Data Structures and Algorithms Design

Assignment – 2 – PS18 – G018 – Batting Order

**Design**:

The design considerations include the following

1. Generate all possible permutations of the players using recursion function

2. Use native data types to implement Hash table with double hashing to avoid clustering and collision.

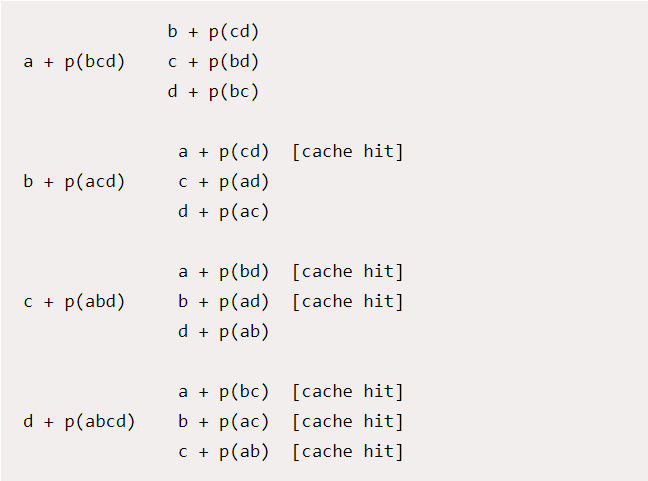
3. Save the intermediate results from recursive function to the hash table.(Memoisation)

4. Lookup the hash table to reduce the processing. Hash table has a time complexity of O(1)

5. The time complexity of permutation operation is O(n!) but after applying dynamic programming it is O(n^2)

6. Optimize execution time, by fine tuning big (O)

Eg., For 'abcd' , please find the below inner working of dynamic programming with memoisation



**Permutation Implementation:**

1. Read the player positions from the input file and handle all possible error that might occur in the input file.

2. Call the permute() function in recursion to generate the different permutation of players.

The time complexity is O(n!). So, for 11 players, the possible permutations are 11! = 39916800.

3. Implement the Memoisation using the hash table. The intermediate values are saved in the hash table and retrieved when necessary without doing duplicate processing.

3. Check the list generated with the player positions from the input file and filter the unique positions.

4. write the total count to the output file

**HashTable Implementation:**

Hash table stores key-value pairs and the key is generated through a hashing function. The address or the index value of the data element is generated from a hash function to store the data in buckets -List

**Hash Function:**

We use the python inbuilt function ORD() to generate the integer for the Unicode characters and add everything together and modulo by the total size of the Hash table.

hash = sum([ORD(key)]) % Size

**Collision Handling:**

There are different methods to handle collision in Hash tables. We first started with the chaining method, but the overall time complexity is O(n) if collision handling is not done properly.

Since linear and quadratic probing caused clustering. We used Double hashing probing method with the second hash function integrating the key from the first hash function

i = h(k) – first hash function

i + f(j)) mod size] – Second hash function

where f(j) =j h'(k) and h'(k) = q - (k mod q) for j= 1,2,3…. and q – prime number

**Run time analysis - Time complexity:**

**Hash table Insertion:**

**Best case scenario : O(1)** - If there is no collision - The insertion takes place in O(1)

**Worst case scenario: O(n)** - Double Hashing is used for collision management and in a worst-case scenario when the whole hash table has to be probed.

**Hash table Retrieval:**

**Best case scenario : O(1)** - If there is no collision - Retrieval will be O(1)

**Worst case scenario: O(n)** - If there is collision, then Double Hashing is used for collision management. It will probe the entire hash table in worst case scenario.

**Average number of probes required for insertion or retrieval or deletion:**

E = 1/(1-alpha) Where, alpha = load factor = N/M,

M - is the size of hash table

N - is the number of elements currently in the hash table

In Our case, we consider the load factor of 0.5 for rehashing to a bigger hash table.

**Permutation:**

**Without dynamic programming:** The time complexity is O(n!)

**With dynamic programming:** The time complexity is O(n^2) with Memoisation.