ループ不変図式に基づく文芸的プログラミングのための会話型支援システムに向けて

真野 芳久
南山大学 数理情報学部 情報通信学科
ymano@it.nanzan-u.ac.jp

1 はじめに

意味のあるプログラムは、同一コードを繰り返し実行する部分を含む。この部分は while 文に代表される繰返し文あるいは再帰呼出しの形をとる。本稿では繰返し文に焦点を当てて議論する。本稿で取り上げる話題についての再帰における同様の問題は、稿を改めて述べることにする。

プログラム中の同一地点であるにもかかわらず、繰り返し実行される種々の時点における状況は異なっている。ここで“状況”とは、ある時点を指定した時に、プログラム中で使われている変数とその値との組の集合を指すものとする。こうして、プログラムの実行とは、与えられた初期状況から期待する最終状況へ変換する動作であり、その制御を行なうのがプログラムであると捉えることができる。

既に存在する完成したプログラムに対してその動作を理解しようとすると、時間とともに変化する様々な状況を想定することになる。プログラムの地点を固定しても同様であり、個々の地点に対して多数の状況の集合体が対応する。限られた記憶能力しか持たない人にとってその把握は困難である。この問題に対しては、時間とともに変化する多数の状況を集合体としてではなく、プログラムの各地点に対応して抽象化した 1 つの状況を考え、具体的な状況の集合体はその具体例の集合であると捉えることで改善できる。この抽象化された状況はその地点における表明 (assertion) と呼ばれ、繰返しの中にある場合にはそれを強調してループ不変式 (loop invariant, ループ不変条件とも) と呼ばれる。表明、特にループ不変式は、既に存在するプログラムの部分正当性の証明のための重要な概念であり、これに基づく研究は多数ある。

本稿ではプログラム作成、すなわちプログラミングを問題とする。創造的活動であるプログラミング、特に正しいプログラムの作成は、既存プログラムの理解や証明以上に困難な作業である。ここでは、プログラミングの困難さを代表する繰返し部分の作成に焦点を当てて、プログラミングの困難に立ち向かう 1 つのアプローチとその支援システムに向けての考察を行なう。

初期状況から最終状況に向けて変数の値が時間とともに徐々に変化していく多数の状況の集合体を想定してプログラムを作ることは、人の能力を越えている。ここでもやはり抽象化した 1 つの状況を考え、時間とともに生じる多数の状況はその具体例であると捉えることで困難なプログラミングに一筋の光明を見ることができるようになる。

例えば Dijkstra[1] は、目的とする状況からそれを作り出すための最弱事前条件への変換系 wp について考察し、非決定性を持つ言語を使って“証明しつつのプログラミング”を提唱した。そこにおいても、繰返し部分のプログラム作成ではループ不変式が決定的な役割を果たしている。

一方、人の自然な思考を自然に反映させることでスムーズなプログラム作成を容易にし、理解性が高く“読む”ことのできるプログラムを得られるプログラミング方法として、文芸的プログラミング (literate programming) が提唱されている [4][5]。そこでは、まとまりのある処理単位を section として自由に導入・参照しつつ、プログラムコードと説明文書を同時に作成する。こうして、開発時の検討内容が自然な流れの中で説明文書として記述され、それらがコードと直結していることで、理解性・信頼性・保守性の高いプログラムを得やすい方法と言える。

本稿では、ループ不変式を用いることが現実的なプログラミングの場においても有効な方法であり、それを非形式的ではあるが十分正確に記述したものは文芸的プログラミングにおける説明文書としてふさわしいことを示す。2 章では、ループ不変式を用いたプログラミング方法について述べる。3 章では、文芸的プログラミングにおける説明文書の一部として、理解しやすい表現形式のループ不変式が効果的であることを、プログラミング例を使って示す。4 章では、ここで示したプログラミング法を支援するシステムへの展望を述べる。

