グラフとアルゴリズムとプログラムのやさしいおはなし

渡邉敏正

2022 年 6 月 11 日

第 13 回

1　部分和問題の解を求めてみよう

第 12 回で部分和問題の解の存在性を YES、NO で判定をするアルゴリズム (漸化式) を設計し、それに基づいて C 言語プログラムを作成しました。今回は一歩進めて、部分和問題の解集合を具体的に求めるアルゴリズムの設計に挑戦します。最終的は、設計したアルゴリズムに基づいた C 言語プログラムの作成までいきます。

部分和問題 (決定問題バージョン) は以下の定義でした。

部分和問題 (決定問題バージョン) の定義

(入力) $n\,(\geq 1)$ 個の正整数の集合 $A = \{a_1, \cdots, a_n\}$ および目標値 (正の整数) b

(出力) 以下を満たす添字集合 S_n が存在するとき YES；そうでないときは NO：

$$\sum_{i \in S_n} a_i \ = b \text{ かつ } S_n \subseteq I_n = \{1, \cdots, n\}$$

言い換えますと

> n 個の正整数の集合 $A = \{a_1, \cdots, a_n\}$ から適当に何個か選んでそれらの総和が b になるようにできれば YES、そうでなければ NO と答えなさい

という決定問題でした。なお、上記の定義における和の記述の仕方としては、

$$\sum_{a_i \in T} a_i \ = b \text{ かつ } T \subseteq \{a_1, \cdots, a_n\}$$

を満たす集合 T が存在するとき、という表現も可能ですが、ここでは添字集合で扱うことにしています。

定義における添字集合 S_n を**解添字集合**、解を構成する要素の集合を**解要素集合**とよぶことにします。ただし、混乱の恐れがない場合は S_n, T を単に**解集合**とよぶ場合があります。

第 12 回で示した例題 12.1 を再び使いますので、再掲しておきます。

例題 12.1 (再掲)

(入力) $n\,(=4)$ 個の正の整数 $\{a_1 = 3, a_2 = 2, a_3 = 6, a_4 = 5\}$、添字集合 $I_4 = \{1, 2, 3, 4\}$、目標値 $b\,(=7)$

(出力) 以下を満たす集合 S_4 が存在するとき YES；そうでないときは NO：

$$\sum_{i \in S_4} a_i \ = \ b\,(=7) \ \text{ かつ } S_4 \subseteq I_4 = \{1, \cdots, 4\}$$

この例題について、たとえば $S_4 = \{1, 2, 3\}$ ならば $\sum_{i \in S_4} a_i = a_1 + a_2 + a_3 = 11 \neq b$ ですし、$S_4 = \{2, 4\}$ ならば $\sum_{i \in S_4} a_i = a_2 + a_4 = 7 = b$ です。したがって、この例題については YES となります。$S_4 = \{2, 4\}$ が解添字集合で、$T = \{a_2, a_4\}$ が解要素集合です。

第 12 回は YES, NO を出力する決定問題を解きましたが、今回は部分和問題の解集合 S (および T) を具体的に求めるアルゴリズムを設計します。さらにこのアルゴリズムに基づいて部分和問題の解集合を求める C 言語プログラムを作成します。

今回の目的に沿って、決定問題バージョンの出力が YES の場合に解集合 S_n を求める形式に部分和問題の定義を拡張します。

部分和問題 (求解問題バージョン) の定義

(入力) $n\,(\geq 1)$ 個の正整数の集合 $A = \{a_1, \cdots, a_n\}$ および目標値 (正の整数) b

(出力) 以下を満たす添字集合 S_n が存在する否か判定し、存在する場合には解集合 S_n を求めなさい：

$$\sum_{i \in S_n} a_i = b \text{ かつ } S_n \subseteq I_n = \{1, \cdots, n\}$$

1.1 第 12 回の復習

第 12 回で説明した部分和問題 (決定問題バージョン) の解法に関して、簡単な復習をしておきます。

1.1.1 論理変数 $P(w, m)$ とその導入目的

- この問題を解くために 1 または 0 のいずれかの値をとる論理変数 $P(w, m)$ を考えます。ここで、$I_m = \{1, \cdots, m\}$ $(1 \leq m \leq n)$ とします。なお、この論理変数は $0 \leq w \leq b$ および $1 \leq m \leq n$ なる w および m すべてについて考えるのが基本ですが、議論の都合上、

  $w < 0$ の場合は、$P(w, m) = 0$ ($1 \leq m \leq n$ なるすべての整数 m について)

  と設定します。
- 論理変数 $P(w, m)$ 導入は以下の $\Longrightarrow$ 左側の命題 (添字集合 S_m が存在するか否か) を簡潔に表現することが目的の 1 つです (「ならば」を $\Longrightarrow$ と表記します) :

  ($w \leq b$ および $1 \leq m \leq n$ なるすべての整数 w および m について)

  (a) $\sum_{i \in S_m} a_i = w$ を満たす添字集合 $S_m \subseteq I_m$ が存在する $\Longrightarrow$ $P(w, m) = 1$

  (b) $\sum_{i \in S_m} a_i = w$ を満たす添字集合 $S_m \subseteq I_m$ が存在しない $\Longrightarrow$ $P(w, m) = 0$
- 例題 12.1 については、たとえば $S_3 = \{1, 2\} \subseteq I_3$ に対して $\sum_{i \in S_3} a_i = a_1 + a_2 = 3 + 2 = 5$ が成り立ちますので $P(5, 3) = 1$、あるいは $S_4 = \{2, 4\} \subseteq I_4$ に対して $\sum_{i \in S_4} a_i = a_2 + a_4 = 7 = b$ が成り立ちますので $P(7, 4) = 1$ などの表現になります。
- まず、目標値、添字集合、論理変数の値を表 7 にまとめておきます。そのあと補足説明をします。

注意 13.1 目標値 $w < 0$ の場合には (a)、(b) が共に成り立ちます。このことは第 12 回の注意 12.3 で含意命題として言及しています。(含意命題の詳細は第 14 回で説明の予定です。)

- 注意 13.1 により、以後は特に断らない限り、目標値 $w \geq 0$ として (a)、(b) が成り立つか否かについて考えていきます。なお、目標値は w と表記されるだけでなく $w - a_m$ などの形で現れることもありま

表 7　目標値、添字集合、論理変数と値のまとめ

目標値	$\sum_{i\in S_m} a_i = w$ を満たす $S_m \subseteq I_m$	論理変数の値
$w \geq 0$	存在する $\Longrightarrow$	$P(w,m)=1$
	存在しない $\Longrightarrow$	$P(w,m)=0$
$w < 0$	存在しない	$P(w,m)=0$

す。このときも $w - a_m \geq 0$ として考えます。また以後、$w < 0$ の場合に (a)、(b) が共に成り立つことは断りなしに使います。

- (a) と (b) が共に成り立っているとしますと、

$$(c)\quad \sum_{i\in S_m} a_i = w \text{を満たす添字集合} S_m \subseteq I_m \text{が存在する} \iff P(w,m)=1$$

および

$$(d)\quad \sum_{i\in S_m} a_i = w \text{を満たす添字集合} S_m \subseteq I_m \text{が存在しない} \iff P(w,m)=0$$

が成り立ちます[*1]。

- 見かけは違いますが (c) と (d) は同じことを述べています。(c) (あるいは (d)) によって、$\sum_{i\in S_m} a_i = w$ を満たす $S_m \subseteq I_m$ が存在するか否かの判定と $P(w,m)$ が 1 か 0 かの判定が同じことになります。

1.1.2　部分和問題の解法に向けて

- 上述の通り、論理変数 $P(w,m)$ の導入目的は条件を満たす集合 S_m が存在するか否かを $P(w,m)$ が 1 か 0 かで判断できるようにする、ということですが、これはあくまで定義の話、導入の目的です。これにより正確な判定ができることを証明する必要があります。そのために、

    $P(w,m)$ **が正しく計算されている** (あるいは、$P(w,m)$ **は正しく計算される)**

  という表現を使います。この意味は上述の命題 (a), (b) が共に成り立つことです。
- これらは $\Longrightarrow$ の前後を入れ替えた以下の対偶命題が共に成り立つことと同じことで、必要に応じて選択して使います。

    (b) の対偶：　$P(w,m)=1 \implies \sum_{i\in S_m} a_i = w$ を満たす添字集合 $S_m \subseteq I_m$ が存在する

    (a) の対偶：　$P(w,m)=0 \implies \sum_{i\in S_m} a_i = w$ を満たす添字集合 $S_m \subseteq I_m$ が存在しない

- 部分和問題を解くには「$P(b,n)$ **が正しく計算されている**」ことがポイントになります。例題 12.1 について言えば、$P(7,4)$ $(w=7,\ m=n=4)$ が正しく計算されているとすると

$$P(7,4)=1 \Longleftrightarrow \sum_{i\in S_4} a_i = 7 \geq 0 \text{を満たす} S_4 \subseteq I_4 \text{が存在する}$$

  ですので、

    $P(7,4)=1$ ならば YES　　$P(7,4)=0$ ならば NO

[*1] $\Longleftrightarrow$ は左側が成り立てば右側が成り立ち、かつ右側が成り立てば左側が成り立つことを表します。

を出力すればこの例題を解いたことになります。

- 論理変数 $P(w,m)$ の導入によって、$\sum_{i\in S_n} a_i = b$ を満たす添字集合 $S_n \subseteq I_n$ が存在するか否かの判定問題を $P(n,b)$ の値が 1 か 0 かで判定する問題に置き換えて解くということです。

1.2 漸化式に基づく解法

1.2.1 初期値設定と漸化式

「$P(b,n)$ は正しく計算される」ことを示すために $P(w,m)$ について以下の初期値設定と漸化式を考えました。

- $w < 0$ のとき
  $P(w,m) = 0$ （$1 \le m \le n$ なるすべての整数 m について）
- $m = 1$ のとき
  $P(0,1) = 1,\ P(a_1,1) = 1$ (例題 12.1 では、$a_1 = 3$)
  $P(w,1) = 0$ ($1 \le w \le b$ かつ $w \ne a_1$ なるすべての整数 w について)
- $2 \le m \le n$ のとき
  $P(w,m) = P(w,m-1) \vee P(w-a_m,m-1)$ （$0 \le w \le b$ なるすべての整数 w について）
  (ここで、$\vee$ は「または」を表す)

