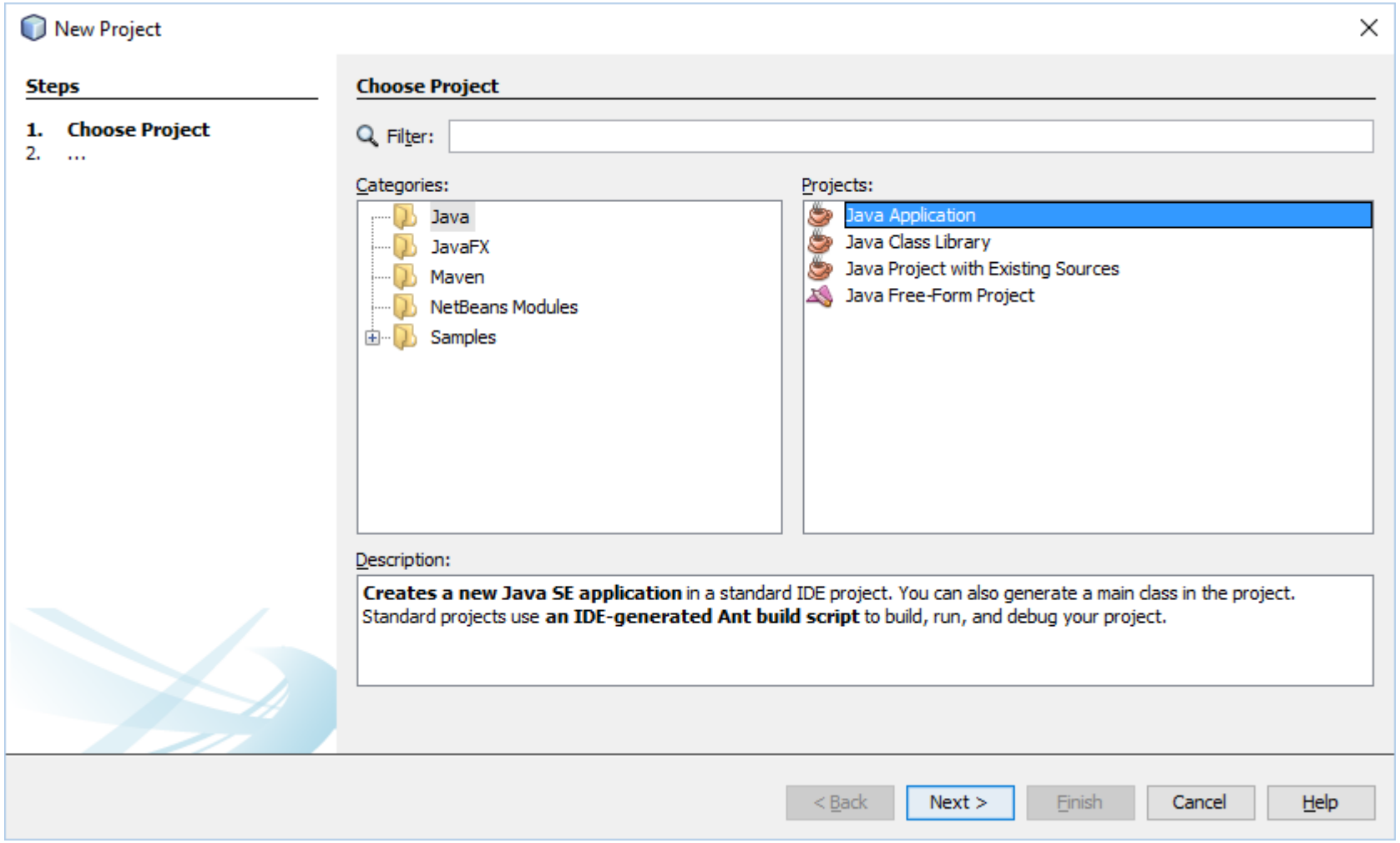
Lab 10

CPS501 – Advanced Programming and Data Structures

Objective

- Practice more with Recusion and Dynamic Programming

Create Lab10 project in NetBeans

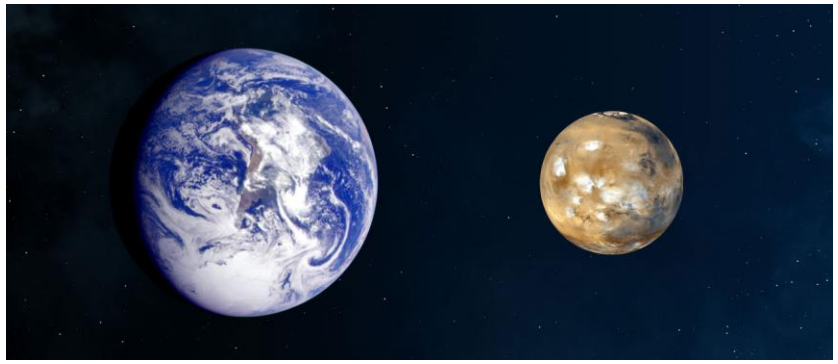


Problem: Counting coins

- To find the **minimum number** of Mars coins (1¢, 5¢, 10¢, 21¢, 25¢) to make any amount, the greedy method always works
- At each step, just choose the largest coin that does not overshoot the desired amount, for example, V = 31¢.
  - 31¢→25
  - 6¢→5
  - 1¢→1