2 ループ不変式利用のプログラミング

2.1 ループ不変式とプログラミング方法

繰返し構造として、図1の流れ図と同一の実行を行なう

S_i; **while** B **do** S; S_f

を考える。ここで S_i, S, S_f は代入文を含む文、B は論理式である。

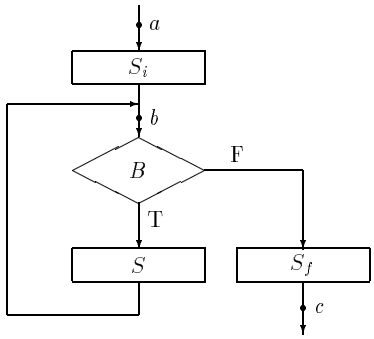

図 1: 繰返し構造の流れ図

図1の地点 b における表明、すなわちこの地点を通過する時点で常に成立する条件がループ不変式である。実行の進行とともに変数群の作る状況は時間とともに変化するが、抽象化された条件であるループ不変式はどの時点においても成立する。

図1の地点 a はこの部分を実行する直前であるが、この地点において変数群が満たすべき条件を事前条件 (precondition) と呼ぶ。また、図1の地点 c はこの部分を実行することによって得たい状況を作っている筈であるが、その条件を事後条件 (postcondition) と呼ぶ。

事前条件を P、事後条件を Q として、ループ不変式利用のプログラミングは図2のように記述できる。

ループ不変式を得るための方法として、Gries[2] は “事後条件中の定数を変数にして一般化する” などのいくつかの方法を検討しているが、一般的な方法はないであろう。従って繰返し構造のプログラムの作成は相変わらず困難な問題であることに変わりはないが、ループ不変式が見つかっていれば、図2に示したように明確な方向付けをもってプログラミングを進めることができる。すなわち、プログラミングという困難な知的創造活動をループ不変式の発見とそれに基づくプログラミングとに分割することで、より系統的に実行できるようになることを意味している。

2.2 ループ不変式を利用したプログラミングの例

前節で述べたプログラミング方法を具体的に示すために、「配列 A があり、添字が 1 から N までの配列要素に整数値が入っている。値が 0 の要素は削除して 0 でない要素を配列前部に詰める」という配列の圧縮問題を例として考える。なお、Pascal 風の言語を用いてプログラムを記述する。

この問題に対する事前条件 P、事後条件 Q を厳密に記述することは意外に難しい。ここでは実現可能性を重視する立場から、非形式的にしかし十分に正確に記述してみる。

- 事前条件 P：配列 A の添字 $1 \sim N$ の部分に整数値が入っている
- 事後条件 Q：配列 A の添字 $1 \sim n$ の部分に 0 でない整数値が入っており、列 A[1] $\sim$ A[n] は元の配列の列から 0 を除いた列に等しい。

図2の “ループ不変式利用のプログラミング方法” に従って、この問題のプログラムを作っていく。箇条書きの番号は図2の番号に対応する。

プログラミング経験の浅い学生の場合、$O(N^2)$ の実行時間を必要とするプログラムを作りやすいようである [3] が、このような手順を踏んでプログラミングを行なうことによって、$O(N)$ の効率の良い正しいプログラムを作りやすくなっていること、また、ループの本体を作る 3. の段階がプログラミングの難関と言えそうであるが、ループ不変式が存在する状況ではさほど困難さはないことが理解できよう。

1. P および Q の両方を含み、P から Q への途中状況を表現する I として、元の配列値の意味で A[1] $\sim$ A[m] を圧縮した結果が、新しい配列値の意味での A[1] $\sim$ A[n] となっている状況を考える。
2. I を成立させる。最も単純な方法として、何も処理していないことを示す $m = 0, n = 0$ の状況とすることが考えられる。こうして、S_i として $m := 0; n := 0$ が得られる。
3. 処理を進めるということは m が大きくなることであり、ほんの僅かでも進めばよいので m を 1 だけ進めることを考える。A[m] は未処理の要素であり、これが 0 か否かで処理が分かれる。A[m] $= 0$ であれば除かれる要素であり、特に何もしなくてもよい。A[m] $\neq 0$ であればこの要素は前の方に詰められる。以上の考察から

1. P および Q から問題解決方法を検討し、ループ不変式 I を考える。
2. 前処理 S_i として、P の状況から I を成立させる実行文を考える。I を成立させることが目的であり、単純な方法で構わない。
3. I を維持したまま目標に向けて前進しているコードを作る。目標に向かって僅かでも前進していればよい。
4. 終了条件を考える。繰返しが終了した時点で成立していてほしい条件が I と終了条件とから導きだせることを確認する。
5. 繰返しが終了した時点で成立している筈の条件から事後条件 Q を達成させるコードを後処理 S_f として作る。