1.2.2 論理変数、漸化式および添字集合について

$2 \le m \le n$ のときの論理変数、漸化式および添字集合には以下の関係があります。

- $P(w,m) = P(w,m-1) \vee P(w-a_m,m-1) = 1$ が意味するのは
  $P(w,m-1) = 1$ (したがって、$w \ge 0$) である
  または
  $P(w-a_m,m-1) = 1$ (したがって、$w \ge a_m$) である
  が成り立つことです。
- つまり、
  $\sum_{i\in S_m} a_i = w$ を満たす $S_m \subseteq I_m$ が存在する
  ということは
  $\sum_{i\in S_{m-1}} a_i = w$ を満たす $S_{m-1} \subseteq I_{m-1}$ ($\subseteq I_m$) が存在する
  ($S_m \leftarrow S_{m-1}$ と考えればよい)
  あるいは
  $\sum_{i\in S_{m-1}} a_i = w - a_m$ を満たす $S_{m-1} \subseteq I_{m-1}$ が存在する
  ($S_m \leftarrow S_{m-1} \cup \{m\}$ と考えればよい)
  のどちらかが成り立つことと同じであるということです。
- 一方、$w \ge 0$ の場合に、$P(w,m) = 0$ が意味するのは以下が成り立つことです。
  $P(w,m-1) = 0$　かつ　$P(w-a_m,m-1) = 0$
- つまり、
  $\sum_{i\in S_m} a_i = w$ を満たす $S_m \subseteq I_m$ が存在しない

ということは

$\sum_{i\in S_{m-1}} a_i = w$ を満たす $S_{m-1} \subseteq I_{m-1}$ が存在しない

および

$\sum_{i\in S_{m-1}} a_i = w - a_m$ を満たす $S_{m-1} \subseteq I_{m-1}$ が存在しない

の両方が成り立つことと同じであるということです。

1.2.3 漸化式とアルゴリズム

- 上記の初期値設定と漸化式は $P(b,n)$ の値を計算するための 1 つのアルゴリズムになっています。
- したがって、漸化式で計算された $P(w,m)$ の値 1、0 と、$\sum_{i\in S_m} a_i = w$ を満たす添字集合 $S_m \subseteq I_m$ が存在する、しないが正確に対応していることを示すことが正当性のポイントです。
- すなわち、上記の漸化式を部分和問題を解くアルゴリズムとしてみた場合の正当性は、$w \leq b,\ 1 \leq m \leq n$ なるすべての整数 w, m について以下を証明することになります：

  $P(w,m)$ **は正しく計算される**

上述の漸化式に基づいて次の命題が成り立つことを第 12 回で示しました。

命題 12.1 $w \leq b,\ 1 \leq m \leq n$ なるすべての整数 w, m について $P(w,m)$ は正しく計算される。

1.2.4 漸化式に基づいた C 言語プログラムと例題 12.1 に対する実行結果

上述の漸化式に基づいて第 12 回で示した C 言語プログラム `subsetsum-8` が図 134 です。行数を抑えるために { や } のカッコをできるだけ省略して記述しています。なお、行番号は説明用に入れており、実際のプログラムに記述するとエラーになります。

例題 12.1 に対する `subsetsum-8.c` の実行結果が図 135 で、出力は YES です。解は添字集合としては $\{2,4\}$ であり、要素集合として $\{a_2 = 2,\ a_4 = 5\}$ です。

1.3 部分和問題の解探索

1.3.1 解集合を求めるための準備

解集合を求めるためにいくつかの準備をします。まず、部分和問題 (求解問題バージョン) の定義、論理変数 $P(w,m)$ および漸化式を再掲しておきます。論理変数、漸化式は判定問題の場合と同様ですが、解集合を求める操作に合わせて定義を少し拡張します。ただし、拡張部分は解の存在判定には使用せず、解の存在確定後に解探索を行う場合に使います。この定義の拡張に関してはあとで補足説明をします。

部分和問題 (求解問題バージョン) の定義

(入力) $n\,(\geq 1)$ 個の正整数の集合 $A = \{a_1, \cdots, a_n\}$ および目標値 (正の整数) b

(出力) 以下を満たす添字集合 S_n が存在する否か判定し、存在する場合には S_n を求めなさい：

$$\sum_{i\in S_n} a_i = b \text{ かつ } S_n \subseteq I_n = \{1, \cdots, n\}$$

論理変数 $P(w,m)$　($w \leq b$ および $1 \leq m \leq n$ なるすべての整数 w, m について)

$P(w,m) = 1 \iff \sum_{i\in S_m} a_i = w$ を満たす添字集合 $S_m \subseteq I_m = \{1, \cdots, m\}$ が存在する

```
1   // Solving the SUBSET_SUM problem
2   #include <stdio.h>
3   #define n 4 // #Elements
4   #define b 7 // Target value
5
6   int main(void)
7   {
8   int a[n+1] = { 0,3,2,6,5 }; // a[0] は未使用
9   // int c[n+1];  ここでの YES, NO の判定では使用しない；あとで解を求める際に使用
10  int P[b+1][n+1];
11  int i, w, k, m;
12  // 初期値設定
13  for (w = 0; w <= b; w++) {
14      if ( (w==0) || (w == a[1]) )
15          P[w][1] = 1;
16      else
17          P[w][1] = 0;  // 0:false  1:true
18  }
19  for (w = 0; w <= b; w++)
20      P[w][0] = 0;
21  // 漸化式の計算
22  for (m = 2; m <= n; m++) {
23      for (w = 0; w <= b; w++) {
24          if ((P[w][m - 1] == 1) || ((w - a[m] >= 0) && (P[w - a[m]][m - 1] == 1)))
25              P[w][m] = 1;
26          else
27              P[w][m] = 0;
28      }
29  }
30  //  入力データと計算結果の表示
31  printf("\n****** The SUBSET_SUM problem *****\n#Elements n=%d,
    Target value b=%d\n", n, b);
32  for (w = 1; w <= n; w++)
33      printf("a[%d]=%d  ", w, a[w]);
34  printf("\n\n");
35  for (w = 0; w <= b; w++) {
36      for (m = 1; m <= n; m++)
37          printf("P(%d,%d)=%d  ", w, m, P[w][m]);
38      printf("\n");
39  }
40  printf("\nThe answer is ");
41  if (P[b][n] == 1)
42      printf("YES\n");
43  else
44      printf("NO\n");
45  printf("\n");
46
47  return 0;
48  }
```

図 134　部分和問題を解く C 言語プログラム subsetsum-8.c

```
Toshimasa-no-MacBook-Pro:subset_sum_new watanabe$ gcc -o subsetsum-8 subsetsum-8.c
Toshimasa-no-MacBook-Pro:subset_sum_new watanabe$ ./subsetsum-8

****** The SUBSET_SUM problem *****
#Elements n=4,  Target value b=7
a[1]=3  a[2]=2  a[3]=6  a[4]=5

P(0,1)=1  P(0,2)=1  P(0,3)=1  P(0,4)=1
P(1,1)=0  P(1,2)=0  P(1,3)=0  P(1,4)=0
P(2,1)=0  P(2,2)=1  P(2,3)=1  P(2,4)=1
P(3,1)=1  P(3,2)=1  P(3,3)=1  P(3,4)=1
P(4,1)=0  P(4,2)=0  P(4,3)=0  P(4,4)=0
P(5,1)=0  P(5,2)=1  P(5,3)=1  P(5,4)=1
P(6,1)=0  P(6,2)=0  P(6,3)=1  P(6,4)=1
P(7,1)=0  P(7,2)=0  P(7,3)=0  P(7,4)=1

The answer is YES
```

図 135　例題 12.1 に対する subsetsum-8.c の実行結果

$P(w,m)$ **に対する漸化式** (初期値設定と論理式)

($w<0$ のとき)

$P(w,m)=0$ ($1\leq m\leq n$ なるすべての整数 m について)

($m=1$ のとき)

$P(0,1)=1,\ P(a_1,1)=1\,(a_1=3)$

$P(w,1)=0$ ($1\leq w\leq b$ かつ $w\neq a_1$ なるすべての整数 w について)

($2\leq m\leq n$ のとき)

$P(w,m)=P(w,m-1)\ \vee\ P(w-a_m,m-1)$ ($0\leq w\leq b$ なるすべての整数 w について)

(ここで、$\vee$ は「または」を表す)

$P(w,m)$ **の定義の拡張**：$m=0$ のときの値設定 (解の存在判定には使いません)

$P(w,0)=0$ ($w\leq b$ なるすべての整数 w について)

注意 13.2 この定義の拡張に関連して補足説明をしておきます。

- まず、$w<0$ の場合についてです。漸化式計算の整合性の観点から、$w<0$ の場合にも論理変数 $P(w,m)$ は定義しています。第 12 回でも何度か言及していますが (例えば、1.4.1 プログラム記述における漸化式の扱い、など)、$w<0$ なる論理変数 $P(w,m)$ は $P(b,n)$ の値が 1 か 0 かの判定操作では実質的には使いませんし、C 言語プログラム作成では $w\geq 0$ の場合のみを扱ってよいことも説明しています。また、これから扱う解探索は $P(b,n)=1$ の場合に実行されます。このときの判定操作、およびその後の解探索ではすべて $w\geq 0$ なる論理変数 $P(w,m)$ が使われます。したがって、アルゴリズム設計およびプログラム記述では $w\geq 0$ の場合のみを扱ってよいことになります。
- 第 12 回で示した C 言語プログラム subsetsum-8.c では $P(w,m)\,(0\leq w\leq b,\ 1\leq m\leq n)$ については整数型 2 次元配列 P[b+1][n+1] を宣言し、そこに値を格納しました。その際に P[0][0] 〜

P[b][0] の値は 0 に初期化しましたが、$P(b,n)$ の値が 1 か 0 かを判定するための漸化式計算では未使用でした。(この初期化は、これから取り組む解探索のために予め組み込んでおいたことです。) また、$P(w,0)\,(0 \le w \le b)$ は未定義で解の存在判定の議論では使っていませんでした。

