**Bt ngoài**

Tên : Trịnh Tâm Như

Mssv: 3123411215

**1.Two Sum**

1. **Ý tưởng giải quyết**Cách đầu tiên cũng là cách thông thường, chúng ta sẽ sử dụng thuật toán Brute Force (thuật toán vét cạn) để kiểm tra tổng mỗi cặp phần tử trong mảng nums xem chúng có bằng target không? Nếu bằng nhau thì trả về mảng chứa hai ví trị index của chúng.
2. **Ví dụ minh họa**Input:  
   nums = [2, 7, 11, 15]  
   target = 9

Output:  
[0, 1]

Giải thích:

Trong mảng nums, nums[0] = 2 và nums[1] = 7.

Tổng của 2 và 7 là 9, nên chỉ số của hai số này là [0, 1].

1. **Code**

class Solution {

public int[] twoSum(int[] nums, int target) {

int[] res = new int[2];

for (int i = 0; i < nums.length - 1; i++) {

for (int j = i + 1; j < nums.length; j++) {

if (nums[i] + nums[j] == target){

res[0] = i;

res[1] = j;

}

}

}

return res;

}

}

1. **Kết quả chạy code**A screenshot of a test results

   AI-generated content may be incorrect. **2.Add Two Number  
   a) ý tưởng thuật toán**Cộng từng cặp chữ số của hai danh sách. Nếu một danh sách kết thúc trước, coi các chữ số còn lại của danh sách còn lại là 0.Sau khi cộng hai chữ số, nếu tổng vượt quá 9 (tức là có nhớ), cần giữ lại phần nhớ để cộng vào chữ số tiếp theo.

Nếu sau khi cộng hết các chữ số của cả hai danh sách mà còn nhớ, sẽ phải tạo một nút mới để lưu chữ số đó.

**b) ví dụ minh họa**

Input:

* l1 = [2, 4, 3] (tương đương với số 342)
* l2 = [5, 6, 4] (tương đương với số 465)

Output:

* [7, 0, 8] (tương đương với số 807)

Giải thích:

* Cộng từ chữ số ít nhất:
  + 2 + 5 = 7 (không có nhớ)
  + 4 + 6 = 10, viết 0, nhớ 1.
  + 3 + 4 + 1 (nhớ) = 8.

Kết quả là [7, 0, 8] (tương đương với 807).  
  
**c) Code**  
class Solution {

 public:

  ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2) {

    ListNode dummy(0);

    ListNode\* curr = &dummy;

    int carry = 0;

    while (l1 || l2 || carry) {

      if (l1 != nullptr) {

        carry += l1->val;

        l1 = l1->next;

      }

      if (l2 != nullptr) {

        carry += l2->val;

        l2 = l2->next;

      }

      curr->next = new ListNode(carry % 10);

      carry /= 10;

      curr = curr->next;

    }

    return dummy.next;

  }

};  
**d) Kết quả chạy code**

A screenshot of a phone

AI-generated content may be incorrect.  
  
**3. Longest Substring Without Repeating Characters  
a) ý tưởng thuật toán**tìm kiếm độ dài của chuỗi con dài nhất trong một chuỗi cho trước mà không chứa ký tự lặp lại.

**Khởi tạo hai con trỏ**: start và end đều chỉ vào vị trí đầu của chuỗi.

**Duyệt qua chuỗi**: Mỗi lần di chuyển con trỏ end, kiểm tra nếu ký tự đó đã có trong cửa sổ hiện tại (dùng một set hoặc dictionary để theo dõi).

**Cập nhật con trỏ start**: Nếu ký tự lặp lại, dịch start về phía trước cho đến khi cửa sổ lại hợp lệ.

**Lưu độ dài lớn nhất**: Mỗi lần end di chuyển, tính toán độ dài của cửa sổ và lưu lại giá trị lớn nhất.

**b) Ví dụ minh họa  
Input**:  
"abcabcbb"

**Output**:  
3

**Giải thích**:

* Chuỗi con dài nhất không chứa ký tự lặp lại là "abc", có độ dài là 3.

**c) Code**class Solution {

 public:

  int lengthOfLongestSubstring(string s) {

    int ans = 0;

    vector<int> count(128);

    for (int l = 0, r = 0; r < s.length(); ++r) {

      ++count[s[r]];

      while (count[s[r]] > 1)

        --count[s[l++]];

      ans = max(ans, r - l + 1);

    }

    return ans;

  }

};  
**d) Kết quả**  
A screenshot of a computer

AI-generated content may be incorrect.

