|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **說明**  假設有一個背包的負重最多可達8公斤，而希望在背包中裝入負重範圍內可得之總價物品，假設是水果好了，水果的編號、單價與重量如下所示：   |  |  |  |  | | --- | --- | --- | --- | | 0 | 李子 | 4KG | NT$4500 | | 1 | 蘋果 | 5KG | NT$5700 | | 2 | 橘子 | 2KG | NT$2250 | | 3 | 草莓 | 1KG | NT$1100 | | 4 | 甜瓜 | 6KG | NT$6700 |   **解法**  背包問題是關於最佳化的問題，要解最佳化問題可以使用「動態規劃」（Dynamic programming），從空集合開始，每增加一個元素就先求出該階段的最佳解，直到所有的元素加入至集合中，最後得到的就是最佳解。   以背包問題為例，我們使用兩個陣列value與item，value表示目前的最佳解所得之總價，item表示最後一個放至背包的水果，假設有負重量 1～8的背包8個，並對每個背包求其最佳解。   逐步將水果放入背包中，並求該階段的最佳解：   * 放入李子  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 背包負重 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | value | ０ | ０ | ０ | 4500 | 4500 | 4500 | 4500 | 9000 | | item | － | － | － | ０ | ０ | ０ | ０ | ０ |  * 放入蘋果  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 背包負重 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | value | ０ | ０ | ０ | 4500 | 5700 | 5700 | 5700 | 9000 | | item | － | － | － | ０ | 1 | 1 | 1 | ０ |  * 放入橘子  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 背包負重 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | value | ０ | 2250 | 2250 | 4500 | 5700 | 6750 | 7950 | 9000 | | item | － | 2 | 2 | ０ | 1 | 2 | 2 | ０ |  * 放入草莓  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 背包負重 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | value | 1100 | 2250 | 3350 | 4500 | 5700 | 6800 | 7950 | 9050 | | item | 3 | 2 | 3 | ０ | 1 | 3 | 2 | 3 |  * 放入甜瓜  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 背包負重 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | value | 1100 | 2250 | 3350 | 4500 | 5700 | 6800 | 7950 | 9050 | | item | 3 | 2 | 3 | ０ | 1 | 3 | 2 | 3 |   由最後一個表格，可以得知在背包負重8公斤時，最多可以裝入9050元的水果，而最後一個裝入的 水果是3號，也就是草莓，裝入了草莓，背包只能再放入7公斤（8-1）的水果，所以必須看背包負重7公斤時的最佳解，最後一個放入的是2號，也就 是橘子，現在背包剩下負重量5公斤（7-2），所以看負重5公斤的最佳解，最後放入的是1號，也就是蘋果，此時背包負重量剩下0公斤（5-5），無法 再放入水果，所以求出最佳解為放入草莓、橘子與蘋果，而總價為9050元。  **實作：**[**C**](http://openhome.cc/Gossip/AlgorithmGossip/KnapsackProblem.htm#C)[**Java**](http://openhome.cc/Gossip/AlgorithmGossip/KnapsackProblem.htm#Java)[**Python**](http://openhome.cc/Gossip/AlgorithmGossip/KnapsackProblem.htm#Python)**[Scala](http://openhome.cc/Gossip/AlgorithmGossip/KnapsackProblem.htm" \l "Scala)**[**Ruby**](http://openhome.cc/Gossip/AlgorithmGossip/KnapsackProblem.htm#Ruby)[**JavaScript**](http://openhome.cc/Gossip/AlgorithmGossip/KnapsackProblem.htm#JavaScript)[**Haskell**](http://openhome.cc/Gossip/AlgorithmGossip/KnapsackProblem.htm#Haskell)   * C   #include <stdio.h>  #include <stdlib.h>   #define LIMIT 8 // 重量限制   typedef struct {   char name[20];   int weight;   int price;  } Fruit;   void knapsack(Fruit\*, int\*, int\*, int, int); int min(Fruit\*, int);   int main(void) {   Fruit fruits[] = {{"李子", 4, 4500},   {"蘋果", 5, 5700},   {"橘子", 2, 2250},   {"草莓", 1, 1100},   {"甜瓜", 6, 6700}};  int items[LIMIT + 1] = {0};   int values[LIMIT + 1] = {0};     int length = sizeof(fruits) / sizeof(fruits[0]);  knapsack(fruits, values, items, length, LIMIT);   printf("物品\t價格\n");   int i;  for(i = LIMIT; i >= min(fruits, length); i -= fruits[items[i]].weight) {  printf("%s\t%d\n", fruits[items[i]].name, fruits[items[i]].price);   }   printf("合計\t%d\n", values[LIMIT]);    return 0;  }   void knapsack(Fruit\* fruits, int\* values, int\* items,   int length, int limit) {  int i, w;  for(i = 0; i < length; i++) {   for(w = fruits[i].weight; w <= limit; w++) {  int p = w - fruits[i].weight;  int newValue = values[p] + fruits[i].price;   if(newValue > values[w]) { // 找到階段最佳解   values[w] = newValue;   items[w] = i;   }  }   } }  int min(Fruit\* fruits, int length) {  int i, m;  for(i = 0, m = fruits[0].weight; i < length; i++) {  if(fruits[i].weight < m) {  m = fruits[i].weight;  }  }  return m; } |