CS10003

PROGRAMMING AND DATA STRUCTURES

Spring 2024

Computer Science & Engineering Department

IIT Kharagpur

Recursion – Advanced Examples

Sudeshna Sarkar

12 Feb 2024

Paying with fewest coins

- A country has coins of denomination 3, 5 and 10, respectively.
- We are to write a function canchange(k) that returns –1 if it is not possible to pay a value of k using these coins.
  - Otherwise it returns the minimum number of coins needed to make the payment.

- For example, canchange(7) will return –1.
- On the other hand, canchange(14) will return 4 because 14 can be paid as 3+3+3+5 and there is no other way to pay with fewer coins.
- Finally, 15 can be changed as 3+3+3+3+3, 5+5+5, 5+10, so canchange (15) will return 2.

Paying with fewest coins

```
int canchange( int V)
{
        int a;
        if (V==0) return 0;
        if ( (V ==3) || (V == 5) || (V == 10) ) return 1;
        if (V< 3)  return –1  ;

        a = canchange( V– 10 ); if (a > 0) return a+1 ;
        a = canchange( V – 5 ); if (a > 0) return a+1 ;
        a = canchange( V – 3 ); if (a > 0) return a+1 ;
        return –1;
 }
```

Exercise: Rewrite this code if the denominations are 3, 8, and 10. Do you see a problem? Repair it.

Find minimum number of coins to make a given value using Recursion

The minimum number of coins for a value V can be computed using the recursive formula below.

- **If V == 0:**

  0 coins required

- **If V > 0:**

  minCoins(coins[0..m-1], V) = min { 1 + minCoins(V-coin[i])}

  where, 0 <=i <= m-1 and coins[i] <= V.

Paying with fewest coins

```
int canchange( int V)
{
        int a, min;
        if (V==0) return 0;
        if ( (V ==3) || (V == 8) || (V == 10) ) return 1;
        if (V< 3)  return –1  ;
        min = 0;
        a = canchange( V– 10 ); if (a> 0) min = a; ;
        a = canchange( V – 8 ); if ((a > 0) && (a < min)) min = a;
        a = canchange( V – 3 ); if ((a > 0) && (a < min)) min = a;
        if (min > 0)
            return a;
         return -1
 }
```

Paying with fewest coins

```
int canchange( int V, int coins[], int numcoins)  {
        int a, i, min=-1;
        if (V==0) return 0;
        if (V<0)  return -1;
        for (i=0; i<numcoins; i+=) {
            a= canchange( V– coins[i], coins, numcoins);
            if (a==-1) continue;
            if ((a+1 < min) || (min==-1))  min = a+1;
        }
        return min;
}
```

```
int main()  {
    int coins[3]={10,8,3} ;
    int V;
    scanf ("%d", &V) ;
    printf ("min coins = %d\n",
                canchange (V,coins, 3));
    return 0;
}
```

Count number of ways to make a change of V

- **Note: Assume that you have an infinite supply of each type of coin.**

Examples:

- **Input: sum = 4, coins[] = {1,2,3},**
- **Output: 4**
- **Explanation: there are four solutions:**
  {1, 1, 1, 1}, {1, 1, 2}, {2, 2}, {1, 3}.

Count number of ways to make a change of V

- <u>For each coin, there are 2 options.</u>
- **Include the current coin**: Subtract the current coin's denomination from the target sum and call the count function recursively with the updated sum and the same set of coins i.e., count(coins, n, sum – coins[n-1])
- **Exclude the current coin**: Call the count function recursively with the same sum and the remaining coins. i.e., count(coins, n-1,sum).

- **Recurrence Relation**

Count(coins,n,sum) = count(coins,n,sum-count[n-1]) + count(coins,n-1,sum)

```c
// Returns the count of ways we can sum coins[0...n-1]
// coins to get  "sum"
int count(int coins[], int n, int sum)  {
    // If sum is 0 then there is 1 solution (do not include any coin)
    if (sum == 0)
            return 1;
    // If sum is less than 0 then no solution exists
    if (sum < 0)
          return 0;
    // If there are no coins and sum is greater than 0,
    //then no solution exist
    if (n <= 0)
          return 0;
   return count(coins, n - 1, sum)+ count(coins, n, sum - coins[n - 1]);
}
```

```c
// Driver program
int main()
{
   int i, j;
   int coins[] = { 1, 2, 3 };
   int n = 3;
   printf("%d ", count(coins, n, 5)) ;
   return 0;
}
```

Generate all unique partitions of an integer

- **Given a positive integer n, the task is to generate all possible unique ways to represent n as sum of positive integers.**
- **Examples:**

Input: 4

Output:

4
3 1
2 2
2 1 1
1 1 1 1

Input: 3

Output:

3
2 1
1 1 1

Generate all unique partitions of an integer

```
void partition (int arr[], int size, int n) {
        int i;
         if (n==0) {
                for (i=0; i<size; i++)
                        printf ("%3d", arr[i]) ;
                        printf ("\n") ;
                        return;
        }
        for (i=n; i>0; i--) {
                arr[size] = i;
                partition (arr, size+1, n-i) ;
        }
}
```

```
int main(){
        int n;
        int A[100] ;
        scanf ("%d", &n);
        printf (“n = %d\n", n) ;
        partition (A, 0, n) ;
        return 0;
}
```

```
n=4
 4
 3  1
 2  2
 2  1  1
 1  3
 1  2  1
 1  1  2
 1  1  1  1
```

```c
#include <stdio.h>
void partition (int arr[], int size, int n, int max) {
        int i;    if (n<0) return;
        if (n==0) {
            for (i=0; i<size; i++)
                printf ("%3d", arr[i]) ;
                printf ("\n") ;
                return;
        }
        for (i=max; i>0; i--) {
                arr[size] = i;
                partition (arr, size+1, n-i, i) ;
        }
}
```

```c
int main(){
    int n, size = 0, max;
    int A[100] ;
    scanf ("%d", &n);
    printf ("n = %d\n", n) ;
    max = n;
    partition (A, 0, n, max) ;
    return 0;
}
```

```
n = 7
  7
  6  1
  5  2
  5  1  1
  4  3
  4  2  1
  4  1  1  1
  3  3  1
  3  2  2
  3  2  1  1
  3  1  1  1  1
  2  2  2  1
  2  2  1  1  1
  2  1  1  1  1  1
  1  1  1  1  1  1  1
```