This note gives an example of the design pattern introduced in the note on ACTIVITY SELECTION, for designing a greedy algorithm by finding a first choice that satisfies the greedy invariant, analyzing the recursive algorithm using optimal substructure, and analyzing the iterative algorithm using the greedy invariant.

Here we consider the problem of MAKING CHANGE (for a given total, using given denominations, using the minimum number of coins). We consider the largest-coin-first algorithm, and show that it works for U.S. denominations. We analyze a recursive formulation of the algorithm using greedy-choice and optimal-substructure lemmas, then analyze an iterative version of the algorithm using the greedy invariant.

MAKING CHANGE

input: A pair $I = (D, T)$ where $D = \{d_1, d_2, \ldots, d_k\}$ is a set of *denominations*, and T is a target (all non-negative integers, with $1 \in D$).

output: A minimum-size sequence $C = (c_1, \ldots, c_n)$ of *coins* s.t. $c_i \in D$ for all i and $\sum_i c_i = T$.

Given an instance I, define a sequence C to be a *feasible solution (for I)* if it meets the constraints ($c_i \in D$ for each i and $\sum_{i=1}^{n} c_i = T$). A feasible solution is a correct solution if it has minimum size. Let $\mathsf{opt}(I)$ be the minimum size of any feasible solution: $\mathsf{opt}(I) = \min\{|C| : C \textit{ is feasible for } I\}$. Given any feasible solution C, say that C is *optimal (for I)* if $|C| = \mathsf{opt}(I)$. So, given I, the problem is to find a feasible solution of minimum size, that is, an optimal solution.

A greedy algorithm for this problem should maintain a sequence X of coins, initially empty, then add coins one by one until the target is reached, hopefully maintaining the greedy invariant (as it specializes here) throughout. It should then return the final sequence X.

Here we consider the *largest-coin-first* greedy choice. Given an instance $I = (D, T)$, it chooses the first coin $x_1 = \max\{d_i \in D : d_i \le T\}$ to have maximum value not exceeding T. The recursive algorithm chooses this first coin x_1, then recurses to choose the rest (which must total $T - x_1$).

$\mathsf{change}\big(I = (D = \{d_1, d_2, \ldots, d_k\}, T)\big)$:	— *recursive algorithm for* MAKING CHANGE —
1. if $T = 0$: return ().	
2. let $x_1 = \max\{d_i \in D : d_i \le T\}$	*(A coin of maximum value among those not exceeding T.)*
3. let $Y = (y_1, y_2, \ldots, y_k) = \mathsf{change}(D, T - x_1)$	*(Recursively compute change for $T - x_1$.)*
4. return $X = (x_1, y_1, y_2, \ldots, y_k)$	*(The sequence obtained by prepending x_1 to Y.)*

To build intuition, consider examples. Try different denominations. Find an example for which the algorithm fails!

In what follows, we consider only the following U.S. denominations: $D = \{1, 5, 10, 25\}$, called *pennies*, *nickels*, *dimes*, and *quarters*. Our goal is to show that the algorithm works for these denominations and any target T. We start by showing that the first step chooses a coin that is in some optimal solution. First we prove a utility lemma:

Lemma 1. *Let $I = (D, T)$ be any instance with $D = \{1, 5, 10, 25\}$ and any total $T > 0$. Let C be any optimal solution. Then*

(i) C has at most 4 pennies.
(ii) C has at most 1 nickel.
(iii) C has at most 2 dimes.
(iv) C has at most 2 dimes or nickels.

Proof (short form). Property (i) holds, because otherwise exchanging 5 pennies for a nickel gives a smaller feasible solution, contradicting the optimality of C. Property (ii) holds — otherwise replacing 2 nickels by a dime gives a better solution. Property (iii) holds, because otherwise replacing 3 dimes by a quarter and nickel improves the solution. Property (iv) holds — otherwise, by (ii) and (iii), C has 1 nickel and 2 dimes, and replacing them by a quarter would give a smaller feasible solution. □

Now we prove that the first coin x_1 selected by the algorithm is always in some optimal solution.

Lemma 2 (greedy choice)**.** *For $D = \{1, 5, 10, 25\}$ and any total $T > 0$, the coin $x_1 = \max\{d_i \in D : d_i \leq T\}$ is in some optimal solution C.*

Proof (short form). Let C be any optimal solution (one clearly exists).

We consider cases, depending on the value of $x_1 \in \{1, 5, 10, 25\}$.

In the case that $x_1 = 1$ (the first coin is a penny), by inspection of the algorithm the total T must satisfy $1 \leq T \leq 4$. So the optimal solution C consists of T pennies. So $x_1 = 1 \in C$.

