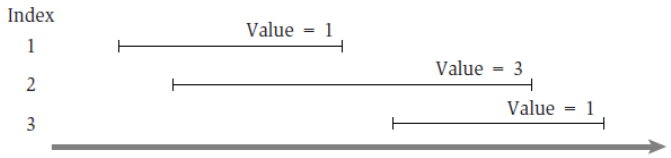
برنامه نویسی پویا

۱ مقدمه

در تکنیک برنامه نویسی پویا، بر خلاف استراتژی حریصانه، الگوریتم بطور ضمنی کل فضای جواب را بررسی میکند. این کار در واقع با شکستن مسئله به زیرمسائل کوچکتر انجام میشود. زیرمسائل کوچکتر ابتدا پردازش شده و سپس از حل آنها برای حل زیرمسائل بزرگتر و در نهایت حل مسئله اصلی استفاده میشود. بشکستن مسئله به زیرمسائل کوچکتر از طریق یک رابطه بازگشتی انجام میشود بطوریکه جواب مسئله در حل نهایی این رابطه بازگشتی پیدا میشود.
برای تبیین بهتر این تکنیک، بهتر است از یک نمونه شروع کنیم.

۲ مسئله زمانبندی بازه های با ارزش غیر یکسان

مسئله زمانبندی بازه ها در فصل مربوط به الگوریتمهای حریصانه معرفی شد. در این مسئله یک مجموعه از بازه با ارزش یکسان داریم که میخواهیم حداکثر تعداد را از آنها انتخاب کنیم بطوریکه بازه های انتخابی همپوشانی نداشته باشند. در آنجا دیدیم که استراتژی حریصانه (بازه ای که زودتر خاتمه پیدا میکند اولویت دارد) جواب بهینه را بدست می آورد.
اگر همین مسئله را به حالتی تعمیم دهیم که بازه ها ارزش غیر یکسان داشته باشند و بخواهیم دسته ای از بازه ها را انتخاب کنیم که در مجموع ارزش بیشینه داشته باشند، به شرطی که همپوشانی نداشته باشند، استراتژی حریصانه ای که گفته شد و خیلی از استراتژیهای حریصانه دیگر جوابگو نخواهد بود. برای نمونه شکل زیر را ببینید. انتخاب بازه ها با معیار زمان خاتمه زودتر به یک جواب غیر بهینه می انجامد.



برای بیان رسمی تر مسئله، فرض کنید I شامل n بازه باشد.

$$I = \{(s_1, f_1, v_1), (s_2, f_2, v_2), \ldots, (s_n, f_n, v_n)\}$$

در اینجا s_i زمان شروع، f_i زمان خاتمه و v_i ارزش بازه است. میخواهیم یک زیرمجموعه $S \subseteq \{1, \ldots, n\}$ انتخاب کنیم بطوریکه $\sum_{i \in S} v_i$ ماکزیمم شود و هیچ دو بازه در S همپوشانی نداشته باشند.

۱.۲ رابطه بازگشتی برای حل مسئله

یک تفکر بازگشتی برای حل مسئله زمانبندی بازه ها میتواند به این صورت باشد. فرض کنید بازه ها بر اساس زمان خاتمه از کوچک به بزرگ مرتب شده اند. دو حالت داریم یا جواب بهینه شامل بازه n ام هست یا نیست.

تعریف: $OPT(j)$ ارزش جواب بهینه مسئله ای است که فقط شامل بازه های 1 تا j است.
تعریف: $p(j)$ شماره آخرین بازه ای است که قبل از شروع بازه j خاتمه یافته است.

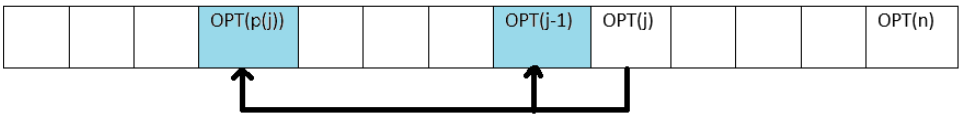
با استفاده از تعریف بالا میتوانیم بنویسیم.

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

و به همین طریق، برای هر j می‌توانیم بنویسیم

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

لذا کافی است برای محاسبه $OPT(n)$ که ارزش جواب بهینه است، آرایه OPT را محاسبه کنیم. آرایه را از چپ به راست پر می‌کنیم.



قدمهای الگوریتم در زیر خلاصه شده است.

۱. بازه ها بر اساس زمان خاتمه از کوچک به بزرگ مرتب کن. فرض کن

$$I' = (s_1, f_1), \ldots, (s_n, f_n)$$

لیست مرتب بازه ها باشد.

۲. برای هر j، مقدار $p(j)$ را محاسبه کن.

۳. آرایه OPT را طبق آنچه در بالا گفته شد تکمیل کن. دقت کنید $OPT(0) = 0$.

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

۴. در انتهای کار $OPT(n)$ جواب الگوریتم است.

برای محاسبه جواب بهینه O، دقت کنید اگر $OPT(n) = OPT(p(n)) + v_n$ آنگاه می‌توانیم نتیجه بگیریم $n \in O$. در غیر اینصورت $n \notin O$. به همین طریق می‌توانیم عضویت یا عدم عضویت بقیه بازه‌ها در یک جواب بهینه را تشخیص دهیم. البته جواب بهینه می‌تواند منحصر بفرد نباشد. برای مثال این وقتی اتفاق می‌افتد که یک j وجود داشته باشد که

$$OPT(j) = OPT(j-1) = OPT(p(j)) + v_j$$

۲.۲ زمان اجرای الگوریتم

قدم اول الگوریتم یعنی مرتب سازی بازه ها در زمان $O(n \log n)$ قابل انجام است. قدم دوم الگوریتم هم در زمان $O(n \log n)$ قابل انجام است. قدم سوم که پر کردن آرایه OPT است در زمان $O(n)$ انجام می‌شود. دقت کنید آرایه $n+1$ خانه دارد. هر خانه آرایه هم در زمان $O(1)$ با بررسی دو خانه قبلی قابل محاسبه است. پس در مجموع الگوریتم پیشنهادی در زمان $O(n \log n)$ قابل پیاده سازی است.