1. 2 Add Two Numbers
   Problem.

```
class Solution {
public:
  ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    int s = 0;
    ListNode* ans = new ListNode;
    ListNode* ansTmp = ans;

    while(l1 !=nullptr || l2 !=nullptr){
      ListNode* node = new ListNode;
      ansTmp->next = node;
      ansTmp = node;

      int tmp = 0;
      if(l1 !=nullptr && l2 !=nullptr){
        tmp = l1->val + l2->val + s;
        l1 = l1->next;
        l2 = l2->next;
      }
      else if(l1 !=nullptr){
        tmp = l1->val + s;
        l1 = l1->next;
      }
      else{
        tmp = l2->val + s;
        l2 = l2->next;
      }

      s = 0;

      if(tmp >= 10){
        s = 1;
        tmp -= 10;
      }

      ansTmp->val = tmp;
    }

    if(s==1){
      ListNode* node = new ListNode;
      ansTmp->next = node;
      ansTmp = node;
      ansTmp->val = 1;
    }

    ans = ans->next;
    return ans;
  }
};
```

2. 5 Longest Palindromic Substring
   Problem.
   (input) $\boldsymbol{a} \quad ; a_i \in \mathbb{Z}$

$$
\begin{aligned}
\max_{\boldsymbol{x} \in \mathbb{Z}^{1:\dim(\boldsymbol{a})}} \quad & n \\
s.t. \quad & n = \dim(\boldsymbol{x}) \\
& x_i = a_{s+i} \quad ; i = 1:n && \text{(子序列约束)} \\
& x_i = x_{n-i} \quad ; i = 1:n && \text{(回文约束)}
\end{aligned}
$$

Algorithm.

- 动态规划

$$\begin{aligned} f\left(s,e\right) &= \begin{cases} f\left(s-1,e+1\right)+2 & f\left(s,e\right)>0 \ and \ a_{s-1}=a_{e-1} \\ 0 & other. \end{cases} \\ f\left(s,s\right) &= 1 \\ f\left(s,s+1\right) &= 2 \quad ;a_s = a_{s+1} \end{aligned} \tag{初始易知值}$$

- $f\left(\right)$: $a_{s:e}$ 的回文字数, 不是回文序列则为 0.

```
class Solution {
public:
  string longestPalindrome(string s) {
    int n = s.length();

    int f[n][n];

    for (int i = 0; i < n; i++) {
      for (int j = i; j < n; j++) {
        f[i][j] = 0;
      }
    }

    for (int i = 0; i < n; i++) {
      f[i][i] = 1;

      if (i != n - 1 && s[i] == s[i + 1]) {
        f[i][i + 1] = 2;
      }
    }

    for (int i = 0; i < n; i++) {
      int k = 1;
      while (i - k >= 0 && i + k < n && s[i - k] == s[i + k]) {
        f[i - k][i + k] = f[i - k + 1][i + k - 1] + 2;
        k++;
      }
    }

    for (int i = 0; i < n - 1; i++) {
      int k = 1;
      while (s[i] == s[i + 1] && i - k >= 0 && i + 1 + k < n && s[i - k] == s[i + 1
          + k]) {
        f[i - k][i + 1 + k] = f[i - k + 1][i + 1 + k - 1] + 2;
        k++;
      }
    }

    int max = 0, I = 0;
    for (int i = 0; i < n; i++) {
      for (int j = i; j < n; j++) {
        if (max < f[i][j]) {
          max = f[i][j];
          I = i;
        }
      }
    }

    return s.substr(I, max);

  }
};
```

3. 22 Generate Parentheses

Problem.

