ロープ飛び移りゲームと動的計画法

情報処理学会・学会誌「情報処理」
2023年6月5日 09:21

久野　靖（電気通信大学）

おなじみ「教科『情報』の入試問題って?」，今回は，大学入学共通テストの2023年本試験「情報関係基礎」第3問を取り上げます（この問題を含むさまざまな問題のアーカイブは文献1）にあります）．

なぜこの問題？ということですが，「情報 I」の試験が始まっていない現時点では，「情報関係基礎」の問題が参考にできる問題の最有力候補です．そして，毎回，第3問は「プログラミング」を題材としているので，取り上げてみよう，ということです．

そして，特にこの問題は，難しい解法をどれくらい知っているべきか，という悩ましい議論とかかわりがありますので，それについても後半で取り上げます．で

は，始めましょう．

目次

ロープ飛び移りゲームって?

この問題では，「ロープ飛び移りゲーム」というものを規定し，その得点を高くするためのアルゴリズムとプログラムを考えていきます．最後には表題にある「動的計画法」（後で詳しく説明します）を使って，最適解を求めるところまでいくのですが，始めはもっと簡単なやり方から着手します．でもその前に，ゲームの説明から始めないと，ですね．

Aさんは，天井から地上まで吊り下げられたロープをキャラクターに順次飛び移らせてゴールを目指すゲーム(図1)を遊んでいる。ロープは全部で11本あり，一列に並んでいる。ゲームはキャラクターが1本目のロープの高さ55 mの地点にいる状況から始まり，11本目のロープの高さ0 mの地点がゴールである。キャラクターができることは，ロープを降りることと，同じ高さのまま次のロープに飛び移ることの二つのみであり，ロープを登ったり，前のロープに戻ったりすることはできない。各ロープには1か所ずつリボンが巻かれており，キャラクターがこのリボンに触れるたびに得点を1点獲得できる。各ロープにリボンが巻かれている高さ(以下，**リボンの高さ**と呼ぶ。)はあらかじめ決まっており，表1のとおりとなっている。ゴールに到達して最後のリボンの得点を加えたゲーム終了時の得点をできるだけ高くするため，Aさんは手続きを作成して，どのようにキャラクターを動かしたらよいか検討することにした。

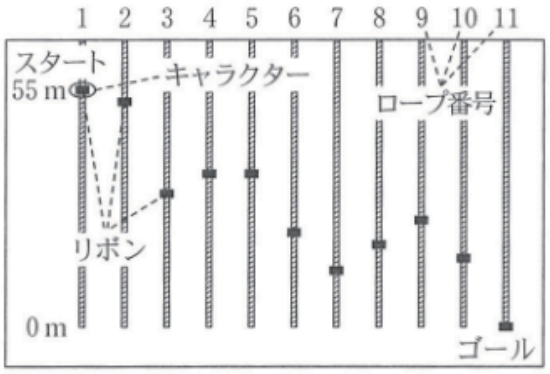


図1　ゲーム画面

問題文：ゲームの説明

いかがでしょうか．要は，最初のロープで55mの高さにいて，次々にロープに飛び移り，どのロープでも下にだけ移動できて（下に移動しなくてもよい），最後のロープの0mの高さでゴールします．その間に，ロープに巻かれたリボンに触れると得点でき，その数をできるだけ多くしたいわけです．

ロープが11本，最初のリボンの高さが55m，最後のリボンの高さが0mに決められているのは考えやすくするということで分かりますが，途中のリボンの高さも**表-1**のように決まっていることになっています．アルゴリズム的にはもっとさまざまな配置に対応できるのですが，配置を限定した方が考えやすいという配慮からでしょうか．ちなみに，私の近辺ではこれについて「複数の配置を示すべきだ」「始めはもっと容易な配置がよい」「終わりの方ではもっと難しい配置を出してほしい」等，さまざまな意見が聞かれます．

ロープ番号	1	2	3	4	5	6	7	8	9	10	11
リボンの高さ(m)	55	53	31	37	37	22	13	19	25	16	0

表1　リボンの高さ

配置が与えられたので，すこし具体的な動かし方を考えて見ます．ここで，水平移動とロープを降りるだけ，というのがポイントになります．たとえば最初のロープから55，53，31と連続してリボンに触れると，次の37，37の2本は31より高いので触れることができません．3本目の31をパスすれば，55，53，37，37と5本目までのロープで4点を獲得できます．このように各ロープで触れたりパスしたりというのを選択して高得点を目指す，というのがこの種の問題の要点です．