図 2: ループ不変式を利用したプログラミングの方法

$m := m + 1$;
if A[m] $\neq 0$ **then**
begin $n := n + 1$; A[n] := A[m] **end**
が得られる。
この実行後にも、「元の配列値の意味で A[1] 〜 A[m] を圧縮した結果が、新しい配列値の意味での A[1] 〜 A[n] となっている」というループ不変式 I が成立している。

4. 未処理部分がなくなれば処理を終了してよい。これは m が N に一致した場合である。
5. I かつ $m = N$ から「元の配列値の意味で A[1] 〜 A[N] を圧縮した結果が、新しい配列値の意味での A[1] 〜 A[n] となっている」が得られ、これは元の問題の要求にあっており、この例では事後処理は必要ない。

2.3 プログラミング問題の例

前節で “ループ不変式を利用したプログラミング方法” の実践例を示した。本節ではいくつかのプログラミング問題を示す。困難な問題ではないが容易とも言えず、プログラミング経験の浅い学生であれば大いに悩みそうな問題である。その課題を解くプログラム作成に役立つループ不変式の例は次節で示すが、それらはプログラミングに対する大きなヒントになっていることが分かる。

2.3.1 べき乗計算

実数 x と正整数 n が与えられている。大きな n に対しても効率良く x^n を計算するプログラムを作れ。ただし、log 関数は使用できないものとする。

2.3.2 配列中の最も大きな 10 個の値を求める

十分に大きな N と 1 次元の整数値配列 A[1..N] が与えられている。A に含まれる値の中で最も大きな 10 個の値を求めよ。整列した時の大きな方の 10 個を求める問題であるが、整列アルゴリズムを使うのは効率的でない。

2.3.3 オランダ国旗問題

1 次元の配列 A[1..N] があり、その要素値として Red, White, Blue の値がランダムに入っている。これらのすべての値が Red, White, Blue の順に並ぶように並び換えよ。ただし、配列に対して使用できる操作は、配列要素の値を見ることと、2 つの配列位置を指定して両者にある色を交換することだけである。

2.3.4 2 つの昇順配列に対してある条件を満たす対の個数

昇順に整数値が入っている 2 つの配列 A, B と値 min, max が与えられている。条件 $min \leq$ A[ia] + B[ib] $\leq max$ を満たす対 (ia, ib) の個数を求めよ。

2.3.5 線形リストの反転

ポインタ $head$ が指す線形リストがある。ポインタの付け替えによって、この線形リストの並び順を逆にせよ。

2.4 プログラミング問題に対するループ不変式

2.4.1 べき乗計算

$x^n = px^c$ という第1のループ不変式を用いると、

$p := 1;\ c := n;$
while $c > 0$ **do**
  begin $p := p * x;\ c := c - 1$ **end**

が得られる。

$x^n = pq^c$ という第2のループ不変式からも上と同様のプログラムを得ることはできるが、q を変数として導入していることから

$p := 1;\ q := x;\ c := n;$
while $c > 0$ **do**
  if odd(c) **then**
    begin $p := p * q;\ c := c - 1$ **end**
  else
    begin $q := q * q;\ c := c$ **div** 2 **end**

という効率の良いプログラムを得ることができる。

以下ではループ不変式だけを示し、そこから得られるプログラムは記述しない。

2.4.2 配列中の最も大きな 10 個の値を求める

A[1] 〜 A[i] ($i \geq 10$) が既に調べられており、その中の最大の 10 個が A[1] 〜 A[10] に降順に入っている。

2.4.3 オランダ国旗問題

$1 \leq r \leq w \leq b \leq N$ で、A[1] 〜 A[$r-1$] に Red が、A[r] 〜 A[$w-1$] に White が、A[$b+1$] 〜 A[N] に Blue がある。

2.4.4 2 つの昇順配列に対してある条件を満たす対の個数

$1 \leq ia \leq NA$ として、A[ia] に対して条件を満たす B の要素は、添字が $ib1+1$ 〜 $ib2$ のものである。

2.4.5 線形リストの反転

元のリストの先頭から p が指す要素までは既に反転している (p が先頭を指す線形リストになっている)。元のリストで p の指す要素の次を q が指しており、それ以降は元のリストと変わっていない。

3 説明文書としてのループ不変式の利用

ループ不変式は開発の道順を導いてくれる貴重な指針となることを前章までに示した。このことはまた、開発されたプログラムを理解する際にループ不変式は貴重な資料になりうることを示唆している。

