الگوریتم های حریصانه

روش حریصانه یا الگوریتم حریصانه یکی از روش‌های مشهور و پرکاربرد طراحی الگوریتم‌ها است که با ساختاری ساده در حل بسیاری از مسائل استفاده می‌شود. این روش اغلب در حل مسائل بهینه‌سازی به کار می‌رود و در پاره‌ای مواقع جایگزین مناسبی برای روش‌هایی مانند برنامه‌ریزی پویا است. در حالت کلی این روش سرعت و مرتبهٔ اجرایی بهتری نسبت به روش‌های مشابه خود دارد؛ اما متناسب با مسئله ممکن است به یک جواب بهینهٔ سراسری ختم نشود.

در روش حریصانه رسیدن به هدف در هر گام مستقل از گام قبلی و بعدی است؛ یعنی در هر مرحله برای رسیدن به هدف نهایی، مستقل از این که در مراحل قبلی چه انتخاب‌هایی صورت گرفته و انتخاب فعلی ممکن است چه انتخاب‌هایی در پی داشته باشد، انتخابی صورت می‌پذیرد که در ظاهر بهترین انتخاب ممکن است .به همین دلیل است که به این روش، روش حریصانه گفته می‌شود. به‌طور مثال، زمانی که یک دزد عجول و حریص وارد خانه‌ای می‌شود، در مسیر حرکت خود هر وسیله و کالای با ارزشی را داخل کیسه می‌اندازد. وی در این حالت چندان توجهی نمی‌کند که چه اشیایی را قبلاً برداشته و ممکن است در آینده چه اشیاء گران‌بهاتری به دست آورد. او در هر گام تنها از بین اشیای دم‌دست خود باارزش‌ترین آن را انتخاب کرده و به وسایل قبلی اضافه می‌کند.

- مساله کوله پشتی 1-0
  - $S = \{item_1, item_2, \ldots, item_n\}$
  - w_i = weight of $item_i$
  - p_i = profit of $item_i$
  - W = maximum weight the knapsack can hold
- تعیین یک زیر مجموعه مانند A از S به طوری که:

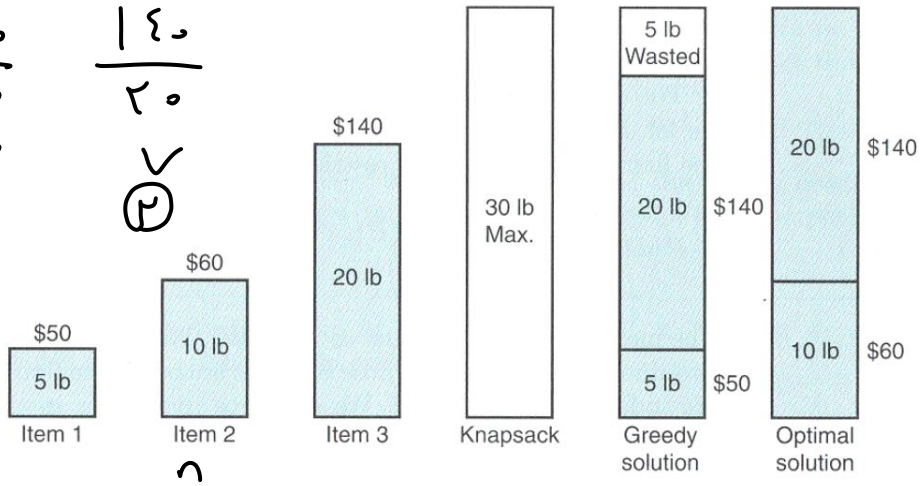
$$\sum_{item_i \in A} p_i \quad \text{is maximized subject to} \quad \sum_{item_i \in A} w_i \le W$$

– الگوریتم *Brute force* : $O(2^n)$ – زیرا 2^n زیر مجموعه وجود دارد.