問1：意外な動かし方

なのですが，問1では「意外な」動かし方が出てきます．見てみましょう．

Aさんは手始めとして，全部で11本のロープで合計55m降りるのだから，各ロープを5mずつ降りるという動かし方を考えた。このゲームはロープの降り始めや降り終わりでもリボンに触れたとみなすので，1本目のロープで55mから50mまで降りるときは，55mの高さのリボンに触れたこととなり，1点獲得できる。2本目では50mから45mまで降りるが，ここではリボンに触れない。また，6本目では［アイ］mから［ウエ］mまで降りるが，ここでもリボンに触れない。

この動かし方での得点を求めるためにAさんが作成した手続きが図2である。なお，各ロープにおけるリボンの高さは配列**Ribon**に格納されており，**Ribon[i]**は**i**本目のロープのリボンの高さを表す。また，変数**tokuten**には得点を，変数**takasa**にはキャラクターが今いる高さを格納する。手続きを実行し，ゲーム終了時の得点は［キ］点であることがわかった。

問1：問題文

つまり，各ロープで5mずつ降りるというのです．リボンの状況にかかわらず決まった動き方をする，という点で「意外な」と私は感じましたが，皆様はどうでしょうか．もっとも，このような「決まった」動かし方が，最初は分かりやすくてよい，という考えもあるでしょう．

さてそうすると，各ロープで降りる前の高さは順に，55m，50m，45m，40m，35m，30mとなりますから，6本目の【アイ】は「30」，【ウエ】は降りた後ですから「25」となります．

得点は，各ロープで「降りる範囲」にリボンがあるものだけ1点を獲得しますから，表1と「降りる範囲」を照らし合わせた結果，ロープ1 (55m～50m)，ロープ4 (40m～35m)，ロープ11 (5m～0m) の3本が該当し，【キ】は「3」となります．

さて次は，この動かし方による得点を計算するプログラムです．配列や変数の使い方がまず説明されていましたね（あと，ストーリー的には【キ】もプログラムを動かした結果となっていますが，実際には上でやったように，手で解く方が素直でしょう）．

