ECON381 Fall 2024

Homework Assignment 2

Okey Game Questions and Solutions

Question 1: Adding and Removing Keys on the Board

What kind of operations are required for adding and removing keys during gameplay?

Adding Keys

- **Operation:** Add a key (tile) to the player's board.
    - This involves checking if there is enough space (max 14 tiles).
    - The key can be appended at the end or inserted at the correct position based on sorting rules (e.g., by color and number).

Removing Keys

- **Operation**: Remove a specific key from the player's board.
    - The key must be identified (by its number and color).
    - The structure holding the keys must be updated (e.g., shifting elements in an array).

Implementation Notes

- **In Arrays**: Adding/removing requires shifting elements, especially when maintaining order.
- **In Linked Lists:** No need for shifting; insertion and deletion are efficient but require traversal to find the correct position.

Question 2: Determining if a Player is Done

What checks are required to determine if a player has completed the game?

Condition A: Blocks of 3 or More

- All 14 keys must be organized into valid blocks:
    - **Runs**: Consecutive numbers of the same color (e.g., 2, 3, 4 in red).
    - **Sets**: Same number but different colors (e.g., 11 in red, black, and yellow).

Condition B: Seven Pairs

- Alternatively, the player can win by forming seven pairs:
    - Each pair consists of two identical tiles (same number and color).

Algorithm

1. Sort keys by color and number.
2. Check for runs and sets using iterative or recursive logic.
3. If a valid combination is formed:
    - All 14 keys form blocks (Condition A), OR
    - Keys form seven pairs (Condition B), the player is done.

Question 3: Data Structure Choice

Would you use a single fixed-size array, multiple arrays, or linked lists to hold the 14 keys?

Option 1: Single Fixed-Size Array

- **Advantages:**
    - **Simplicity:** Easy to initialize and manage**.**
    - **Fast access:** Random access is O(1).
    - **Memory efficient:** Fixed size avoids dynamic memory overhead.
- **Disadvantages: Inflexible:** Hard to manage dynamic insertions/deletions without shifting elements.

Option 2: Multiple Arrays

- **Advantages:**
    - Can separate keys into blocks (e.g., one array for runs, another for sets).
    - Easier to validate conditions like blocks and pairs.
- **Disadvantages:**
    - More complex to manage.
    - Requires extra memory and logic to merge and split blocks.

Option 3: Linked Lists

- **Advantages:**
    - Dynamic resizing: No need for shifting when inserting or removing keys.
    - Simplifies operations like reordering or grouping keys into blocks.
- **Disadvantages:**
    - Slower access time: Traversal is O(n).
    - Higher memory usage: Each node requires additional storage for pointers.

Recommendation

- A single fixed-size array is suitable for simplicity and performance if the game's rules (like sorting and forming blocks) are implemented with helper methods.
- Use linked lists if the game frequently requires dynamic insertion/removal or flexible grouping.

Implementation Notes for OkeyKey Class

The given OkeyKey class is already well-implemented and supports:

1. **Validation**: Ensures numbers are between 1 and 13 and colors are not null.
2. **Equality and Hashing**: Allows comparison and use in collections like HashSet.
3. **String Representation**: Facilitates debugging.

Here's an example of its usage:

```
OkeyKey key1 = new OkeyKey(3, OkeyKey.Color.RED);
OkeyKey key2 = new OkeyKey(3, OkeyKey.Color.RED);

// Example usage in a list
List<OkeyKey> board = new ArrayList<>();
board.add(key1);
board.add(key2);

// Check for equality
System.out.println(key1.equals(key2)); // true
```

Yaren BOLAT 19232810032