Greedy Algorithms

- Knapsack
- Coin Change
- Huffman Code
- Scheduling

Optimization Problems

- Optimization problem: a problem of finding the best solution from all feasible solutions.

- Two common techniques:
  - Greedy Algorithms
  - Dynamic Programming (global)

Elements of Greedy Strategy

- ***Greedy-choice property***: A global optimal solution can be arrived at by making locally optimal (greedy) choices
- ***Optimal substructure***: an optimal solution to the problem contains within it optimal solutions to sub-problems

Greedy Algorithms

A greedy algorithm works in phases. At each phase:

- You take the best you can get right now, without regard for future consequences
- You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum

Greedy algorithms typically consist of

- A set of ***candidate solutions***
- ***Function*** that checks if the candidates are ***feasible***
- ***Selection function*** indicating at a given time which is the most **promising candidate** not yet used
- ***Objective function*** giving the value of a solution; this is the function we are trying to optimize

Huffman Codes

Text Compression (Zip)

– On a computer: changing the representation of a file so that it takes less space to store or/and less time to transmit.

– Original file can be reconstructed exactly from the compressed representation

- Very effective technique for compressing data, saving 20% - 90%.

First Approach

- Consider the word ABRACADABRA
- How can we write this string in a most economical way?
- Since it has 5 letters, we would need 3 bits to represent each character. For example.

  A = 000
  B = 001
  C = 010
  D = 011
  R = 100
- Since there are 11 letters in ABRACADABRA it requires 33 bits.
- Is there a better way?

Of Course!!

- Magic word: ABRACADABRA
- LET A = 0
  B = 100
  C = 1010
  D = 1011
  R = 11
- Thus, ABRACADABRA = 01001101010010110100110
- So 11 letters demand 23 bits < 33 bits, an improvement of about 30%.

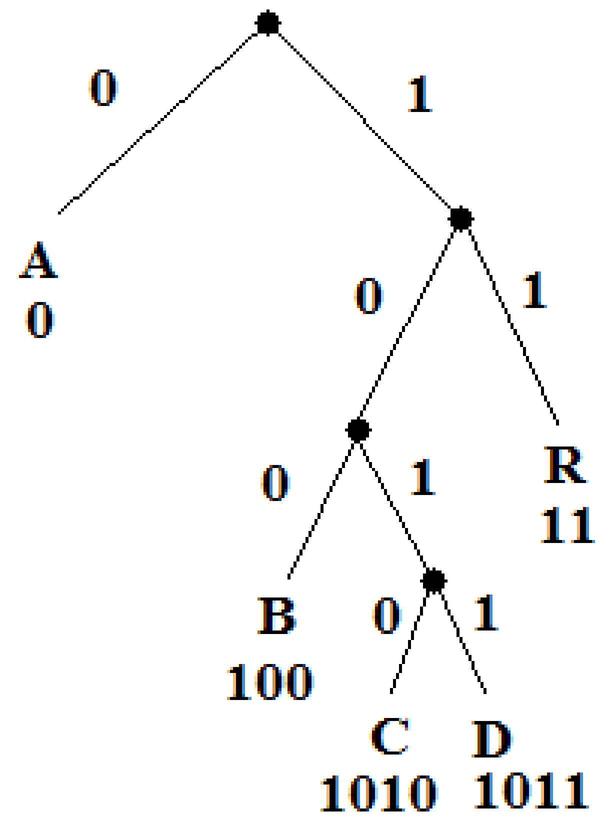
However…

- There are some concerns…
- Suppose we have
  - A-> 01
  - B-> 0101
- If we have 010101, is this AB? BA? Or AAA?
- Therefore: prefix codes, no codeword is a prefix of another codeword, is necessary

Prefix Codes

- Any prefix code can be represented by a full binary tree
- Each leaf stores a symbol.
- Each node has two children – left branch means 0, right means 1.
- codeword = path from the root to the leaf interpreting suitably the left and right branches

For Example



A = 0

B = 100

C = 1010

D = 1011

R = 11

Decoding is unique and simple!

How do we find the optimal coding tree?

- It is clear that the two symbols with the smallest frequencies must be at the bottom of the optimal tree, as children of the lowest internal node

- This is a good sign that we have to use a bottom-up manner to build the optimal code!

