# GREEDY ALGORITHMS

# الگوریتمهای حریصانه

* تکنیک یا نگرش الگوریتم حریصانه (سود جویانه) یکی از روش‌های طراحی الگوریتم برای حل مسائل **بهینه‌سازی ترکیبیاتی** است.

$$\begin{cases} \min\limits_{X \in U} f(X) \\ \text{s.t.} \quad X \in C \end{cases}$$

$C \subseteq U$ . مجموعه انتخاب‌ها

(گسسته و متناهی)

* feasibility شدنی بودن (قابل قبول بودن)

اگر $X \in C$ باشد می‌گوییم $X$ یک جواب شدنی است.

و می‌گوییم (optimal) بهینه است اگر شدنی باشد و $f(X)$ کمینه باشد.

معمولاً جواب‌ها به صورت مجموعه یا برداری از مؤلفه‌های جواب بازنمایی می‌شود.

$X = \langle x_1, x_2, \dots, x_n \rangle$ $\boxed{x_i: \text{یک مؤلفه جواب}}$

الگوی کلی یک الگوریتم حریصانه

**function** *greedy* ($C$ : *set*) : *set*
  {$C$ is the set of all the candidates}
  $S \leftarrow \varnothing$  {$S$ is a set in which we construct the solution}
  **while not** *solution* ($S$) **and** $C \neq \varnothing$ **do**
    $x \leftarrow$ an element of $C$ "maximizing" *select* ($x$)
    $C \leftarrow C \setminus \{x\}$
    **if** *feasible* ($S \cup \{x\}$) **then** $S \leftarrow S \cup \{x\}$
  **if** *solution* ($S$) **then return** $S$ ✓
      **else return** "*there are no solutions*" ✗

Solution(S) → True
اگر یک جواب کامل توسط S فراهم شده باشد.

به بیان ساده، منظور از انتخاب حریصانه (Greedy Choice)، انتخاب از بین گزینه‌های موجود با هدف کسب بیشترین سود در لحظه است.

در نتیجه، نگرش حریصانه لزوماً شما را به یک جواب بهینه در پایان نمی‌رساند.

# 4. Greedy Algorithms I

- *coin changing*
- *interval scheduling*
- *scheduling to minimize lateness*
- *optimal caching*

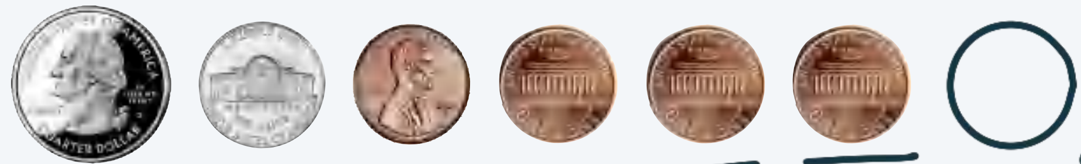
## Coin changing

مسئله خرد کردن پول

**Goal.** Given currency denominations: $C\{1, 5, 10, 25, 100\}$ devise a method to pay amount to customer using fewest number of coins.

یک مبلغ $\underline{n}$ دلاری را با استفاده از حداقل تعداد سکه‌ها خرد کنید.

**Ex.** 34¢.



25¢ ~~10¢~~ 5¢ 1¢ 1¢ 1¢ 1¢

$n = 34$

$25 + 5 + 1 \times 4 = 34$

**Cashier's algorithm.** At each iteration, add coin of the largest value that does not take us past the amount to be paid.

**Ex.** $2.89.

1\$ 1\$ 25¢ 25¢ 25¢ 10¢ 1¢ × 4

$n = 2.89$

# Cashier's algorithm

الگوریتم صندوقدار

At each iteration, add coin of the largest value that does not take us past the amount to be paid.

