6.10 UNIQUE PERMUTATIONS OF AN ARRAY

Question:

Given a collection of numbers, nums, that might contain duplicates, return all possible unique permutations in any order.

AIM

To generate all unique permutations of a given array of integers (which may contain duplicates) using backtracking.

ALGORITHM

1. Sort the array to make it easier to handle duplicates.
2. Use backtracking to generate permutations.
3. Maintain a 'used' array to track whether an element is included in the current path.
4. At each step, iterate through nums:
5. - Skip already used elements.
6. - Skip duplicates by checking if the current element is the same as the previous and the previous has not been used.
7. Add the current number to the path and recurse.
8. When the path length equals nums length, record the permutation.
9. Backtrack by marking the element as unused and removing it from the path.

PROGRAM

```
def permute_unique(nums):
    result = []
    nums.sort()
    used = [False] * len(nums)
    def backtrack(path):
        if len(path) == len(nums):
            result.append(path[:])
            return
        for i in range(len(nums)):
            if used[i]:
                continue
            if i > 0 and nums[i] == nums[i-1] and not used[i-1]:
                continue
            used[i] = True
            path.append(nums[i])
            backtrack(path)
            path.pop()
            used[i] = False
    backtrack([])
    return result

nums = list(map(int, input("Enter numbers: ").split()))
result = permute_unique(nums)
print("Unique permutations:")
for perm in result:
    print(perm)
```

Input:

nums = [1,1,2]

Output:

```
Enter numbers: 1 1 2
Unique permutations:
[1, 1, 2]
[1, 2, 1]
[2, 1, 1]
>>>
```

RESULT:

Thus, the program is successfully executed and verified to generate all unique permutations of the array.

PERFORMANCE ANALYSIS:

Time Complexity: O(n * n!) in the worst case where n is the number of elements.

Space Complexity: O(n) for recursion depth and path storage, plus O(n) for the 'used' array.