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

Input: n = 3

Output: [”((()))”,”(()())”,”(())()”,”()(())”,”()()()”]

```
class Solution {
public:
  vector<string> generateParenthesis(int n) {
    vector<string> ans;

    string a = magenta"magenta";
    for(int j = 0;j<n;j++){
      a += magenta"magenta(magenta";
    }

    ans.push_back(a);

    for(int i=0;i<n-1;i++){
      int m = ans.size();

      for(int j=0;j<m;j++){
        string b = ans[j].substr(0, i + (ans[j].length() - n) + 1);
        int kn = n + i + 1- ans[j].length();

        for(int k = 0;k< kn;k++){
          b += magenta'magenta)magenta';
          ans.push_back(
            b + ans[j].substr(i + (ans[j].length() - n) + 1, ans[j].length() - (i +
                (ans[j].length() - n) + 1))
          );
        }
      }
    }

    int m = ans.size();

    for (int j = 0; j < m; j++) {
      for (int i = ans[j].length(); i < 2 * n; i++) {
        ans[j] += magenta'magenta)magenta';
      }
    }

    for (int j = 0; j < m; j++) {
      for (int i = j + 1; i < m; i++) {

        if (ans[i] == ans[j]) {
          ans.erase(ans.begin() + i);
          i--; m--;
        }
      }
    }

    return ans;

  }
};
```

4. 39 Combination Sum

Problem.

(input) $\boldsymbol{a} \in mathbbZ_+^n, b \in mathbbZ_+ \quad ; a_i \leq b$

$$\boldsymbol{x}^T \boldsymbol{a} = b$$
$$\boldsymbol{x} \in \mathbb{N}^n$$

Algorithm.

```
class Solution {
public:
  int dot(vector<int>& a, vector<int>& b) {
    int n = a.size(), ans = 0;

    for (int i = 0; i < n; i++) {
      ans += a[i] * b[i];
    }

    return ans;
  }

  vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> ans;
    vector<int> x;

    int n = candidates.size();
    for (int i = 0; i < n; i++) {
      x.push_back(0);
    }

    fun(0, x, candidates, target, ans);

    return ans;
  }

  void fun(int ind, vector<int>& x, vector<int>& a, int target, vector<vector<int>>&
      ans) {
    int n = a.size();

    if (ind == n - 1) {

      while (1) {
        int t = dot(x, a);

        for (int i = 0; i < x.size(); i++) {
          printf(magenta"magenta%magentadmagenta magenta", x[i]);
        }printf(magenta"magenta>%magentadmagenta magenta\magentanmagenta", t);

        if (t > target) {
          x[ind] = 0;
          return;
        }

        else if (t == target) {
          getAns(x, a, ans);

          x[ind] = 0;
          return;
        }

        x[ind]++;
      }
    }

    while (1) {
```

```
      fun(ind + 1, x, a, target, ans);

      x[ind]++;

      int t = dot(x, a);
      if (t > target) {
        x[ind] = 0;
        return;
      }
    }

    x[ind] = 0;
    return;
  }

  void getAns(vector<int>& x, vector<int>& a, vector<vector<int>>& ans) {
    vector<int> t;
    int  n = a.size();

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < x[i]; j++) {
        t.push_back(a[i]);
      }
    }
    ans.push_back(t);
  }
};
```

5. 45 Jump Game II

Problem.

(input) $\boldsymbol{a} \quad ; a_i \in \mathbb{N}$

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{N}^{1:(\dim \boldsymbol{a}-1)}} \quad & \dim \boldsymbol{x} \\ s.t. \quad & x_i \leq a\left(\sum_{k=0}^{i-1} x_k\right) \\ & \mathbf{1}^T \boldsymbol{x} = \dim \boldsymbol{a} - 1 \end{aligned}$$

Algorithm.

- 动态规划

$$\begin{aligned} f(i) &= \min\left\{ f(i-k) + 1 \middle| k \in 1 : \min(a_i, i), f(i-k) > 0 \ when \ i-k > 0 \right\} \\ => f(i+k) &\leftarrow \min\left(f(i+k), f(i) + 1\right) \quad ; k \in 1 : \min(a_i, \dim \boldsymbol{a} - i) \\ f(0) &= 0 && \text{(初始易知值)} \\ f(\dim \boldsymbol{a}) &= \dim \boldsymbol{x}^* && \text{(答案)} \end{aligned}$$

- $f(k)$: 当 $\mathbf{1}^T \boldsymbol{x} = i$ 时的优化问题的解, 无解时为 0.

```
class Solution {
public:
  int jump(vector<int>& nums) {
    int n = nums.size();
    int f[n];

    for (int i = 0; i < n; i++) {
      f[i] = 0xFFFFFFF;
```

```
    }
    f[0] = 0;

    for (int i = 0; i < n - 1; i++) {
      for (int k = 1; k <= nums[i]; k++) {
        if (i + k >= n)
          break;

        f[i + k] = min(f[i + k], f[i] + 1);
      }
    }

    return f[n - 1];
  }
};
```