Problem: Counting coins

- The greedy method would not work if we did not have 5¢ coins
  - For V = 31 cents, we can have seven coins (25+1+1+1+1+1+1), but we can do it with four (10+10+10+1)
- The greedy method also would not work if we did not have a 21¢ coin
  - For 63 cents, we can have six coins (25+25+10+1+1+1), but we can do it with three (21+21+21)
- How can we find the minimum number of coins for any given coin set?



Coin set for examples

- For the following examples, we will assume coins in the following denominations:

  1¢ 5¢ 10¢ 21¢ 25¢

- We'll use V = 63¢ as our goal

Recursion Solution

- We always need a 1¢ coin, otherwise no solution exists for making one cent
- If V == 0, then 0 coins required.
- If V > 0

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

where i varies from 0 to m-1

and coin[i] <= V

```java
// m is size of coins array (number of different coins)
// V is the expected amount
static int minCoins(int coins[], int m, int V)
{
    // base case
    if (V == 0) return 0;

    // Initialize result
    int res = Integer.MAX_VALUE;

    // Try every coin that has smaller value than V
    for (int i=0; i<m; i++)
    {
      if (coins[i] <= V)
      {
          int sub_res = minCoins(coins, m, V-coins[i]);

          // Check for Integer.MAX_VALUE to avoid overflow and see if
          // result can minimized
          if (sub_res != Integer.MAX_VALUE && sub_res + 1 < res)
             res = sub_res + 1;
      }
    }
    return res;
}
```

```java
public static void main(String args[])
{
   int coins[] =  {25, 21, 10, 5, 1};
   int m = coins.length;
   int V = 63;
   System.out.println("Minimum coins
    required: "+ minCoins(coins, m, V));
}
```

Recursion Solution

- This algorithm can be viewed as brute force
  - This solution is very recursive
  - It requires exponential work

A dynamic programming solution

- Idea: Solve first for one cent, then two cents, then three cents, etc., up to the desired amount
  - *Save each answer in an array !*
- For each new amount N, compute all the possible pairs of previous answers which sum to N
  - For example, to find the solution for 13¢,
    - First, solve for all of 1¢, 2¢, 3¢, ..., 12¢
    - Next, choose the best solution among:
      - Solution for 1¢ + solution for 12¢
      - Solution for 2¢ + solution for 11¢
      - Solution for 3¢ + solution for 10¢
      - Solution for 4¢ + solution for 9¢
      - Solution for 5¢ + solution for 8¢
      - Solution for 6¢ + solution for 7¢

Example

- Suppose coins are 1¢, 3¢, and 4¢
  - There's only one way to make 1¢ (one coin)
  - To make 2¢, try 1¢+1¢ (one coin + one coin = 2 coins)
  - To make 3¢, just use the 3¢ coin (one coin)
  - To make 4¢, just use the 4¢ coin (one coin)
  - To make 5¢, try
    - 1¢ + 4¢ (1 coin + 1 coin = 2 coins)
    - 2¢ + 3¢ (2 coins + 1 coin = 3 coins)
    - The first solution is better, so best solution is 2 coins
  - To make 6¢, try
    - 1¢ + 5¢ (1 coin + 2 coins = 3 coins)
    - 2¢ + 4¢ (2 coins + 1 coin = 3 coins)
    - 3¢ + 3¢ (1 coin + 1 coin = 2 coins) – best solution
  - Etc.

```java
// m is size of coins array (number of different coins)
// V is the expected amount
static int minCoinsDP(int coins[], int m, int V)
{
    // table[i] will be storing the minimum number of coins
    // required for i value.  So table[V] will have result
    int table[] = new int [V+1];

    // Base case (If given value V is 0)
    table[0] = 0;

    // Initialize all table values as Infinite
    for (int i=1; i<=V; i++)
        table[i] = Integer.MAX_VALUE;

    // Compute minimum coins required for all
    // values from 1 to V
    for (int i=1; i<=V; i++)
    {
        // Go through all coins smaller than i
        for (int j=0; j<m; j++)
          if (coins[j] <= i)
          {
              int sub_res = table[i-coins[j]];
              if (sub_res != Integer.MAX_VALUE && sub_res + 1 < table[i])
                  table[i] = sub_res + 1;
          }
    }
    return table[V];
}
```

```java
public static void main(String args[])
{
   int coins[] =  {25, 21, 10, 5, 1};
   int m = coins.length;
   int V = 63;
   System.out.println("Minimum coins
required: "+ minCoinsDP(coins, m, V));
    }
```

How good is the algorithm?

- The first algorithm is recursive and it takes exponential time, with a large base
- The dynamic programming algorithm is O(N*K), where N is the desired amount and K is the number of different kinds of coins

Q&A