DAA ASSIGNMENT

Given By: Dr.Mohammed Javed

Guided by: Mr. Bhavesh Kumar Bohora

Submitted By:

Shivani Pal – IIT2021504

Rhythm Jain – IIT2021505

Aditya Yalamanchi – IIB2021001

Nitesh Kumar Shah – IIB2021002

QUESTION

The task is to implement a recursive algorithm in C++ that generates the n U r permutations with unrestricted repetitions in reverse lexical order.

The algorithm should generate all possible permutations of r elements from a set of n distinct elements, allowing repetitions.

Algo

1. Define a recursive function generate Permutations(n, r, perm) that takes three arguments:
   a. n: the number of distinct elements in the set
   b. r: the desired length of the permutation
   c. p: a vector to store the permutation

2. Check if r == 0. If it is, print the permutation in reverse lexical order and return.

3. Otherwise, for each element i from 1 to n, do the following:
   a. Add i to the end of the permutation vector.
   b. Recursively call permute(p,n, r - 1).
   c. Remove the last element from the permutation vector.

4. In the main function:
   a. Initialize n and r.
   b. Call Permute(p,n, r) to generate the permutations.

Code :

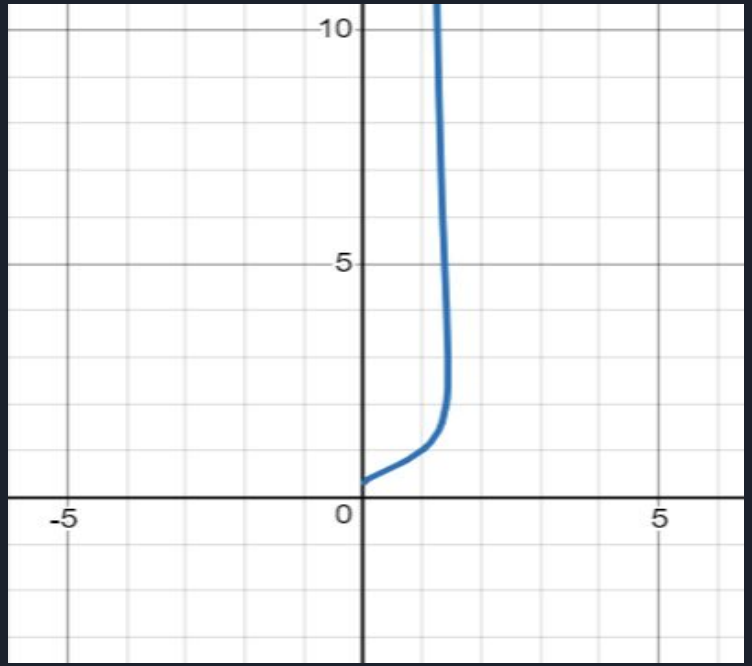
```cpp
#include <iostream>
#include <vector>

using namespace std;

void permute(vector<int> &p, int n, int r)
{
    if (r == 0)
    {

        for (int i = p.size() - 1; i >= 0; i--)
        {
            cout << p[i] << " ";
        }
        cout << endl;
    }
    else
    {
        for (int i = 1; i <= n; i++)
        {
            p.push_back(i);
            permute(p, n, r - 1);
            p.pop_back();
        }
    }
}
int main()
{
    int n, r;
    cout << "Enter n and r: ";
    cin >> n >> r;

    vector<int> p;
    permute(p, n, r);

    return 0;
}
```

Time Complexity

The time complexity of the given code is O(n^r), where n is the number of elements to choose from and r is the number of elements in each permutation.

The reason for this complexity is that the code generates all possible permutations with unrestricted repetitions, and there are n^r such permutations.

The recursive function permute is called r times (the depth of the recursion), and at each level of the recursion, there are n branches (the for loop with i from 1 to n). Therefore, the total number of function calls is n^r.



Space Complexity

The space used by the algorithm depends on the maximum depth of the recursive tree, which is equal to r.

At each level of the tree, a new integer is added to the permutation vector, which has a maximum size of r.

Therefore, the space complexity of the algorithm is O(r), which is relatively small compared to the time complexity.

APOSTERIORI ANALYSIS

We are using a variable t to count the number of operations required for a particular value of n;

For n=3 r=2 number of Operations=144

For n=3 r=3 number of Operations=558

For n=4 r=3 number of Operations=1272

For n=4 r=4 number of Operations=6136

For n=5 r=3 number of Operations=2430

We observe that as n increases number of Operations also increase proportional to O(n^r).

Conclusion

The C++ implementation of the recursive algorithm for generating n U r permutations with unrestricted repetitions in reverse lexical order provides a useful tool for enumerating all possible permutations of r elements chosen from a set of n distinct objects.

While the time and space complexity of the algorithm can be limiting for large values of n and r, it is efficient and accurate for small to medium-sized inputs. The algorithm can be easily adapted to other problems that require generating all possible combinations of a set of objects, making it a useful reference for researchers, engineers, and programmers.