- これから示す解探索アルゴリズムの設計では、subsetsum-8.c によって解の存在判定をしますので、$P(w,m)\,(0 \le w \le b,\ 1 \le m \le n)$ の値を整数型 2 次元配列 P[b+1][n+1] に格納することは同じです。
- さらに $P(w,0) = 0\ (0 \le w \le b)$ と $P(w,m)$ の定義を拡張しましたが、これらの値は P[0][0] 〜 P[b][0] に格納します。(実際には宣言した時点で値が 0 に初期化されます。) したがって、上記の拡張でこれまでの解の存在判定の操作が影響を受けることはありません。
- なお、P[0][0] 〜 P[b][0] にある $P(0,0)$〜$P(b,0)$ の値 (すべて 0) は探索の終点を知らせる一種の記号としてのみ利用します。
- 説明の都合で、$0 \le w \le b, 0 \le m \le n$ なる w, m に対する $P(w,m)$ の値の集合を 2 次元配列 P[b+1][n+1] の形状イメージでとらえます。$P(w,0)$〜$P(w,n)$ を w の行 (横並び)、$P(0,m)$〜$P(b,m)$ を m の列 (縦並び) とよぶことにします。「w の」や「m の」は省略する場合があります。$P(w,m)$ の値を 2 次元配列 P[b+1][n+1] に格納する場合の対応を考えれば混乱は生じないと思います。

$P(w,m)$ の定義と漸化式によって明らかなことですが、$P(w,m)$ の値の分布について述べておきます。

- $w \ge 0$, $m \ge 2$ のとき、$P(w,m-1) = 1$ ならば漸化式によって $P(w,m) = 1$ です。
- $P(0,m) = 1\,(1 \le m \le n)$ と解添字集合との関係も述べておきます。$P(0,m) = 1$ のとき、$S = \emptyset$ としますと

  $\sum_{i\in S} a_i = 0$ かつ $S = \emptyset \subseteq I_m\,(1 \le m \le n$ なるすべての整数 m について $)$

  ですので、$S = \emptyset$ は

  $P(0,m) = 1 \quad \Longleftrightarrow \quad \sum_{i\in S_m} a_i = w = 0$ を満たす添字集合 $S_m \subseteq I_m$ が存在する

  なる関係における S_m の条件を満たします。しかしながら、$i \in S$ なる要素 a_i は存在しません。これから説明していく解探索の操作の中では、$i \in S_m$ なる添字 i については解添字の候補として記憶するために $c[i] \leftarrow 1$ なる特性ベクトルの更新をしますが、この場合は特性ベクトルの更新はありません。したがって解添字集合の要素構成に影響を与えません。
- 漸化式の初期値設定から、$P(0,1) = P(a_1,1) = 1$ であり、$1 \le w \le b$ かつ $w \ne a_1$ ならば $P(w,1) = 0$ です。

以上のことを次の命題としてまとめておきます。

命題 13.1 $P(w,m)$ について以下が成り立つ。

(1) $0 \le w \le b,\ 2 \le m \le n$ なる整数 w, m について、$P(w,m-1) = 1$ ならば $P(w,m) = 1$ である。

(2) $1 \le m \le n$ なる整数 m について、$P(0,m) = P(a_1,1) = 1$ である。

(3) $1 \le w \le b$ かつ $w \ne a_1$ なる整数 w について、$P(w,1) = 0$ である。

命題 13.1 に基づいて、$P(w,m)$ の値のいくつかが図 136 に示す分布になることがわかります。$m = 0$ の列が定義の拡張部分です。$m = 1$ の列から $m = n$ の列までの部分に解の存在判定における $P(w,m)$ の値が格納されます。$m = 0$ の列は解の存在判定では使いません。解集合を求める場合に使います。

$w \diagdown m$	0	1	2	$\cdots$	n
0	0	1	1	$\cdots$	1
1	0	0			
$\vdots$	$\vdots$	$\vdots$			
	0	0			
a_1	0	1	1	$\cdots$	1
	0	0			
$\vdots$	$\vdots$	$\vdots$			
b	0	0			

図 136　$P(w,m)$ の値の分布 (基本となる部分のみ)；図では $1 < a_1 < b$ の場合を示しています

1.3.2　決定ペアと関数 leftsearch

ここで、$P(w,m-1)=0$ かつ $P(w,m)=1\,(\,0 \leq w \leq b,\ 1 \leq m \leq n\,)$ のとき、これらを**決定ペア** (decision pair) と名付けておきます。

- 各 $w\,(\geq 0)$ の行について、$P(w,0)=0$ から始まりますし、一度 $P(w,m)=1$ となれば命題 13.1 (1) により以後はこの行の右方向には 1 が続きます。
- したがって、各行には決定ペアは高々 1 組存在します。また、もし $P(w,m)=1$ が存在すればこの行で $P(w,m)=1$ を含めた左方向に決定ペアが存在します。

$P(w,m)=1\,(\,w \geq 0,\, m \geq 1,\,)$ なる変数が存在したとき、同じ行を左方向に移動して決定ペアを探す関数 **leftsearch** を示しておきます。

関数 **leftsearch(P, w, m)**　　/* $P(w,m)=1\,(\,w \geq 0,\, m \geq 1)$ */

Step1.　$[(P(w,m)=1)]$ が成り立つ間 ($m \geq 1$ である)、以下の操作を反復する：

$m \leftarrow m-1;$

Step2.　反復操作の終了後、/* $P(w,m)=0,\ P(w,m+1)=1$ */

$k \leftarrow m+1;$

と設定して、k を戻り値として呼び出し側に返す。

/* 決定ペア $P(w,k-1)=0,\, P(w,k)=1$ */

上述のことから、$P(w,m)=1\,(\,w \geq 0,\, m \geq 1,\,)$ なる変数から開始すれば関数 **leftsearch** により必ず決定ペア $P(w,k-1)=0,\, P(w,k)=1\,(\,k \geq 1\,)$ を見つけることができますので、関数 **leftsearch** の正当性は明らかです。

関数 leftsearch を開始する場合に着目した $P(w,m)=1$ なる論理変数 (および値) を、この変数から関数 **leftsearch** あるいは解の探索を始めるという意味で、**開始変数**と呼ぶことにします。以下では、「$P(w,m)=1$ を開始変数として leftsearch を開始した」などと表現します。

このあと、関数 leftsearch を利用した解探索を説明します。まず例題 12.1 に対して解探索を説明してイ

メージを掴んでいただこうと思います。そのあとで一般的な解探索アルゴリズムを示します。

1.3.3 例題に対する解探索

まず、皆さんに解探索のイメージをつかんでいただくために、例題 12.1 ($b = 7$ で出力が YES) の解探索を実行してみます。加えて、例題 12.1 で目標値を $b = 11$ と変更した場合 (出力は YES) についても解探索を実行してみます。これらの 2 つの例で解の探索方法やそのパターンが見えてくると思います。そのあとで、解探索の一般的なルールを説明します。

注意 13.3 これ以降、A の要素 a_i を $a[i]$ と表記する場合があります。今後 a_i の添字 i を k_j などと表記する必要が出てきます。その場合には添字が小さくて見にくくなりますので、それを避けるのが大きな理由です。a_{k_j} よりは $a[k_j]$ の方が見やすいと思います。数式などではできるだけ a_i の形で表記し、プログラムのデータのイメージの場合には $a[i]$ と表記する、といった区別は意識しますが、厳密ではないことをお断りしておきます。プログラム作成では A の要素 a_i は整数型の 1 次元配列要素 `a[i]` に格納しますので、このような表記をしても混乱は生じないと思います。

(1) 例題 12.1 において $b = 7$ の場合

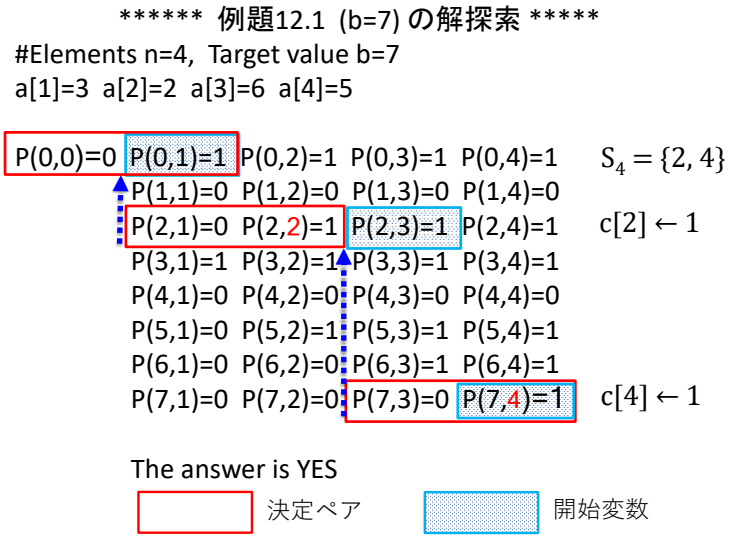

図 137　例題 12.1 における $b = 7$ の場合の解探索

図 137 には、例題 12.1 で $b = 7$ の場合における $P(w, m)$ の値を 2 次元配列状に並べています。図 135 で示した `subsetsum-8.c` の実行結果に拡張部分の $P(0, 0) = 0$ を追加しています。この上で解探索を説明します。

1. $P(7, 4) = 1$ を開始変数として **leftsearch** によって解探索を始めます。長さ $n + 1$ の特性ベクトル c の $c[1]$〜$c[n]$ の部分を $c[m] \leftarrow 0$ ($1 \leq m \leq n$ なるすべての整数 m について) と初期設定しておきます。$c[0]$ は使用しません。
2. (`leftsearch` により) 開始変数から左に移動して、$P(7, m-1) = 0$ かつ $P(7, m) = 1$ なるペア (つまり決定ペア) を探します。いま、$P(7, 3) = 0$, $P(7, 4) = 1$ が決定ペアです。$c[4] \leftarrow 1$ とします。
3. ここで $a[4] = 5$ かつ $P(7, 4) = P(7, 3) \vee P(7 - a[4], 3) = P(7, 3) \vee P(2, 3) = 1$ です。$P(7, 3) = 0$ かつ $P(2, 3) = 1$ ですから、必ず $P(7, 3) = 0$ と同じ列の上方に $P(2, 3) = 1$ がありますのでそこに移動

します。

- ここで、$c[4] \leftarrow 1$ とする理由を説明します。
- $P(7,3)=0$, $P(7,4)=1$ が決定ペアで、$P(7,4)=P(7,3) \vee P(7-a[4],3)=P(7,3) \vee P(2,3)=1$ ですので、$P(7-a[4],3)=P(2,3)=1$ ($a[4]=5$) です。したがって以下の 3 命題が成り立ちます。
  - $\sum_{i \in S_3} a_i = 7$ を満たす添字集合 $S_3 \subseteq I_3$ は存在しない
  - $\sum_{i \in S_4} a_i = 7$ を満たす添字集合 $S_4 \subseteq I_3$ が存在する
  - $\sum_{i \in R_3} a_i = 7-a[4]=2$ を満たす添字集合 $R_3 \subseteq I_3$ が存在する
- このとき $R_4 \leftarrow R_3 \cup \{4\}$ とすると、$4 \in R_4$ であり、かつ $R_4 \subseteq I_4$ は $\sum_{i \in R_4} a_i = 7$ を満たします。つまり、決定ペア $P(7,3)=0$, $P(7,4)=1$ は目標値が 7 のとき添字 4 を含む解添字集合が存在することを示しています。そこで $c[4] \leftarrow 1$ として添字 4 を解添字の候補として記憶します。