D. E. Knuth は、プログラムの開発と説明文書の作成とを同時に行ない、両者が密接に結び付いたプログラムの作成法を文芸的プログラミング (literate programming) として提唱した。そこではプログラミング言語の要求する記述順序の制約を受けず、問題解決のための自然な流れとして、プログラム作成者の思考過程がまとまった役割を持つ section という単位の列で記述される。このように記述された説明文書付きのプログラムは WEB 文書と呼ばれる。

文芸的プログラミング法に基づいてプログラムを作成することによって、プログラムを作る際の考察を十分に行なうこと、それらの考察を読める形で表現することが助長され、結果的に種々の面で優れたプログラムが作られやすくなっていると言える。しかし、説明文書に書くべき内容についてはプログラム作成者にまかされている。Knuth らによる多くの WEB 文書例などから説明文書の良い書き方を学ぶことができるとはいうものの、より具体的な指針があると実践しやすいであろう。

2.3.4 節で述べた “2 つの昇順配列に対してある条件を満たす対の個数” 問題を文芸的プログラミング法によって解いたプログラムの例を付録に載せておく。説明文書の一部としてループ不変式を図式的に記述しており、このような記述方法がプログラム作成やプログラム理解に役立っていることを示している。なお、これは CWEB システムの CWeave プログラムによって、WEB 文書を TEX に変換して可視化したものである。ただし、そこに含まれるべき 3 つの図は、都合によって図3、図 4、図5として最後に載せている。

4 支援システムに向けて

前章までに、ループ不変式がプログラム作成時の良い指針となること、またそれは説明文書の一部として有効であること、従って、文芸的プログラミングに基づくプログラム作成に使用することによってそれらの効果を一層発揮させることができるであろうことを見てきた。我々は文芸的プログラミング法を軸とするプログラミング支援システムの開発を検討しているが、前章の例を参照しつつ、そこで目指している支援システムおよび関連する支援機能の設計・実現に向けての考察を行なう。

4.1 プログラム作成時の文芸的プログラミング支援の必要性

文芸的プログラミングは優れたプログラミング手法と考えられるにもかかわらず、現実のプログラミングの場ではそれほど利用されていない。その大きな理由として、動作するプログラムの開発を主目的と考えると説明文書の同時作成は負担が大きいという面に加えて、プログラム開発時の支援機能が乏しいことが挙げられよう。

前者は良い説明文書を作るために払うべきものとしてやむを得ない労力であるが、後者のシステム支援機能を充実させることでその労力を軽減させることはできる。文芸的プログラミングのための支援システムとしては WEB、CWEB[6] などが知られている。これらのシステムは、WEB 文書から目的言語のプログラムを抽出する Tangle と、WEB 文書から説明文書とプログラムコードを整形された形で抽出する Weave とから構成される。すなわち、完成した、あるいはある程度完成した WEB 文書を対象にした支援機能である。

文芸的プログラミングは優れた手法であり WEB あるいは CWEB システムの支援があるけれど、

- まとまりのある役割を持つ section 単位によって構成されるけれど、section のサイズは小さなものなので、最終的には多数の section が作られることになり、それらを把握しつつプログラム作成を進めることはその進行に伴って困難となる。
- マークアップ言語方式で WEB 文書が記述され、そほ記述に基づいて Weave による整形支援が得られる。しかし Weave は、作成している編集段階での利用を想定したものではなく、それまでの記述内容を再考したり、そこにある情報を整理してその時点で抱えている問題に役立たせることには不向きである。

という問題点を挙げることができる。プログラムを作成している困難な段階に対するシステム支援が強く望まれる。

多くの section 群の構造の作成・理解を会話型で支援するには hypertext の概念を利用するのが自然のように思われる。この観点から Aalborg 大学の K.-Østerbye[8] と K. Nørmark[9] による会話型システムの提案などがあり、LEO のように文芸的プログラミングを支援するエディタも開発されている [7]。