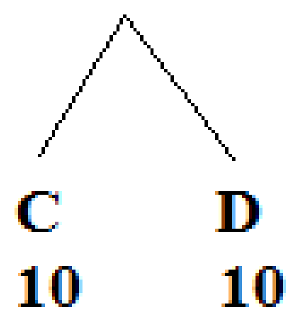
- Huffman's idea is based on a greedy approach, using the previous notices.

Constructing a Huffman Code

- Assume that frequencies of symbols are

  A: 50 B: 15 C: 10 D: 10 R: 18

- Smallest numbers are 10 and 10 (C and D)

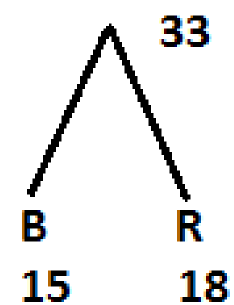


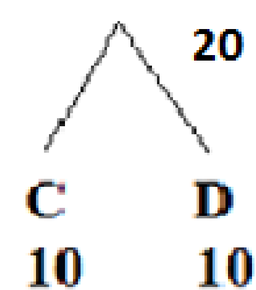
Constructing a Huffman Code

- Now Assume that frequencies of symbols are
  A: 50 B: 15 C+D: 20 R: 18

- C and D have already been used, and the new node above them (call it C+D) has value 20

- The smallest values are B + R

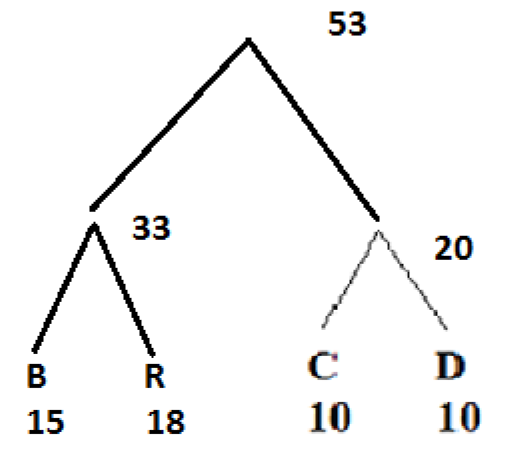




Constructing a Huffman Code

- Now Assume that frequencies of symbols are
  A: 50 B+R: 33 C+D: 20

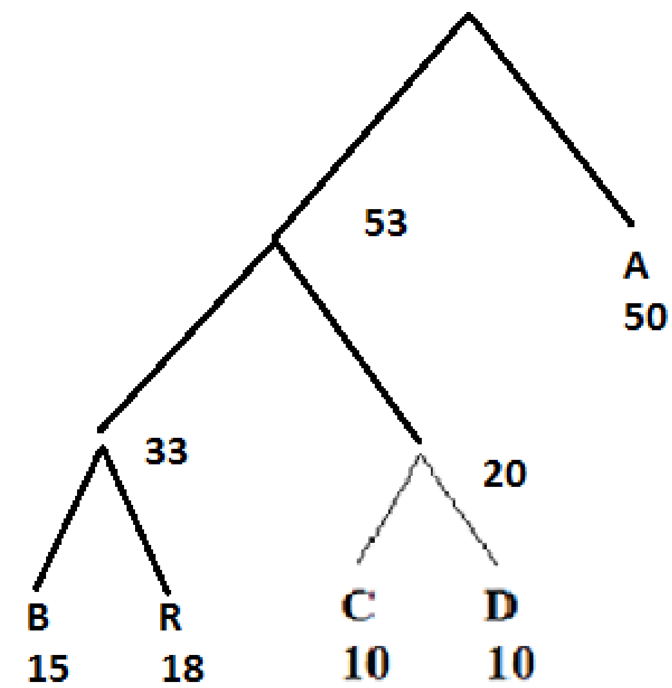
- The smallest values are
  (B + R)+(C+D)=53



