

Consider the coin-change problem: Given a set of coin types and an amount of change to be returned, determine the **fewest number** of coins for this amount of change.

1) What "greedy" algorithm would you use to solve this problem with US coin types of {1, 5, 10, 25, 50} and a change amount of 29-cents?

2) Do you get the correct solution if you use this algorithm for coin types of {1, 5, 10, 12, 25, 50} and a change amount of 29-cents?

3) One way to solve this problem in general is to use a divide-and-conquer algorithm. Recall the idea of **Divide-and-Conquer** algorithms.

Solve a problem by:

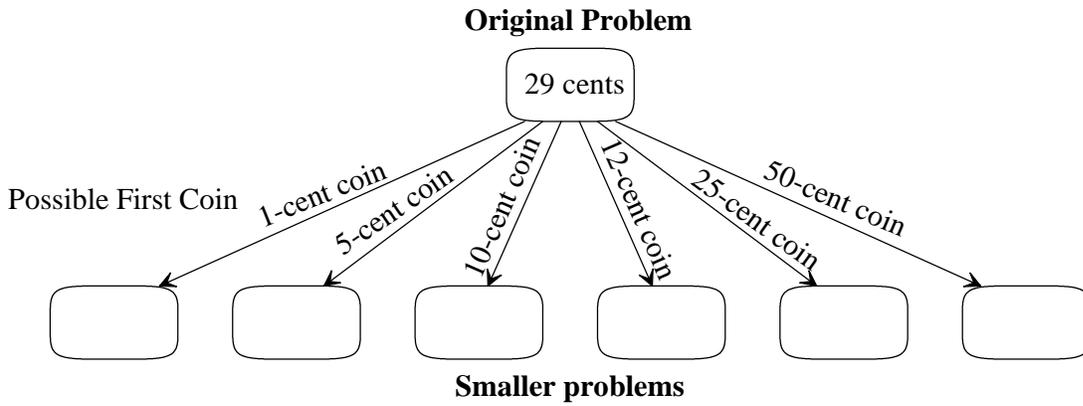
- dividing it into smaller problem(s) of the same kind
- solving the smaller problem(s) recursively
- use the solution(s) to the smaller problem(s) to solve the original problem

a) For the coin-change problem, what determines the size of the problem?

b) How could we divide the coin-change problem for 29-cents into smaller problems?

c) If we knew the solution to these smaller problems, how would be able to solve the original problem?

4) After we give back the first coin, which smaller amounts of change do we have?

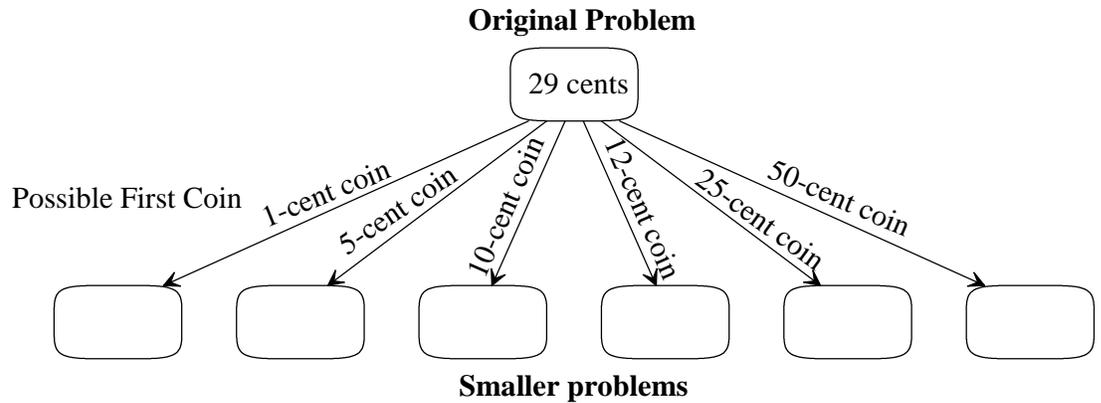


5) If we knew the fewest number of coins needed for each possible smaller problem, then how could determine the fewest number of coins needed for the original problem?