6. 54

```
class Solution {
public:
  vector<int> spiralOrder(vector<vector<int>>& matrix) {
    vector<int> ans;

    int m = matrix.size(), n = matrix[0].size();

    int x = 0, y = 0, dimX = 0, dimY = 1;

    for(int i=0;i<m*n;i++){
      ans.push_back(matrix[x][y]);
      matrix[x][y] = 0xFFFFFFF;

      if(dimY > 0 && (y + dimY == n || matrix[x][y + dimY] == 0xFFFFFFF)){
        dimX = 1;
        dimY = 0;
      }

      else if(dimY < 0 && (y + dimY == -1 || matrix[x][y + dimY] == 0xFFFFFFF)){
        dimX = -1;
        dimY = 0;
      }

      else if(dimX > 0 && (x + dimX == m || matrix[x + dimX][y] == 0xFFFFFFF)){
        dimX = 0;
        dimY = -1;
      }

      else if(dimX < 0 && (x + dimX == -1 || matrix[x + dimX][y] == 0xFFFFFFF)){
        dimX = 0;
        dimY = 1;
      }

      x += dimX;
      y += dimY;
    }

    return ans;
  }
};
```

7. 62 Unique Paths

Problem.

(input) $x_0, y_0 \in \mathbb{N}$

方块图，$(0,0) \to (x_0, y_0)$，只能走右、下的所有路径数.

Algorithm.

- 动态规划

$$
\begin{aligned}
&f(x,y) = f(x-1,y) + f(x,y-1) \\
&f(0,y) = 1 && \text{(初始易知值)} \\
&f(x,0) = 1
\end{aligned}
$$

- $f(x,y)$: 位置 x,y 处的路径数.

```cpp
class Solution {
public:
  int uniquePaths(int m, int n) {
    int f[m][n];

    for(int i = 0;i<m;i++){
      f[i][0] = 1;
    }

    for(int i = 0;i<n;i++){
      f[0][i] = 1;
    }

    for(int i = 1;i<m;i++){
      for(int j=1;j<n;j++){
        f[i][j] = f[i][j-1] + f[i-1][j];
      }
    }

    return f[m-1][n-1];
  }
};
```

8. 145

```cpp
class Solution {
public:
  bool is(int* a) {
    int t = a[0] * a[4] * a[8] + a[1] * a[5] * a[6] + a[2] * a[3] * a[7]
        - a[2] * a[4] * a[6] - a[1] * a[3] * a[8] - a[0] * a[5] * a[7];
    return t == 0 ? 1 : 0;
  }

  int maxPoints(vector<vector<int>>& points) {
    int n = points.size(), ans = 0;
    if(n == 1)
      return 1;
    if(n == 2)
      return 2;

    int a[9];
    a[2] = a[5] = a[8] = 1;

    for (int i = 0; i < n; i++) {
      a[0] = points[i][0];
      a[1] = points[i][1];

      for (int j = i + 1; j < n; j++) {
        a[3] = points[j][0];
        a[4] = points[j][1];

        int num = 2;

        for (int k = 0; k < n; k++) {

          if (k == i || k == j)
            continue;
```

```
            a[6] = points[k][0];
            a[7] = points[k][1];

            if (is(a))
              num++;
          }

          ans = max(num, ans);
        }
      }

      return ans;
    }
};
```

9. 233 Number of Digit One
   Problem.
   (input) $n \in \mathbb{N}$
   求所有比 n 小 (包括 n) 的正整数中，含数字 1 的个数.
   Property.
   - 第 i 出现 1 的现象, 是一个周期 10^{i+1} 宽 10^i 高 1 的矩形波.
     第 i 位 1 的矩形位于整个周期的第 $10^i+1 \sim 2\times 10^i$ 个数上,

$$\begin{pmatrix} \text{数位 i} & \text{周期} & \text{波峰宽度} & \text{波峰高度} \\ 0 & 10 & 1 & 1 \\ 1 & 100 & 10 & 1 \\ 2 & 1000 & 100 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ i & 10^{i+1} & 10^i & 1 \end{pmatrix}$$

   eg.

