背包問題

- 給定 n 個數對：(a_1,b_1)，……，(a_n,b_n)，其中 a_i 為第 i 個東西的重量、 b_i 為第 i 個東西的價值，皆為整數。

  我們要選擇一些東西裝進背包裡，在不超過背包最大負重 W 的情況下，最多可裝進多少價值的東西？

使用動態規劃
(Dynamic Programming)

典型的步驟：

- 想像最佳解的樣子 （結構、如何組成等等）
- 適當地定義(切割出)子問題、
並明確區分出每個參數、以及欲求取的最佳解。
- 透過觀察，寫下(最佳解的)遞迴式及邊界條件。
- 依據遞迴式， 由下而上計算最佳解。

動態規劃的特徵

能適用動態規劃(DP)解決的問題，
必定具有下述的特徵：

- 問題的最佳解，
  必然可由子問題的最佳解組合出來。

  (此關係，即是我們希望拆解出的遞迴式)

  (Optimal Substructure)

背包問題(Knapsack)的 DP 遞迴式

- 定義 opt(i,w) 為：

  在重量總和恰好等於 w 的情況下，
  從前 i 個東西裡挑選, 可組合出的最大價值。

背包問題(Knapsack)的 DP 遞迴式

- 定義 opt(i,w) 為：

  在重量總和恰好等於 w 的情況下，
  從前 i 個東西裡挑選，可組合出的最大價值。

- 則

$$opt(i,w) = \max\left\{\begin{array}{c} opt(i-1,w) \\ opt(i-1,w-a_i)+b_i \end{array}\right\}, \text{if } i > 1,$$

$$opt(i,w) = \left\{\begin{array}{ll} b_1, & \text{if } w = a_1 \\ -\infty, & \text{otherwise} \end{array}\right\}, \text{if } i = 1.$$

背包問題(Knapsack)的 DP 遞迴式

- 定義 opt(i,w) 為：

  在重量總和恰好等於 w 的情況下，
  從前 i 個東西裡挑選，可組合出的最大價值。

  - 此方法所需要的計算量為 O(n * W)

動態規劃的計算量

- 不同類型的問題，可能會需要不同的動態規劃 DP 遞迴式，相對應地，需要的計算量也會不同
- 最長遞增子序列
  - LIS(i) : 以 a_i 為結尾的 LIS
  - 計算量為 $O(n^2)$
- 背包問題
  - opt(i,w) : 重量w, 使用前i個的最大價值
  - 計算量為 $O(n * W)$

問題與解法的分類

問題與解法的分類

- 透過簡單的方法、規則、計算法,
  能夠直接、有效率地求得最佳解的問題

- 最佳解的範圍有限、
  且可以有效率回答每個單點值的合法性的問題。
  (可透過 binary search 搜尋得最佳解)

  以上兩類的問題，皆可有效率計算求得解答。
  (Computationally-Tractable)

- 例如： LIS (DP) 、 迷宮探索尋寶 、
  W10 (greedy或 binary search)

問題與解法的分類

- 有一類的問題 （背包問題是其中之一），

  除了使用窮舉法，考慮所有解答的可能性之外，我們不知道是否能有效率計算求取最佳解。

  (Computationally-Intractable)

- 當 input 的數值比較小的時候，可能可以透過一些方法求解
  - *Ex. Dynamic Programming (動態規劃)*