3.7 節末問題 3.7 的解答

問題 3.7.1

答案如下。

如果不理解，可以回到3.7.1項～3.7.3項進行確認。

元素	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
值	1	1	1	3	5	9	17	31	57	105

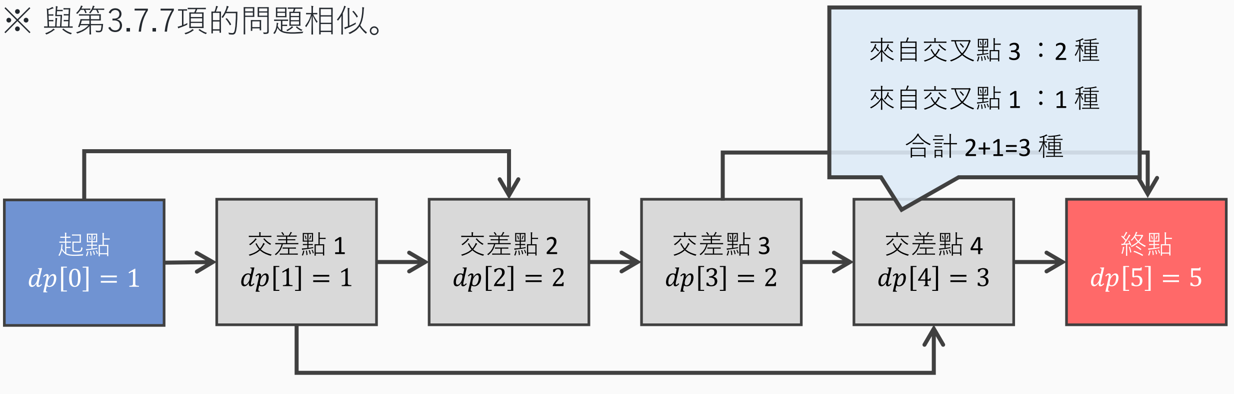
問題 3.7.2

答案是 5 種。

令 $dp[i]$ =（行進到交叉點 i 為止的方法數）來進行動態規劃法，可以得到答案。

其中，將起點設為交叉點 0，終點設為交叉點5。

※ 與第3.7.7項的問題相似。

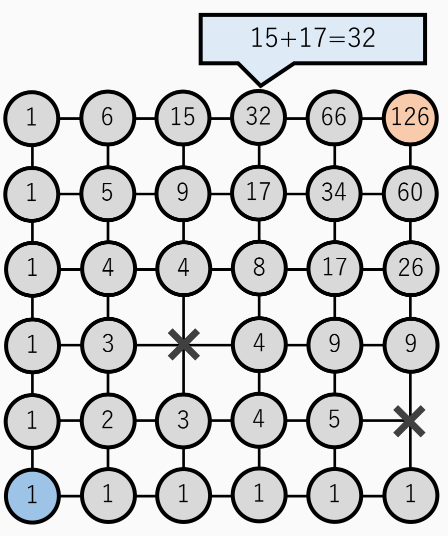


問題 3.7.3

答案是126 種。與問題3.7.2用相同的方針進行動態規劃法即可。

另外，注意從起點到終點以最短路徑（10步）移動時，只能向上和向右移動。

朝從左開始的第i列、從下開始第 j 行的格子 (i,j) 移動時，前一個格子為 $(i-1,j)$ 或 $(i,j-1)$。



問題 3.7.4

部分和問題可以用類似於背包問題（→**3.7.8項**）的以下方法來解決。

準備的陣列（二維陣列）

$\mathrm{dp}[i][j]$ ：從左開始到第 i 個卡片（以下稱為卡 i）之中，如果存在總和為 j 的組合，則為 `true` ，否則為`false`。

動態規劃法的轉換（$i = 0$）

顯然只有「什麼都不選」這種方法，因此：

- $\mathrm{dp}[0][j] = \texttt{true}$ $(j = 0)$
- $\mathrm{dp}[0][j] = \texttt{false}$ $(j \neq 0)$

