خرد کردن پول با رویکرد حریصانه

بخش اول: ثابت کردن بهینگی الگوریتم حریصانه برای سکه‌های {1, 10, 25}

مجموعه سکه‌های {1, 10, 25} یک سیستم پولی کاننیکال (canonical coin system) است. در چنین سیستم‌هایی، الگوریتم حریصانه همیشه راه‌حل بهینه تولید می‌کند. این سیستم‌ها طوری طراحی شده‌اند که هیچ ترکیب خاصی از سکه‌های کوچک‌تر نتواند سکه‌های بزرگ‌تر را بهینه‌تر جایگزین کند.

اثبات:

می‌خواهیم نشان دهیم که هیچ راه‌حل دیگری با تعداد کمتری از سکه‌ها وجود ندارد.
فرض می‌کنیم یک مقدار پول N داریم و دو نوع راه‌حل برای آن:

- G: الگوریتم حریصانه که سکه‌هایی مثل 25، 10، 1 را از بزرگ‌ترین به کوچک‌ترین انتخاب می‌کند.
- O: یک راه‌حل بهینه‌تر که کمتر از G سکه مصرف می‌کند.

می‌خواهیم نشان دهیم چنین راه‌حل O اصلاً وجود ندارد.

سکه‌های {1، 10، 25} به‌گونه‌ای هستند که 10 برابر 10 تا 1 است و 25 برابر 2 تا 10 بعلاوه 5 تا 1 است. بنابراین استفاده از یک سکه‌ی 25 همیشه بهتر از استفاده از دو سکه 10 و 5 سکه 1 است و همچنین استفاده از یک سکه 10 بهتر از 10 سکه 1 است. یعنی هیچ ترکیبی از سکه‌های کوچکتر نمی‌تواند با سکه بزرگ‌تر جایگزین شود و باعث کاهش تعداد سکه‌ها شود.

نتیجه‌گیری:

در هر مرحله اگر الگوریتم حریصانه سکه‌ی بزرگ‌تری را انتخاب کند، و ما بخواهیم از سکه‌های کوچکتر به‌جای آن استفاده کنیم، مجبور به استفاده‌ی تعداد بیشتری از سکه‌ها هستیم. پس هیچ راه‌حل جایگزینی وجود ندارد که با تعداد سکه کمتر به همان مقدار برسد.

بخش دوم: شکست الگوریتم حریصانه برای سکه‌های {1, 7, 10}

N = 14 سنت مثال نقض رویکرد حریصانه برای این مجموعه سکه است.

الگوریتم حریصانه:

۱۰ * ۱ + ۱ * ۴ = ۱۴ → ۵ سکه

راه‌حل بهینه:

۷ * ۲ = ۱۴ → ۲ سکه

چرا الگوریتم حریصانه شکست می‌خورد؟

الگوریتم حریصانه تنها روی بهینه بودن محلی تمرکز دارد (در هر مرحله بهترین انتخاب)، ولی این بهینگی همیشه به سراسری بودن نمی‌انجامد. در مثال بالا، انتخاب سکه‌ی 10 در ابتدا باعث می‌شود بعداً مجبور شویم از تعداد زیادی سکه‌ی کوچک استفاده کنیم، در حالی که نادیده گرفتن سکه‌ی 10 و استفاده از دو سکه‌ی 7 انتخاب بهتری بود.

بخش سوم: حل با برنامه‌نویسی پویا

dp[i] را حداقل تعداد سکه‌ها برای ساخت مقدار i تعریف می‌کنیم.

حالت پایه: dp[0]=0

برای i=1 تا N و همه c_i ها به شرطی که عبارت داخل براکت منفی نشود:

$$dp[i] = \min(dp[i-c_i] + 1)$$

این راه‌حل همیشه جواب بهینه را تضمین می‌کند.

البته از آنجایی که زیرمساله را برای تمام مقادیر پول و همه زیرمجموعه‌های سکه‌ها باید حساب کنیم، می‌توانستیم معادل با جواب بالا، آرایه را دو بعدی تعریف کنیم.