- **Applications:** One of these areas is scripting and automation tasks. Consider the case of writing a script to manage the hard disk storage of a server. Your script must maintain as many important files in the server's disk as possible, deleting less important files to make space for more important ones. Judging these files' importance could be based on their frequency of access, their sizes, or other business-specific metrics. This scenario is, in reality, a 0/1 Knapsack Problem. The hard disk represents the knapsack, and the files represent the items with their individual sizes and values.

- Dynamic resource allocations in virtual networks.

شکست روش حریصانه



$$\max \sum_{i=1}^{n} p_i x_i \quad \sum_{i=1}^{n} w_i x_i \le W, \quad x_i = 0 \text{ یا } 1$$

0-1 Knapsack

Relaxation $\Longrightarrow$

$$\max P^T x \quad W^T x \le W \quad 0 \le x_i \le 1$$

الگوریتم حریصانه: ۱- $\frac{p_i}{w_i}$ را به صورت نزولی مرتب کنید ۲- ۳- ۴- ۵

$$\sum_{i=1}^{r} w_i < W, \quad \sum_{i=1}^{r+1} w_i > W$$

$$x_i = 1, \; i = 1, 2, \ldots, r \qquad x_{r+1} = \frac{W - \sum_{i=1}^{r} w_i}{w_{r+1}}$$