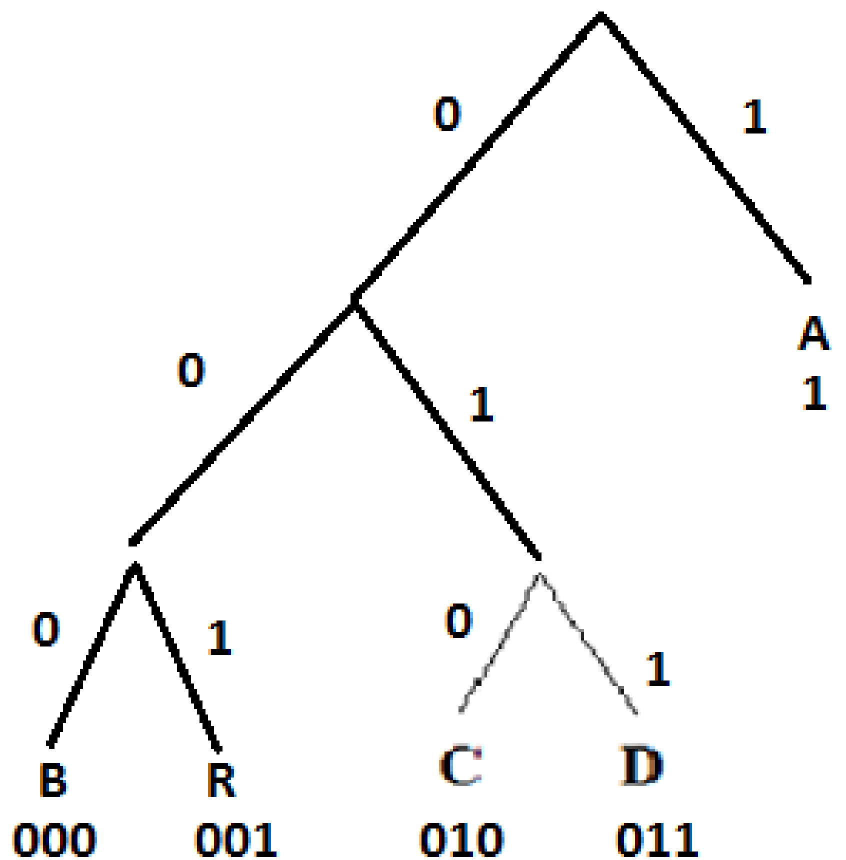
Constructing a Huffman Code

- Now Assume that frequencies of symbols are
  A: 50 (B+R) + (C+D): 53

- The smallest values are
  A+ ((B + R)+(C+D))=103



Constructing a Huffman Code

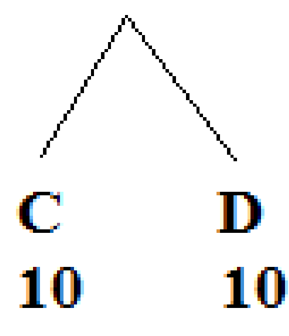


Constructing a Huffman Code

Assume that frequencies of symbols are

A: 50 B: 20 C: 10 D: 10 R: 30

Smallest numbers are 10 and 10 (C and D)

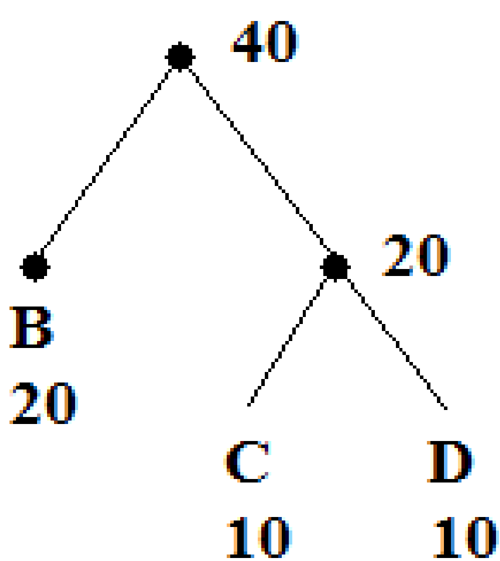


Constructing a Huffman Code

Assume that frequencies of symbols are

A: 50 B: 20 C: 10 D: 10 R: 30

- C and D have already been used, and the new node above them (call it C+D) has value 20
- The smallest values are B, C+D

