LC 53. Maximum Subarray

```
class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: An integer indicate the sum of max subarray
     */
    int maxSubArray(std::vector<int> &nums) {
        // assume nonempty array
        int dp0 = nums.front(), dp1 = nums.front();

        for (auto i = nums.cbegin() + 1; i != nums.cend(); ++i) {
            dp0 = std::max(dp0, dp1);
            if (dp1 > 0) dp1 += *i;
            else         dp1  = *i;
        }

        return std::max(dp0, dp1);
    }
};
```

LC 746. Min Cost Climbing Stairs

```
class Solution {
public:
  int minCostClimbingStairs(std::vector<int>& cost) {
    int cost0 = cost[0], cost1 = cost[1];

    for (std::size_t i = 2; i != cost.size(); ++i)
    {
      int cost2 = std::min(cost0, cost1) + cost[i];
      cost0 = cost1;
      cost1 = cost2;
    }

    return std::min(cost0, cost1);
  }
};
```

LC 213. House Robber II

```
class Solution {
public:
    int rob(std::vector<int>& nums) {
        // Corner cases
        if      (nums.empty())     return 0;
        else if (nums.size() == 1) return nums[0];
        else return std::max(dp(nums, 0, nums.size() - 1),
                             dp(nums, 1, nums.size()    ));
    }

    int dp(const std::vector<int>& nums,
              std::size_t lo,
              std::size_t hi) {

        int dp0 = 0, dp1 = 0, dp2 = nums[lo];

        for (auto i = lo + 1; i != hi; ++i) {
            dp0 = dp1;
            dp1 = dp2;
            dp2 = std::max(dp1, dp0 + nums[i]);
        }

        return dp2;
    }
};
```

LC 1143. Longest Common Subsequence

```
class Solution {
public:
    /**
     * @param A: A string
     * @param B: A string
     * @return: The length of longest common subsequence of A and B
     */
    int longestCommonSubsequence(std::string text1, std::string text2) {
        if (text2.size() < text1.size()) return lCS(text1, text2);
        else                             return lCS(text2, text1);
    }

    int lCS(const std::string& t1, const std::string& t2) {
        // Corner case
        if (t2.empty()) return 0;

        std::vector<int> dp0(t2.size() + 1, 0),
                         dp1(t2.size() + 1, 0);

        return dp(t1, t2, dp0, dp1, 0);
    }

    int dp(const std::string& t1,
           const std::string& t2,
                 std::vector<int>& src,
                 std::vector<int>& dst,
                 std::size_t i) {
        // Base case
        if (i == t1.size()) return src.back();

        for (std::size_t j = 1; j <= t2.size(); ++j) {
            if (t1[i] == t2[j - 1]) dst[j] = src[j - 1] + 1;
            else                    dst[j] = std::max(src[j], dst[j - 1]);
        }

        return dp(t1, t2, dst, src, i + 1);
    }
};
```

LC 1092. Shortest Common Supersequence

```
class Solution {
public:
   std::string shortestCommonSupersequence(std::string str1,
                                           std::string str2) {

      if (str2.size() < str1.size()) return sCS(str1, str2);
      else                          return sCS(str2, str1);
   }

   std::string sCS(const std::string& s1, const std::string& s2) {
      // Corner case
      if (s2.empty()) return s1;

      const auto lcs = lCS(s1, s2);
      std::string scs{};
      auto i = s1.cbegin(), j = s2.cbegin();

      for (auto k = lcs.cbegin(); k != lcs.cend(); ++k) {
         while (*i != *k) scs += *i++; ++i;
         while (*j != *k) scs += *j++; ++j;

         scs += *k;
      }

      scs.append(i, s1.cend());
      scs.append(j, s2.cend());

      return scs;
   }

   std::string lCS(const std::string& s1, const std::string& s2) {

      std::vector<std::string> dp0(s2.size() + 1),
                               dp1(s2.size() + 1);

      return dp(s1, s2, dp0, dp1, 0);
   }

   std::string dp(const std::string& s1,
               const std::string& s2,
                  std::vector<std::string>& src,
                  std::vector<std::string>& dst, std::size_t i) {
```

```
        // Base case
        if (i == s1.size()) return src.back();

        for (std::size_t j = 1; j <= s2.size(); ++j) {
            if (s1[i] == s2[j - 1])
                dst[j] = src[j - 1] + s1[i];
            else
                dst[j] = src[j].size() > dst[j - 1].size() ? src[j] : dst[j - 1];
        }

        return dp(s1, s2, dst, src, i + 1);
    }
};
```