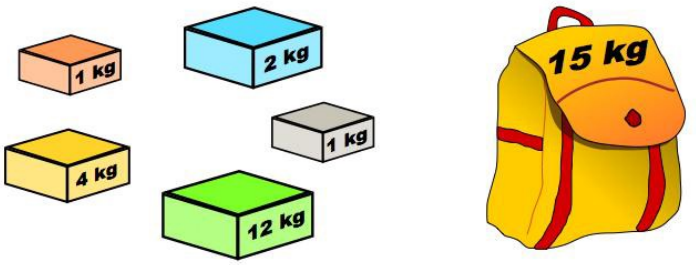
درس طراحی الگوریتم
دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی. پاییز ۱۴۰۱
کتاب مرجع: طراحی الگوریتم. جان کلاینبرگ و اوا تاردش

۱ مسئله کوله پشتی the knapsack problem



یک کوله پشتی به همراه n شیء با وزن و ارزش معین داریم.

$$A = \{(w_1, v_1), \ldots, (w_n, v_n)\}$$

اینجا w_i و v_i به ترتیب وزن و ارزش شیء i ام است. فرض ما بر این است که وزنها و ارزشها اعداد صحیح مثبت هستند.

یک پارامتر W هم داریم که نشان دهنده آستانه تحمل کوله پشتی از لحاظ وزن است.

در این مسئله می‌خواهیم از میان n شیء داده شده، تعدادی را انتخاب کنیم که در مجموع بیشترین ارزش را داشته باشند بشرطی که وزن اشیاء انتخاب شده از W بیشتر نباشد. به عبارت دیگر دنبال مجموعه $S \subseteq \{1, \ldots, n\}$ می‌گردیم بطوریکه

$$\sum_{i \in S} v_i$$

ماکزیمم شود بشرطی که

$$\sum_{i \in S} w_i \leq W$$

۱.۱ یک مثال

جدول زیر مشخصات 7 شیء را نشان می‌دهد.

number	1	2	3	4	5	6	7
weight	4	2	3	5	7	1	1
value	18	10	5	15	7	6	3

با فرض اینکه مقدار آستانه $W = 15$ بیشترین ارزشی که می‌توانیم با کوله پشتی برداریم چقدر است؟

۲ ایده های حریصانه برای مسئله کوله پشتی

مثال زیر نشان می‌دهد که چند ایده حریصانه، از قبیل اولویت دادن به اشیاء با وزن کمتر، اولویت دادن به اشیا با ارزش بیشتر و اولویت دادن به اشیا با نسبت ارزش به وزن بیشتر در رسیدن به جواب بهینه ناموفق است.

$W = 105$						obtained value
weight	20	30	40	50	60	
value	20	30	44	55	60	
value/weight	1	1	1.1	1.1	1	
$\min w_i$	✓	✓	✓			94
$\max v_i$			✓		✓	104
$\max \frac{v_i}{w_i}$			✓	✓		99
optimal solution	✓	✓		✓		105

۳ حل مسئله کوله پشتی با تکنیک برنامه نویسی پویا

ابتدا $OPT(j,t)$ را تعریف می‌کنیم. با فرض اینکه اشیا از 1 تا n شماره گذاری شده‌اند، $OPT(j,t)$ برابر با ارزش جواب بهینه برای j شئ اول $\{1,\ldots,j\}$ است با فرض اینکه مقدار آستانه $W = t$ می باشد.

با توجه به تعریف بالا، در حل مسئله کوله پشتی در واقع دنبال مقدار $OPT(n,W)$ هستیم. البته این فقط مقدار ارزش بهینه را نشان می‌دهیم. علاوه بر این ما دنبال زیرمجموعه ای هستیم که این ارزش را بدست می‌دهد.
در هر صورت، با استفاده از تفکر بازگشتی می‌توانیم بگوییم اگر O زیرمجموعه بهینه باشد، آنگاه دو حالت داریم:

- در حالت اول $n \in O$. این به این معنی است که

$$OPT(n,W) = v_n + OPT(n-1, W-w_n)$$

به شرطی که $w_n \leq W$.

