3.7 節末問題 3.7 の解答

問題 3.7.1

答えは以下のようになります。

分からない人は、3.7.1 項～3.7.3 項に戻って確認しましょう。

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

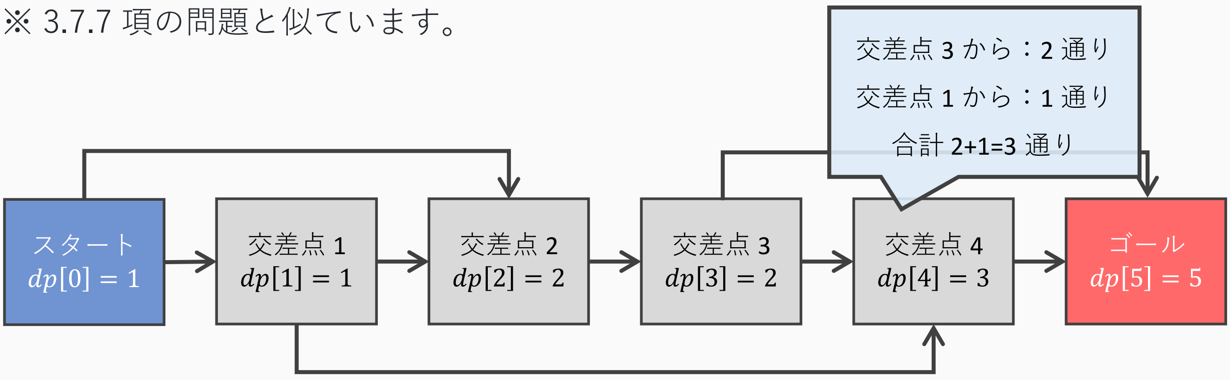
問題 3.7.2

答えは 5 通りとなります。

$dp[i]$ = (交差点 i まで行く方法の数) として動的計画法をすると、答えが分かります。

ただし、スタートを交差点 0、ゴールを交差点 5 とします。

※ 3.7.7 項の問題と似ています。

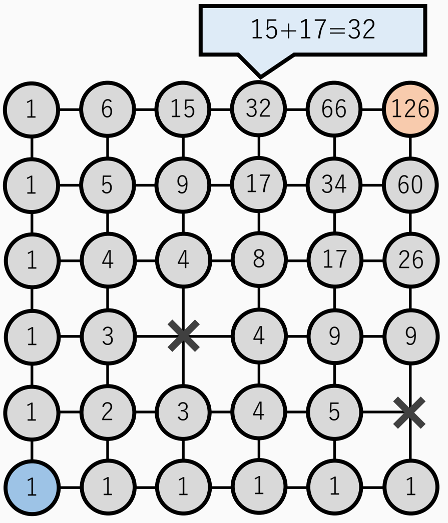


問題 3.7.3

答えは 126 通りとなります。問題 3.7.2 と同様の方針で、動的計画法をすると良いです。

なお、スタートからゴールまで最短経路（10 手）で移動するには、上方向と右方向にしか移動することができないことに注意してください。

左から i 行目・下から j 列目のマス (i,j) に行くときの直前のマスは、$(i-1,j)$ または $(i,j-1)$ です。



問題 3.7.4

部分和問題は、ナップザック問題（→**3.7.8項**）と似た、以下のような方法で解くことができます。

用意する配列（二次元配列）

$\mathrm{dp}[i][j]$：左から i 番目のカード（以下、カード i とする）までの中から、和が j になる組合せが存在するならば `true`、そうでなければ `false`