CASHIERS-ALGORITHM $(x, c_1, c_2, \ldots, c_n)$

x مبلغی که می‌خواهیم خرد کنیم

~~SORT $n$ coin denominations so that $c_1 < c_2 < \ldots < c_n$~~

$C = \{c_1^{\infty}, c_2^{\infty}, \ldots, c_n^{\infty}\}$

$S \leftarrow \phi$ ← set of coins selected

not Solution (S)

WHILE $x > 0$

→ Maximum select → feasible

$k \leftarrow$ largest coin denomination $c_k$ such that $c_k \leq x$

IF no such $k$, RETURN "no solution"

ELSE

$x \leftarrow x - c_k$

$S \leftarrow S \cup \{ k \}$

RETURN $S$

اگر فرض $c_1 = 1$ برقرار باشد

آنگاه این الگوریتم همواره به جواب می‌رسد

اما در مورد اینکه این جواب بهینه است بستگی به اجزاء پول در سیستم پولی دارد

find largest
$O(n)$

\# iteration while
$O(X)$

بدون Sort در ابتدا

$T(n,x) = O(nx)$

با انجام Sort در ابتدا

$T(n,x) = O(n \lg n) + O(x)$

**Q.** Is cashier's algorithm optimal?

## Properties of optimal solution

الگوریتم حریصانه لزوما همیشه بهینه نیست

مثال

$C = \{1, 7, 9, 10\}$

$x = 18$

$10 + 7 + 1$ جواب حریصانه $\underline{3}$

$9 + 9$ جواب بهینه $2$

Property. Number of pennies ≤ 4.

Pf. Replace 5 pennies with 1 nickel.

Property. Number of nickels ≤ 1.

Property. Number of quarters ≤ 3.

Property. Number of nickels + number of dimes ≤ 2.

Pf.

- Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;
- Replace 2 dimes and 1 nickel with 1 quarter.
- Recall: at most 1 nickel.

# Analysis of cashier's algorithm

Theorem. Cashier's algorithm is optimal for U.S. coins: 1, 5, 10, 25, 100.

Pf. [by induction on x]

- Consider optimal way to change $c_k \leq x < c_{k+1}$ : greedy takes coin $k$.
- We claim that any optimal solution must also take coin $k$.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to $x$
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by cashier's algorithm. ▪

| k | $c_k$ | all optimal solutions must satisfy | max value of coins $c_1, c_2, \ldots, c_{k-1}$ in any OPT |
|---|---|---|---|
| 1 | 1 | $P \leq 4$ | – |
| 2 | 5 | $N \leq 1$ | 4 |
| 3 | 10 | $N + D \leq 2$ | 4 + 5 = 9 |
| 4 | 25 | $Q \leq 3$ | 20 + 4 = 24 |
| 5 | 100 | *no limit* | 75 + 24 = 99 |

# Cashier's algorithm for other denominations

$\{1, a, a^2, a^3, \dots, a^s\}$

Q. Is cashier's algorithm for any set of denominations?

A. No. Consider U.S. postage: {1, 10, 21, 34, 70, 100, 350, 1225, 1500}
- Cashier's algorithm: 140¢ = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1.
- Optimal: 140¢ = 70 + 70.

الگوریتم حریصانه

۸ قطعه تمبر

در حالیکه جواب بهینه

۲ قطعه تمبر


A. No. It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9.
- Cashier's algorithm: 15¢ = 9 + ???.
- Optimal: 15¢ = 7 + 8.

1

مسئله کمینه سازی متوسط زمان انتظار

# Minimizing Average Waiting Time in the System

- A single server (a processor, a petrol pump, a cashier in a bank, and so on) has ***n*** customers to serve.

چون در آن واحد پردازنده تنها می‌تواند مشغول انجام یک پردازه باشد سایر پردازه‌ها باید در یک صف منتظر بمانند

- The service time required by each customer is known in advance:

$t_i$ : طول مدت زمان لازم برای پردازش i‌ام (داده مفروض)

*customer **i** will take time t_i, , 1 ≤ **i** ≤ **n**.*

- We want to minimize

هدف. می‌خواهیم صف پردازش را به قسمی بچینیم که متوسط زمان انتظار کمینه شود

$$T = \sum_{i=1}^{n} (\text{time in system for customer } i).$$

مدت انتظار در صف + طول زمان اجرای خود پروسس‌ها

- Since the number of customers is fixed, minimizing the total time in the system is equivalent to minimizing the average time.