**9. Palindrome Number**

**a) Ý tưởng thuật toán**kiểm tra xem một số nguyên có phải là số đối xứng (palindrome) hay không. Một số nguyên được gọi là palindrome nếu nó đọc từ trái sang phải và từ phải sang trái đều giống nhau.

**Chuyển số về dạng chuỗi**: Một cách đơn giản là chuyển số thành chuỗi và so sánh chuỗi đó với chuỗi đảo ngược của nó.

**Kiểm tra số âm**: Nếu số là âm, ngay lập tức trả về False vì số âm không thể là số đối xứng.

**Kiểm tra số có chữ số 0**: Nếu số có chữ số 0 và không phải là 0, thì nó không thể là số đối xứng (ví dụ: 10 không phải palindrome).

**b) Ví dụ minh họa**

Input:

* x = 121

Output:

* True (121 là số đối xứng)

Input:

* x = -121

Output:

* False (-121 không phải là số đối xứng vì dấu "-" không thể đứng ở cuối)

**c) code**

class Solution {

 public:

  bool isPalindrome(int x) {

    if (x < 0)

      return false;

    long reversed = 0;

    int y = x;

    while (y) {

      reversed = reversed \* 10 + y % 10;

      y /= 10;

    }

    return reversed == x;

  }

};

**d)Kết quả**

**A white rectangular object with black lines

AI-generated content may be incorrect.**

[**10. Regular Expression Matching**](https://leetcode.com/problems/regular-expression-matching/)

**a)Ý tưởng thuật toán**

xác định xem một chuỗi có khớp với một mẫu biểu thức chính quy (regular expression) hay không.

**Khởi tạo bảng DP**: Dùng bảng dp[i][j] để lưu trữ kết quả kiểm tra sự khớp giữa chuỗi con s[0..i-1] và mẫu p[0..j-1].

**Cập nhật bảng DP**:

* Nếu p[j-1] == s[i-1] hoặc p[j-1] == '.', thì dp[i][j] = dp[i-1][j-1].
* Nếu p[j-1] == '\*', có thể xét hai trường hợp:
  + dp[i][j] = dp[i][j-2] (khi bỏ qua phần trước dấu \*).
  + Nếu p[j-2] == s[i-1] hoặc p[j-2] == '.', thì dp[i][j] = dp[i-1][j].

**b) ví dụ minh họa**

Input:

* s = "aa"
* p = "a\*"

Output:

* True

Giải thích: "a\*" có thể khớp với một chuỗi chứa bất kỳ số lượng "a" nào, bao gồm không có "a", nên "aa" khớp với mẫu "a\*".

**c) code**

class Solution {

 public:

  bool isMatch(string s, string p) {

    const int m = s.length();

    const int n = p.length();

    vector<vector<bool>> dp(m + 1, vector<bool>(n + 1));

    dp[0][0] = true;

    auto isMatch = [&](int i, int j) -> bool {

      return j >= 0 && p[j] == '.' || s[i] == p[j];

    };

    for (int j = 0; j < p.length(); ++j)

      if (p[j] == '\*' && dp[0][j - 1])

        dp[0][j + 1] = true;

    for (int i = 0; i < m; ++i)

      for (int j = 0; j < n; ++j)

        if (p[j] == '\*') {

    const bool noRepeat = dp[i + 1][j - 1];

          const bool doRepeat = isMatch(i, j - 1) && dp[i][j + 1];

          dp[i + 1][j + 1] = noRepeat || doRepeat;

        } else if (isMatch(i, j)) {

          dp[i + 1][j + 1] = dp[i][j];

        }

    return dp[m][n];

  }

};

[**12. Integer to Roman**](https://leetcode.com/problems/integer-to-roman/)

**a)Ý tưởng thuật toán**

chuyển một số nguyên thành dạng ký tự La Mã. Trong hệ thống La Mã, mỗi số nguyên được biểu diễn bằng các ký tự từ I, V, X, L, C, D, M, và các ký tự này có giá trị cố định. Ngoài ra, một số giá trị còn có các quy tắc đặc biệt (như "IV" cho 4, "IX" cho 9, "XL" cho 40, v.v.).

**Danh sách các ký tự La Mã**: Ta cần tạo một danh sách các giá trị La Mã từ lớn đến nhỏ cùng với các ký tự tương ứng (ví dụ: 1000 → "M", 900 → "CM", 500 → "D",...).

