VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

1. SỬA BÀI TẬP:

- **Nhóm 4**: Bạn Phan Nhật Tân mặc dù chỉ làm bài Problem 1 nhưng mà bài code của bạn khá ổn. Tuy vậy vẫn có một số phần bạn cần sửa đổi:

  - Mình thấy rằng bạn viết code theo tiếp cận đệ quy có ghi nhớ, và về cơ bản thì logic của bài code của bạn là đúng, nhưng mà bạn cần phải xem xét lại các đặc trưng của bài toán:
    - **Tính chất của bài toán:**
      - Bài toán yêu cầu kiểm tra xem chuỗi `s3` có thể được tạo thành từ sự xem kẽ (interleaving) của 2 chuỗi `s1` và `s2` hay không.
    - **Ràng buộc đầu vào:**
      - Độ dài tổng của `s1` và `s2` phải bằng độ dài của `s3` tức là `s1.lenght() + s2.lenght() = s3.lenght()`
      - Các chuỗi chỉ bao gồm ký tự chữ cái từ bảng chữ cái Tiếng Anh.
    - **Bài toán con:**
      - Bài toán này có thể được chia nhỏ thành các bài toán con:
        - Xác định xem một phần của `s3` (đến vị trí `i+j`) có thể được tạo thành từ `s1` (đến vị trí `i`) và `s2` (đến vị trí `j`)
      - Quyết định tại mỗi bước:
        - Lấy một ký tự từ `s1` hoặc lấy một ký tư từ `s2`
    - **Tính chất tối ưu con:**
      - Kết quả của bài toán tại trạng thái `(i, j)` (đến vị trí `i` trong `s1` và `j` trong `s2`) chỉ phụ thuộc vào trạng thái trước đó:
        - `(i - 1, j)` (nếu lấy ký tự từ `s1`),
        - `(i, j - 1)` (nếu lấy ký tự từ `s2`).
    - **Các bài toán con gối nhau:**
      - Với mỗi cặp `(i, j)`, ta phải tính xem phần chuỗi `s3` đến `i + j` có thể được tạo thành hay không. Điều này dẫn đến nhiều trạng thái giống nhau được tính đi tính lại nếu không sử dụng ghi nhớ (memoization).

    Với những đặc trưng trên thì sẽ tốt hơn nếu chúng ta tiếp cận bài toán với Quy hoạch động.

  - Tuy bạn chọn tiếp cận bài toán với đệ quy có ghi nhớ nhưng bạn lại mắc lỗi khi xử lý sai sót trong memo.

```
19          if (i == m && j == n) {
20              return true;
21          }
22          if (memo[i][j] != -1) {
23              return memo[i][j] == 1;
24          }
25
26          memo[i][j] = 0;
27          int k = i + j;
28          if (i < m && s1[i] == s3[k] && dfs(i + 1, j)) {
29              memo[i][j] = 1;
30          }
31
32          if (!memo[i][j] && j < n && s2[j] == s3[k] && dfs(i, j + 1)) {
33              memo[i][j] = 1;
34          }
```

    Đây là bước kiểm tra trạng thái ghi nhớ trong `memo`. Tuy nhiên khi bạn khai báo như thế này thì chương trình chỉ trả về giá trị trực tiếp `memo[i][j] == 1`. Trong trường hợp ghi nhớ sai trạng thái, kết quả trả về sẽ không chính xác. Trong trường hợp này mình nghĩ một số case của bạn sai là do giá trị trong `memo[i][j]` không được cập nhật đúng, ví dụ có thể là do quên xử lý cả hai nhánh của `dfs`.

  - Mình nghĩ bạn nên sửa để đảm bảo `memo[i][j]` được cập nhật đúng với logic sau:

    `memo[i][j] = dfs(i + 1, j)|| dfs(i, j + 1)`

- Sau khi thử qua một số case đặc biệt, mình cũng nhận ra rằng bạn đã không xử lý kỹ các trường hợp biên. Bạn phải đảm bảo các trường hợp đặc biệt được kiểm tra ngay từ đầu:

```
if (s1.emty()) return s2 == s3;
if (s2.emty()) return s1 == s3;
if (s3.emty()) return s1.emty && s2.emty();
```