以下ではループ不変式の利用を中心にして、プログラミング時の困難さを軽減することに重点を置く会話型システムの実現に向けての考察を行なう。

4.2 ループ不変式の図式化の必要性

繰返し内のある地点における常に成立する状況を表現したものがループ不変式であった。これを正確に表現するためには、論理式などの数学的に裏付けのある厳密な表現形式を使用することが必要である。このような形式的表現を利用することによって、作成されたプログラムの正当性の証明、作成時における一部コードの自動生成などのシステム支援機能を期待することが可能となる。例えば、論理式を操作する Dijkstra の wp 変換系を用いるプログラム作成 [1] は、ループ不変式と繰返し内の実行文とから繰返し条件を機械的に導出することができ、プログラムが完成した瞬間にその正しさも示されている。

しかし、この効果を得るには、正確かつ詳細に形式的記述をしなければならず、また、人にとっては暗黙の了解事項と思われることも記述することが要求される。小さなプログラムであれば実践は可能であろうが、現実のプログラミングの場に適用するには負担が大きいと言えよう。

一方、前章の例で見たように、理解しやすい表現でループ不変式を記述する方法も考えられる。このような記述では正当性の証明などの形式的な扱いは困難であるが、証明はされないものの正しさをほぼ確信しつつプログラム作成を進めることができる。もし正しさの証明が要求されるとすれば、この表現をもとに形式的記述を作ることはそれほど困難ではないであろう。

こうして、形式的処理を可能とするための一歩を進めずに図式的に表現することで、理解性の高い良い説明文書を与えるとともに、プログラムの信頼性と正しさへの十分な確信も得られる。

4.3 図式記述とそれに基づく支援機能

図式としての記述自体がプログラム開発の指針として、そして説明文書として役立つものであるが、単なる図形としての記述ではシステムによる知的な支援を期待することは難しい。

ループ不変式に基づくシステム支援機能として、作られたプログラムの部分正当性の証明、Dijkstra の wp 変換系を利用しての終了条件の作成といった形式的記述を前提とした支援がまず挙げられる。非形式的な記述の場合にこれらを厳密な意味で期待することは難しいが、いくつかの現実的な支援を期待することはできるであろう。

繰返し本体の先頭での状況がループ不変式であるが、これを利用することで、例えば、

- 繰返し本体の文の実行とともい変化する状況をスナップショット的な表示する。
- 繰返し本体が大きい場合、そお途中の状況の図式の作成を支援する。
- 繰返しの進行方向が与えられているとして、ループ不変式の示す状況から繰返し本体を1回実行した後の状況を作り出してそれらの差を理解しやすい形で表示する。
  “条件を満たす対の個数” 問題の例で言うと、図5のループ不変式の図から ia の進行方向、$ib1$ と $ib2$ の変化方向を与えることで図4に示されるような変化図を作成し、説明文書の一部として利用できるようにする機能である。
- これらの機能を統合し連続的に表示することで、プログラムの動作を動画的に表示する。

などが考えられる。

単なる図としてではなく、これらの機能の実現が可能となるようにしたいと考えている。そのためには、図に含まれる種々の要素の論理的な意味や相互の関係を人が表現して与える記述形式を、その編集方法とともに検討する必要がある。

4.4 他の支援機能について

これまではループ不変式に基づく支援機能を検討してきたが、その土台となっているのは文芸的プログラミングの会話型支援システムであった。ここでは、この支援システムに持たせようと検討しているいくつかの支援機能について述べる。

4.4.1 WYSIWYG 風の会話型支援システム

WYSIWYG 方式の考え方に基づいて、プログラム作成時に hypertext 形式で WEB 文書を表示・編集できる会話型支援システム。

これについては既に実験システムを作成しており、その使用経験に基づいて更に改善し、本研究の土台として発展させていきたい。

4.4.2 section 構造の理解支援

section 群は問題あるいは解決法の構造を反映した構造を持っている筈であるが、WEB 文書では section の列として表現されている。この構造を利用して、例えば、section 構造の木構造表示、木構造レベルでの移動、編集などの表示・編集機能が期待される。

4.4.3 section 単位での理解支援

大きなプログラムは多数の section から構成される。必要な時に関連する section を参照することになるが、section の外からは抽象化されたものとして、すなわちその役割だけが明確な形で容易に知ることができるようになっていることが望まれる。

また、プログラム作成のある時点で section を導入するが、この時点ではその section に期待する役割を強く意識している筈である。しかし文芸的プログラミングではその時点でこの役割を記述する場所がない。

