

1 Making Change

Suppose we want to make change for \( n \) cents using pennies (1 cent), nickels (5 cents), dimes (10 cents), and quarters (25 cents), but no other denomination. The goal is to find a combination of coins of these types with total value \( n \) that minimizes the number of coins used. Give a \( O(n) \) time algorithm for finding such an allocation of coins.

Question 1. What algorithmic strategy that we’ve seen so far seems most promising to solve this problem?
Answer: Greedy is a natural first choice. There does not seem to be an efficient naïve algorithm for this problem, so divide-and-conquer generally doesn’t look promising. Dynamic programming will find an optimal solution – it is only more general – but of course greedy is usually more efficient when it works, so it is more likely to yield the \( O(n) \) running time.

Question 2. What is a plausible algorithms of this form for this problem? Is it \( O(n) \)?
Answer: A natural candidates for a greedy algorithm is as follows:

Suppose the amount left to change is \( m \). Add to the solution the largest-denomination coin whose value is no more than \( m \), subtract this coin’s value from \( m \), and repeat until \( m = 0 \).

For the running time, since each iteration of the loop is \( O(1) \) time and reduces \( m \) by at least 1, the algorithm runs in time \( O(n) \), which is sufficient.

Question 3. Is the proposed algorithm optimal if the only available denominations for making change are 1, 10, and 25 cents? What makes this case behave differently from the problem we posed with four denominations?
Answer: The algorithm is definitely not optimal here. For \( n = 30 \), the algorithm will take a quarter (25 cents) and then be stuck taking five pennies (1 cent each) whereas three dimes would suffice. Of course, adding the nickel back defeats this counterexample, since a quarter and a nickel is better than using three dimes.
**Question 4.** For the actual denominations we are considering, what property of the choices the algorithm is making that could help establish that it is optimal?

*Answer:* one such property is as follows. Let’s order the set of denominations from largest to smallest value. Now given two sets of coins which sum up to some total value $m$, consider the biggest denomination at which these two sets differ. We must prove that the set which has more coins of this denomination has fewest coins.

**Question 5.** How can you prove this property? *Hint:* you will probably need to consider a number of cases separately. What is unusual about your argument?

*Answer:* There are cases depending on the largest denomination in which the sets differ:

- Clearly, if we use the same number of quarters and dimes, then we can improve a solution that uses too few nickels by trading five pennies for a nickel, so such a solution can’t be optimal.

- Let’s next suppose that the solution has the same number of quarters, but too few dimes. Then if there are more than two nickels, we can trade them for a dime, reducing the number of coins by one, or there is a nickel and five pennies, in which case we can reduce the number by five by trading these for the dime, or there are at least ten pennies, in which case we can reduce the number by nine by trading these for the dime. In any case, the solution that uses too few dimes is sub-optimal.

- Finally, let’s consider a solution that uses too few quarters. We’ve already established that among the smaller denominations, any optimal solution must use as many dimes as possible, then as many nickels as possible, with the rest being covered by fewer than five pennies; any solution that does not use this allocation for the smaller denominations can be improved by some exchange with the smaller denominations, and hence is surely not optimal. Now, since the original solution used fewer quarters than possible, these smaller denomination coins total at least 25 cents, so in an optimal allocation of these smaller denomination coins, there must be at least two dimes available, and either a third dime or a nickel. We can exchange these, respectively, for either a quarter and a nickel or a quarter (respectively), reducing the number of coins used by at least one, and thus the solution that uses too few quarters is not optimal.

What is unusual is that the “inductive steps” (the exchange we make) depends on not just the denomination of the coin, but also the value remaining.

**Question 6.** Now, how can you use this property to prove that the algorithm under consideration is optimal?
2 Coverage with the fewest towers

Suppose you work for the cellular provider ScrewNET, “bringing people together for the lowest price™.” You are looking to extend your network’s coverage out along the one road through a sparsely populated rural area. Surveying has provided you with a set of \( n \) candidate locations along the road, at mile numbers \( \ell_1, \ell_2, \ldots, \ell_n \) where you could build the towers. Along the way, there are residences at mile numbers \( r_1, r_2, \ldots, r_m \). Let’s suppose that a cell tower can serve a residence as long as the residence is within 10 miles of the cell tower; assume the road is roughly straight, for simplicity, so the cell tower can serve the residence if and only if the mile numbers are within 10 of one another. It turns out that every residence is within the range of some candidate location for a cell tower. Your task is to give an algorithm for finding the smallest set of towers that can serve all of the residences.

**Question 1.** What algorithmic strategy that we’ve seen so far seems most promising to solve this problem?

**Answer:** Greedy is a natural choice. There does not seem to be an efficient naïve algorithm for this problem, so divide-and-conquer generally doesn’t look promising. Dynamic programming – for those that have seen it – is another candidate, but of course greedy is simpler (and often more efficient, when it works).

**Question 2.** What are some plausible algorithms of this form for this problem?

**Answer:** Some natural candidates for greedy algorithms might be to add the first location that serves some unserved residence; to choose the location that serves the most currently-unserved residences; to choose a location that serves some residence furthest from the current service area; and to choose the furthest location that serves the first unserved residence. (You might have others.)

**Question 3.** Which of these algorithms can you rule out by constructing a counterexample?
Question 4. What might be true of the remaining plausible algorithm that could help establish that it is optimal?

Answer: Similarly to the analysis of the earliest-finish algorithm for interval scheduling, we might establish that the algorithm that chooses the furthest location that serves the first unserved residence covers all residences up to a larger mile $m$ than any other set of locations.

Question 5. How can you prove this property?

Answer: Again, by induction on the tower locations in increasing mile numbers: suppose that some other set $O$ that serves all residences contains locations $\ell_{j_1}, \ldots, \ell_{j_u}$ and your set contains locations $\ell_{i_1}, \ldots, \ell_{i_u}$ in increasing order. Then we can show that for the first $w$ locations, $\ell_{i_w} \geq \ell_{j_w}$ by induction on $w$: indeed, we choose $i_1$ so that $\ell_{i_1}$ is the largest mile number of a location that serves the first residence, and hence since $j_1$ must be at or before the mile number of some location in $O$ that serves the first residence, $\ell_{i_1} \geq \ell_{j_1}$. Now, supposing that this is true of the first $w$ locations in $O$ and $A$, we see that since $\ell_{i_w} \geq \ell_{j_w}$, neither one has served the next residence $r^*$ that we choose a tower to cover in $A$ using $\ell_{i_{w+1}}$. Since $O$ must also choose a tower to over this residence, and $\ell_{i_{w+1}}$ is the furthest location that covers $r^*$, the next location in $O$ cannot be further than $\ell_{i_{w+1}}$, or else $r^*$ would not be served by any tower in $O$. Thus, $\ell_{i_{w+1}} \geq \ell_{j_{w+1}}$, as needed.

Question 6. Now, how can you use this property to prove that the algorithm under consideration is optimal?

Answer: We want to prove that the greedy algorithm will choose fewer or the same number of towers as any other solution. Suppose not (for contradiction). Say greedy algorithm chose $k$ towers and some other solution chose $m$ towers where $k > m$. Look at the first $m$ towers chosen by the greedy solution — these towers must cover further locations than the $m$ towers of the other solution. Therefore, these towers must already cover everything the other solution could have covered, which is everything. Therefore, the greedy solution would not choose the extra $k - m$ towers reaching a contradiction on the definition of $k$. 