- Khi bạn sử dụng đệ quy có thể vướng phải lỗi về hiệu suất khi đệ quy sâu. Nếu chuỗi đầu vào lớn(ví dụ `s1` và `s2` có độ dài gần 100), số lần gọi đệ quy sẽ tăng rất nhanh. Điều này có thể dẫn đến:
  - **Stack overflow** do đệ quy quá nhiều
  - **Hiệu suất kém**, đặc biệt là nếu trạng thái trong `memo` không được lưu chính xác.
- Chỉ số `k = i + j` được sử dụng để truy cập `s3`. Nếu `i + j` vượt phạm vi `s3` sẽ dẫn đến lỗi truy cập ngoài mảng phạm vi. bạn có thể sửa lại code bằng cách cho chỉ số `k` luôn được bảo đảm nằm trong phạm vi hợp lệ bởi điều kiện `if ((m + n) != s3.size())`. Bạn cũng có thể thêm điều kiện kiểm tra: `if (k >= s3.size()) return false;`
- Mình nhận thấy rằng bạn chưa kiểm tra các trường hợp đặc biệt; cụ thể là các trường hợp chuỗi rỗng; kí tự trùng khớp giữa `s1`, `s2`và `s3`, còn các trường hợp đặc biệt khác đã được xử lý.

- Nhóm 11: Bạn Cao Lê Công Thành đã làm Problem 1, Problem 2 nhưng chưa làm Problem 3. Nhưng mà bài code của bạn còn một số việc cần sửa lại:

Problem 1:

- Trong vòng lặp tính giá trị của `dp[i][j]` của bạn:

```
30
31      // Ãiá»n bá°£ng DP
32      for (int i = 1; i <= m; ++i) {
33          for (int j = 1; j <= n; ++j) {
34              dp[i][j] = (dp[i - 1][j] && s1[i - 1] == s3[i + j - 1]) ||
35                         (dp[i][j - 1] && s2[j - 1] == s3[i + j - 1]);
36          }
37      }
38
39      return dp[m][n];
```

  Chỉ số `i + j - 1` được sử dụng để truy cập chuỗi `s3`. Nếu `i + j - 1 >= s3.size()`, chương trình sẽ cố gắng truy cập ngoài phạm vi mảng, gây lỗi. Mình thấy điều kiện `if (m + n != s3.size())` ở đầu hàm `isInterleave` đã giúp tránh truy cập ngoài phạm vi của `s3` nếu tổng độ dài của `s1` và `s2` không bằng độ dài `s3` nhưng trong vòng lặp DP, chỉ số `i + j - 1` vẫn có thể gây lỗi nếu không kiểm tra cẩn thận logic.
- Bạn đã không kiểm tra cẩn thận trường hợp đặc biệt chuỗi rỗng. Nếu `s1` hoặc `s2` là chuỗi rỗng, chương trình vẫn tiếp tục xử lý với mảng DP và có thể dẫn đến lỗi logic, bạn cũng chưa có kiểm tra riêng cho các trường hợp đặc biệt:
  - `s1 = ""` và `s2 = s2`.
  - `s2 = ""` và `s3 = s1`.
  - `s3 = ""`.
- Khi kiểm tra qua, mình thấy bạn đã khởi tạo logic đúng, nhưng cần đảm bảo rằng vòng lặp tính toán DP không vi phạm điều kiện chỉ số hoặc logic khớp ký tự.
- Mình cũng thấy rằng bạn đã không tối ưu hóa không gian, điều này không tối ưu về mặt bộ nhớ.

Problem 2:

- Sau khi kiểm tra code của bạn, mình nhận thấy rằng bạn đã không kiểm soát điều kiện cạnh biên đúng cách. Các điều kiện kiểm tra của bạn có:

```
7       if (k == word.size()) return true;
8       if (i < 0 || i >= board.size() || j < 0 || j >= board[0].size() || board[i][
9           return false;
```

```
bool exist(vector<vector<char>>& board, const string& word) {
    for (int i = 0; i < board.size(); ++i) {
        for (int j = 0; j < board[0].size(); ++j) {
            if (backtrack(board, word, i, j, 0)) return true;
        }
    }
    return false;
```

```
cktrack(vector<vector<char>>& board, const string& word, int i, int j, int k) {
(k == word.size()) return true;
(i < 0 || i >= board.size() || j < 0 || j >= board[0].size() || board[i][j] != word[k])
 return false;
```

```
if (i < 0 || i >= board.size() || j < 0 || j >= board[0].size() || board[i][j]
!= word[k])
return false;
```

đảm bảo tránh truy cập ngoài phạm vi lưới, tuy nhiên khi lưới lớn hoặc gần sát biên, cần chắc chắn rằng kiểm tra này xảy ra trước khi truy cập `board[i][j]`.