動的計画法の遷移（$i = 0$）

明らかに「何も選ばない」という方法しか存在しないので、

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

となります。

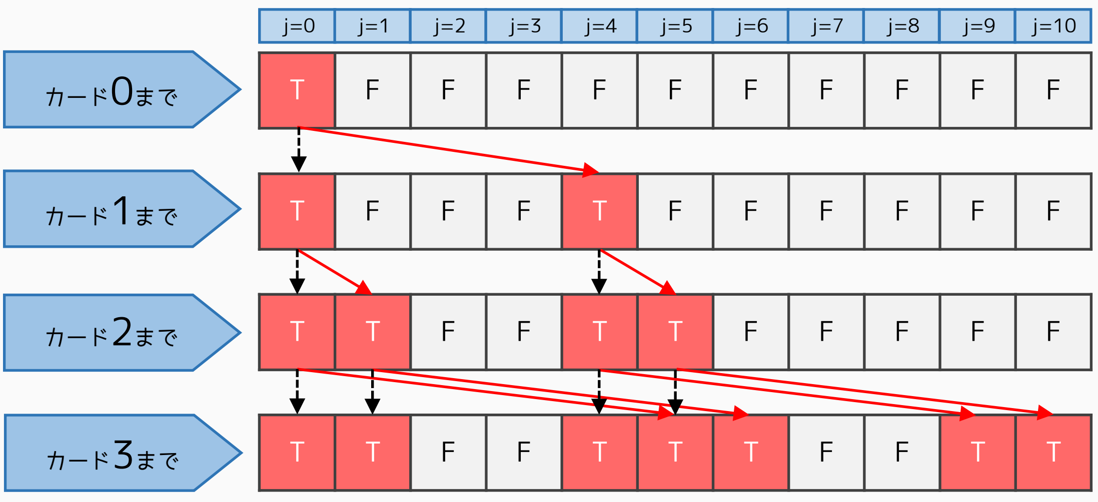
動的計画法の遷移（$i = 1, 2, \dots, N$ の順に計算）

総和が j になるようにカード i までの中から選ぶ方法は、以下の 2 つがあります。（最後の行動［カード i を選ぶか］で場合分けします）

- カード $i-1$ までの総和が $j - A_i$ であり、カード i を選ぶ
- カード $i-1$ までの総和が j であり、カード i を選ばない

したがって、$\mathrm{dp}[i-1][j-A_i], \mathrm{dp}[i-1][j]$ のうち少なくとも一方が true の場合 $\mathrm{dp}[i][j] =$ `true`、そうでなければ `false` となります。

たとえば、$N = 3, (A_1, A_2, A_3) = (4, 1, 5)$ の場合、配列 dp は次のようになります。ここで、$\mathrm{dp}[N][S] = \texttt{true}$ のとき、総和が S となるような選び方が存在します。



この解法を C++ で実装すると、以下のようになります。ナップザック問題のコード 3.7.3 とは異なり、配列 dp が bool 型であることに注意してください。

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

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

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

    // 配列の初期化
    dp[0][0] = true;
    for (int i = 1; i <= S; i++) dp[0][i] = false;

    // 動的計画法
    for (int i = 1; i <= N; i++) {
        for (int j = 0; j <= S; j++) {
            // j < A[i] のとき、カード i は選べない
            if (j < A[i]) dp[i][j] = dp[i-1][j];
            // j >= A[i] のとき、選ぶ / 選ばない 両方の選択肢がある
            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;
            }
        }
    }

    // 答えを出力
    if (dp[N][S] == true) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