4. $P(2,3)=1$ を改めて開始変数と設定して、同様の操作を反復します。ここでは目標値を $7-a[4]=7-5=2$ と設定します。目標値が変更される点には注意してください。
5. (`leftsearch` により) 開始変数から左に移動して、決定ペア $P(2,1)=0$, $P(2,2)=1$ を見つけます。$c[2] \leftarrow 1$ とします。
6. ここで $a[2]=2$ かつ $P(2,2)=P(2,1) \vee P(2-a[2],1)=P(2,1) \vee P(0,1)=1$ です。$P(2,1)=0$ かつ $P(0,1)=1$ ですから、$P(2,1)=0$ と同じ列の上方に $P(0,1)=1$ がありますので $P(0,1)=1$ に移動します。
   - $c[2] \leftarrow 1$ とする理由は上記 $c[4] \leftarrow 1$ の場合と同様ですが、再度説明しておきます。
   - $P(2,1)=0$, $P(2,2)=1$ が決定ペアで、$P(2,2)=P(2,1) \vee P(2-a[2],1)=P(2,1) \vee P(0,1)=1$ ですので、$P(2-a[2],1)=P(0,1)=1$ ($a[2]=2$) です。したがって以下の 3 命題が成り立ちます。
     - $\sum_{i \in S_1} a_i = 2$ を満たす添字集合 $S_1 \subseteq I_1$ は存在しない；
     - $\sum_{i \in S_2} a_i = 2$ を満たす添字集合 $S_2 \subseteq I_2$ が存在する；
     - $\sum_{i \in R_1} a_i = 2-a[2]=0$ を満たす添字集合 $R_1 \subseteq I_1$ が存在する (このとき $R_1=\emptyset$ です)；
   - ここで $R_2 \leftarrow R_1 \cup \{2\}=\{2\}$ とすると、$2 \in R_2$ であり、かつ $R_2 \subseteq I_2$ は $\sum_{i \in R_2} a_i = 2$ を満たします。つまり、決定ペア $P(2,1)=0$, $P(2,2)=1$ は目標値が 2 のとき添字 2 を含む解添字集合が存在することを示しています。そこで $c[2] \leftarrow 1$ として添字 2 を記憶します。
7. $P(0,1)=1$ を改めて開始変数と設定します。ここでは目標値を $2-a[2]=2-2=0$ と設定します。
8. 開始変数から左に移動して、決定ペア $P(0,0)=0$, $P(0,1)=1$ を見つけます。目標値は $2-a[2]=0$ となっており、この場合は特性ベクトル c の更新はしません。$P(w,0)=0$ なる変数に到着しましたので、解探索をここで終了します。
9. 決定ペア $P(0,0)=0$, $P(0,1)=1$ について特性ベクトル c の更新をしない理由を説明します。命題 13.1 の前で述べた $P(w,m)$ と解添字集合との関係の説明でも言及していますが、$P(0,1)=1$ は空集合 $\emptyset$ が $\sum_{i \in S_1} a_i = 0$ を満たす添字集合 $S_1 \subseteq I_1$ に該当することを意味します。空集合は解集合の構成には関与しませんので、添字 1 を解添字の候補に加えることはしません。したがって特性ベクトルの更新はありません。これは注意すべきことです。なお、終了時に添字を解添字の候補に加える操作 (特性ベクトルの更新) をする場合はあります。このあとの説明で出てきます。
10. 解探索が終了したとき、$c[4]=1$, $c[2]=1$ なる添字 4, 2 に着目しますと
    - 添字 $4 \in R_4$ による目標値の更新が $7-a[4]=7-5=2$、

- 添字 $2 \in R_2$ による目標値の更新が $2 - a[2] = (7 - a[4]) - a[2] = 7 - a[4] - a[2] = 7 - 5 - 2 = 0$ ですので、
- $7 = a[4] + a[2]$ が成り立ちます。
- すなわち、$S_4 = \{2, 4\} \subseteq I_4$ は $\sum_{i \in S_4} a_i = 7$ を満たします。

11. 終了時の添字集合 $S_4 = \{2, 4\}$ を解添字集合 ($\{a[2] = 2, a[4] = 5\}$ を解要素集合) として出力します。

(2) 例題 12.1 において $b = 11$ の場合

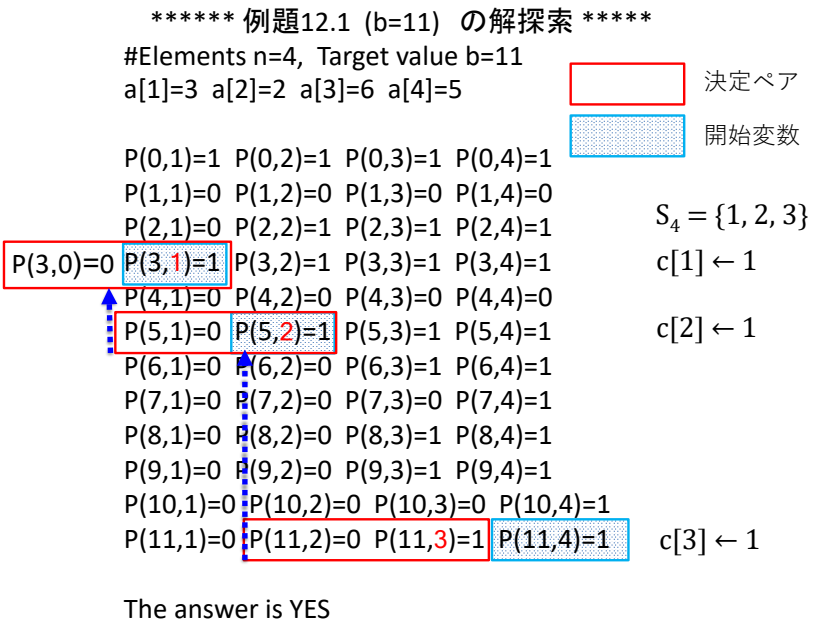

図 138　例題 12.1 における $b = 11$ の場合の解探索

図 138 には、例題 12.1 で $b = 11$ と変更した場合の $P(w, m)$ の値を 2 次元配列状に並べています。ここでは `subsetsum-8.c` の実行結果に拡張部分の $P(3, 0) = 0$ を追加しています。この上でもう一度解探索を説明します。(1) の場合と異なり、終了時に特性ベクトルの更新をします。

1. $P(11, 4) = 1$ を開始変数として **leftsearch** によって解探索を始めます。特性ベクトルを $c[m] \leftarrow 0$ ($1 \leq m \leq n$ なるすべての整数 m について) と初期設定しておきます。
2. (`leftsearch` により) 開始変数から左に移動して、決定ペアを探します。いま、$P(11, 2) = 0$, $P(11, 3) = 1$ が決定ペアです。$c[3] \leftarrow 1$ とします。
3. ここで $a[3] = 6$ かつ $P(11, 3) = P(11, 2) \vee P(11 - a[3], 2) = P(11, 2) \vee P(5, 2) = 1$ です。$P(11, 2) = 0$ かつ $P(5, 2) = 1$ ですから、$P(11, 2) = 0$ と同じ列の上方に $P(5, 2) = 1$ がありますので、そこに移動します。
4. $P(5, 2) = 1$ を改めて開始変数とします。ここでは目標値を $11 - a[3] = 11 - 6 = 5$ と設定します。
5. (`leftsearch` により) 開始変数から左に移動して決定ペア $P(5, 1) = 0$, $P(5, 2) = 1$ を見つけます。$c[2] \leftarrow 1$ とします。
6. ここで $a[2] = 2$ かつ $P(5, 2) = P(5, 1) \vee P(5 - a[2], 1) = P(5, 1) \vee P(3, 1) = 1$ です。$P(5, 1) = 0$ かつ $P(3, 1) = 1$ ですから、$P(5, 1) = 0$ と同じ列の上方に $P(3, 1) = 1$ がありますので、$P(3, 1) = 1$ に

移動します。

7. $P(3,1)=1$ を改めて開始変数とします。ここでは目標値を $5-a[2]=5-2=3$ と設定します。
8. (`leftsearch` により) 開始変数から左に移動して決定ペア $P(3,0)=0$, $P(3,1)=1$ を見つけます。この場合は、目標値は $3=a[1]$ ですので、$c[1]\leftarrow 1$ と特性ベクトルの更新をします。$P(w,0)=0$ なる変数に到着しましたので、解探索はここで終了します。
9. 決定ペア $P(3,0)=0$, $P(3,1)=1$ について特性ベクトルの更新をする理由を説明します。ここでは目標値が $a[1]=3$ ですので、$S_1=\{1\}\subseteq I_1$ は $\sum_{i\in S_1} a_i = 3 = (a[1])$ を満たします。すなわち、目標値が $a[1]$ のとき添字 1 を含む解添字集合が存在します。したがって、$c[1]\leftarrow 1$ として、添字 1 を解添字の候補として記憶します。
10. 解探索が終了したとき、$c[3]=1$, $c[2]=1$, $c[1]=1$ なる添字 $3,2,1$ に着目しますと
    - 添字 3 による目標値の更新が $11-a[3]=11-6=5$、
    - 添字 2 による目標値の更新が $5-a[2]=(11-a[3])-a[2]=11-a[3]-a[2]=11-6-2=3$ です。
    - さらに $a[1]=3$ ですので、$3-a[1]=((11-a[3])-a[2])-a[1]=11-a[3]-a[2]-a[1]$ $=11-6-2-3=0$ です。
    - したがって、$11=a[3]+a[2]+a[1]$ が成り立ちます。
    - すなわち、$S_4=\{1,2,3\}\subseteq I_4$ は $\sum_{i\in S_4} a_i = 11$ を満たします。
11. 終了時の添字集合 $S_4=\{1,2,3\}$ を解添字集合 ($\{a[1]=3,\ [2]=2,\ a[3]=6\}$ を解要素集合) として出力します。

