2024 하반기

전남대학교 PIMM 알고리즘 파티 대회

생성형 AI 제출 간이 평가

전남대학교
게임개발동아리 PIMM
알고리즘 연구회

AI 제출로 평가하는 경우

- 전체 의심군 중 이 대회에서, 그리고 최근 2주 내 제출에서 아래 유형의 사용 빈도가 급격하게 늘어나는 경우에 대해, 운영진 네 명 이상의 동의를 받아 AI 제출로 판단합니다.

1. 알고리즘 문제 풀이에서 쉽게 보기 어려운 코드 유형
2. 코드 전반에 주석이 존재하면서, 매우 자명하고 짧은 코드에 주석이 붙음
3. 지나치게 주석이 많음
4. 다른 제출과 지나치게 유사함

다음 코드는 네 개 유형을 모두 만족하는 AI 제출로 평가합니다.

```
// 필요한 패키지 불러오기
import java.io.BufferedReader;
...
import java.io.OutputStreamWriter;

// Main 클래스 정의
public class Main {
  // main() 메서드 정의
  public static void main(String[] args) throws IOException {
    // BufferedReader 및 BufferedWriter 객체를 불러와 각 변수에 할당
    BufferedReader in = new ...
    BufferedWriter out = new ...
    // readLine() 및 parseInt() 메서드를 사용해 입력 받은 문자열의 개수를...
    int stringNum = Integer.parseInt(in.readLine());
    // 팰린드롬인 문자열의 개수를 저장할 변수 count 초기화
    int count = 0;
    // while 반복문을 사용해 각 문자열을 순회
    while (stringNum-- > 0) {
      // readLine() 메서드를 사용해 입력 받은 문자열을 변수 string에 할당
      String string = in.readLine();
      // 해당 문자열이 팰린드롬인 경우 팰린드롬인 문자열의 개수를 갱신
      if (palindromeChecker(string))
        count++;
    }
    // valueOf() 및 write() 메서드를 사용해 더블팰린드롬 현상을 일으킬 수 있는 쌍의 개수를 출력
    out.write(String.valueOf(count * (count - 1)));
    // close() 메서드를 사용해 각 객체 종료
    in.close();
```

```
    out.close();
  }
  // ---------------------------------------------------------------------
  // palindromeChecker() 메서드 정의
  public static boolean palindromeChecker(String string) {
    // for 반복문을 사용해 문자열의 절반을 순회
    for (int idx = 0; idx < string.length() / 2; idx++) {
      // 해당 문자열이 팰린드롬이 아닌 경우 false 반환
      if (string.charAt(idx) != string.charAt(string.length() - idx - 1))
        return false;
    }
    // 해당 문자열이 팰린드롬인 경우 true 반환
    return true;
  }
}
```

평가 결과

사용자	제출	AC AI 제출 제거 후
A	**맞았습니다!** (2)	**맞았습니다!** (0)
B	틀렸습니다 (4) 시간 초과 (6) 런타임 에러 (1)	—
C	**맞았습니다!** (1) 틀렸습니다 (2) 런타임 에러 (2) 시간 초과 (3)	**맞았습니다!** (0) 틀렸습니다 (2) 런타임 에러 (2) 시간 초과 (3)
D	시간 초과 (2)	—
E	**맞았습니다!** (1) 런타임 에러 (2) 시간 초과 (6)	**맞았습니다!** (1) 런타임 에러 (2) 시간 초과 (6)

84173258, 84173199, 84173181

84173258

```python
import sys

input = sys.stdin.readline
sys.setrecursionlimit(int(1e5))

def is_palindrome(s):
    return s == s[::-1]

def count_double_palindromes(strings):
    count = 0
    n = len(strings)

    for i in range(n):
        if is_palindrome(strings[i]):
            # 문자열 s_i를 절반으로 나누기
            X1 = strings[i][:len(strings[i])//2]
            X2 = strings[i][len(strings[i])//2:]

            for j in range(n):
                if i!=j:
                    # 문자열 s_j를 Y로 사용
                    Y = strings[j]

                    # Z = X1 + Y + X2가 팰린드롬인지 확인
                    Z = X1 + Y + X2

                    if is_palindrome(Z):
                        count += 1

    return count

# 입력 받기
N = int(input())
strings = [input().strip() for _ in range(N)]

# 결과 출력
print(count_double_palindromes(strings))
```

