WIKIPEDIA
25 years of the free encyclopedia

Change-making problem

The **change-making problem** addresses the question of finding the minimum number of coins (of certain denominations) that add up to a given amount of money. It is a special case of the integer knapsack problem, and has applications wider than just currency.

It is also the most common variation of the *coin change problem*, a general case of partition in which, given the available denominations of an infinite set of coins, the objective is to find out the number of possible ways of making a change for a specific amount of money, without considering the order of the coins.

It is weakly NP-hard, but may be solved optimally in pseudo-polynomial time by dynamic programming.[1][2]

Mathematical definition

Coin values can be modeled by a set of n distinct positive integer values (whole numbers), arranged in increasing order as w_1 through w_n. The problem is: given an amount W, also a positive integer, to find a set of non-negative (positive or zero) integers $\{x_1, x_2, ..., x_n\}$, with each x_j representing how often the coin with value w_j is used, which minimize the total number of coins $f(W)$

$$f(W) = \sum_{j=1}^{n} x_j$$

subject to

$$\sum_{j=1}^{n} w_j x_j = W.$$

Non-currency examples

An application of change-making problem can be found in computing the ways one can make a nine dart finish in a game of darts.

Another application is computing the possible atomic (or isotopic) composition of a given mass/charge peak in mass spectrometry.

Methods of solving

Simple dynamic programming

A classic dynamic programming strategy works upward by finding the combinations of all smaller values that would sum to the current threshold.[3] Thus, at each threshold, all previous thresholds are potentially considered to work upward to the goal amount *W*. For this reason, this dynamic programming approach requires a number of steps that is O(*nW*), where *n* is the number of types of coins.

Implementation

The following is a dynamic programming implementation (with Python 3) which uses a matrix to keep track of the optimal solutions to sub-problems, and returns the minimum number of coins, or "Infinity" if there is no way to make change with the coins given. A second matrix may be used to obtain the set of coins for the optimal solution.

```python
def _get_change_making_matrix(set_of_coins, r: int):
    m = [[0 for _ in range(r + 1)] for _ in range(len(set_of_coins) + 1)]
    for i in range(1, r + 1):
        m[0][i] = float("inf")  # By default there is no way of making change
    return m

def change_making(coins, n: int):
    """This function assumes that all coins are available infinitely.
    if coins are only to be used once, change m[c][r - coin] to m[c - 1][r - coin].
    n is the number to obtain with the fewest coins.
    coins is a list or tuple with the available denominations.
    """
    m = _get_change_making_matrix(coins, n)
    for c, coin in enumerate(coins, 1):
        for r in range(1, n + 1):
            # Just use the coin
            if coin == r:
                m[c][r] = 1
            # coin cannot be included.
            # Use the previous solution for making r,
            # excluding coin
            elif coin > r:
                m[c][r] = m[c - 1][r]
            # coin can be used.
            # Decide which one of the following solutions is the best:
            # 1. Using the previous solution for making r (without using coin).
            # 2. Using the previous solution for making r - coin (without
            #      using coin) plus this 1 extra coin.
            else:
                m[c][r] = min(m[c - 1][r], 1 + m[c][r - coin])
    return m[-1][-1]
```

Greedy method

For many real-world coin systems, such as those used in the US and many other countries, a greedy algorithm of picking the largest denomination of coin which is not greater than the remaining amount to be made will produce the optimal result. This is not the case for arbitrary coin systems or even some real world systems, though. For instance, if we consider the old (now withdrawn) Indian coin denominations of 5, 10, 20 and 25 paise, then to make 40 paise, the greedy algorithm would choose three coins (25, 10, 5) whereas the optimal solution is two coins (20, 20). Another example is attempting to make 40 US cents without nickels (denomination 25, 10, 1) with similar result — the greedy chooses seven coins (25, 10, and 5 × 1), but the optimal is four (4 × 10). A coin system is called "canonical" if the greedy algorithm always solves its change-making problem optimally. It is possible to test whether a coin system is canonical in polynomial time.[4]

Related problems

The "optimal denomination problem"[5] is a problem for people who design entirely new currencies. It asks what denominations should be chosen for the coins in order to minimize the average cost of making change, that is, the average number of coins needed to make change. The version of this problem assumed that the people making change will use the minimum number of coins (from the denominations available). One variation of this problem assumes that the people making change will use the "greedy algorithm" for making change, even when that requires more than the minimum number of coins. Most current currencies use a 1-2-5 series, but some other set of denominations would require fewer denominations of coins or a smaller average number of coins to make change or both.

See also

- List of knapsack problems
- Coin problem
- The coin collector's problem

References

1. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). *Introduction to Algorithms*. MIT Press. Problem 16-1, p. 446.
2. Goodrich, Michael T.; Tamassia, Roberto (2015). *Algorithm Design and Applications*. Wiley. Exercise A-12.1, p. 349.
3. Wright, J. W. (1975). "The change-making problem" (https://doi.org/10.1145%2F321864.321874). *Journal of the Association for Computing Machinery*. **22** (1): 125–128. doi:10.1145/321864.321874 (https://doi.org/10.1145%2F321864.321874). S2CID 22568052 (https://api.semanticscholar.org/CorpusID:22568052).
4. Pearson, David (2005). "A polynomial-time algorithm for the change-making problem". *Operations Research Letters*. **33** (3): 231–234. doi:10.1016/j.orl.2004.06.001 (https://doi.org/10.1016%2Fj.orl.2004.06.001). hdl:1813/6219 (https://hdl.handle.net/1813%2F6219). MR 2108270 (https://mathscinet.ams.org/mathscinet-getitem?mr=2108270).
5. J. Shallit (2003). "What this country needs is an 18c piece" (https://www.cs.uwaterloo.ca/~shallit/Papers/change2.pdf) (PDF). *Mathematical Intelligencer*. **25** (2): 20–23. doi:10.1007/BF02984830 (https://doi.org/10.1007%2FBF02984830). S2CID 123286384 (https://api.semanticscholar.org/CorpusID:123286384).

Further reading

- M. Adamaszek, A. Niewiarowska (2010). "Combinatorics of the change-making problem". *European Journal of Combinatorics*. **31** (1): 47–63. arXiv:0801.0120 (https://arxiv.org/abs/0801.0120). doi:10.1016/j.ejc.2009.05.002 (https://doi.org/10.1016%2Fj.ejc.2009.05.002). S2CID 13527488 (https://api.semanticscholar.org/CorpusID:13527488).