1.3.4 決定ペアと解探索

解探索アルゴリズの設計のポイントとなる、関数 `leftsearch`、決定ペアおよび解探索操作の関係を一般的に説明しておきます。

$P(w,m-1)=0$, $P(w,m)=1$ $(0\le w\le b,\ 1\le m\le n)$ を決定ペアとします。決定ペアの状況とそれに応じた解探索の操作について次の 3 通りの場合があります。(1), (2) では解探索の終了につながりますが、(3) では解探索が継続します。

(1) (m=1 かつ $w=0$ (したがって $w\neq a_1$) のとき) $P(0,0)=0, P(0,1)=1$

(2) (m=1 かつ $w=a_1$ のとき) $P(a_1,0)=0, P(a_1,1)=1$

(3) ($2\le m\le n$ かつ $w\ge a_m$ のとき) $P(w,m-1)=0$, $P(w,m)=1$[*2]

それぞれの状況を操作手順に重点をおいて記述します。操作手順の妥当性については後述するアルゴリズムの正当性の中で説明します。

(1) (m=1 かつ $w=0$ (したがって $w\neq a_1$) のとき) 決定ペア: $P(0,0)=0, P(0,1)=1$ (図 139 参照)

$w=0$ の行では $P(0,k)=1$ $(1\le k\le n$ なるすべての整数 k について) であり、$\sum_{i\in S_k} a_i = w=0$ を満たす添字集合 $S_k\subseteq I_k$ が存在しますが、$S_k=\emptyset$ です。この場合は添字集合に関する操作は何もせず、ここで解探索の終了に進みます。

(2) (m=1 かつ $w=a_1$ のとき) 決定ペア: $P(a_1,0)=0, P(a_1,1)=1$ (図 139 参照)

[*2] $2\le m\le n$ のときに $a_m > w\ (\ge 0)$ と仮定すると、漸化式の定義から $P(w-a_m,m-1)=0$ となって $P(w,m)=P(w,m-1)\vee P(w-a_m,m-1)=1$ に矛盾します。

$P(a_1,1)=1$ であり、$\sum_{i\in S_1} a_i = a_1$ を満たす添字集合 $S_1 \subseteq I_1$ が存在します。このとき $S_1=\{1\}$ です。すなわち、目標値が a_1 のとき添字 1 を含む解添字集合が存在しますので、特性ベクトルを $c[1] \leftarrow 1$ と設定して添字 1 が解添字の候補であることを記憶し、ここで解探索の終了に進みます。

$w \diagdown m$	0	1	$\cdots$
0	$P(0,0)=0$	$P(0,1)=1$	($w=0$ のとき)
1	0	0	
$\vdots$	$\vdots$	$\vdots$	
	0	0	
a_1	$P(a_1,0)=0$	$P(a_1,1)=1$	($w=a_1$ のとき)
	0	0	
$\vdots$	$\vdots$	$\vdots$	
b	0	0	

図 139　$m=1$ の場合の決定ペア $P(w,0)=0$, $P(w,1)=1$ ($w=0$, $w=a_1$ の 2 通りの場合を合わせて記載しています)

(3) ($2 \le m \le n$ かつ $w \ge a_m$ のとき) 決定ペア: $P(w,m-1)=0$, $P(w,m)=1$

このとき、必ず $P(w-a_m,m-1)=1$ です。2 つの場合があります。

(i) $w=a_m$ のとき (図 140 参照)

決定ペアは $P(a_m,m-1)=0$, $P(a_m,m)=1$ であり、かつ $P(w-a_m,m-1)=P(0,m-1)=1$ です。$P(0,m-1)=1$ と同じ行の左方向に決定ペア $P(0,0)=0$, $P(0,1)=1$ が存在します。

添字集合でみると、

- $\sum_{i\in S_{m-1}} a_i = a_m$ を満たす添字集合 $S_{m-1} \subseteq I_{m-1}$ は存在しませんが、
- $\sum_{i\in S_m} a_i = a_m$ を満たす添字集合 $S_m \subseteq I_m$ が存在します。このとき $S_m=\{m\}$ です。
- すなわち、目標値が a_m のときは添字 m を含む解添字集合が存在しますので、特性ベクトルを $c[m] \leftarrow 1$ として添字 m が解添字の候補であることを記憶し、$P(0,m-1)=1$ を開始変数として **leftsearch** を実行します。
- その結果、決定ペア $P(0,0)=0$, $P(0,1)=1$ が見つかりそこで解探索の終了に進みます。

$w \diagdown m$	0	1	$\cdots$	$m-1$	m
0	$P(0,0)=0$	$P(0,1)=1$	$\leftarrow$	$P(0,m-1)=1$	$P(0,m)=1$
$\vdots$				$\uparrow$	
a_m				$P(a_m,m-1)=0$	$P(a_m,m)=1$

図 140　$w=a_m$ かつ $2 \le m \le n$ の場合の決定ペアは $P(a_m,m-1)=0$, $P(a_m,m)=1$ であり、このとき $P(w-a_m,m-1)=P(0,m-1)=1$ です。$P(0,m-1)=1$ と同じ行の左方向に決定ペア $P(0,0)=0$, $P(0,1)=1$ が存在します

(ii) $w > a_m$ のとき (図 141 参照)

決定ペアは $P(w,m-1)=0\,P(w,m)=1$ であり、かつ $P(w-a_m,m-1)=1$ です。$P(w-a_m,m-1)=1$ と同じ行の左方向に決定ペア $P(w-a_m,k-1)=0,\ P(w-a_m,k)=1\,(1\le k\le m-1)$ が存在します。

添字集合でみると、

- $\sum_{i\in S_{m-1}} a_i = w$ を満たす添字集合 $S_{m-1}\subseteq I_{m-1}$ は存在しませんが、
- $\sum_{i\in S_m} a_i = w$ を満たす添字集合 $S_m\subseteq I_m$ が存在します。
- さらに、$\sum_{i\in R_{m-1}} a_i = w-a_m$ を満たす添字集合 $R_{m-1}\subseteq I_{m-1}$ が存在し、$R_{m-1}\cup\{m\}\subseteq I_m$ が上記の S_m の条件を満たします。
- すなわち、目標値が w のとき添字 m を含む解添字集合が存在しますので、特性ベクトルを $c[m]\leftarrow 1$ として添字 m が解添字の候補であることを記憶し、$P(w-a_m,m-1)=1$ を開始変数として **leftsearch** を実行し、解探索を継続します。

$w\diagdown m$	$k-1$	k	$\cdots$	$m-1$	m
0					
$\vdots$					
$w-a_m$	$P(w-a_m,k-1)=0$	$P(w-a_m,k)=1$	$\leftarrow$	$P(w-a_m,m-1)=1$	$P(w-a_m,m)=1$
$\vdots$				$\uparrow$	
w				$P(w,m-1)=0$	$P(w,m)=1$

図 141　$w>a_m$ かつ $2\le m\le n$ の場合の決定ペアは $P(w,m-1)=0,\ P(w,m)=1$ であり、このとき $P(w-a_m,m-1)=1$ です。$P(w-a_m,m-1)=1$ と同じ行の左方向に決定ペア $P(w-a_m,k-1)=0,\ P(w-a_m,k)=1\,(1\le k\le m-1)$ が存在します

前述したプログラム `subsetsum-8.c` は部分和問題の解の存在判定をします。ただ、$P(b,n)=1$ は $\sum_{i\in S_n} a_i = w$ を満たす解集合 $S_n\subseteq I_n$ が存在することは教えてくれますが、具体的な解の構成要素は与えてくれません。一方で、$0\le w\le b,\ 1\le m\le n$ なるすべての整数 $w,\ m$ について `subsetsum-8.c` によって $P(w,m)$ の値は既に求められています。また、$m=0$ の場合に $P(w,0)=0$ と定義して、$P(w,m)$ の定義を拡張しています。

以下では、これらの $P(w,m)$ の値、上述の関数 `leftsearch`、決定ペアと開始変数に関する (1)〜(3) の操作に基づいて部分和問題の具体的な解を求めるためのアルゴリズムを設計します。

1.4　解探索アルゴリズム subsetsum-sol

まず、部分和問題の解を具体的に求める解探索アルゴリズム `subsetsum-sol` を示します。次に、アルゴリズムの正当性を仮定して C 言語プログラムの作成について説明します。その後でいつかの準備をしてアルゴリズムの正当性を示します。正当性の証明は数学的色彩の濃い内容になりますので、後方に配置しました。数学的内容は苦手な方あるいは正当性はよく理解している方などはこの部分をスキップして差し支えありません。

アルゴリズムは以下の記述になります。前述の例題で説明した動きがほぼそのまま入っていますので、動作はよくわかると思います。関数 `leftsearch` も記載しておきます。

関数 **leftsearch(P, w, m)**　/* $P(w,m)=1$ ($w \geq 0$, $m \geq 1$) */

Step1.　[$(P(w,m)=1)$] が成り立つ間 ($m \geq 1$ である)、以下の操作を反復する：

$m \leftarrow m-1$;

Step2.　反復操作の終了後、/* $P(w,m)=0,\ P(w,m+1)=1$ */

$k \leftarrow m+1$;

と設定して、k を戻り値として呼び出し側に返す。

/* 決定ペア $P(w,k-1)=0$, $P(w,k)=1$ */

解探索アルゴリズム subsetsum-sol(P, c, w, m)　// $P(b,n)=1$

Step1.　$w \leftarrow b$;　// $b \geq 1$;

$m \leftarrow n$;　// $n \geq 1$;

Step2.　[$(w \geq 0)$ かつ $(m \geq 1)$] が成り立つ間、以下の (a), (b) を反復する：

(a)　$k \leftarrow$ **leftsearch(P, w, m)**;

// 決定ペア $P(w,k-1)=0$, $P(w,k)=1$ $(k \geq 1)$ が求められる

(b)　$k \geq 2$ ならば **(i)〜(iii)** を実行する :

(i)　$c[k] \leftarrow 1$;　// 添字 k を特性ベクトル c に格納する

(ii)　$w \leftarrow w-a[k]$;　// $w-a[k] \geq 0$ (注意 13.4 参照)

(iii)　$m \leftarrow k-1$;　// 次の開始変数 $P(w,m)=1 \leftarrow P(w-a[k],k-1)=1$

そうでない (つまり $k=1$) ならば **(iv)** および **(v)** を実行する :

(iv)　$w=a[k]$ ならば $c[k] \leftarrow 1$;　// 添字 k を特性ベクトル c に格納する

(v)　$m \leftarrow k-1$;　// 更新後は $m=0$ となって **Step2** は終了となる

Step3.　終了する;