In the case $x_1 = 5$ (the first coin is a nickel) the total T satisfies $5 \leq T \leq 9$. The optimal solution C contains only pennies and nickels, because the total value of C is $T \leq 9$. By Lemma 1 (i), C contains at most 4 pennies. Since $T > 4$, there must be a nickel in C. So $x_1 = 5 \in C$.

In the case that $x_1 = 10$, the total T satisfies $10 \leq T \leq 24$. The optimal solution C contains no quarters ($c_i \in \{1, 5, 10\}$), because the total value of C is $T \leq 24$. By Lemma 1 (i) and (ii), the total value of the pennies and nickels in C is at most $4 + 5 = 9$. Since $T \geq 10$, it follows that C must contain a dime. So $x_1 = 10 \in C$.

In the case that $x_1 = 25$, the total T must satisfy $25 \leq T$. By Lemma 1 (i), the total value of the pennies in C is at most 4. By Lemma 1 (iii), the total value of the nickels and dimes is at most $2 \cdot 10 = 20$. So the total value of the coins in C other than quarters is at most $4 + 20 = 24$. Since $T \geq 25$, it follows that C must contain a quarter. So $x_1 = 25 \in C$. □

We leave it as an exercise to give long-form proofs for the two lemmas above.

Extending from the first step. By Lemma 2, given denominations $D = \{1, 5, 10, 25\}$, the algorithm's choice of x_1 maintains the greedy invariant at the end of the first iteration. That is, by the lemma, there is an optimal solution of the form $C = (x_1, y_1, y_2, \ldots, y_k)$ for some $(y_1, \ldots, y_k)$. So the remaining problem is to compute $Y = (y_1, \ldots, y_k)$ so that $C = (x_1, y_1, \ldots, y_k)$ is optimal.

To make C feasible, Y should satisfy $\sum_{i=1}^{k} y_i = T - x_1$. Subject to that contraint, to make C optimal, Y should have minimum size k. In other words, Y should be an optimal solution for the MAKING CHANGE instance $I' = (D, T')$ where $T' = T - x_1$. Next we prove that this suffices.

Lemma 3 (optimal substructure)**.** *Let instance $I = (D, T)$ (with $T > 0$) be as defined above. Let $x_1 = \max\{d_i \in D : d_i \leq T\}$. Let $Y = (y_1, \ldots, y_k)$ be any optimal solution for the* MAKING CHANGE *instance $I' = (D, T')$ where $T' = T - x_1$. Then $C = (x_1, y_1, y_2, \ldots, y_k)$ is an optimal solution for I.*

Proof (long form).

1. Suppose for contradiction that $C = (x_1, y_1, y_2, \ldots, y_k)$ is not optimal for I.
2. Since Y is feasible for I', each y_i is in D, and $\sum_{i=1}^{k} y_i = T' = T - x_1$.
3. From this it follows that C is also feasible for I.
4. Since C is not optimal, any optimal solution for I is smaller.
5. Let $C^* = (x_1, z_1, \ldots, z_\ell)$ be an optimal solution for I with first coin x_1 (C^* exists by Lemma 2.)
6. Let $Z = (z_1, z_2, \ldots, z_\ell)$ be obtained from C^* by removing the first coin x_1.
7. Because C^* is feasible for I, it follows that Z is feasible for I' (verify).
8. Since C^* is smaller than C, it follows that Z is smaller than Y.
9. This contradicts the optimality of Y for I'. □

The converse also holds: if $(x_1, y_1, y_2, \ldots, y_k)$ is any optimal solution for I then $(y_1, y_2, \ldots, y_k)$ is optimal for I'. This is because the feasible solutions for I that start with x_1 correspond one-to-one with the feasible solutions for I', via the bijection $(x_1, y_1, y_2, \ldots, y_k) \leftrightarrow (y_1, y_2, \ldots, y_k)$.

The correctness of the recursive algorithm follows from Lemma 3 by a simple induction.

Theorem 1 (correctness of recursive algorithm)**.** *Given any instance $I = (D, T)$ with $D = \{1, 5, 10, 25\}$, the recursive algorithm returns an optimal solution.*

Proof (long form).

1. The proof is by induction on T.
2. For the proof, since $D = \{1, 5, 10, 25\}$, we identify any instance $I' = (D, T')$ by just its total T'.
3. For total $T = 0$, the only solution is (), so the algorithm is correct.