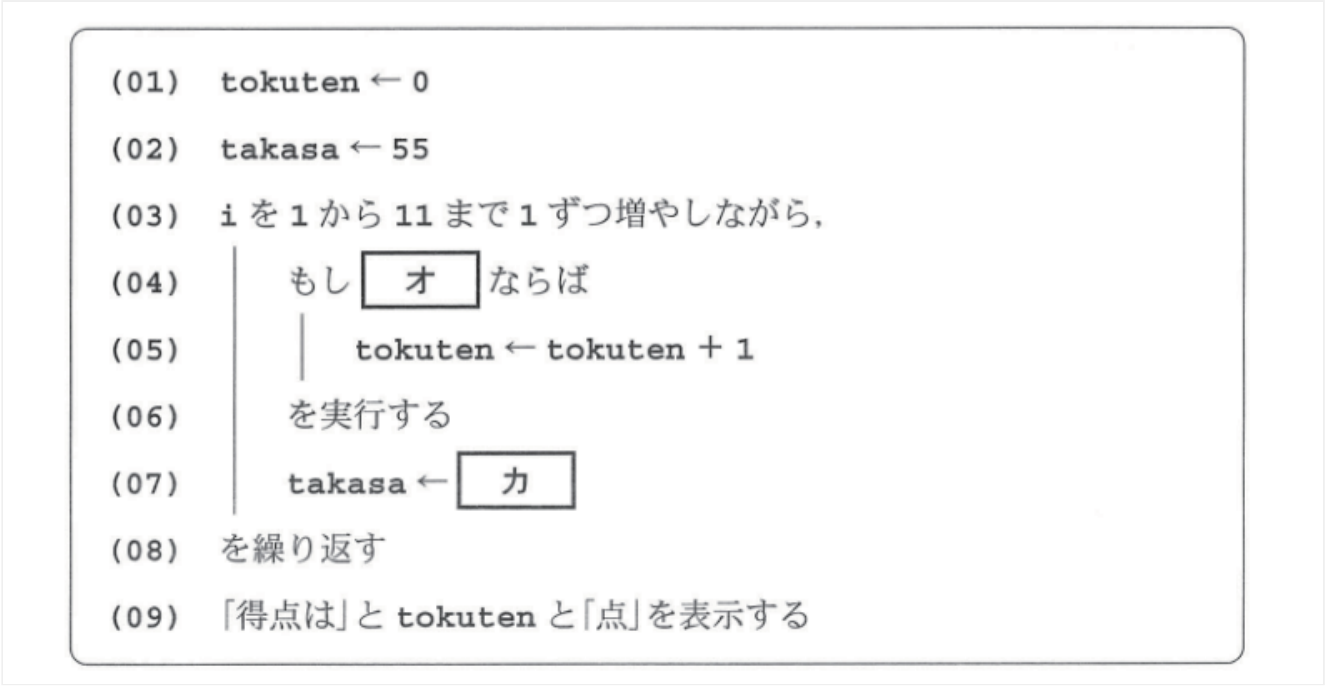


図2　5mずつ降りるときの得点を求める手続き

コードを見てみましょう．tokutenとtakasaは最初は0点と55mですね．次はiを

ループ番号1〜11まで変えながら繰り返します．その中は，【オ】なら得点を1増やす，そして高さを【カ】で置き換える，となっています．これらの穴の解答群は次にあります．

問1：解答群

【オ】ですが，得点できるのは「リボンの高さが，降りる前の高さ以下で，かつ，降りた後の高さ以上」です．まだtakasaを更新していないことに注意すれば，「降りる前の高さ」がtakasaで，「降りた後の高さ」がtakasa - 5と分かります．そこで「Ribon[i] ≦ takasa かつ Ribon[i] ≧ takasa - 5」が正解としたいのですが，「かつ」の両側が入れ換えてあるので，「Ribon[i] ≧ takasa - 5 かつ Ribon[i]

≦ takasa」つまり②が正解です.

【カ】はもっと簡単で，高さを前の高さより5mずつ減らしていくので，「takasa - 5」つまり③が正解です.

問1の範囲で振り返ってみると，【ア〜エ】はこの動かし方が読み取れれば正解できます．【キ】は表1のデータを通して得点を計算できれば正解できます．これらはアルゴリズムの問題です．【オ〜カ】はアルゴリズムをどのようにプログラムに翻訳するかが分かっている必要がある，プログラムの問題です.

問2：普通の動かし方

さて，問2に進むと別の動かし方が出てきます．実際には，こちらの動かし方の方が「まず思いつく」普通のやり方だと思われます.

Aさんは，**問1**の動かし方では1本目のロープでリボンに触れた後に5m降りたことで，2本目のロープのリボンに触れ損ねて損をしていることに気がついた。そこでAさんは毎回5m降りるという動かし方をやめ，次の新しい動かし方を考えた。

【新しい動かし方】 キャラクターが今いるロープでリボンに触れることができるときは，リボンの高さまで降りてリボンに触れた後に次のロープに飛び移る。そうでないときは，ロープを降りずにそのまま次のロープに飛び移る。これを最後のロープまで順次繰り返す。

この動かし方での得点を求めるために作成した手続きが図3である。

問2：問題文

つまり，「リボンの高さが現在の高さ以下（等しい場合も含む）なら，リボンの高さまで降りて（0m降りる場合も含む），点数も1増やす」わけです．

今度は，この動かし方での得点を計算するプログラムをまず作ります．書いてありませんが，変数 tokuten，takasa，そしてロープ番号iの役割は先の問題と同じです．