**Duyệt qua các giá trị**: Duyệt qua danh sách giá trị và ký tự, trừ đi giá trị của mỗi phần tử vào số num cho đến khi num giảm xuống 0. Mỗi lần trừ đi, thêm ký tự tương ứng vào chuỗi kết quả.

**Tiếp tục cho đến khi hoàn thành**: Duyệt qua tất cả các giá trị từ lớn đến nhỏ để có thể biểu diễn số nguyên dưới dạng La Mã.

**b) Ví dụ minh họa**

Input:  
num = 58

Output:  
"LVIII"

Giải thích:

* 58 có thể được phân tách thành 50 (L) và 8 (VIII), nên kết quả là "LVIII".

**c) code**

class Solution {

public:

string intToRoman(int num) {

const vector<pair<int, string>> valueSymbols{

{1000, "M"}, {900, "CM"}, {500, "D"}, {400, "CD"}, {100, "C"},

{90, "XC"}, {50, "L"}, {40, "XL"}, {10, "X"}, {9, "IX"},

{5, "V"}, {4, "IV"}, {1, "I"}};

string ans;

for (const auto& [value, symbol] : valueSymbols) {

if (num == 0)

break;

while (num >= value) {

num -= value;

ans += symbol;

}

}

return ans;

}

};

**d)Kết quả**

**A white background with text

AI-generated content may be incorrect.**

[**13. Roman to Integer**](https://leetcode.com/problems/roman-to-integer/)

**a)Ý tưởng bài toán**

Chuyển đổi một chuỗi ký tự La Mã thành số nguyên. Các ký tự La Mã như I, V, X, L, C, D, M có các giá trị cố định, và có các quy tắc đặc biệt khi các ký tự nhỏ hơn đứng trước ký tự lớn hơn.

**Duyệt từ trái sang phải**: sẽ duyệt qua từng ký tự trong chuỗi La Mã.

**Kiểm tra các ký tự liền nhau**: Nếu ký tự hiện tại có giá trị lớn hơn hoặc bằng ký tự tiếp theo, ta cộng giá trị của ký tự đó vào kết quả. Nếu không, ta trừ giá trị của ký tự hiện tại (đây là trường hợp đặc biệt, ví dụ như "IV" hay "IX").

**Cộng hoặc trừ các giá trị**: Dựa vào quy tắc này, chúng ta có thể tính giá trị của số La Mã bằng cách cộng hoặc trừ các giá trị tương ứng.

**b) Ví dụ minh họa**

Input:  
"III"

Output:  
3

Giải thích:

* "III" tương đương với 1 + 1 + 1 = 3.

**c) Code**

class Solution {

 public:

  int romanToInt(string s) {

    int ans = 0;

    vector<int> roman(128);

    roman['I'] = 1;

    roman['V'] = 5;

    roman['X'] = 10;

    roman['L'] = 50;

    roman['C'] = 100;

    roman['D'] = 500;

    roman['M'] = 1000;

    for (int i = 0; i + 1 < s.length(); ++i)

      if (roman[s[i]] < roman[s[i + 1]])

        ans -= roman[s[i]];

      else

        ans += roman[s[i]];

    return ans + roman[s.back()];

  }

};

**d)Kết quả**

**A white background with black text

AI-generated content may be incorrect.**

[**14. Longest Common Prefix**](https://leetcode.com/problems/longest-common-prefix/)

**a)Ý tưởng thuật toán**

Tìm chuỗi tiền tố chung dài nhất giữa một tập hợp các chuỗi.

**so sánh từng chuỗi** trong danh sách và tìm tiền tố chung của tất cả các chuỗi.

**Cách làm:**

1. **Sắp xếp danh sách chuỗi**: Một cách đơn giản để tìm tiền tố chung là sắp xếp danh sách chuỗi theo thứ tự từ điển. Sau khi sắp xếp, chuỗi tiền tố chung dài nhất sẽ là tiền tố chung của chuỗi đầu tiên và chuỗi cuối cùng trong danh sách.
2. **So sánh chuỗi đầu tiên và chuỗi cuối cùng**: Sau khi sắp xếp danh sách, chuỗi tiền tố chung của tất cả các chuỗi sẽ là tiền tố chung của chuỗi đầu tiên và chuỗi cuối cùng (bởi vì sau khi sắp xếp, chuỗi đầu tiên và chuỗi cuối cùng sẽ có khả năng có tiền tố chung dài nhất).