4.1. Consider any total $T > 0$.

4.2.1. Assume that the algorithm is correct for all totals $T' < T$.

4.2.2. Consider the execution of the algorithm on I.

4.2.3. Let x_1 and $Y = (y_1, \ldots, y_k)$ be as computed by the algorithm in Lines 2 and 3.

4.2.4. By the inductive assumption, Y is optimal for the total $T' = T - x_1$.

4.2.5. So, by Lemma 3, the solution $X = (x_1, y_1, y_2, \ldots, y_k)$ returned by change(I) is optimal for T.

4.3. By Block 4.2, if change is correct for all totals $T' < T$, then it is correct for T.

5. By Block 4, for any $T \geq 1$, if change is correct for all totals $T' < T$, then change is correct for T.
6. By Step 3, change is correct for $T = 0$, so, inductively, change is correct for all totals $T \geq 0$. □

We proved correctness of the recursive algorithm using the greedy-choice / optimal-substructure approach. Here is the iterative version, obtained by unwinding the tail-recursion:

ichange$\big(I = (D = \{d_1, d_2, \ldots, d_k\}, T)\big)$: — *iterative algorithm for* MAKING CHANGE —

1. initialize $X \leftarrow ()$ and $t \leftarrow 0$
2. while $\sum_{i=1}^{t} x_i \neq T$:

3.1. $\quad t \leftarrow t + 1$

3.2. $\quad x_t \leftarrow \max\{d_i \in D : d_i \leq T - \sum_{i=1}^{t-1} x_i\}$ *(Add next piece to partial solution.)*

4. return $X = (x_1, x_2, \ldots, x_t)$

To be correct, the algorithm must maintain the generic greedy invariant, which in this case is

greedy invariant: *The current sequence $X = (x_1, x_2, \ldots, x_t)$ is a prefix of some optimal solution.*

The invariant is true before the first iteration (when $t = 0$ and $X = ()$), and by Lemma 2 the first step maintains the invariant. We generalize Lemma 2 to show that every step does.

Lemma 4 (invariant (generalizes Lemma 2))**.** *When run on any instance $I = (D, T)$ with $D = \{1, 5, 10, 25\}$, the iterative algorithm maintains the greedy invariant.*

Proof (long form).

1. Consider the execution of the algorithm on any instance $I = (D, T)$ with $D = \{1, 5, 10, 25\}$.
2. The invariant holds at the start of the first iteration, when $t = 0$ and $X = ()$.

3.1. Consider any iteration t such that the invariant holds at the start of the iteration.
3.2. Let $X = (x_1, \ldots, x_{t-1})$ be the coins chosen so far.
3.3. By the invariant, X is a prefix of some optimal solution C. That is:
3.4. For some $Y = (y_1, y_2, \ldots, y_\ell)$, the solution $C = (x_1, \ldots, x_{t-1}, y_1, y_2, \ldots, y_\ell)$ is optimal for T.
3.5. Then Y is a feasible solution for $T' = T - \sum_{i=1}^{t-1} x_i$ (the remaining total).
3.6. Let $x_t = \max\{d_i \in D : d_i \le T'\}$ be the coin computed in Line 3.2 of the algorithm.
3.7. Let $Z = (z_1, z_2, \ldots, z_k)$ be an optimal solution for total T' such that $z_1 = x_t$ (by Lemma 2).
3.8. Then $C' = (x_1, \ldots, x_{t-1}, z_1, \ldots, z_k)$ is a feasible solution for total T.
3.9. Since Z and Y are feasible for total T', and Z is optimal for T', it follows that $|Z| \le |Y|$.
3.10. By inspection of C and C', it follows that $|C'| \le |C|$.
3.11. By Steps 3.4 and 3.8, C' and C are feasible for total T.
3.12. Since C is optimal, it follows that C' is also optimal for total T.
3.13. Let $X' = (x_1, \ldots, x_{t-1}, x_t)$ be the coins chosen by the end of the iteration.
3.14. Then (using $z_1 = x_t$) X' is a prefix of C'. So the invariant is maintained.

4. By Block 3, each iteration that starts with the invariant true ends with it true.
5. And the invariant holds initially. So it holds throughout. □

Correctness follows easily from the invariant.

Theorem 2 (correctness of iterative algorithm)**.** *The iterative algorithm is correct.*

Proof (long form).

1. Consider the execution of the algorithm on any instance I.
2. Consider the end of the final iteration t.
3. Let $X = (x_1, \ldots, x_t)$ be the sequence returned.
4. By the termination condition, X is a feasible solution for total T.
5. By Lemma 3, the invariant holds then.
6. That is, X is a prefix of some optimal solution C for total T.
7. So $|X| \le |C|$. But X is feasible and C is optimal, so $|C| \le |X|$.
8. So $X = C$, and algorithm returns an optimal solution for total T. □

Exercises

1. Find an instance of the form $I = (D = \{1, d_2, d_3\}, T)$ on which the algorithm fails.
2. Is the algorithm correct for all instances with $D = \{1, 5, 10, 25, 100\}$? Prove your answer.
3. Give long-form proofs of Lemmas 1 and 3.
4. Give short-form proofs of Lemmas 3 and 4, and of Theorems 1 and 2.