Dash

All

Articles

Videos

Problems

Quiz

« Prev

Next »

Tail call elimination in Quick Sort

Tail Recursion a recursive function is tail recursive if the recursive call is the last thing executed by the function.

C++

```
 #include <bits/stdc++.h>
using namespace std;
void print(int n)
{
    if (n < 0)
       return;
    cout<<n;

    // The last executed statement is recursive call
    print(n-1);
}
```

Tail-recursive is better than a non-tail recursive as tail-recursion can be optimized by modern compilers. Modern compiler basically does **tail call elimination** to optimize the tail-recursive code.

If we take a closer look at the above function, we can remove the last call with goto. Below are examples of tail call elimination.

Dash
All
Articles
Videos
Problems
Quiz

C++

```
#include <bits/stdc++.h>
using namespace std;

void print(int n)
{
start:
    if (n < 0)
       return;
    cout<<n;

    // Update parameters of recursive call
    // and replace recursive call with goto
    n = n-1;
    goto start;
}
```

QuickSort : One more example
QuickSort is also tail recursive (Note that MergeSort is not tail recursive, this is also one of the reasons why QuickSort performs better)

C++

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

The above function can be replaced by following after tail call elimination.

C++

```
void quickSort(int arr[], int low, int high)
{
start:
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
```

```
            quickSort(arr, low, pi - 1);

            // Update parameters of recursive call
            // and replace recursive call with goto
            low = pi+1;
            high = high;
            goto start;
        }
    }
```

Therefore job for compilers is to identify tail recursion, add a label at the beginning and update parameter(s) at the end followed by adding the last goto statement.

Function stack frame management in Tail Call Elimination :
Recursion uses a stack to keep track of function calls. With every function call, a new frame is pushed onto the stack which contains local variables and data of that call. Let's say one stack frame requires O(1) i.e, constant memory space, then for **N** recursive call memory required would be O(N).

Tail call elimination reduces the space complexity of recursion from O(N) to O(1). As function call is eliminated, no new stack frames are created and the function is executed in constant memory space.

It is possible for the function to execute in constant memory space because, in tail recursive function, there are no statements after call statement so preserving state and frame of parent function is not required. Child function is called and finishes immediately, it doesn't have to return control back to the parent function.

As no computation is performed on the returned value and no statements are left for execution, the current frame can be modified as per the requirements of the current function call. So there is no need to preserve stack frames of previous function calls and function executes in constant memory space. This makes tail recursion faster and memory-friendly.

Marked as Read

Dash

All

Articles

Videos

Problems

Quiz

Report An Issue

If you are facing any issue on this page. Please let us know.