84172722, 84172466, 84172446

84172722

```python
# 팰린드롬 여부를 확인하는 함수
def is_palindrome(s):
    return s == s[::-1]

# 입력 받기
N = int(input())
strings = [input().strip() for _ in range(N)]

# 팰린드롬인 문자열의 개수 세기
palindrome_count = 0
for s in strings:
    if is_palindrome(s):
        palindrome_count += 1

# 더블 팰린드롬 쌍의 개수 계산 (i ≠ j인 경우만)
# 팰린드롬 문자열들 중에서 두 개를 고르는 경우의 수는 palindrome_count *
(palindrome_count - 1)
if palindrome_count > 1:
    ans = palindrome_count * (palindrome_count - 1)
else:
    ans = 0

# 결과 출력
print(ans)
```

84174246, 84174104, 84174041, 84173958, 84173869, 84173788

84174246

```python
def manacher(s):
    """Manacher's algorithm to find all palindromes in a string."""
    A = '@#' + '#'.join(s) + '#$'
    Z = [0] * len(A)
    center = right = 0
    for i in range(1, len(A) - 1):
        if i < right:
            Z[i] = min(right - i, Z[2 * center - i])
        while A[i + Z[i] + 1] == A[i - Z[i] - 1]:
            Z[i] += 1
        if i + Z[i] > right:
            center, right = i, i + Z[i]
    return Z[2:-2:2]  # Extract only the useful part and ignore separators.

def is_palindrome(s, left, right):
    """Check if s[left:right+1] is a palindrome."""
    while left < right:
        if s[left] != s[right]:
            return False
        left += 1
        right -= 1
    return True

def double_palindrome_count(strings):
    count = 0

    for i in range(len(strings)):
        X = strings[i]
        if len(X) % 2 != 0:  # X는 짝수 길이어야 함
            continue

        # Manacher's algorithm to find all palindromes in X
        Z = manacher(X)

        mid = len(X) // 2
        X1 = X[:mid]
        X2 = X[mid:]

        for j in range(len(strings)):
```

```
            if i == j:  # 동일한 문자열은 선택하지 않는다.
                continue

            Y = strings[j]
            if len(Y) % 2 != 0:  # Y도 짝수 길이어야 함
                continue

            # Z = X1 + Y + X2를 만들고 팰린드롬인지 확인
            Z = X1 + Y + X2
            if is_palindrome(Z, 0, len(Z) - 1):
                count += 1

    return count

# 테스트 입력
n = int(input())
strings = [input() for _ in range(n)]

# 더블팰린드롬 쌍의 개수 출력
result = double_palindrome_count(strings)
print(result)
```

84173310

84173310

```java
// 필요한 패키지 불러오기
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

// Main 클래스 정의
public class Main {

  // main() 메서드 정의
  public static void main(String[] args) throws IOException {

    // BufferedReader 및 BufferedWriter 객체를 불러와 각 변수에 할당
    BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
    BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(System.out));

    // readLine() 및 parseInt() 메서드를 사용해 입력 받은 문자열의 개수를 변수
stringNum에 할당
    int stringNum = Integer.parseInt(in.readLine());

    // 팰린드롬인 문자열의 개수를 저장할 변수 count 초기화
    int count = 0;

    // while 반복문을 사용해 각 문자열을 순회
    while (stringNum-- > 0) {

      // readLine() 메서드를 사용해 입력 받은 문자열을 변수 string에 할당
      String string = in.readLine();

      // 해당 문자열이 팰린드롬인 경우 팰린드롬인 문자열의 개수를 갱신
      if (palindromeChecker(string))
        count++;
    }

    // valueOf() 및 write() 메서드를 사용해 더블팰린드롬 현상을 일으킬 수 있는 쌍의 개수
를 출력
    out.write(String.valueOf(count * (count - 1)));
```

```
    // close() 메서드를 사용해 각 객체 종료
    in.close();
    out.close();
  }

  //
----------------------------------------------------------------------------------

  // palindromeChecker() 메서드 정의
  public static boolean palindromeChecker(String string) {

    // for 반복문을 사용해 문자열의 절반을 순회
    for (int idx = 0; idx < string.length() / 2; idx++) {

      // 해당 문자열이 팰린드롬이 아닌 경우 false 반환
      if (string.charAt(idx) != string.charAt(string.length() - idx - 1))
        return false;
    }

    // 해당 문자열이 팰린드롬인 경우 true 반환
    return true;
  }
}
```