これらの要求・問題点の解決のために、各 section の仕様記述のための WEB 文書の形式の検討とそのためのシステム支援機能を考えていきたい。

4.4.4 説明文書の一部としての入出力例の支援

一般的な記述での理解が困難な場合、人は例を要求する。また、問題解決が困難な場合、具体的な例を用いることで解決の見通しが得られる場合も多い。“条件を満たす対の個数” 問題では、図3の例を考えることによって、以降の問題解決の方法が得やすくなっていると言えよう。

プログラムにおける例、すなわち入力と出力の例は、この意味でプログラムの説明文書の一部として有効なものになりうることが分かる。

section はまとまりの仕事を行なうものであるので、section 単位で入力・出力データの例を記述することも可能であろう。その場合の文書の形式とその利用形態を検討していきたい。section の理解を助ける情報として役立つが、次の部分実行と組み合わせることによって、section 単位で入出力データの例を記述する価値は高まる。

4.4.5 部分実行機能による動作確認

最初から正しくプログラムを記述していくことが理想であり、文芸的プログラミングなどの方法論はその支援を1つの大きな目標とする。しかし、人は間違いを犯すものであり、優れた方法論に基づいていることを理由にその記述に間違いがないとは断言できない。間違いがないことを保証するものではないが、信頼性を高める現実的な手法がテストである。

プログラム全体を作成してからではなく、途中の時点で部分的に確認したいことは起こる。不完全なプログラムコードの実行のためには人との協調作業が必要であるが、まとまりのある section を単位として部分実行することによってその動作を確認でき

るのであれば、より安心感をもってそれ以降の作成を続けることができる。

4.5 支援システムの設計・実現に向けて

これまでに述べた考えに基づいて、南山大学経営学研究科の修士論文研究および経営学部情報管理学科の卒業研究において検討を進め、いくつかの実験システムを作成してきた。今後は更に詳細に検討を進め、文芸的プログラミングのための会話型支援システムを中心とする統合システムにすることを目指すものの、まずは個々の機能を優れたプログラミングツールとして実現していきたい。

5 おわりに

本稿ではまず、ループ不変式に基づくプログラミング方法、およびいくつかの問題例に対するループ不変式の例を示した。系統的にプログラミングを行なえるようになること、作成されるプログラムが目的とする仕様を満たすことに確信をもってプログラミングを進めることができることなどの利点がある。

次に、文芸的プログラミングにおける説明文書として図的に示したループ不変式が有効であることをプログラミング例によって示した。文芸的プログラミングは、説明文書とともにプログラムを作るすぐれたプログラミング方法であるが、説明文書としてふさわしい内容は作者にまかされていた。ここでは繰返しというプログラミング上の困難な部分について一つの記述内容と形式を例によって示した。このようなプログラミングを実践するにはシステムの支援が不可欠であり、そこにおける支援機能とシステム構成についても検討した。また、文芸的プログラミングの支援システムとして望まれる機能についてもいくつか指摘した。今後はここで述べた方法論を更に洗練し、支援システムの具体化・実現へと進めていきたい。

謝辞

本研究は、南山大学経営学研究科および経営学部情報管理学科の真野ゼミの中で行なっている修士論文と卒業研究での種々のテーマの基礎となるものであり、真野ゼミ生との議論の中で目標やアプローチが明確になってきたものである。真野ゼミ生諸君に感謝する。

参考文献

[1] E. W. Dijkstra： A Discipline of Programming, Prentice Hall 1976.
浦, 土居, 原田 (訳)：プログラミング原論, サイエンス社 1983.

[2] David Gries： The Science of Programming, Springer-Verlag 1981.
筧 (訳)：プログラムの科学, 培風館 1991.

[3] Owen Astrachan： Pictures as Invariants, SIGCSE Technical Symposium on Computer Science Education 1991 pp.112–118.

[4] Literate Programming, Computer Journal V.27 pp.97-111 May 1984.

[5] Donald E. Knuth： Literate Programming, Center for the Study of Language and Information (1992).
有澤訳：文芸的プログラミング、アスキー 1994.

[6] Donald E. Knuth, Silvio Levy： The CWEB System of Structured Documentation, Addison-Wesley 1993.
`http:/www.literateprogramming.com/cweb.pdf` に最新版 (Ver3.63, Jan 2001) がある.

[7] Literate Programming Home Page, `http://www.literateprogramming.com/`

[8] Kasper Østerbye： Literate Smalltalk Programming Using Hypertext, IEEE Transrations on Software Engineering, Vol.21, No.2 (1995.2).