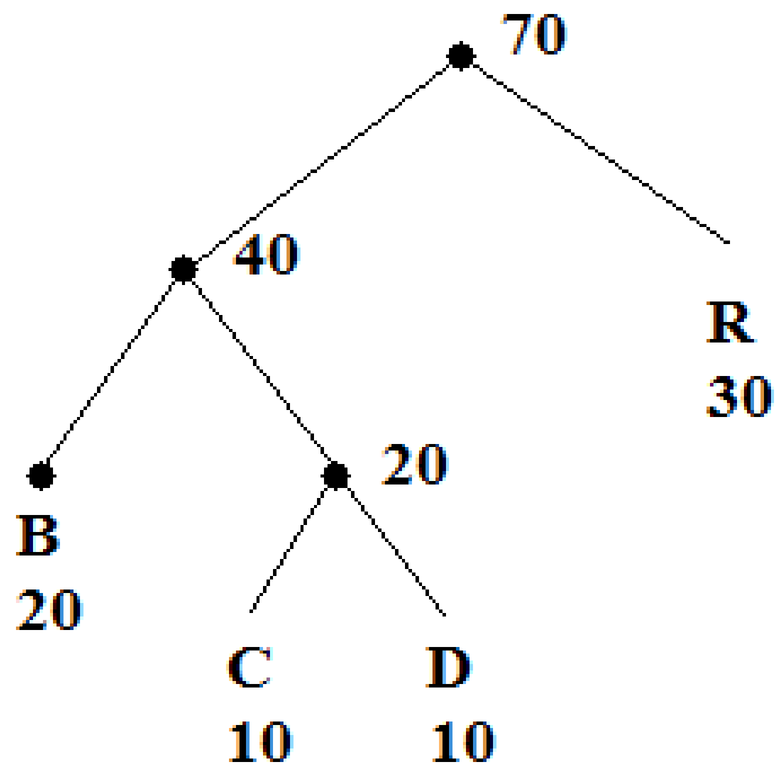


Constructing a Huffman Code

Assume that frequencies of symbols are

A: 50 B: 20 C: 10 D: 10 R: 30

Next, B+C+D (40) and R (30)