مثال.

$n = 3 \qquad t_1 = 5, \quad t_2 = 3, \quad t_3 = 10$

صف $1,2,3$ : $T = 5 + (5+3) + (5+3+10) = 31$

صف $2,3,1$ : $T = 3 + (3+10) + (3+10+5) = 34$

⋮

صف $2,1,3$ : $T = 3 + (3+5) + (3+5+10) = 29$

اثبات می‌شود اگر صف پردازه‌ها را بر حسب طول زمان اجرا به ترتیب صعودی مرتب کنیم،
جواب بهینه بدست می‌آید؛ یعنی از قاعده «هر کس زمان کمتری کار دارد جلوتر بیاید» استفاده می‌کنیم.

# اثبات درستی الگوریتم حریصانه در مسئله کمینه سازی میانگین زمان انتظار

○ قرار دهید $r_i$ نشان دهندهٔ موقعیت پردازش $i$ ام در صف باشد. داریم:

$$T = \sum_{i=1}^{n} (n+1-r_i) \cdot t_i$$

در مثال قبل اگر صف 2,3,1 را در نظر بگیریم؛ یعنی

$$r_1 = 3 \quad , \quad r_2 = 1 \quad , \quad r_3 = 2$$

و

$$T = t_2 + (t_2 + t_3) + (t_2 + t_3 + t_1)$$

$$= 3t_2 + 2t_3 + t_1$$

$$= \sum (n+1-r_i) \cdot t_i$$

○ مسئله در فرم یک مسئله بهینه سازی ترکیباتی به صورت زیر صورتبندی می شود:

$$\text{Min } T_{\langle r_1, r_2, \dots, r_n \rangle}$$

s.t.

$$\langle r_1, r_2, \dots, r_n \rangle \in S_n$$

که $S_n$ نشان دهندهٔ مجموعهٔ جایگشتهای $\{1, 2, \dots, n\}$ است.

## ادامهٔ اثبات درستی الگوریتم حریصانه در حل مسئلهٔ کمینه سازی میانگین زمان انتظار

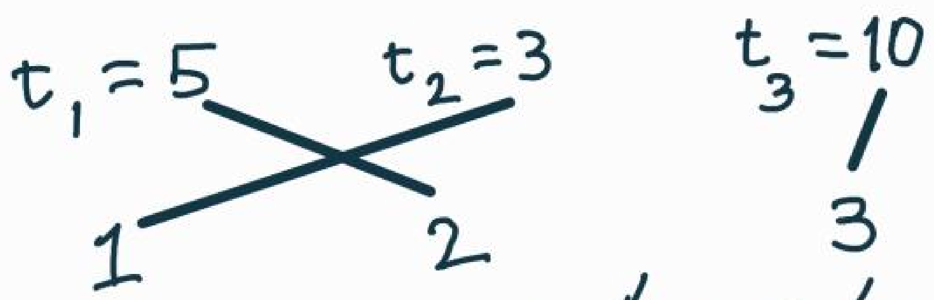
$$\min_{\langle r_1,\dots,r_n\rangle \in S_n} T = \min \sum_{i=1}^{n} (n+1-r_i)\cdot t_i$$

$$= \underbrace{n(n+1)\,t_i}_{\text{constant}} - \min \sum_{i=1}^{n} r_i t_i$$

$$\Longrightarrow \operatorname*{Argmin}_{\langle r_1,\dots,r_n\rangle} T = \operatorname*{Argmax}_{\langle r_1,\dots,r_n\rangle} \sum_{i=1}^{n} r_i t_i$$

حال به سادگی ملاحظه می شود بیشینه سازی $\sum r_i t_i$ وقتی $r_i$ ها مقید به اعداد صحیح $\{1,2,\dots,n\}$ هستند زمانی بدست می آید که بزرگترین عدد در $\{1,2,\dots,n\}$ را ضریب بزرگترین عدد بین $\{t_1,t_2,\dots,t_n\}$ قرار دهیم و همین ترتیب تا کوچکترین.

در مثال قبل




این دقیقاً همان انتخابی است که توسط الگوریتم حریصانه صورت می گیرد.