$$\begin{array}{lr} 0\quad 1\quad 2\quad 3\quad 4\quad 5\quad 6\quad 7\quad 8\quad 9 & (\text{一位 1 在第 2 个}) \\ 0\quad 1\quad ...\quad 10\quad ...\quad 19\quad ...\quad ...\quad ...\quad 99 & (\text{二位 1 在第 } 11-20 \text{ 个}) \\ 0\quad 1\quad ...\quad 100\quad ...\quad 199\quad ...\quad ...\quad ...\quad 999 & (\text{三位 1 在第 } 101-200 \text{ 个}) \end{array}$$

   Algorithm.
   - 将 $0 \sim n$ 的数分解为每一位上的 1 的矩形波, 并求和波峰即可.
   - 步骤
     - 首先, 求完成的周期数, 周期数 $\times$ 波峰宽便是第一部分数字 1 的数量.
     - 然后, 分析正在进行的周期, 求余剩下的数的长度 rest
       * $rest \geq 2\times 10^i$ 时, 波峰部分已完成, 直接加波峰宽度即可.
       * $10^i+1 \leq rest < 2\times 10^i$ 时, 波峰部分正在进行, 加剩余数的数目即可
       * $rest < 10^i+1$ 时, 波峰部分未开始, 不用计算.

```
class Solution {
public:
  long long countDigitOne(long long n) {
    long long ans = 0;

    long long nt = n, dim = 0;
    while (nt != 0) {
      dim++;
```

```
      nt /= 10;
    }

    for (long long i = 0; i < dim; i++) {
red       red//red red完red成red的red周red期
      long long tmp = (n + 1) / (long long)pow(10, i + 1) * pow(10, i);

red       red//red red正red在red进red行red的red周red期
      long long rest = (n + 1) % (long long)pow(10, i + 1),
        tmp2 = rest / pow(10, i);

      if (tmp2 > 1) { red     red//red red完red成red一red个red波red峰
        tmp += pow(10, i);
      }
      else if (tmp2 == 1) { red red//red red正red在red完red成red一red个red波red峰
        tmp += rest % (long long)pow(10, i);
      }
red                    red//red red未red完red成red一red个red波red峰red red(red不red用red计red算red)
      ans += tmp;
    }

    return ans;
  }
};
```

10. 322

Problem.

(input) $\boldsymbol{a}, b$; $a_i, b \in \mathbb{N}$

$$\begin{aligned} \min_{x} \quad & \|\boldsymbol{x}\|_1 = \mathbf{1}^T \boldsymbol{x} \\ s.t. \quad & \boldsymbol{a}^T \boldsymbol{x} = b \\ & x_i \in \mathbb{N} \end{aligned}$$

Algorithm.

- 动态规划

$$f(k) = \begin{cases} f(k - a_i) + 1 & f(k - a_i) > 0 \ or \ k - a_i = 0 \\ 0 & other \end{cases} \qquad k \in 1 : \dim(\boldsymbol{a}_i)$$

$$f(0) = 0 \qquad \text{(初始易知值)}$$

$$f(b) = \min_{x} \|\boldsymbol{x}\|_1 \qquad \text{(答案)}$$

- $f(k)$: 当 $b \leftarrow k$ 时的优化问题的解, 无解时为 0.

```
class Solution {
  public:
    int coinChange(vector<int>& coins, int amount) {
      if (amount == 0)
        return 0;

      sort(coins.begin(), coins.end());

      int table[amount + 1];
      table[0] = 0;
      for (int i = 1; i < amount + 1; i++) {
        table[i] = 0xFFFFFFF;
```

```
        }

        int size = coins.size();

        for (int i = 1; i <= amount; i++) {
          for (int j = 0; j < size; j++) {
            if (coins[j] > i)
              break;

            int tmp = table[i - coins[j]] + 1;

            table[i] = table[i] < tmp ? table[i] : tmp;
          }
        }

        return table[amount] == 0xFFFFFFF ? -1 : table[amount];
      }
    };
```

end