Maven

Codelity - 2019 Nov30 *(3 questions in 3 hours)*

Question 1 Given a vector of integers, count the number of subsequences with zero sum by dynamic programming.

Solution

Given input vector x[1:N], suppose :

$$
\begin{array}{rcll}
g(x[1:N]) & = & \text{number of zero-sum subsequences starting with } x[1] & \textit{reduced problem that is recursive} \\
f(x[1:N]) & = & \text{number of zero-sum subsequences} & \\
& = & \sum_{n\in[1,N]} g(x[n:N]) & \textit{which gives our answer} \\
g(x[1:1]) & = & \delta_{x[1]=0} & \textit{which gives our boundary case}
\end{array}
$$

Making use of the property that $0+0=0$, that is :

$$
\begin{array}{llcll}
\textit{If} & sum(x[1:n]) & = & 0 & \\
& sum(x[1:n']) & = & 0 & \textit{where} \quad n<n' \\
\textit{then} & sum(x[n+1:n']) & = & 0 &
\end{array}
$$

The following recursion on g holds :

$$
\begin{array}{lrcl}
& g(x[1:N]) & = & 1+g(x[n+1:N]) \\
\textit{where} & n & = & \textit{min index such that} \quad sum(x[1:n])=0
\end{array}
$$

```
// Slow method in O(N^2)
int solve_g_subproblem(const std::vector<int>& x, int offset)
{
        int count = 0;
        int sum = 0;
        for(int n=offset; n!=x.size(); ++n)
        {
                sum += x[n];
                if (sum == 0) ++count;
        }
        return count;
}
int f(const std::vector<int>& x)
{
        std::vector<int> g_subproblem(x.size(), 0);
        for(int n=0; n!=x.size(); ++n) g_subproblem[n] = solve_g_subproblem(x, n);
        return std::accumulate(g_subproblem.begin(), g_subproblem.end());
}
```

```
// A little faster method in O(N^2), proved to be equivalent to the above
void solve_g_subproblem(const std::vector<int>& x, int offset, std::vector<int>& g_subproblem)
{
        int sum = 0;
        for(int n=offset; n!=x.size(); ++n)
        {
                sum += x[n];
                if (sum == 0)
                {
                        if (n+1 < x.size())     g_subproblem[offset] = 1 + g_subproblem[n+1];
                        else                    g_subproblem[offset] = 1;
                        return;
                }
        }
}
int f(const std::vector<int> x)
{
        std::vector<int> g_subproblem(x.size(), 0);
        for(int n=x.size()-1; n>=0; --n) solve_g_subproblem(x, n, g_subproblem);
        return std::accumulate(g_subproblem.begin(), g_subproblem.end());
}
```

A generic version of this question is asked by Facebook on 03Apr2020. For optimal solution, read Algorithm3.doc, which involves construction of a set of cumulative sum and some passing-car logic.

Question 2 Merge time of two arrays with size *N* and *M* is *N*+*M*. Given the size of multiple arrays (as a vector of integers), find the minimum merge time using dynamic programming. There is no need to return the merge sequence, for example given three vectors with size *N*, *M*, *K*, there are three merging strategies with different total merge time :

strategy 1	*time to merge 1 with 2*	=	*N+M*
	time to merge 1&2 with 3	=	*(N+M)+K*
	total	=	*2(N+M)+K*
strategy 2	*time to merge 2 with 3*	=	*M+K*
	time to merge 2&3 with 1	=	*(M+K)+N*
	total	=	*2(M+K)+N*
strategy 3	*time to merge 3 with 1*	=	*K+N*
	time to merge 3&1 with 2	=	*(K+N)+M*
	total	=	*2(K+N)+M*

__Solution__

```cpp
int merge_time(const std::vector<int>& sizes)
{
        std::priority_queue<int> q;
        for(auto& x:sizes) q.push(x);

        int output = 0;
        while(q.size()>=2)
        {
                int z0 = q.top(); q.pop();
                int z1 = q.top(); q.pop();
                q.push(z0+z1);
        }
        return output;
}
```

Question 3 In the game of battleship, write a function that returns the number of sunk ships and the number of hit ships given an array of ship corrdinates (one for upper-left and one for lower-right) and an array of hit corrdinates, corrdinates convention is Excel-type and all inputs / ouputs are strings (rather than vector of integers).

```
ships = ”1A 2D, 15D 18D, 21D 24E”
hits = ”1B 4D 21D 22D 4E”
```

space between upper-left and lower-right, comma between two ships

space between two hits

Solution

```
std::string solution(int grid_size, const std::string& ships, const std::string& hits)
{
        std::vector<bool> hit_map(grid_size*grid_size, false);
        int num_sunk = 0;
        int num_hit = 0;

        size_t n=0;
        while(n < hits.size())
        {
                int y, x;
                n = get_coordinate(hits, n, y, x);
                hit_map[y * grid_size + x] = true;
        }

        n = 0;
        while(n < ships.size())
        {
                int y0, x0;
                int y1, x1;
                n = get_coordinate(ships, n, y0, x0);
                n = get_coordinate(ships, n, y1, x1);

                bool is_sunk = true;
                bool is_hit = false;
                for(int y=y0; y<=y1; ++y)
                {
                        for(int x=x0; x<=x1; ++x)
                        {
                                if (hit_map[y * grid_size + x]) is_hit = true;
                                else is_sunk = false;
                        }
                }

                if (is_sunk) ++num_sunk;
                else if (is_hit) ++num_hit;
        }

        std::stringstream ss;
        ss << “num_sunk = “ << num_sunk << “ num_hit = “ << num_hit;
        return ss.str();
}

size_t get_coordinate(const std::string& str, size_t start_pos, int& y, int& x)
{
        size_t pos = str.find_first_of(” ,”, start_pos);

        if (pos != std::string::npos)
        {
                auto s = str.substr(start_pos, pos-1-start_pos);
                y = std::stoi(s)-1;
                x = str[pos-1] - ‘A’;
                return pos+1;
        }
        else
        {
                auto s = str.substr(start_pos, str.size()-1-start_pos);
                y = std::stoi(s)-1;
                x = str[str.size()-1] - ‘A’;
                return str.size();
        }
}
```