注意 13.4 (既に 1.2.4 決定ペアと解探索の (3) で言及したことですが) $k \geq 2$ のとき $P(w,k)=P(w,k-1) \vee P(w-a[k],k-1)=1$ ですので、決定ペア $P(w,k-1)=0$, $P(w,k)=1$ については $P(w-a[k],k-1)=1$ です。したがって $w-a[k] \geq 0$ です。なお、$a[k]$ は a_k の別表現です。

アルゴリズムの正当性は後述の **1.6 解探索アルゴリズム subsetsum-sol の正当性** で示します。以下ではこの正当性を仮定して、`subsetsum-sol` を C 言語による解探索プログラム `subsetsum-sol-mod.c` として記述します。さらにこのプログラムを例題 12.1 を含む 3 つの例題に対して実行してその実行結果を提示します。

1.5　C 言語プログラムの作成

まず関数 `leftsearch` を説明し、そのあとで `subsetsum-sol-mod.c` 作成上のポイントを説明します。

1.5.1　関数 `leftsearch` について

関数 `leftsearch` は、整数型 2 次元配列 `SP[][n+1]`、整数型変数 `sw` および `sm` を仮引数とします。ここでは繰り返しの動作を `while` 文で記述しています。動作としては、

- SP[sw][sm] == 1 が成り立つ間、sm の値を 1 ずつ減少させます。(漸化式の定義から、この間は sm >= 1 が成り立ちます。)
- この減少操作の終了後に sm+1 の値を戻り値として呼び出し側に返します。
- 減少操作が終了したときは SP[sw][sm] == 0、SP[sw][sm+1] == 1 が成り立ちます。
- main の中では、leftsearch は実引数 P[][n+1]、w、s を

  SP ← P;

  sw ← w;

  sm ← m;

  と仮引数に受け取って P、w、m について動作します。
- 減少操作が終了したときは P[w][m] == 0 かつ P[w][m+1] == 1 で呼び出し側への戻り値は m+1 ですので、main 側で

  k ← leftsearch(P, w, m)

  として受け取ったとすると k の値は m+1 です。
- したがって

  P[w][k-1]=0, P[w][k]=1

  が決定ペアとして選ばれたことになります。
- なお、データの受け取り確認のため、動作開始前に実引数を表示させています。

以下の図 142 が C 言語での記述例です。行番号は subsetsum-sol-mod.c に合わせています。

```
6   int leftsearch(int SP[][n+1], int sw, int sm)
7   {
8       printf("P[%d][%d]=%d\n", sw, sm, SP[sw][sm]);  //受け取ったデータの表示
9       while (SP[sw][sm] == 1)  //  sm >= 1 holds
10          sm--;
11      return sm+1;  // main 側で k <- sm+1
12  }
```

図 142　関数 leftsearch の記述

1.5.2　解探索プログラム subsetsum-sol-mod.c について

プログラムとしては、初めに解の存在判定のために subsetsum-8.c を実行して出力が YES の場合に解探索を実行する形式にします。(前述の解探索アルゴリズムの記述にはこの部分は入っていません。) 具体的には、$P(b,n) = 1$ のときに解探索を実行する if 文の形式にしています。

まずプログラム subsetsum-sol-mod.c を図 143 と図 144 に示します。ここでは、記述を短くするために”{”や”}” なども含めていろいろと省略した形で示していることをお断りしておきます。なお、行番号は説明用に入れています。プログラムに記述するとエラーになります。

次に、実行結果を図 145〜図 147 に示します。図 145 は例題 12.1 ($b = 7$)、図 146 は例題 12.1 で $b = 11$ と変更した場合、図 146 は例題 12.2 それぞれの実行結果です。例題 12.2 の場合は表示すべき $P(w,m)$ の個数が非常に多いのでその表示は省略し、プログラム名を subsetsum-sol-mod-10.c と変更しています。

以下がプログラム作成に関する説明です。

- 図 143 の 1 行〜4 行および図 143 から図 144 にまたがる 14 行〜53 行の部分については宣言と 45 行以外は図 134 の `subsetsum-8.c` と同じですので説明は省略します。6 行〜12 行は関数 `leftsearch` です。
- 宣言の違いは 17 行に特性ベクトル格納用に整列型 1 次元配列 `c[n+1]`、19 行に総和の計算用に整数型変数 `s` を追加したことの 2 つだけです。これらも特に説明の必要はないと思います。
- 45 行では、$P(w,m)$ の定義を $m = 0$ の場合に拡張したことに対応して、2 重ループの内側で (`subsetsum-8.c` では 1 から n まで繰り返しであった) m を 0 から n まで繰り返すように変更して拡張部分の $P(0,0) = 0 \sim P(b,0) = 0$ を表示させています。
- 図 144 の 55 行〜87 行に関して説明します。この部分が 1.4 で示した解探索アルゴリズム **subsetsum-sol** の記述に対応する C 言語プログラムです。
- プログラムの 55 行〜72 行については、60 行〜62 行以外の部分は 1.4 でのアルゴリズム記述をそのまま C 言語で書いた形ですので理解は容易かと思います。73 行〜84 行はデータの表示です (表示の形式については、図 145〜図 147 を参照してください)。以下に少し補足しておきます。
- 56 行〜58 行で $P(b,n) = 1$ を最初の開始変数と設定します。
- 59 行〜72 行は、`leftsearch` による決定ペアの選択のあとで、目標値と特性ベクトルの更新、次の開始変数の設定などの一連の操作を if 文で $k \geq 2$ と $k = 1$ の場合で分岐実行させることを while 文によって反復させる形式で記述しています。
- 60 行〜62 行で関数 `leftsearch` の動作をチェックするためのデータを表示させています。60 行で呼び出し前の実引数を "`leftsearch(P,w,m)`" の形で、さらに 62 行で呼び出し後の戻り値を "`k=(`**戻り値**`)`" の形でそれぞれ表示させています。この 2 つのデータの間に記述する形で関数 `leftsearch` が受け取った実引数を "`P[m][w]=1`" の形で表示させます。この表示は関数 `leftsearch` の 8 行目の printf 文で実行されます。これで呼び出し側と受け取り側のデータ、および戻り値が確認できます。
- 73 行〜76 行は解添字集合を特性ベクトルで "`c[i]=0`" または "`c[i]=1`" の形で横並びに表示させています。
- 77 行〜79 行は解のチェックを兼ねて解要素の総和を求めています。
- 80 行〜84 行は解要素の総和を "`b=(`**総和の値**`)`" の形で表示して改行し、解要素集合を "`a[i]=(`**要素の値**`)`" の形で横並びに表示させています。

```
1   // Solving the SUBSET_SUM problem
2   #include <stdio.h>
3   #define n 4 // #Elements
4   #define b 7 // Target value
5
6   int leftsearch(int SP[][n+1], int sw, int sm)
7   {
8       printf("P[%d][%d]=%d\n", sw, sm, SP[w][m]);
9       while ((SP[sw][sm] == 1) && (sm >= 1))
10          sm--;
11      return sm+1;  // main 側で k <- sm+1
12  }
13
14  int main(void)
15  {
16      int a[n+1] = { 0,3,2,6,5 }; // a[0] は未使用
17      int c[n+1]; // c[0] は未使用
18      int P[b+1][n+1];
19      int i, w, k, m, s;
20      // 初期設定
21      for (w = 0; w <= b; w++) {
22          if ( (w==0) || (w == a[1]) )
23              P[w][1] = 1;
24          else
25              P[w][1] = 0;  // 0:存在しない  1:存在する
26      }
27      for (w = 0; w <= b; w++)
28          P[w][0] = 0;
29      for (m = 0; m <= n; m++)
30          c[m] = 0;
31      // 漸化式の計算
32      for (m = 2; m <= n; m++) {
33          for (w = 0; w <= b; w++) {
34              if ((P[w][m - 1] == 1) || ((w - a[m] >= 0) && (P[w - a[m]][m - 1] == 1)))
35                  P[w][m] = 1;
36              else
37                  P[w][m] = 0;
38          }
39      }
```

図 143　部分和問題を解く C 言語プログラム (解探索操作を含む) subsetsum-sol-mod.c (その 1) (注：行数を減少させるため { } を省略した箇所が多数あります)

```
40      //  入力データと計算結果の表示
41      printf("\n****** The SUBSET_SUM problem *****\n#Elements n=%d,
Target value b=%d\n", n, b);
42      for (w = 1; w <= n; w++) printf("a[%d]=%d  ", w, a[w]);
43      printf("\n\n");
44      for (w = 0; w <= b; w++) {
45          for (m = 0; m <= n; m++) printf("P(%d,%d)=%d  ", w, m, P[w][m]);
46          printf("\n");
47      }
48      printf("\nThe answer is ");
49      if (P[b][n] == 1)
50          printf("YES\n");
51      else
52          printf("NO\n");
53      printf("\n");
54
55      // 出力が "YES" のときは解探索に進みます。
56      if (P[b][n] == 1) {
57          w = b;
58          m = n;
59          while ( (w >=0) && (m >=1) ){
60              printf("***calling leftsearch(P,%d,%d)\n", w, m);
61              k = leftsearch(P, w, m);   //P(w,k-1)==0 and P(w,k)==1
62              printf("check at main:the returned value  k=%d\n", k);
63              if (k>=2) {
64                  c[k]=1;
65                  w = w - a[k];
66                  m = k - 1;
67              }
68              else {  //k==1
69                  if (w==a[k]) c[k]=1;
70                  m=k-1;  //m==0
71              }
72          }
73          // 解となる添字集合の表示（特性ベクトルとして）
74          printf("\nA solution is given by a characteristic vector c[1..%d].\n", n);
75          for (m = 1; m <= n; m++) printf("c[%d]=%d  ", m, c[m]);
76          printf("\n");
77          // 解集合の要素の和を計算（b になるか否かの検算を兼ねて）
78          s=0;
79          for (m = 1; m <= n; m++) s+=c[m]*a[m];
80          // 解となる要素集合の表示
81          printf("The sum of the following elements is equal to b=%d\n", s);
82          for (m = 1; m <= n; m++)
83              if (c[m] == 1) printf("a[%d]=%d  ", m, a[m]);
84          printf("\n");
85      }
86      return 0;
87  }
```