- Bạn đã thực hiện xử lý hoàn tác rất tốt tuy vậy nếu có lỗi logic trong kiểm tra điều kiện hoặc việc đánh dấu `''` không chính xác, kết quả trả về có thể sai hoặc gây lỗi logic (như không thể khôi phục trạng thái ban đầu).
- Trong code của bạn đã sử dụng đệ quy để tìm kiếm tất cả các nhánh có thể xuất hiện từ một ô khởi đầu đồng thời đã kiểm tra từng ô trong lưới `board` làm điểm bắt đầu (trong vòng lặp ở hàm `exit`):

  tuy vậy bạn đã không tối ưu hóa các trường hợp mà ký tự đầu tiên của `word` không xuất hiện trong `board` đồng thời bạn cũng không sử dụng bộ nhớ để ghi nhớ các trạng thái đã kiểm tra nhằm tránh lặp lại đệ quy không cần thiết.
- Code của bạn đã không xử lý chuỗi rỗng (`word = ""`) hoặc bảng `board` rỗng. Tuy vậy trong bài toán các ràng buộc không cho phép mảng rỗng (m, n >=1) nên lỗi sẽ không sảy ra. Nhưng bạn quên trường hợp nếu `word` rỗng, chương trình sẽ trả về `False`, trong khi lẽ ra đáp án đúng phải là `True`.
- Trong phần code của bạn không có xử lý phân biệt chữ hoa và chữ thường. Chương trình so sánh các ký tự như:

  điều này có thể đúng nếu `word` và `board` co cùng kiểu chữ. Tuy vậy nếu đầu vào không đồng nhất (chữ hoa và chữ thường), chương trình sẽ trả về kết quả sai.

- **Nhóm 15**: Bạn Nguyễn Nguyên Khang mới chỉ làm Problem 1 mà chưa làm Problem 2 và Problem 3, bài làm của bạn cũng không tốt, cần sửa đổi các phần sau:

  - Ban đầu `dp[0]` được khởi tạo là `True`, nhưng các giá trị khác cần được tính toán dựa trên logic từ chuỗi `s2`. Khi tính giá trị của `dp[j]` trong vòng lặp, giá trị của `dp[j]` phụ thuộc vào `dp[j - 1]` (từ chuỗi `s2`) hoặc `dp[j]` (từ chuỗi `s1`).
  - Mình cũng nhận thấy rằng bạn đã xử lý biên không đầy đủ, trường hợp `s1` hoặc `s2` hoặc `s3` là tập rỗng không được xử lý cụ thể, ví dụ trong tập testcase có 1 case là `s1="",s3 = s2` thì chương trình cần trả về `True` nhưng code của bạn lại không xử lý đúng vì không kiểm tra các trường hợp đặc biệt như thế này.
  - Có một số vấn đề với logic cập nhật `dp[j]`. Mình thấy ràng trong code của bạn có đoạn:

```
        for (int j = 1; j <= n2; j++) {
            dp[j] = (dp[j] && s1[i - 1] == s3[i + j - 1]) ||
                    (dp[j - 1] && s2[j - 1] == s3[i + j - 1]);
        }
    }
```

    Như đã nhắc ở trên, biểu thức này không đảm bảo rằng `dp[j]` chứa các giá trị đúng từ lần lặp trước, vì `dp[j]` đã bị ghi đè trong cùng vòng lặp. Nếu `j = 0`, điều kiện `dp[j-1]` sẽ không hợp lệ dẫn đến lỗi nếu không được kiểm tra rõ ràng.
  - Bạn đã không kiểm tra kĩ `s3`. Khi tính toán chỉ số `s3[i+j-1]` bạn đã không có bước kiểm tra đảm bảo `i+j-1` nằm trong giới hạn của `s3` nên khi `i+j-1` vượt quá giới hạn của `s3` thì thay vì đưa ra kết quả `False` thì chương trình có thể gặp lỗi truy cập mảng ngoài phạm vi.

Các bạn còn lại không làm bài tập trên wecode.

2. NHẬN XÉT:

Về cơ bản mình thấy 3 bạn đã làm bài có ý tưởng tiếp cận đúng với từng bài, tuy vậy bạn Tân có vẻ đã không đánh giá đúng các đặc trưng của bài toán, đồng thời code của các bạn vẫn tồn tại quá nhiều hạn chế khiến điểm của các bạn không cao.

- Các bạn đã xử lý tốt phần logic của bài toán, đồng thời 2/3 bạn làm bài đã xác định được đúng đặc trưng của bài toán để lựa chọn giải thuật hợp lý, đồng thời mình cũng thấy rằng code của các bạn được tổ chức rất tốt, rất dễ đọc thuận tiện cho việc kiểm tra và sửa lỗi.
- Tuy vậy các bạn lại mắc các lỗi lai nghiêm trọng khiến cho bài của các bạn không đạt được kết quả cao. Các bạn không xử lý đầy đủ các trường hợp biên, không xử lý các ngoại lệ cùng với đó là một số lỗi lặt vặt.