$$x_j = 0, \; j = r+2, \ldots, n$$

$$\max \; 2x_1 + 5x_2 + 7x_3 + 3x_4$$

$$3x_1 + 8x_2 + 6x_3 + 5x_4 \le 13$$

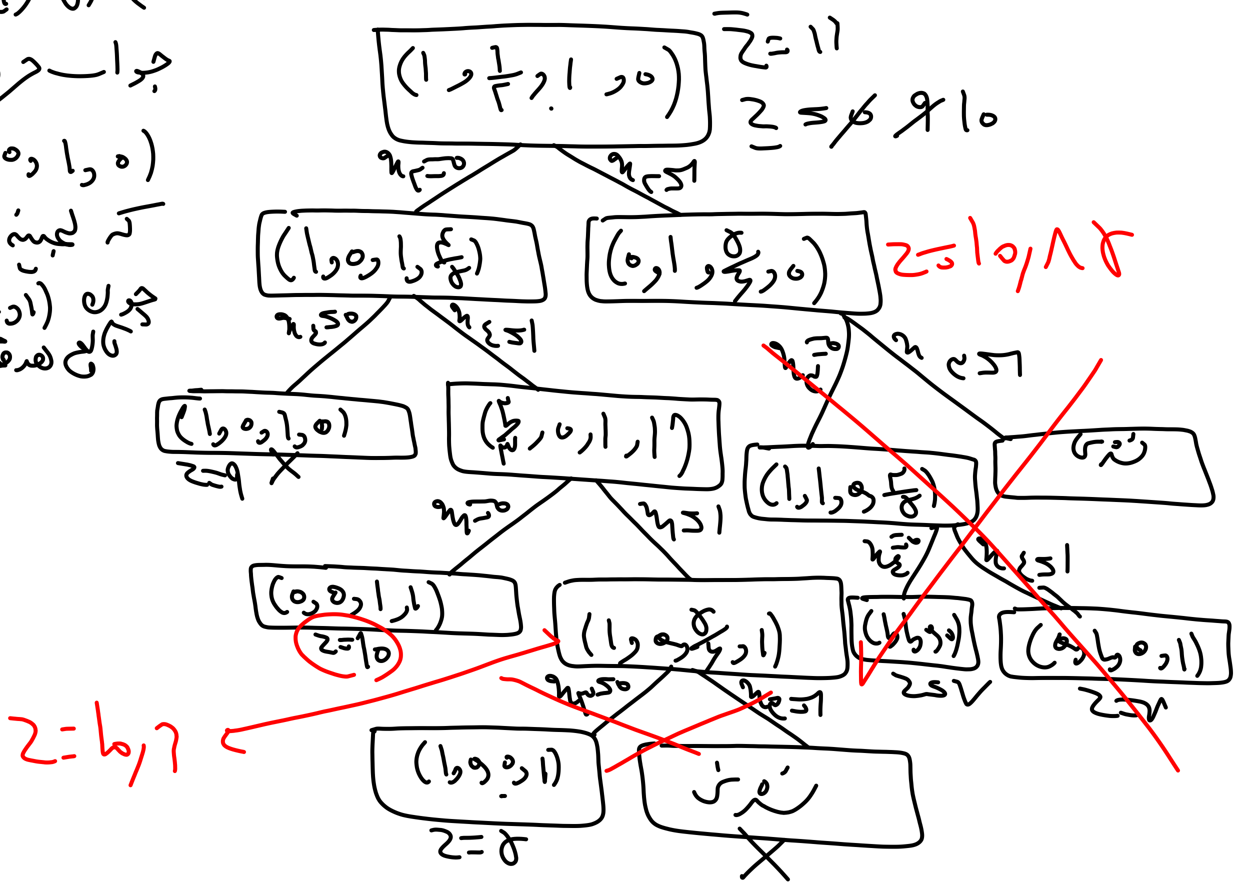
$$x_i = 0, 1$$

$\frac{2}{3}, \frac{5}{8}, \frac{7}{6}, \frac{3}{5}$

(2) (3) (1) (4)

جواب حریصانه
$(1, 0, 1, 0)$
که بهینه نیست

جواب $(0,0,1,1)$ تابع هدف بیشتری دارد



رویکرد برنامه ریزی پویا

- الگوریتم:

$$P[i][w] = \begin{cases} maximum\left(P[i-1][w], p_i + P[i-1][w-w_i]\right) & \text{if } w_i \le w \\ P[i-1][w] & \text{if } w_i > w. \end{cases}$$

- حداکثر سود = $P[n][W]$
- با استفاده از آرایه $P[0..n][0..W]$
- پیچیدگی زمانی:

تعداد درایه های محاسبه شده برابر nW : $\Theta(nW)$

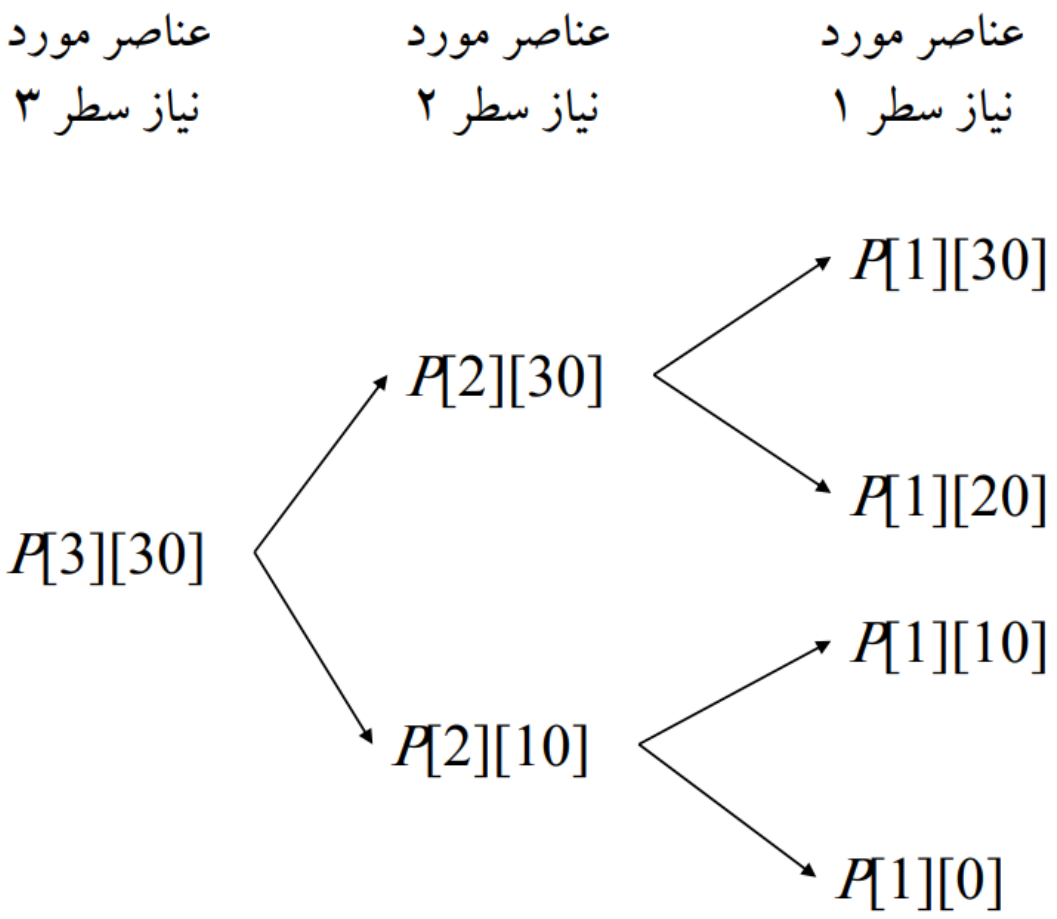
نسخه بهبود یافته

- برگشت به عقب به منظور تعیین مولفه مورد نیاز به دلیل

$$P[n][W] = \begin{cases} maximum\left(P[n-1][W], p_n + P[n-1][W-w_n]\right) & \text{if } w_n \le W \\ P[n-1][W] & \text{if } w_n > W, \end{cases}$$

- مولفه مورد نیاز در سطر n-1، برابر $P[n\text{-}1][W]$ و $P[n\text{-}1][W\text{-}w_n]$ می باشند
- به طور کلی می توانیم از این حقیقت بهره ببریم که $P[i][w]$ از روی $P[i\text{-}1][w]$ و $P[i\text{-}1][w\text{-}w_i]$ محاسبه شده است.
- مگر اینکه $n = 1$ و یا $w \le 0$

مثال:



- محاسبه سطر ۱:

$$P[1][w] = \begin{cases} \text{maximum}(P[0][w],\ \$50 + P[0][w-5]) & 5 \le w \\ P[0][w] & 5 > w \end{cases}$$

$$= \begin{cases} \$50 & 5 \le w \\ 0 & 5 > w \end{cases}$$

- بنابراین:

- $P[1][0] = \$0$
- $P[1][10] = \$50$
- $P[1][20] = \$50$
- $P[1][30] = \$50$

- محاسبه سطر ۲:

$$P[2][10] = \begin{cases} \text{maximum}(P[1][10],\ \$60 + P[1][0]) & 10 \le 10 \\ P[1][10] & 10 > 10 \end{cases}$$
$$= \$60$$

$$P[2][30] = \begin{cases} \text{maximum}(P[1][30],\ \$60 + P[1][20]) & 10 \le 30 \\ P[1][30] & 10 > 30 \end{cases}$$
$$= \$60 + \$50 = \$110$$

- محاسبه سطر ۳:

$$P[3][30] = \begin{cases} \text{maximum}(P[2][30],\ \$140 + P[2][10]) & 20 \le 30 \\ P[2][30] & 20 > 30 \end{cases}$$
$$= \$140 + \$60 = \$200$$

- حداکثر 2^i درایه در سطر $(n-1)$ محاسبه می شود.

- تعداد کل درایه های محاسبه شده حداکثر برابر است با:

$$1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1 = \Theta(2^n)$$

- نتیجه: تعداد عناصر محاسبه شده در بدترین حالت:

$$O(\text{minimum}(2^n, nW))$$

Discrete optimization

0-1 variables

Max x_n

$2\sum_{i=1}^{n-1} x_i + x_n = n$

x_i s.t. $0 \le x_i \le 1$

$x_n = 0$