図 144　部分和問題を解く C 言語プログラム (解探索操作を含む) subsetsum-sol-mod.c (その 2) (注：行数を減少させるため { } を省略した箇所が多数あります)

```
****** The SUBSET_SUM problem *****
#Elements n=4,  Target value b=7
a[1]=3  a[2]=2  a[3]=6  a[4]=5

P(0,0)=0  P(0,1)=1  P(0,2)=1  P(0,3)=1  P(0,4)=1
P(1,0)=0  P(1,1)=0  P(1,2)=0  P(1,3)=0  P(1,4)=0
P(2,0)=0  P(2,1)=0  P(2,2)=1  P(2,3)=1  P(2,4)=1
P(3,0)=0  P(3,1)=1  P(3,2)=1  P(3,3)=1  P(3,4)=1
P(4,0)=0  P(4,1)=0  P(4,2)=0  P(4,3)=0  P(4,4)=0
P(5,0)=0  P(5,1)=0  P(5,2)=1  P(5,3)=1  P(5,4)=1
P(6,0)=0  P(6,1)=0  P(6,2)=0  P(6,3)=1  P(6,4)=1
P(7,0)=0  P(7,1)=0  P(7,2)=0  P(7,3)=0  P(7,4)=1

The answer is YES

***calling leftsearch(P,7,4)
P[7][4]=1
check at main:the returned value  k=4
***calling leftsearch(P,2,3)
P[2][3]=1
check at main:the returned value  k=2
***calling leftsearch(P,0,1)
P[0][1]=1
check at main:the returned value  k=1

A solution is given by a characteristic vector c[1..4].
c[1]=0  c[2]=1  c[3]=0  c[4]=1
The sum of the following elements is equal to b=7
a[2]=2  a[4]=5
```

図 145　例題 12.1 (b=7) に対する subsetsum-sol-mod.c の実行結果

```
****** The SUBSET_SUM problem *****
#Elements n=4,  Target value b=11
a[1]=3  a[2]=2  a[3]=6  a[4]=5

P(0,0)=0  P(0,1)=1  P(0,2)=1  P(0,3)=1  P(0,4)=1
P(1,0)=0  P(1,1)=0  P(1,2)=0  P(1,3)=0  P(1,4)=0
P(2,0)=0  P(2,1)=0  P(2,2)=1  P(2,3)=1  P(2,4)=1
P(3,0)=0  P(3,1)=1  P(3,2)=1  P(3,3)=1  P(3,4)=1
P(4,0)=0  P(4,1)=0  P(4,2)=0  P(4,3)=0  P(4,4)=0
P(5,0)=0  P(5,1)=0  P(5,2)=1  P(5,3)=1  P(5,4)=1
P(6,0)=0  P(6,1)=0  P(6,2)=0  P(6,3)=1  P(6,4)=1
P(7,0)=0  P(7,1)=0  P(7,2)=0  P(7,3)=0  P(7,4)=1
P(8,0)=0  P(8,1)=0  P(8,2)=0  P(8,3)=1  P(8,4)=1
P(9,0)=0  P(9,1)=0  P(9,2)=0  P(9,3)=1  P(9,4)=1
P(10,0)=0  P(10,1)=0  P(10,2)=0  P(10,3)=0  P(10,4)=1
P(11,0)=0  P(11,1)=0  P(11,2)=0  P(11,3)=1  P(11,4)=1

The answer is YES

***calling leftsearch(P,11,4)
P[11][4]=1
check at main:the returned value  k=3
***calling leftsearch(P,5,2)
P[5][2]=1
check at main:the returned value  k=2
***calling leftsearch(P,3,1)
P[3][1]=1
check at main:the returned value  k=1

A solution is given by a characteristic vector c[1..4].
c[1]=1  c[2]=1  c[3]=1  c[4]=0
The sum of the following elements is equal to b=11
a[1]=3  a[2]=2  a[3]=6
```

図 146　例題 12.1 の目標値を $b = 11$ に変更した場合の subsetsum-sol-mod.c の実行結果

```
****** The SUBSET_SUM problem *****
#Elements n=10,  Target value b=3754
a[1]=1  a[2]=4  a[3]=16  a[4]=64  a[5]=256  a[6]=1040  a[7]=1041  a[8]=1093
a[9]=1284  a[10]=1344

The answer is YES

***calling leftsearch(P,3754,10)
P[3754][10]=1
check at main:the returned value  k=9
***calling leftsearch(P,2470,8)
P[2470][8]=1
check at main:the returned value  k=8
***calling leftsearch(P,1377,7)
P[1377][7]=1
check at main:the returned value  k=6
***calling leftsearch(P,337,5)
P[337][5]=1
check at main:the returned value  k=5
***calling leftsearch(P,81,4)
P[81][4]=1
check at main:the returned value  k=4
***calling leftsearch(P,17,3)
P[17][3]=1
check at main:the returned value  k=3
***calling leftsearch(P,1,2)
P[1][2]=1
check at main:the returned value  k=1

A solution is given by a characteristic vector c[1..10].
c[1]=1  c[2]=0  c[3]=1  c[4]=1  c[5]=1  c[6]=1  c[7]=0  c[8]=1  c[9]=1  c[10]=0
The sum of the following elements is equal to b=3754
a[1]=1  a[3]=16  a[4]=64  a[5]=256  a[6]=1040  a[8]=1093  a[9]=1284
```

図 147　例題 12.2 に対する subsetsum-sol-mod-10.c の実行結果 (P[w][m] の表示は省略しています)

1.6 解探索アルゴリズム subsetsum-sol の正当性

関数 **leftsearch** とアルゴリズム **subsetsum-sol** はかなり前に記述していますので、以下にこれらを再掲しておきます。

関数 **leftsearch(P, w, m)**　/* $P(w,m)=1\,(\,w\geq 0,\,m\geq 1)$ */

Step1.　$[(P(w,m)=1)]$ が成り立つ間 ($m\geq 1$ である)、以下の操作を反復する：

$m \leftarrow m-1$;

Step2.　反復操作の終了後、/* $P(w,m)=0,\ P(w,m+1)=1$ */

$k \leftarrow m+1$;

と設定して、k を戻り値として呼び出し側に返す。

/* 決定ペア $P(w,k-1)=0,\ P(w,k)=1$ */

解探索アルゴリズム subsetsum-sol(P, c, w, m)　// $P(b,n)=1$

Step1.　$w \leftarrow b$;　// $b\geq 1$;

$m \leftarrow n$;　// $n\geq 1$;

Step2.　[$(w\geq 0)$ かつ $(m\geq 1)$] が成り立つ間、以下の (a), (b) を反復する：

(a)　$k \leftarrow$ **leftsearch(P, w, m)**;

// 決定ペア $P(w,k-1)=0,\ P(w,k)=1\,(k\geq 1)$ が求められる

(b)　$k\geq 2$ ならば **(i)〜(iii)** を実行する :

(i)　$c[k] \leftarrow 1$;　// 添字 k を特性ベクトル c に格納する

(ii)　$w \leftarrow w-a[k]$;　// $w-a[k]\geq 0$

(iii)　$m \leftarrow k-1$;　// 次の開始変数 $P(w,m)=1 \leftarrow P(w-a[k],k-1)=1$

そうでない (つまり $k=1$) ならば **(iv)** および **(v)** を実行する :

(iv)　$w=a[k]$ ならば $c[k] \leftarrow 1$;　// 添字 k を特性ベクトル c に格納する

(v)　$m \leftarrow k-1$;　// 更新後は $m=0$ となって **Step2** は終了となる

Step3.　終了する;

まず、アルゴリズム **subsetsum-sol** の正当性を示すための準備として、アルゴリズムの進行に伴う決定ペア、開始変数、目標値の更新、および解添字候補の更新の状況について詳しく調べます。

アルゴリズムが開始してから終了するまでに **Step2 (a)** の関数 **leftsearch** が $r\,(\geq 1)$ 回実行されたとして、その i 回目の実行での戻り値を k_i とします。すなわち、

$$k_1, \cdots, k_r\ (r\geq 1)$$

とします。ここで、**Step1** で

$$w_1 \leftarrow b\,(\geq 1),\ m_1 \leftarrow n\,(\geq 1)$$

と設定します。添字を付けることで **Step2** の実行回数との対応を明確にします。1 回目の **Step2** は $P(w_1,m_1)$ を開始変数として開始します。**Step2** の 1 回目から r 回目の実行において求めた決定ペア、開始

あるいは継続に向けた開始変数、目標値および特性ベクトルの更新については以下の図 148 に示す状況になります。以下でこれを詳しく調べます。$r = 1$ の場合もありますが、説明のため $r \geq 2$ をイメージして、**Step2** の 1 回目、2 回目、3 回目、$\cdots$、 i 回目、$\cdots$、 $r-1$ 回目、r 回目の実行を順を追って見ていきます。

Step 2 (1 **回目**) $(\, w_1 \geq 1 \,)$ かつ $(\, m_1 \geq 1 \,)$

(a) $k_1 \leftarrow$ **leftsearch(P,** w_1, m_1**)**

// 決定ペア $P(w_1, k_1 - 1) = 0,\ P(w_1, k_1) = 1\ (\, 1 \leq k_1 \leq m_1 = n, \,)$

(b) $k_1 \geq 2$ のとき [*3]

(i) $c[k_1] \leftarrow 1$;

(ii) $w_2 \leftarrow w_1 - a[k_1]\,(\geq 0\,)$;

(iii) $m_2 \leftarrow k_1 - 1\,(\geq 1\,)$

Step 2 (2 **回目**) $(\, w_2 \geq 0 \,)$ かつ $(\, m_2 \geq 1 \,)$

(a) $k_2 \leftarrow$ **leftsearch(P,** w_2, m_2**)**

// 決定ペア $P(w_2, k_2 - 1) = 0,\ P(w_2, k_2) = 1\ (\, 1 \leq k_2 \leq m_2 = k_1 - 1)$

(b) $k_2 \geq 2$ のとき

(i) $c[k_2] \leftarrow 1$;

(ii) $w_3 \leftarrow w_2 - a[k_2]\,(\geq 0\,)$;

(iii) $m_3 \leftarrow k_2 - 1\,(\geq 1\,)$

Step 2 (3 **回目**) $(\, w_3 \geq 0 \,)$ かつ $(\, m_3 \geq 1 \,)$

(a) $k_3 \leftarrow$ **leftsearch(P,** w_3, m_3**)**

// 決定ペア $P(w_3, k_3 - 1) = 0,\ P(w_3, k_3) = 1\ (\, 1 \leq k_3 \leq m_3 = k_2 - 1)$

(b) $k_3 \geq 2$ のとき

(i) $c[k_3] \leftarrow 1$;