84173252, 84173205, 84173127, 84173045

84173252

```python
from collections import defaultdict

def count_double_palindromes(strings):
    prefix_map = defaultdict(list)
    suffix_map = defaultdict(list)
    count = 0

    for index, string in enumerate(strings):
        half_len = len(string) // 2
        X1 = string[:half_len]
        X2 = string[half_len:]

        # 각 절반을 기록
        prefix_map[X1].append(index)
        suffix_map[X2].append(index)

    # 각 문자열에 대해 매칭이 가능한지 검사
    for index, string in enumerate(strings):
        half_len = len(string) // 2
        X1 = string[:half_len]
        X2 = string[half_len:]

        # X2의 역순과 X1이 매칭 가능한지 확인 (i != j 조건 고려)
        reversed_X2 = X2[::-1]
        if reversed_X2 in prefix_map:
            for match_index in prefix_map[reversed_X2]:
                if match_index != index:
                    count += 1

        # X1의 역순과 X2가 매칭 가능한지 확인 (i != j 조건 고려)
        reversed_X1 = X1[::-1]
        if reversed_X1 in suffix_map:
            for match_index in suffix_map[reversed_X1]:
                if match_index != index:
                    count += 1

    return count

# 입력 처리
N = int(input())
strings = [input().strip() for _ in range(N)]
```

```python
# 더블팰린드롬 쌍의 개수 계산 및 출력
result = count_double_palindromes(strings)
print(result)
```

84174067, 84173939

84174067

```
import bisect

def solve(queries):
    tree_positions = []
    current_position = 0
    total_distance = 0

    for query in queries:
        if query[0] == 1:
            # 민규가 x에 쓰레기를 버림
            x = query[1]
            bisect.insort(tree_positions, x)  # 이진 탐색으로 정렬된 리스트에 삽입
        elif query[0] == 2:
            while tree_positions:
                # 현재 위치에서 가장 가까운 나무를 찾는다
                idx = bisect.bisect_left(tree_positions, current_position)

                # 가장 가까운 나무 후보들
                if idx == len(tree_positions):
                    closest_tree = tree_positions[-1]
                elif idx == 0:
                    closest_tree = tree_positions[0]
                else:
                    if abs(tree_positions[idx] - current_position) <
abs(tree_positions[idx - 1] - current_position):
                        closest_tree = tree_positions[idx]
                    else:
                        closest_tree = tree_positions[idx - 1]

                # 그 나무로 이동
                total_distance += abs(current_position - closest_tree)
                current_position = closest_tree
                tree_positions.remove(closest_tree)

    return total_distance

# 입력 받기
N = int(input())
queries = []
for _ in range(N):
    query = list(map(int, input().split()))
```

```
    queries.append(query)

# 결과 출력
print(solve(queries))
```

84173709, 84173611

84173709

```
from sortedcontainers import SortedList

# 입력 받기
N = int(input())
queries = [input().split() for _ in range(N)]

# SortedList를 사용하여 나무 좌표를 관리
trees = SortedList()
current_position = 0
total_distance = 0

# 쿼리 처리
for query in queries:
    if query[0] == '1':
        # 쿼리 1: 나무에 쓰레기를 버림
        x = int(query[1])
        trees.add(x)
    elif query[0] == '2':
        # 쿼리 2: 근성이 모든 나무의 쓰레기를 수거함
        while trees:
            # 가장 가까운 나무를 찾아 이동 (좌표가 같은 경우 더 작은 좌표 우선)
            # 왼쪽 가장 가까운 나무
            closest_left_idx = trees.bisect_left(current_position) - 1
            # 오른쪽 가장 가까운 나무
            closest_right_idx = trees.bisect_left(current_position)

            # 왼쪽과 오른쪽 나무를 결정
            if closest_left_idx >= 0 and closest_right_idx < len(trees):
                # 양쪽에 나무가 있는 경우 더 가까운 나무 선택
                left_distance = abs(trees[closest_left_idx] -
current_position)
                right_distance = abs(trees[closest_right_idx] -
current_position)

                if left_distance <= right_distance:
                    next_position = trees.pop(closest_left_idx)
                else:
                    next_position = trees.pop(closest_right_idx)
            elif closest_left_idx >= 0:
                # 왼쪽에만 나무가 있는 경우
                next_position = trees.pop(closest_left_idx)
```

```
            else:
                # 오른쪽에만 나무가 있는 경우
                next_position = trees.pop(closest_right_idx)

            # 이동한 거리 계산
            total_distance += abs(current_position - next_position)
            # 근성의 현재 위치 업데이트
            current_position = next_position

# 결과 출력
print(total_distance)
```