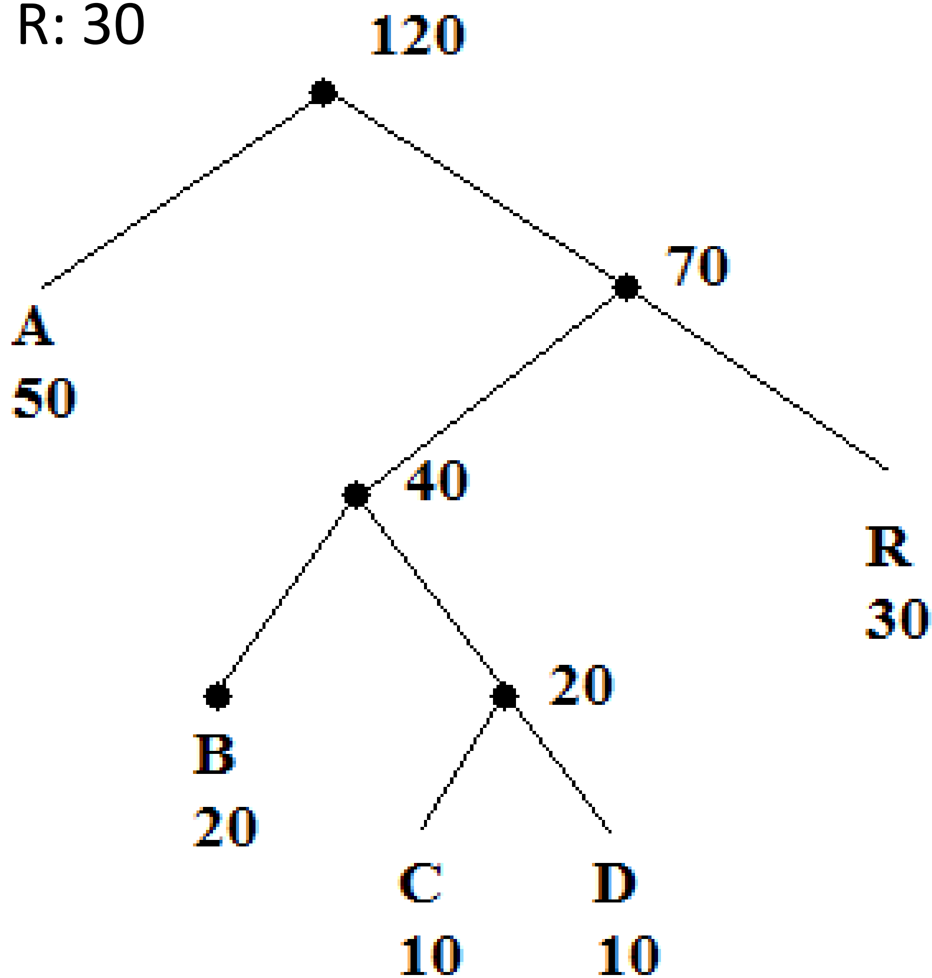


Constructing a Huffman Code

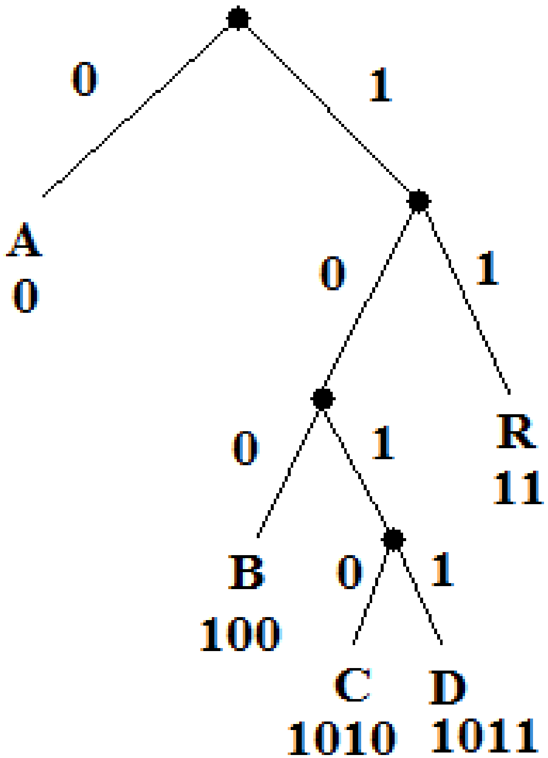
Assume that frequencies of symbols are

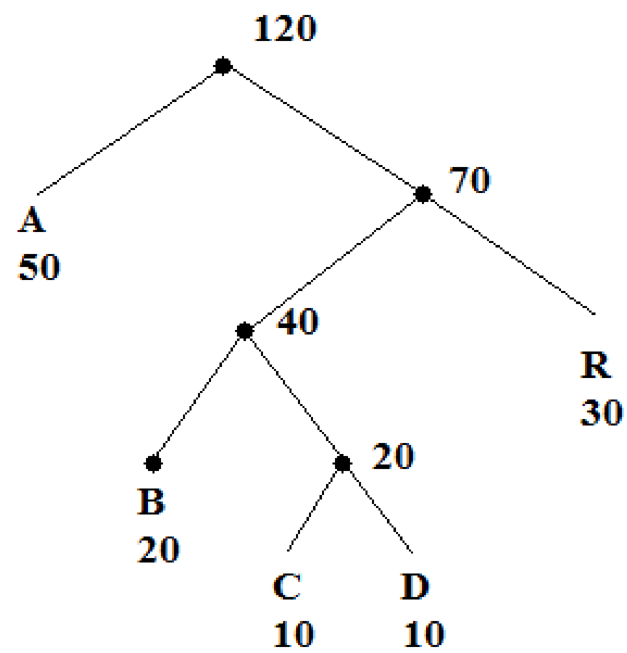
A: 50 B: 20 C: 10 D: 10 R: 30

Finally



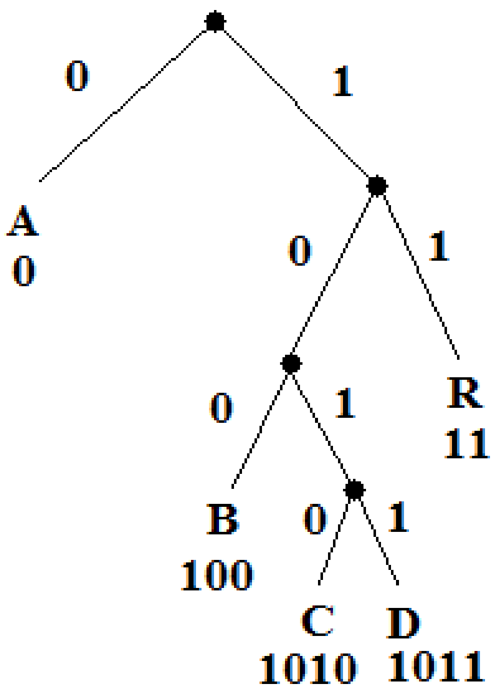
Constructing a Huffman Code





Decode the tree

- Suppose we have the Following code: 10001011



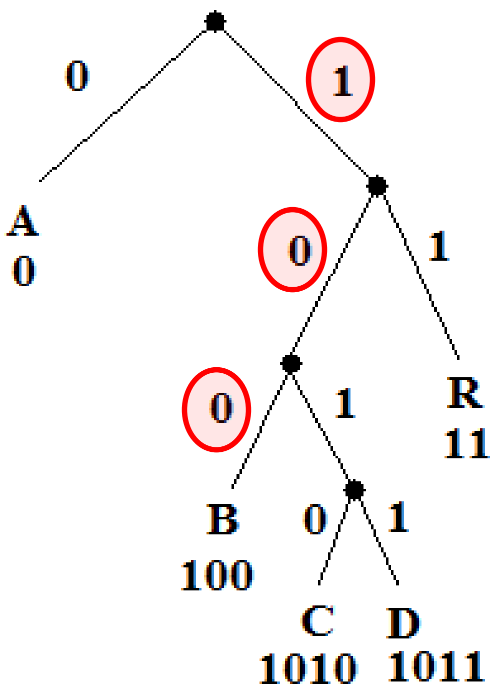
- What is the decode result?

Decode the tree