(ii) $w_4 \leftarrow w_3 - a[k_3]\,(\geq 0\,)$;

(iii) $m_4 \leftarrow k_3 - 1\,(\geq 1\,)$

$\vdots$

Step 2 (i **回目**) $(\, w_i \geq 0 \,)$ かつ $(\, m_i \geq 1 \,)$

(a) $k_i \leftarrow$ **leftsearch(P,** w_i, m_i**)**

// 決定ペア $P(w_i, k_i - 1) = 0,\ P(w_i, k_i) = 1\ (\, 1 \leq k_i \leq m_i = k_{i-1} - 1)$

(b) $k_i \geq 2$ のとき

(i) $c[k_i] \leftarrow 1$;

(ii) $w_{i+1} \leftarrow w_i - a[k_i]\,(\geq 0\,)$;

(iii) $m_{i+1} \leftarrow k_i - 1\,(\geq 1\,)$

$\vdots$

Step 2 ($r-1$ **回目**) $(\, w_{r-1} \geq 0 \,)$ かつ $(\, m_{r-1} \geq 1 \,)$

(a) $k_{r-1} \leftarrow$ **leftsearch(P,** w_{r-1}, m_{r-1}**)**

// 決定ペア $P(w_{r-1}, k_{r-1} - 1) = 0,\ P(w_{r-1}, k_{r-1}) = 1\ (\, 1 \leq k_{r-1} \leq m_{r-1} = k_{r-2} - 1)$

[*3] $k_1 = 1$ とすると (b) (v) において、$m_2 \leftarrow k_1 - 1\,(= 0)$ となり、更新後には $m_2 = 0$ となって **Step2** が終わり、**Step3** に進んでアルゴリズムが終了します。したがって $r \geq 2$ ならば r 回目の **Step2** が実行されないことになり矛盾です。$r = 1$ のときのみ $k_1 = 1$ が成り立ちます。以下、$r \geq 2$ のイメージで実行状況を示すため $k_1 \geq 2$ と記述しています。

(b) $k_{r-1} \geq 2$ のとき

(i) $c[k_{r-1}] \leftarrow 1$;

(ii) $w_r \leftarrow w_{r-1} - a[k_{r-1}]\,(\geq 0)$;

(iii) $m_r \leftarrow k_{r-1} - 1\,(\geq 1)$

Step 2 (r **回目**) $(\,w_r \geq 0\,)$ かつ $(\,m_r \geq 1\,)$

(a) $k_r \leftarrow$ **leftsearch(P,** w_r, m_r**)**

// 決定ペア $P(w_r, k_r - 1) = 0,\ P(w_r, k_r) = 1\ (1 \leq k_r \leq m_r = k_{r-1} - 1)$

(b) $k_r = 1$ のとき [*4]

(iv) $w_r = a[k_r]$ ならば $c[k_r] \leftarrow 1$;

(v) $m_{r+1} \leftarrow k_r - 1\,(= 0)$; // 更新後は $m_{r+1} = 0$ となって **Step2** の反復条件が満たされなくなり、**Step3** に進んでアルゴリズムは終了する

ここで、r 回目の **Step2 (a)** における決定ペアに着目しますと、

$$P(w_r, k_r - 1) = 0,\ P(w_r, k_r) = 1\ (1 \leq k_r \leq m_r = k_{r-1} - 1)$$

ですが、$k_r = 1$ ですので、

$$P(w_r, 0) = 0,\ P(w_r, 1) = 1$$

です。したがって図 136 で示したように

$$w_r = 0 \ \text{または}\ w_r = a[1]\,(= a_1\,)$$

です。すなわち、アルゴリズムが終了するときの決定ペアは

$$P(0,0) = 0,\ P(0,1) = 1 \quad \text{または} \quad P(a[1], 0) = 0,\ P(a[1], 1) = 1$$

の 2 通りとなります。

以上の説明から次の命題が成り立ちます。

命題 13.2 アルゴリズム **subsetsum-sol** の実行について次の (1), (2) が成り立つ。

(1) $1 \leq i \leq r - 1\,(r \geq 1)$ とするとき、i 回目の **Step2** の実行状況、求められた決定ペア、実行あるいは継続に向けた開始変数、目標値および特性ベクトルの更新は図 148 に示す通りである。

(2) r 回目の **Step2** の実行により求められた決定ペアは図 148 に記載の通りであり、かつ $k_r = 1$ である。さらに $w_r = 0$ または $w_r = a[1]$ であり、次の (i) または (ii) が成り立つ。

(i) $w_r = 0$ のときは特性ベクトル c の更新はせず、初期値設定の $c[1] = 0$ のままアルゴリズムが終了する。

(ii) $w_r = a[1]$ のときは特性ベクトルを $c[1] \leftarrow 1$ と更新してアルゴリズムが終了する。

[*4] $k_r \geq 2$ とすると (b) (i) ～ (iii) において $w_{r+1} \leftarrow w_r - a[k_r]\,(\geq 0)$, $m_{r+1} \leftarrow k_r - 1\,(\geq 1)$ となり、$(r+1)$ 回目の **Step2** が実行されることになって矛盾です。

	求めた決定ペア (戻り値 k_i)	実行あるいは継続に向けた開始変数 目標値, 特性ベクトルの更新
開始前		$P(w_1, m_1) = 1\,(\,m_1 = n\,)$
		$w_1(= b)$, なし
1 回目	$P(w_1, k_1 - 1) = 0,\ P(w_1, k_1) = 1$	$P(w_1 - a[k_1], k_1 - 1) = 1\,(2 \le k_1 \le m_1 = n)$
		$w_2 \leftarrow w_1 - a[k_1]\,(\ge 0), \quad c[k_1] \leftarrow 1$
2 回目	$P(w_2, k_2 - 1) = 0,\ P(w_2, k_2) = 1$	$P(w_2 - a[k_2], k_2 - 1) = 1\,(2 \le k_2 \le k_1 - 1)$
		$w_3 \leftarrow w_2 - a[k_2]\,(\ge 0), \quad c[k_2] \leftarrow 1$
$\vdots$		$\vdots$
i 回目	$P(w_i, k_i - 1) = 0,\ P(w_i, k_i) = 1$	$P(w_i - a[k_i], k_i - 1) = 1\,(2 \le k_i \le k_{i-1} - 1)$
		$w_{i+1} \leftarrow w_i - a[k_i]\,(\ge 0), \quad c[k_i] \leftarrow 1$
$\vdots$		$\vdots$
$r-1$ 回目	$P(w_{r-1}, k_{r-1} - 1) = 0,$	$P(w_{r-1} - a[k_{r-1}], k_{r-1} - 1) = 1\,(2 \le k_{r-1} \le k_{r-2} - 1)$
	$P(w_{r-1}, k_{r-1}) = 1$	$w_r \leftarrow w_{r-1} - a[k_{r-1}]\,(\ge 0), \quad c[k_{r-1}] \leftarrow 1$
r 回目	$P(w_r, k_r - 1) = 0,\ P(w_r, k_r) = 1$	なし
	$k_r = 1;\ w_r = 0$ または $w_r = a[k_r]$	w_r (更新なし), $w_r = a[k_r]$ ならば $c[k_r] \leftarrow 1$

図 148 **Step2 (a)** の 1 回目から r 回目の実行において求めた決定ペア、実行あるいは継続に向けた開始変数、目標値および特性ベクトルの更新

ここで、アルゴリズム実行中の目標値 $w_1, \cdots, w_r$ と特性ベクトルに格納した $c[k_i] = 1$ なる解添字の候補 $k_1, \cdots, k_{r-1}$ に着目しましょう。以下にこれらを図 149 に列挙してみます。

$$
\begin{array}{lcll}
w_1 & = & b & \\
w_2 & = & w_1 - a[k_1] & = \quad b - a[k_1] \\
w_3 & = & w_2 - a[k_2] & = \quad b - a[k_1] - a[k_2] \\
 & \vdots & & \vdots \\
w_{i+1} & = & w_i - a[k_i] & = \quad b - a[k_1] - a[k_2] - \cdots - a[k_{i-1}] - a[k_i] \\
 & \vdots & & \vdots \\
w_r & = & w_{r-1} - a[k_{r-1}] & = \quad b - a[k_1] - a[k_2] - \cdots - a[k_{r-2}] - a[k_{r-1}]
\end{array}
$$

図 149 アルゴリズム実行中の目標値 $w_1, \cdots, w_r$ と $c[k_i] = 1$ なる解添字の候補 $k_1, \cdots, k_{r-1}$

さらに、添字 $k_r = 1$ に関しては以下の 2 通りがあります。

(i) $w_r = 0$ かつ $c[k_r] = 0$
(ii) $w_r = a[k_r]$ かつ $c[k_r] = 1$

ここで、添字集合 S_n を以下の 2 つの場合に分けて定めます。

(i) $w_r = 0$ かつ $c[k_r] = 0$ のとき

$S_n = \{k_1, \cdots, k_{r-1}\}$

と定めます。このとき

$w_r = 0 = b - a[k_1] - a[k_2] - \cdots - a[k_{r-2}] - a[k_{r-1}]$

ですので

$b = a[k_1] + a[k_2] + \cdots + a[k_{r-2}] + a[k_{r-1}]$

となります。

(ii) $w_r = a[k_r]$ かつ $c[k_r] = 1$ のとき

$S_n = \{k_1, \cdots, k_{r-1}, k_r\}$

と定めます。このとき

$w_r = a[k_r] = b - a[k_1] - a[k_2] - \cdots - a[k_{r-2}] - a[k_{r-1}]$

ですので

$b = a[k_1] + a[k_2] + \cdots + a[k_{r-2}] + a[k_{r-1}] + a[k_r]$

となります。

ここまでの $a[i]$ を a_i なる表記に戻して記述しますといずれの場合にも上記の S_n について

$\sum_{i \in S_n} a_i = b$

が成り立ちます。したがって、解添字集合に関して次の命題が成り立ちます。

命題 13.3 アルゴリズム **subsetsum-sol** の実行により求められた特性ベクトル c について

$$S_n = \{k_i \in I_n \mid c[k_i] = 1,\ 1 \leq i \leq n\}$$

とする。このとき

$$\sum_{i \in S_n} a_i = b \text{ かつ } S_n \subseteq I_n$$

が成り立つ。すなわち、S_n は部分集合問題の解添字集合である。

以上の命題 13.2 と命題 13.3 によってアルゴリズム **subsetsum-sol** の正当性が示されたことになります。