84175483

84175483

```java
// 필요한 패키지 불러오기
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.StringTokenizer;
import java.util.TreeSet;

// Main 클래스 정의
public class Main {

  // main() 메서드 정의
  public static void main(String[] args) throws IOException {

    // BufferedReader 및 BufferedWriter 객체를 불러와 각 변수에 할당
    BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
    BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(System.out));

    // readLine() 및 parseInt() 메서드를 사용해 입력 받은 쿼리의 개수를 변수 queryNum
에 할당
    int queryNum = Integer.parseInt(in.readLine());

    // 근성이의 현재 위치 및 이동 거리를 저장할 각 변수 초기화
    int currentLoc = 0;
    long totalDist = 0;

    // 민규가 쓰레기를 버린 나무의 위치를 저장할 TreeSet 객체 trashes 초기화
    TreeSet<Integer> trashes = new TreeSet<>();

    // while 반복문을 사용해 각 쿼리를 순회
    while (queryNum-- > 0) {

      // StringTokenizer 객체를 불러와 변수 st에 할당
      StringTokenizer st = new StringTokenizer(in.readLine());

      // nextToken() 및 parseInt() 메서드를 사용해 입력 받은 쿼리의 종류를 변수
category에 할당
      int category = Integer.parseInt(st.nextToken());
```

```
      // 민규가 쓰레기를 버리는 경우 쓰레기를 버린 좌표를 trashes에 추가
      if (category == 1) {
        trashes.add(Integer.parseInt(st.nextToken()));

      // 근성이가 쓰레기를 수거하는 경우
      } else {

        // while 반복문을 사용해 더 이상 쓰레기가 없을 때까지 순회
        while (!trashes.isEmpty()) {

          // ceiling() 및 floor() 메서드를 사용해 현재 근성이의 위치에서 가장 가까운 쓰
레기가 투기된 나무를 각 변수에 할당
          Integer leftTree = trashes.floor(currentLoc);
          Integer rightTree = trashes.ceiling(currentLoc);

          // 근성이가 다음으로 이동할 위치를 저장할 변수 nextLoc 초기화
          int nextLoc;

          // 근성이의 오른쪽에 쓰레기가 투기된 나무가 없는 경우 왼쪽 나무로 이동
          if (rightTree == null) {
            nextLoc = leftTree;

          // 근성이의 왼쪽에 쓰레기가 투기된 나무가 없는 경우 오른쪽 나무로 이동
          } else if (leftTree == null) {
            nextLoc = rightTree;

          // 근성이의 양쪽에 쓰레기가 투기된 나무가 있는 경우
          } else {

            // 왼쪽 나무가 근성이와 더 가깝거나, 두 나무가 근성이로부터 같은 거리만큼 떨어
져 있는 경우 왼쪽 나무로 이동
            if (Math.abs(leftTree - currentLoc) <= Math.abs(rightTree -
currentLoc)) {
              nextLoc = leftTree;

            // 오른쪽 나무가 근성이와 더 가까운 경우 오른쪽 나무로 이동
            } else {
              nextLoc = rightTree;
            }
          }

          // abs() 메서드를 사용해 근성이의 이동 거리를 갱신
          totalDist += Math.abs(nextLoc - currentLoc);
```

```
            // 근성이의 현재 위치를 갱신
            currentLoc = nextLoc;

            // remove() 메서드를 사용해 쓰레기를 치운 나무의 위치를 trashes에서 제거
            trashes.remove(nextLoc);
        }``
      }
    }

    // valueOf() 및 write() 메서드를 사용해 근성이의 이동 거리를 출력
    out.write(String.valueOf(totalDist));

    // close() 메서드를 사용해 각 객체 종료
    in.close();
    out.close();
  }
}
```