※ Python などのソースコードは chap3-7.md をご覧ください。

問題 3.7.5

以下のようにすると、1.1.4 項の問題をナップザック問題に帰着することができます。

- 重さ：品物の値段
- 価値：品物のカロリー
- 重さの上限：500 円

問題 3.7.6

この問題は、以下のような方法で解くことができます。一次元配列を 2 個用意して、1 日目から順番に動的計画法の処理を行う方針です。

用意する配列（二次元配列）

$\mathrm{dp1}[i]$： i 日目に勉強する場合の、これまでの実力アップの最大値

$\mathrm{dp2}[i]$： i 日目に勉強しない場合の、これまでの実力アップの最大値

動的計画法の遷移（$i = 0$）

1 日目から勉強できるので、$\mathrm{dp1}[0] = 0, \mathrm{dp2}[0] = 0$ などの適切な値に設定しておけば良いです。

動的計画法の遷移（$i = 1, 2, \ldots, N$ の順に計算）

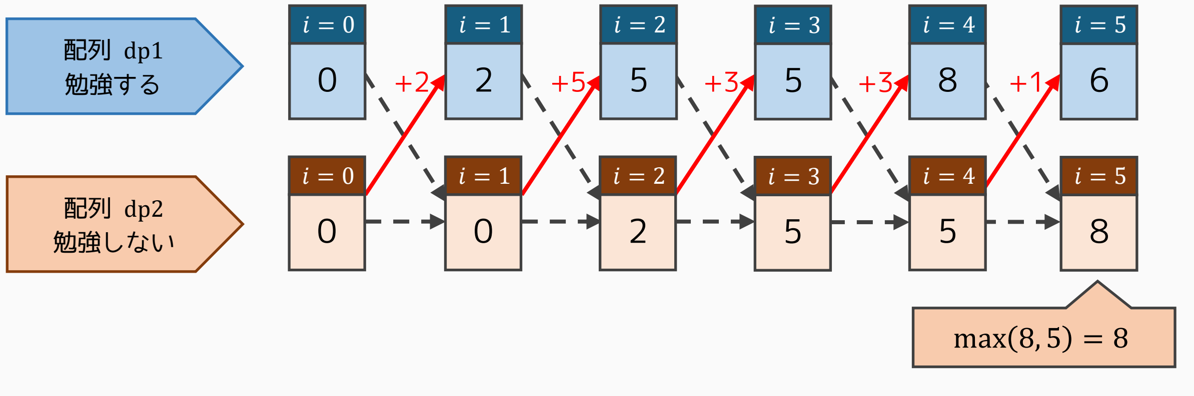
まず、i 日目に勉強する方法は以下の 1 つしかなく、i 日目に勉強すると実力が A_i 上がるため、$\mathrm{dp1}[i] = \mathrm{dp2}[i-1] + A_i$ となります。

- $i-1$ 日目に勉強しない（$\mathrm{dp2}[i-1]$ に対応）

一方、i 日目に勉強しない方法は以下の 2 つがあるため、$\mathrm{dp2}[i] = \max(\mathrm{dp1}[i-1], \mathrm{dp2}[i-1])$ となります。

- $i-1$ 日目に勉強する（$\mathrm{dp1}[i-1]$ に対応）
- $i-1$ 日目に勉強しない（$\mathrm{dp2}[i-1]$ に対応）

たとえば $N = 5, (A_1, A_2, A_3, A_4, A_5) = (2, 5, 3, 3, 1)$ の場合の配列 $\mathrm{dp1}, \mathrm{dp2}$ の遷移は以下のようになります。ここで、求める答え（N 日目を終えた後の実力アップの最大値）は $\max(\mathrm{dp1}[N], \mathrm{dp2}[N])$ なので、この例では答えが 8 となります。



この解法を C++ で実装すると、以下のようになります。なお、制約が $N \leq 500000$, $A_i \leq 10^9$ と大きく、答えが 10^{14} を超える可能性があります。
`int` 型などの 32 ビット整数ではオーバーフローを起こすため、`long long` 型などの 64 ビット整数を利用することが推奨されます。

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

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

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

    // 配列の初期化
    dp1[0] = 0;
    dp2[0] = 0;

    // 動的計画法
    for (int i = 1; i <= N; i++) {
        dp1[i] = dp2[i - 1] + A[i];
        dp2[i] = max(dp1[i - 1], dp2[i - 1]);
    }

    // 答えを出力
    cout << max(dp1[N], dp2[N]) << endl;
    return 0;
}
```

※ Python などのソースコードは chap3-7.md をご覧ください。