `matrix::is_valid(pos,n)` which check if it is valid to fill `n` in `pos`

```
void solve_sudoku(const matrix& mat, std::vector<matrix>& out)
{
    out.clear();
    std::vector<matrix> tmp;

    std::optional<position> pos = mat.get_next_empty_pixel();
    if (!pos) { out.push_back(mat); return; } // boundary case [problem solved, return original matrix]

    // *** Game tree (for loop) *** //
    for(std::uint8_t n=1; n!=10; ++n)
    {
        if (quest.is_valid(pos,n)) // for optimization, filter useless cases
        {
            matrix new_mat = mat; new_mat.fill(pos,n);
            solve_sudoku(new_mat, tmp);
            reduce(pos, n, tmp, out);
        }
    }
}

void reduce(const position& pos, std::uint8_t n, const std::vector<matrix>& tmp, std::vector<matrix>& out)
{
    if (tmp.empty()) // boundary case [unlike Mose code, tmp is never empty]
    for(const auto& x:tmp)
    {
        matrix mat(x); mat.fill(pos,n);
        out.push_back(mat);
    }
}
```

The pattern for sudoku is not that complicated, however it involves plenty of code to implement the matrix.

Question 3 : Complete function that reads two strings, both of which are in DDMMYYYY format of Darian dates, returns an integer denoting the number of dates between them. Darian calendar is defined as :

- there are 24 months each year
- the first 5 months of each quarter have 28 days
- the final month of each quarter has 27 days, except for leap year, when the final month of the year has 28 days
- each decades has six leap year and four non-leap year
- leap year is a year that is odd or evenly divisible by 10

```
int num_days_in_month(int y, int m)
{
    if (m%6!=0) return 28;
    else if (m!=24) return 27;
    else if (y%2 == 1 || y % 10 == 0) return 28;
    else return 27;
}

int calc_days_between(string start, string end)
{
    int d0 = stoi(start.substr(0,2));
    int m0 = stoi(start.substr(2,2));
    int y0 = stoi(start.substr(4));
    int d1 = stoi(end.substr(0,2));
    int m1 = stoi(end.substr(2,2));
    int y1 = stoi(end.substr(4));

    int count = 0;
    while(!(y0==y1 && m0==m1))
    {
        count += num_days_in_month(y0,m0);

        ++m0;
        if (m0==25) { ++y0; m0=1; }
    }
    count += (d1-d0);
    return count;
}
```