6) Complete a recursive relationship for the fewest number of coins.

$$\text{FewestCoins}(\text{change}) = \begin{cases} \min_{\text{coin} \in \text{CoinSet}} (\text{FewestCoins}(\text{change} - \text{coin})) + 1 & \text{if change} \in \text{CoinSet} \\ \infty & \text{if change} \notin \text{CoinSet} \end{cases}$$

7) Complete a couple levels of the recursion tree for 29-cents change using the set of coin types {1, 5, 10, 12, 25, 50},



8) As in the maze problem of homework #2 we could use backtracking and a stack to maintain the list of unexplored choices, but we can also use recursion (with its run-time stack) to drive a backtracking algorithm. The general recursive backtracking algorithm for optimization problems (e.g., fewest number of coins) looks something like:

```
Backtrack( recursionTreeNode p ) {
    treeNode c;
    for each child c of p do
        if promising(c) then
            if c is a solution that's better than best then
                best = c
            else
                Backtrack(c)
            end if
        end if
    end for
} // end Backtrack
```

General Notes about Backtracking:

- the depth-first nature of backtracking only stores information about the current branch being explored so the memory usage is “low”
- Each node of the search space tree maintains the state of a partial solution. In general the state consists of potentially large arrays that change little between parent and child. To avoid having multiple copies of these arrays, a single “global” state is maintained which is updated before we go down to the child (via a recursive call) and undone when we backtrack to the parent.

For the coin-change problem:

a) What defines the current state of a search-space tree node?

b) When would a “child” search-space tree node NOT be promising?

## Coin-change backtracking code in Python:

```
# global current state of the backtrack
numberOfEachCoinType = []
numberOfCoinsSoFar = 0
solutionFound = False
bestFewestCoins = -1
bestNumberOfEachCoinType = None

# profiling
backtrackingNodes = 0

def main():
    changeAmt = int(raw_input("Enter the change amount: "))
    coinTypes = raw_input("Enter the coin types separated by spaces: ")
    coinTypes = coinTypes.split(" ")
    for index in xrange(len(coinTypes)):
        coinTypes[index] = int(coinTypes[index])
    print "Change Amount:",changeAmt," Coin types:", coinTypes
    (fewestCoins, numberOfEachCoinType) = solveCoinChange(changeAmt, coinTypes)
    print "Fewest number of coins",fewestCoins
    print "The number of each type of coin in the solution is:"
    for index in xrange(len(coinTypes)):
        print "number of %s-cent coins is %s"%(str(coinTypes[index]),
                                                str(numberOfEachCoinType[index]))

    return

def solveCoinChange(changeAmt, coinTypes):

    def backtrack(changeAmt):
        global numberOfEachCoinType
        global numberOfCoinsSoFar
        global solutionFound
        global bestFewestCoins
        global bestNumberOfEachCoinType
        global backtrackingNodes
        backtrackingNodes += 1

        for index in range(len(coinTypes)):
            smallerChangeAmt = changeAmt - coinTypes[index]
            if promising(smallerChangeAmt, numberOfCoinsSoFar+1):
                if smallerChangeAmt == 0: # a solution is found
                    # check if its best
                    if (not solutionFound) or numberOfCoinsSoFar + 1 < bestFewestCoins:
                        bestFewestCoins = numberOfCoinsSoFar+1
                        bestNumberOfEachCoinType = [] + numberOfEachCoinType
                        bestNumberOfEachCoinType[index] += 1
                        solutionFound = True
                else:
                    # update global "current state" for child before call
                    numberOfCoinsSoFar += 1
                    numberOfEachCoinType[index] += 1

                    backtrack(smallerChangeAmt)

                    # undo change to global "current state" after backtracking
                    numberOfCoinsSoFar -= 1
                    numberOfEachCoinType[index] -= 1
            # end def backtrack
```

```

def promising(changeAmt, numberOfCoinsReturned):
    if changeAmt < 0:
        return False
    elif changeAmt == 0:
        return True
    else: # changeAmt > 0
        if solutionFound and numberOfCoinsReturned+1 >= bestFewestCoins:
            return False
        else:
            return True

# set-up initial "current state" information
global numberOfEachCoinType
global numberOfCoinsSoFar
global solutionFound
global bestFewestCoins
global bestNumberOfEachCoinType

numberOfEachCoinType = []
for coin in coinTypes:
    numberOfEachCoinType.append(0)
numberOfCoinsSoFar = 0
solutionFound = False
bestFewestCoins = -1
bestNumberOfEachCoinType = None

backtrack(changeAmt)
return (bestFewestCoins, bestNumberOfEachCoinType)

main()
print "Number of Backtracking Nodes:", backtrackingNodes

```