**b) Ví dụ minh họa**

Input:

strs = ["flower", "flow", "flight"]

Output:

"fl"

Giải thích:

* Tiền tố chung dài nhất của các chuỗi "flower", "flow", "flight" là "fl".

**c) Code**

class Solution {

 public:

  string longestCommonPrefix(vector<string>& strs) {

    if (strs.empty())

      return "";

    for (int i = 0; i < strs[0].length(); ++i)

      for (int j = 1; j < strs.size(); ++j)

        if (i == strs[j].length() || strs[j][i] != strs[0][i])

          return strs[0].substr(0, i);

    return strs[0];

  }

};

**d)Kết quả thuật toán**

**A white background with black and white stripes

AI-generated content may be incorrect.**

[**20. Valid Parentheses**](https://leetcode.com/problems/valid-parentheses/)

**a)Ý tưởng thuật toán**

Kiểm tra tính hợp lệ của một chuỗi chứa các dấu ngoặc đơn (), ngoặc vuông [], và ngoặc nhọn {}. Chuỗi có tính hợp lệ nếu các dấu ngoặc đóng và mở phải khớp đúng theo cặp.

Một cách hiệu quả để giải quyết bài toán này là sử dụng **stack (ngăn xếp)**:

1. **Duyệt qua từng ký tự** trong chuỗi:
   * Nếu ký tự là một dấu ngoặc mở ('(', '{', '['), push nó vào stack.
   * Nếu ký tự là một dấu ngoặc đóng (')', '}', ']'), kiểm tra xem stack có trống không và xem phần tử trên cùng của stack có phải là dấu ngoặc mở tương ứng không. Nếu đúng, pop phần tử đó khỏi stack. Nếu không, chuỗi không hợp lệ.
2. **Kết thúc kiểm tra**:
   * Nếu stack trống, nghĩa là tất cả các dấu ngoặc mở đã có dấu ngoặc đóng tương ứng, chuỗi hợp lệ.
   * Nếu stack không trống, nghĩa là còn dấu ngoặc mở không có dấu ngoặc đóng tương ứng, chuỗi không hợp lệ.

**b) Ví dụ minh họa**

**Input**:

s = "()"

**Output**:

True

**Giải thích**:

* Chuỗi "()" hợp lệ vì có một cặp ngoặc mở và ngoặc đóng tương ứng.

**c) Code**

class Solution {

 public:

  bool isValid(string s) {

    stack<char> stack;

    for (const char c : s)

      if (c == '(')

        stack.push(')');

      else if (c == '{')

        stack.push('}');

      else if (c == '[')

        stack.push(']');

      else if (stack.empty() || pop(stack) != c)

        return false;

    return stack.empty();

  }

 private:

  int pop(stack<char>& stack) {

    const int c = stack.top();

    stack.pop();

    return c;

  }

};

**d)Kết quả**

**A white background with black lines

AI-generated content may be incorrect.**

[**21. Merge Two Sorted Lists**](https://leetcode.com/problems/merge-two-sorted-lists/)

**a)Ý tưởng thuật toán**

Hợp nhất hai danh sách liên kết đã được sắp xếp theo thứ tự tăng dần thành một danh sách liên kết duy nhất cũng được sắp xếp theo thứ tự tăng dần.

Bài toán này có thể giải quyết bằng cách sử dụng phương pháp **trượt qua từng phần tử** của hai danh sách liên kết, và chọn phần tử nhỏ hơn của hai danh sách để đưa vào danh sách kết quả. Việc trượt qua này sẽ tiếp tục cho đến khi một trong hai danh sách kết thúc, sau đó ta nối phần còn lại của danh sách chưa kết thúc vào kết quả.

**b) Ví dụ minh họa**

Input:

l1 = [1, 2, 4]

l2 = [1, 3, 4]

Output:

[1, 1, 2, 3, 4, 4]

Giải thích:

* Hợp nhất hai danh sách [1, 2, 4] và [1, 3, 4] cho kết quả là [1, 1, 2, 3, 4, 4].

**c) code**

class Solution {

 public:

  ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2) {

    if (!list1 || !list2)

      return list1 ? list1 : list2;

    if (list1->val > list2->val)

      swap(list1, list2);

    list1->next = mergeTwoLists(list1->next, list2);

    return list1;

  }

};

**d)Kết quả**

**A screenshot of a computer

AI-generated content may be incorrect.**