- در حالت دوم $n \notin O$. این به این معنی است که

$$OPT(n,W) = OPT(n-1,W)$$

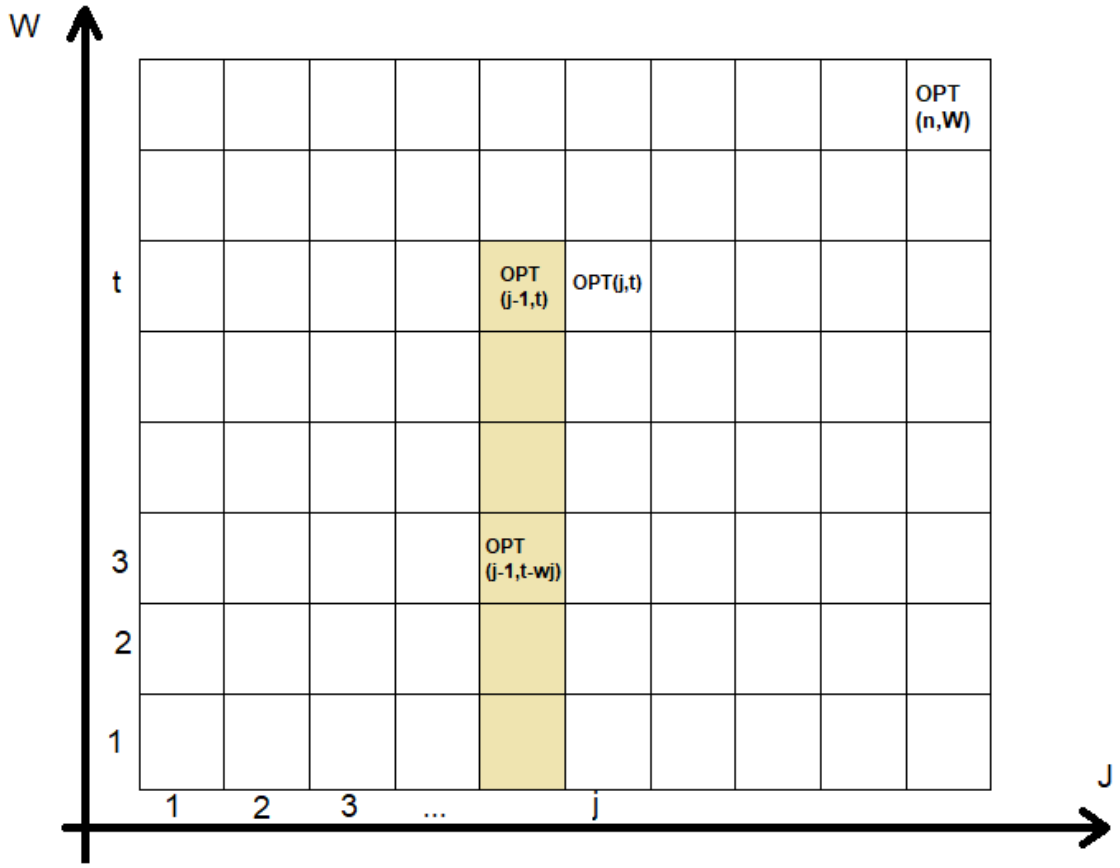
پس می‌توانیم بنویسیم

$$OPT(n,W) = \max\{v_n + OPT(n-1, W-w_n), OPT(n-1,W)\}$$

و اگر همین رابطه بازگشتی را برای j شئ اول بنویسیم داریم

$$OPT(j,t) = \max\{v_j + OPT(j-1, t-w_j), OPT(j-1,t)\}$$

پس در اینجا OPT، که در مسئله بازه ها یک بعدی بود و به شکل یک آرایه بود، در مسئله کوله پشتی حالت دو بعدی دارد و بصورت یک جدول است.



برای یافتن مقدار خانه $OPT(j,t)$
به مقدار خانه های $OPT(j-1,t)$ و خانه‌های $OPT(j-1,t-w_j)$ نیاز داریم. پس جدول OPT را اگر از سمت چپ به راست، ستون به ستون پر کنیم می‌توانیم کل جدول را براحتی تکمیل کنیم. همانطور که گفته شد دنبال خانه $OPT(n,W)$ می‌گردیم.

یافتن زیرمجموعه بهینه. برای پیدا کردن زیرمجموعه بهینه O کافی است توجه کنیم که $OPT(n,W)$ برابر با کدامیک از حالات $OPT(n-1,W)$ و $v_n + OPT(n-1, W-w_n)$ است. اگر

$$OPT(n,W) = v_n + OPT(n-1, W-w_n)$$

یعنی $n \in O$ و کار را با $OPT(n-1,W-w_n)$ دنبال می‌کنیم. اگر $OPT(n,W) = OPT(n-1,W)$ یعنی $n \notin O$ و لذا کار را با بررسی $OPT(n-1,W)$ دنبال می‌کنیم.

زمان اجرا. برای پیدا کردن جواب باید جدول OPT را تکمیل کنیم. جدول در کل nW خانه دارد. دقت کنید مقدار هر خانه از جدول به مقدار دو خانه دیگر وابسته است. لذا زمان پر کردن هر خانه $O(1)$ است. پس در مجموع $O(nW)$ زمان اجرای الگوریتم است. زیرمجموعه بهینه را نیز می‌توان در زمان $O(n)$ محاسبه کرد.

نکته نهایی. توجه کنید زمان اجرای الگوریتم بالا یک زمان چند جمله‌ای polynomial time نیست. گوییم الگوریتمی زمان اجرایش چند جمله‌ای است اگر زمان اجرایش تابعی چند جمله‌ای از اندازه ورودی مسئله باشد. اینجا ورودی مسئله $2n$ عدد (وزنها و ارزشها) به اضافه یک عدد (مقدار آستانه تحمل W) است. پس اندازه ورودی $2n+1$ عدد است. اگر حداکثر وزن W باشد و حداکثر ارزش V باشد، ورودی مسئله را می‌توان با $n\log W + n\log V + \log W$ بیت نمایش داد. از آنجا که زمان اجرای الگوریتم با تکنیک برنامه نویسی پویا برابر با $O(nW)$ پس زمان اجرا نسبت نمایی با یک پارامتر اندازه ورودی یعنی $\log W$ دارد و لذا یک تابع چند جمله‌ای نیست. در واقع مسئله کوله پشتی جزو مسائل $NP-Complete$ است. گمان غالب بر این است که برای هیچ مسئله $NP-Complete$ الگوریتم با زمان چند جمله‌ای وجود ندارد.