[9] The Elucidative Programming Home Page, `http://www.cs.auc.dk/~normark/elucidative-programming/`

1. 2 つの昇順配列に対してある条件を満たす対の個数.

(メイン部の説明とコード：省略)

2. 解こうとしている問題は、“昇順に整数値が入っている 2 つの配列A, Bと値min, maxが与えられている時に、条件 $min \le A[ia] + B[ib] \le max$ を満たす添字の対 (ia, ib) の個数を求める”であり、“条件を満たす対の個数”問題と呼ぶ。

例えば、図 3 のように昇順配列A, Bが与えれていて、 $min = 20$, $max = 25$ とすると、図 3 の両配列要素を結ぶ実線で示す対の総数、すなわち 6 がこの問題の解になる。

(図 3)

3. ここではプログラム全体ではなく、条件を満たす対の個数を関数値とする関数$numPair$として作る。その入力引数として、2 つの昇順配列と要素数をそれぞれA, `NA`, B, `NB`とし、和の範囲を表わす下限と上限をminとmaxとする。こうして、作るべき関数は次のようになる。

⟨ 関数 3 ⟩ ≡
```
int numPair(int A[], int NA, int B[], int NB, int min, int max)
    {
    ⟨ numPair変数宣言 8 ⟩
    ⟨ numPair本体 4 ⟩
    }
```
This code is used in section 1.

4. 関数本体は繰返し構造によって実現できるであろう。

⟨ $numPair$本体 4 ⟩ ≡
```
⟨ 初期設定 7 ⟩
⟨ 繰返し部 10 ⟩
⟨ 後始末 13 ⟩
```
This code is used in section 3.

5. 配列Bが昇順になっていることから、配列Aのある要素$A[ia]$ に対して条件を満たすBの要素の集合は、連続した区間であることが分かる。この区間をBの添字変数$ib1$ と$ib2$ とで表わすことにする。対称形にするため、$ib1$ は $A[ia] + B[b] < min$ を満たす最大のbで、$ib2$ は $A[ia] + B[b] \le max$ を満たす最大のbとする。このように$ib1$ と$ib2$ を定めると、条件を満たす区間は $B[ib1 + 1]$ 〜$B[ib2]$ となる。なお、このような条件を満たすbがない場合の$ib1$ あるいは$ib2$ の値は、今後のプログラムの都合で定めることにする。

配列Aも昇順になっていることから、添字iaを例えば増加の方向にずらしていくと、図 4 が示しているように、条件を満たすBの要素の区間 $[ib1 + 1, ib2]$ は両端の添字が減少の方向にずれていく。

(図 4)

6. これまでの考察から、効率的なプログラムを作る方針が得られる。図 5 に示すように、Aの 1 つの要素$A[ia]$ を決めた時に、条件を満たすBの区間が $B[ib1 + 1]$ 〜$B[ib2]$ となっているように$ib1$ と$ib2$ を決める。$A[ia]$ に対して条件を満たす対の個数は、従って $ib2 - ib1$ である。

図 5 の状況を維持しつつiaを動かし、それに応じて$ib1$ と$ib2$ も現位置からずらしていく。Aのすべての要素について条件を満たす対の個数を求めれば、それらの和が求めるべき数である。iaは単調に増加し、$ib1$ と$ib2$ は単調に減少していくことから、配列のサイズに比例した処理量であることが分かる。

(図 5)

7. この方針でプログラムを作ることにする。初期設定としては、iaをAの先頭とし、$ib1$と$ib2$を図5の示す状況に設定することが考えられる。

```
⟨ 初期設定 7 ⟩ ≡
  ia = 0;
  ⟨ A[0] に対して条件を満たすBの区間を求める 12 ⟩
  sum = ib2 − ib1;
```

This code is used in section 4.

8. これまでにいくつかの変数を導入したので、宣言しておく。

```
⟨ numPair変数宣言 8 ⟩ ≡
  int ia, ib1, ib2;
  int sum;
```

This code is used in section 3.

