# Introduction to Greedy Algorithms

Zezhou Jing

Rev. February 22, 2017

A *greedy algorithm* is an algorithmic paradigm which iteratively makes the locally optimal decisions in the hope of optimizing the global result. Dijkstra's shortest path algorithm is an example of greedy algorithms. Note that Dijkstra's algorithm processes each destination only once and is irrevocable.

A problem may have multiple greedy algorithms in attempt to achieve the global optimal result. Any of them may work or fail. For some problems, a greedy approach may even not be possible. For example, Dijkstra's algorithm may fail to compute the shortest path if the graph contains some edges with negative weighs.

We must prove the correctness of a greedy algorithm to the problem. The proof is frequently by induction or by contradiction.

Running time analysis for greedy algorithms is relatively easier than divide and conquer algorithms.

## Coin Changing

Suppose a cashier is given the U.S. currency denominations 1, 5, 10, 25, 100 and specific amount to pay to a customer. Develop a *Cashier's algorithm* to pay this amount with fewest number of coins.

*Solution.* Following the greedy paradigm, we derive the Cashier's algorithm.

**Algorithm.** At each iteration, add one coin of the largest value from denominations 1, 5, 10, 25, 100, which does not make the current amount exceed the amount to be paid.

**Theorem.** *Cashier's algorithm always produces the global optimum for the coin changing problem, given currency denominations* 1, 5, 10, 25, 100.

*Proof.* We prove that the Cashier's algorithm makes optimal choices for every denomination, namely each of 1, 5, 10, 25, 100. The proof is by induction.

First we may easily observe the following facts for any optimal solution:

- Number of pennies (cent) $p \leq 4$ since we can replace 5 pennies with a nickel.
- Number of nickels (five-cent) $n \leq 1$ since we can replace 2 nickels with a dime.
- (Number of nickels $n$ + number of dimes $d$) $\leq 2$ since we can replace 2 nickels and a dime with 2 dimes; we can replace 2 dimes and a nickel with a quarter; and we can replace 3 dimes with a quarter and a nickel.
- Number of quarters $q \leq 3$ because we can replace 4 quarters with a dollar.

Now we show that the Cashier's algorithm optimally solves the coin changing problem for any amount $s < 100$ while satisfying the above properties.

| Amount to be paid | Cashier's algorithm | Max amount to be paid |
|---|---|---|
| $s < 5$ | $p < 5$ | $4 = 1 * 4$ |
| $s < 10$ | $p < 5$ and $n = 1$ | $9 = 5 + 1 * 4$ |
| $s < 25$ | $p < 5$ and $n + d \leq 2$ | $24 = 10 * 2 + 4$ |
| $s < 100$ | $p < 5$, $n + d \leq 2$, and $q \leq 3$ | $99 = 25 * 3 + 10 * 2 + 4$ |

We see that any optimal solution other than the Cashier's algorithm violates the above properties for $s < 100$. Thus the Cashier's algorithm optimally solves coin changing problem for any $s < 100$. We let the base case be when $s < 100$.

Suppose the Cashier's algorithm is true for some $s$. We must show inductively that we can move from $s$ to $s' > s$ through given denominations 1, 5, 10, 25, 100, which are all individually and optimally solved by the Cashier's algorithm due to the table above. This is true because we can obtain any $s' \geq 100$ from $s < 100$ through finite combinations of the U.S. currency denominations. Remember any step onward by 1, 5, 10, 25, 100 is optimally solved by the Cashier's algorithm.

By principles of proof by mathematical induction, we prove that the Cashier's algorithm is globally optimal for any amount to be paid $s$ under the denominations 1, 5, 10, 25, 100. □

*Remark.* Cashier's algorithm does not produce global optimal solution for any denomination. A counterexample is the U.S. postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

```
if $i \geq maxval$ then
    $i \leftarrow 0$
else
    if $i + k \leq maxval$ then
        $i \leftarrow i + k$
    end if
end if
```

**Interval Scheduling**

Suppose we are given a set of $n$ tasks.Each task has distinct start and finish times $s_i$ and $f_i$, respectively ($s_i < f_i$). Two tasks are compatible if they do not overlap. That is, $f_i < s_j$ or $f_j < s_i$. The goal is to choose a maximum-cardinality subset of the input set in which every pair of tasks are compatible.

*Solution.* We wish to design a greedy algorithm. At each iteration we should choose a task $i$ (which is of course not already chosen) and discard all possible future choices that are not compatible with it.

**Algorithm.** We consider the following selection routines at each iteration.

- Choose one from the remaining tasks with the *earliest start time* $s_i$;
- Choose one from the remaining tasks with the *earliest finish time* $f_i$.
- Choose one from the remaining tasks with the *shortest interval length* $f_i - s_i$;

- Choose one from the remaining tasks with the *longest interval length* $f_i - s_i$.
- Choose one from the remaining tasks with the *fewest conflicts* with other remaining tasks. That is, for each task $j$, iterate through the set of remaining tasks and count the total number of conflicts $c_j$, order the schedule by ascending $c_j$.

However it seems only one greedy algorithm is globally correct.

**Theorem.** *The greedy algorithm ordering the task intervals by ascending finish times is globally optimal for the interval scheduling problem.*

*Proof.* content... □

### Interval Partitioning

### Minimizing Lateness

### Optimal Offline Caching