- Suppose we have the Following code: **100**01011

- What is the decode result?

B

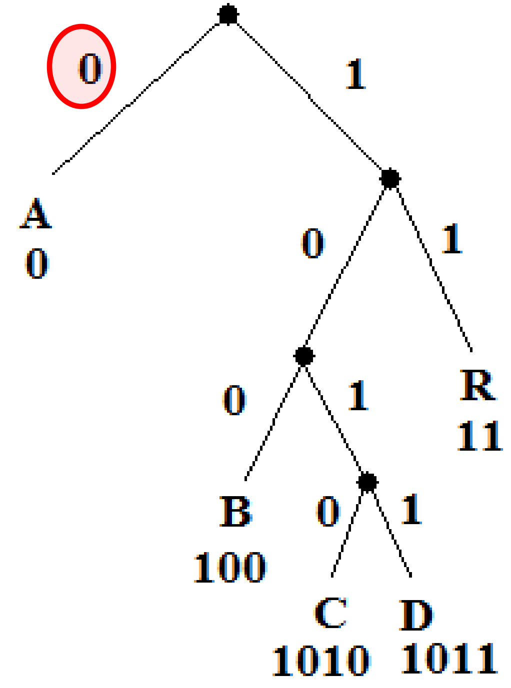


Decode the tree

- Suppose we have the Following code: 10001011

- What is the decode result?

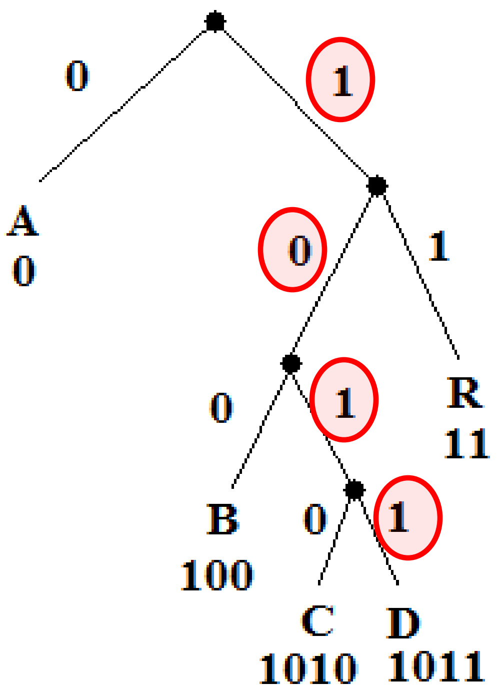
BA



Decode the tree

- Suppose we have the Following code: 10001011
- What is the decode result?

BAD



Decode the tree

- Suppose we have the

Following code:

10001011

- What is the decode

result? BAD