9. $ib1$と$ib2$の定義は既に述べたが、それぞれ条件を満たすbが存在しない場合は、Bの最小添字0でも条件を満たすものがないという意味でも、また$ib2-ib1$で区間内の要素の個数を求めることができるという意味でも、-1と定めるのが自然である。

10. 繰返し部では、まだ調べていないAの要素があればiaを動かし、新しいiaに対して図5を再び成立させるように$ib1$と$ib2$を変更する。前 section で述べた理由で$ib2$が-1になった場合、調べていないAのどの要素に対しても条件を満たす組はもはや存在しない。

```
⟨ 繰返し部 10 ⟩ ≡
  while (++ia < NA ∧ ib2 ≥ 0) {
    ⟨ 新しいib1 とib2 を求める 11 ⟩
    sum += ib2 − ib1;
  }
```

This code is used in section 4.

11. iaが増加して固定された時に、$ib1$と$ib2$を再設定することになるが、それまでに持っていた値以下になることは既に分かっている。新しい$ib1$の値は元の$ib1$の値以下であるが、さらに新しい$ib2$の値以下でもある。

そこで$ib2$の更新を$ib1$の更新前に行なう。次のコードは、前に言及した‘$ib1$と$ib2$を定めるための条件を満たすbが存在しない’場合には、特別な値-1を確かに設定していることに注意されたい。

```
⟨ 新しいib1 とib2 を求める 11 ⟩ ≡
  while (ib2 ≥ 0 ∧ A[ia] + B[ib2] > max) ib2 −−;
  if (ib1 > ib2) ib1 = ib2;
  while (ib1 ≥ 0 ∧ A[ia] + B[ib1] ≥ min) ib1 −−;
```

This code is used in sections 10 and 12.

12. ia=0の場合に$ib1$と$ib2$の値を求めるのは、初期値をうまく与えることで一般の場合と同様にできる。

```
⟨ A[0] に対して条件を満たすBの区間を求める 12 ⟩ ≡
  ib1 = ib2 = NB − 1;
  ⟨ 新しいib1 とib2 を求める 11 ⟩
```

This code is used in section 7.

13. 繰返しが終了した時点で、求めるべき数値はsumに入っている。

```
⟨ 後始末 13 ⟩ ≡
  return sum;
```

This code is used in section 4.

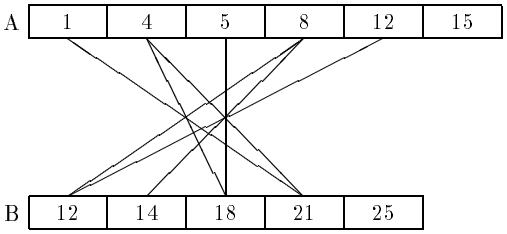

図 3: “条件を満たす対の個数” 問題の例と条件を満たす対

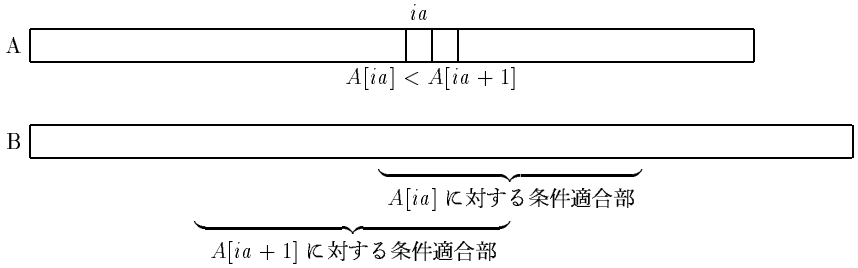

図 4: $A[ia]$ と $A[ia+1]$ に対する B の条件適合部

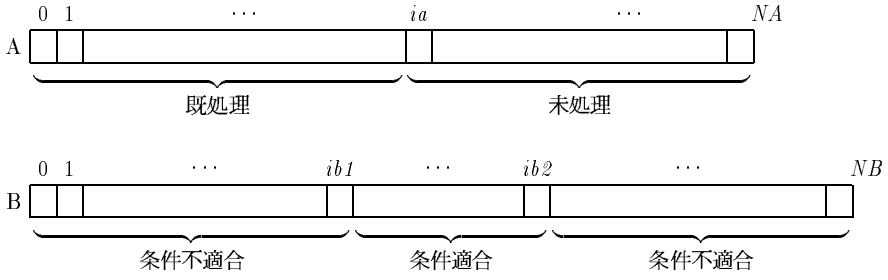

図 5: “条件を満たす対の個数” 問題のためのループ不変式