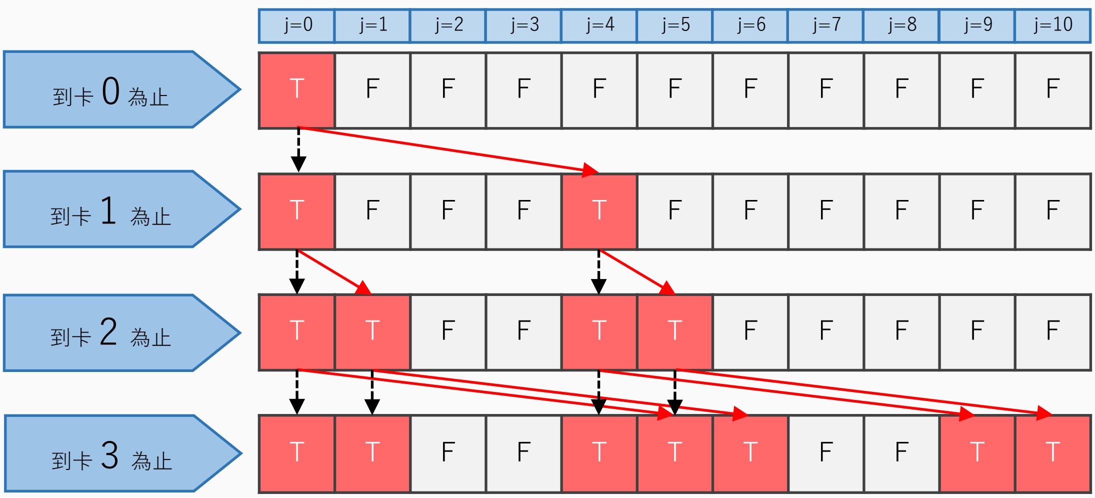
動態規劃的轉換（以 $i = 1, 2, \ldots, N$ 的順序計算）

到卡 i 為止，從中選擇以使總和為 j 的方法有以下兩種。（以最後的行動 [是否選擇卡 i] 來區分））

- 卡 $i-1$ 為止的總和為 $j - A_i$ 選擇卡 i
- 卡 $i-1$為止的總和為 j 不選擇卡 i

因此， $\mathrm{dp}[i-1][j-A_i], \mathrm{dp}[i-1][j]$ 之中至少一個為 true 的時候， $\mathrm{dp}[i][j] =$ `true` ，否則為`false` 。

例如， $N = 3, (A_1, A_2, A_3) = (4, 1, 5)$ 時，陣列 dp 如下所示。在此，當 $\mathrm{dp}[N][S] =$ `true` 時，存在總和為 S 的選擇方法。。



這個解法用C++實作如下。注意與背包問題的程式碼3.7.3不同，陣列 dp 為bool型態。

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int N, S, A[69];
bool dp[69][10009];

int main() {
    // 輸入
    cin >> N >> S;
    for (int i = 1; i <= N; i++) cin >> A[i];

    // 陣列的初始化
    dp[0][0] = true;
    for (int i = 1; i <= S; i++) dp[0][i] = false;

    // 動態規劃法
    for (int i = 1; i <= N; i++) {
        for (int j = 0; j <= S; j++) {
            // j < A[i] 時，無法選擇卡 i
            if (j < A[i]) dp[i][j] = dp[i-1][j];
            // j >= A[i] 時，有選擇 / 不選擇 兩種選項
            if (j >= A[i]) {
                if (dp[i-1][j] == true || dp[i-1][j-A[i]] == true) dp[i][j] = true;
                else dp[i][j] = false;
            }
        }
    }

    // 輸出答案
    if (dp[N][S] == true) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

