

Løsning av 0-1 Knapsack-problemet med dynamisk programmering

Åsmund Eldhuset

Problem

Man har fått oppgitt n objekter der objekt nummer i ($1 \leq i \leq n$) veier $w[i]$ kg og er verdt $v[i]$ kroner (alle vekter og verdier er heltall) og en ryggsekk som har kapasitet W kg. Hva er den maksimale verdien av objekter man kan få med seg i ryggsekken?

Optimal substruktur

La oss formulere problemet slik: Hva er største verdi i sekken hvis man begrenser vekten til maksimalt m kg og kun tillater at man velger mellom objektene fra og med nr. 1 til og med nr. i ? Dersom vekten $w[i]$ til objekt i er større enn m , kan man ikke ta med objekt i ; ellers har man valget mellom enten å ta det med seg eller ikke.

- Hvis man tar med seg objekt i , bruker man opp $w[i]$ kg, og da er det bare $m - w[i]$ kg igjen. Man kan da stille seg spørsmålet: Hva er største verdi jeg kan få med meg hvis jeg kan velge blant objektene fra og med nr. 1 til og med nr. $i - 1$, og har begrenset vekten til $m - w[i]$ kg? Til denne verdien kan man legge $v[i]$, siden man tok med seg objekt i .
- Hvis man ikke tar med seg objekt i , er fremdeles alle de m kiloene ledige, og man kan da spørre seg: Hva er største verdi jeg kan få med meg hvis jeg kan velge blant objektene fra og med nr. 1 til og med nr. $i - 1$, og har begrenset vekten til m kg?

Vi ser altså at vi har en optimal substruktur her: Løsningen av et delproblem består av løsninger av mindre delproblemer. Vi definerer $c[i][m]$ til å være den maksimale verdien i sekken dersom man tillater bruk av objektene fra og med 1 til og med i , og begrenser vekten til m kg. Ut i fra det foregående kan vi da sette opp følgende:

$$c[i][m] = \begin{cases} 0 & \text{hvis } i = 0 \text{ eller } m = 0 \\ c[i - 1][m] & \text{hvis } w[i] > m \\ \max(c[i - 1][m], c[i - 1][m - w[i]] + v[i]) & \text{hvis } i > 0 \text{ og } w[i] \leq m \end{cases}$$

Løsningen på hele problemet er lik $c[n][W]$, da det representerer maksimal verdi i sekken dersom man tillater bruk av alle objektene og vekten er begrenset til ryggsekkens fulle kapasitet.

Implementasjon

Når man har gitt reglene ovenfor, skal det være en relativt grei sak å fylle ut DP-tabellen. Å skrive et program som gjør det overlates til leseren. Er man ikke i humør til det, finnes det nok av kodeeksempler på nettet. Det kan være verdt å merke seg at w - og v -tabellene som vi har brukt her er 1-indekserte, så hvis man koder i Python, Java eller lignende, kan det være greit å dytte inn en eller annen verdi i fronten av de to tabellene slik at de egentlige verdiene havner på indeksene 1 til n i stedet for 0 til $n - 1$. Husk også at c -tabellen har $n + 1$ rader og $W + 1$ kolonner.

Analyse

Kjøretiden til denne algoritmen er $\Theta(nW)$, siden tabellen inneholder $n \times W$ felter, hvert felt besøkes én gang og utregningen av tallet i et felt tar konstant tid. Merk at dette *ikke* er polynomisk; det er derimot pseudopolynomisk. Dette skyldes at W ikke er et mål på størrelsen til input (slik n er), men er selv en del av inputen. Størrelsen til W kan defineres som antall bits d som er nødvendig for å representere W . Skal man være helt nøyaktig, er dette lik $\lceil \lg W \rceil + 1$, men for enkelhets skyld sier vi at $d = \lg W$. Da ser vi at kjøretiden er $\Theta(nW) = \Theta(n \cdot 2^{\lg W}) = \Theta(n \cdot 2^d)$, som er eksponentielt i inputstørrelsen d . Ingen bedre algoritmer er kjent for dette problemet, og det viser seg at 0-1 Knapsack er NP-komplett.