※ Python等原始碼請參閱 chap3-7.md。

問題 3.7.5

如下述，可以將第1.1.4項的問題歸納為背包問題。

- 重量：物品的價格
- 價值：物品的卡路里
- 重量上限：500日元

問題 3.7.6

這個問題可以用以下方法解決。準備 2 個一維陣列，從第 1 天開始按順序進行動態規劃法處理。

準備的陣列（二維陣列）

$\mathrm{dp1}[i]$： 當第 i 天學習時，到目前為止實力提升的最大值

$\mathrm{dp2}[i]$： 最大當第 i 天不學習時，到目前為止實力提升的最大值

動態規劃法的轉換（$i = 0$）

由於從第 1 天開始才可以學習，設置 $\mathrm{dp1}[0] = 0, \mathrm{dp2}[0] = 0$ 等適當的值即可。

動動態規劃的轉換（以 $i = 1, 2, \ldots, N$ 的順序計算）

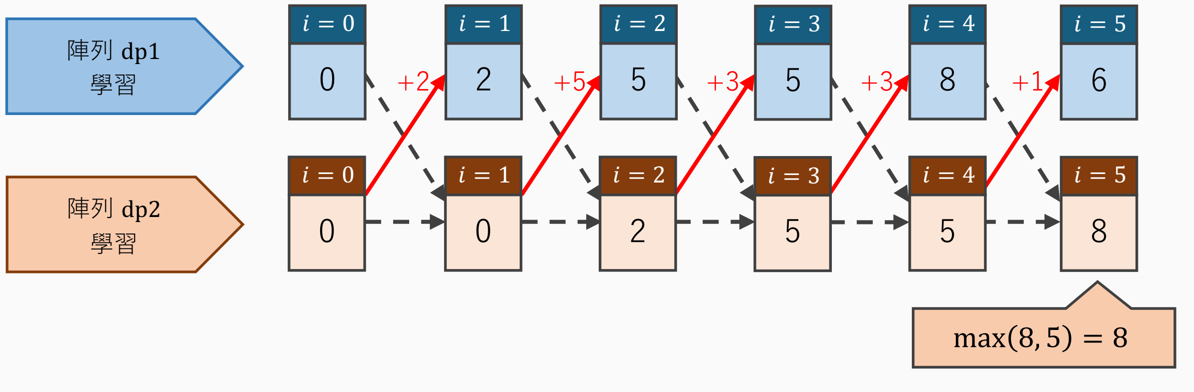
首先，第 i 天學習的方法只有以下一種，由於第 i 天學習的話，實力會提升 A_i，因此 $\mathrm{dp1}[i] = \mathrm{dp2}[i-1] + A_i$ 。

- 第 $i-1$ 天不學習（對應於 $\mathrm{dp2}[i-1]$ ）

另一方面，第 i 天不學習的方法有以下兩種，所以$\mathrm{dp2}[i] = \max(\mathrm{dp1}[i-1], \mathrm{dp2}[i-1])$。

- 第 $i-1$ 天學習（對應於 $\mathrm{dp1}[i-1]$ ）
- 第 $i-1$ 天不學習（對應於 $\mathrm{dp2}[i-1]$ ）

例如，當 $N = 5, (A_1, A_2, A_3, A_4, A_5) = (2, 5, 3, 3, 1)$ 時，陣列dp1 、 dp2的轉換如下所示。在此，由於所求的答案（第 N 天結束後實力提升的最大值）是 $\max(\mathrm{dp1}[N], \mathrm{dp2}[N])$ ，在此例中答案為 8 。



這個解法用 C++ 實作如下。注意限制條件為 $N \leq 500000$， $A_i \leq 10^9$ 之大，答案可能超過 10^{14} 。

由於 `int` 型態等32位元整數會發生溢出，因此建議使用 `long long` 型態等 64 位元整數。

```
#include <iostream>
#include <algorithm>
using namespace std;

long long N, A[500009];
long long dp1[500009], dp2[500009];

int main() {
    // 輸入
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> A[i];

    // 陣列的初始化
    dp1[0] = 0;
    dp2[0] = 0;

    // 動態規劃法
    for (int i = 1; i <= N; i++) {
        dp1[i] = dp2[i - 1] + A[i];
        dp2[i] = max(dp1[i - 1], dp2[i - 1]);
    }

    // 輸出答案
    cout << max(dp1[N], dp2[N]) << endl;
    return 0;
}
```

※ Python等原始碼請參閱 chap3-7.md。。