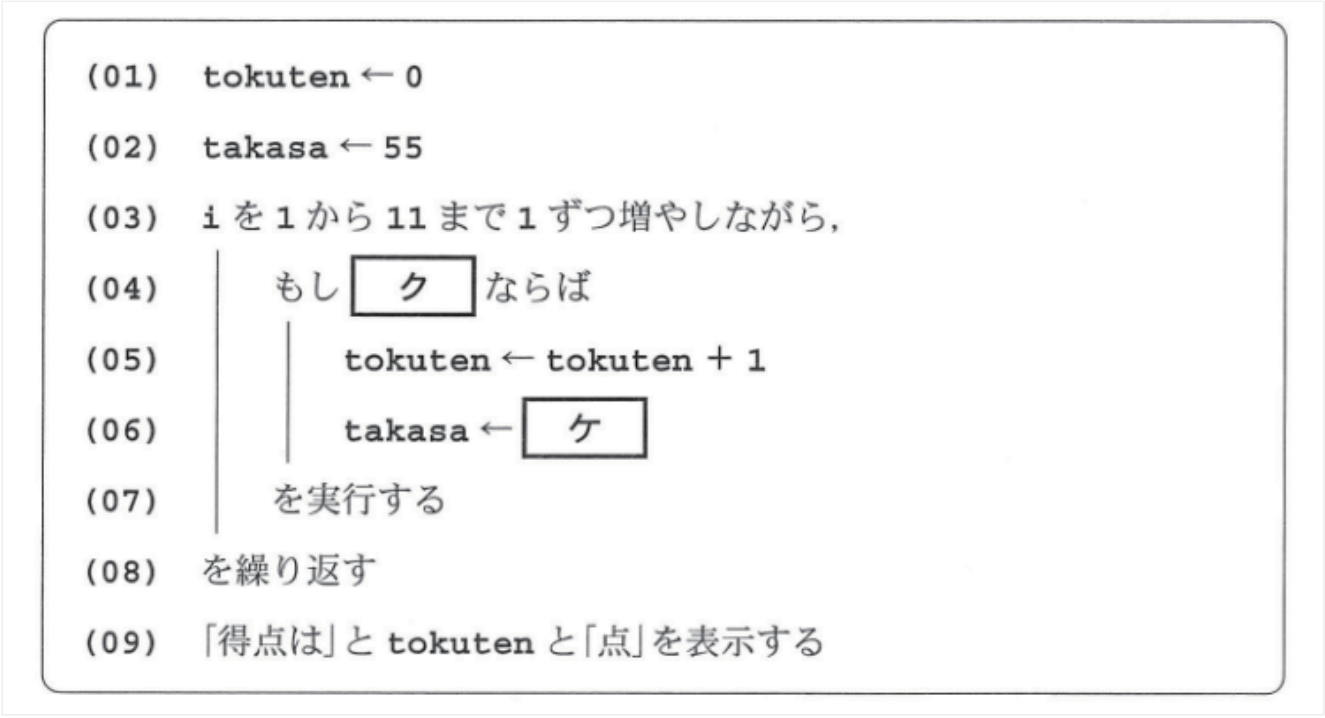


図3　新しい動かし方での得点を求める手続き

穴は枝分かれの条件と高さの置き換え式の2つで，先の問題と同じです．ただし，さっきは「得点はある条件の時のみ，高さは毎回動かす」だったのが，こんどは「ある条件のときのみ，得点して，高さも動かす」になっています．この違い（コードで言うと「を実行する」と「takasa ← ...」の順番の違い），分かりますね？　これらの穴の解答群は次のものです．

問2：解答群

【ク】については，「リボンの高さが、現在の高さ以下（等しい場合も含む）」ですから，「Ribon[i] ≦ takasa」で②となります．【ケ】については，「リボンの高さまで降りる」ですから，「Ribon[i]」で⓪となります．

手続きの流れを確認するため，表2を用意して，図3の(07)行目の直後における i，tokuten，takasa の値を記録した。その結果，i ＝4のときの tokuten の値は［コ］，takasa の値は［サシ］であった。手続きを実行し，ゲーム終了時の得点は［ス］点であることがわかった。

表2を眺めていてAさんは，あるロープで降りすぎると，その後の複数のロープのリボンに触れ損ねて損をすることがあると気がついた。そこで，新たに定数 GENDO（単位はm）を導入し，リボンに触れるために GENDO m 以上降りる必要があるときはロープを降りずにそのまま次のロープに飛び移るように動かし方を改めることにした。ただし，最後のロープではリボンの高さである0mまで必ず降りることとする。試しにAさんは GENDO の値を20としたうえで，図3の(04)行目の［ク］を［セ］と書き換えて手続きを実行した。その結果，ゲーム終了時の得点は［ソ］点となることがわかった。

問2：問題文つづき

次に，プログラムの途中 (i = 4) および最後における変数 tokuten，takasaの値をたずねる問いがあります．しかし，これらの変数は得点と現在の高さですから，アルゴリズムから考えてもできますね．

i	1	2	3	4	5	6	7	8	9	10	11
tokuten	1	2		［コ］							［ス］
takasa	55			［サシ］							

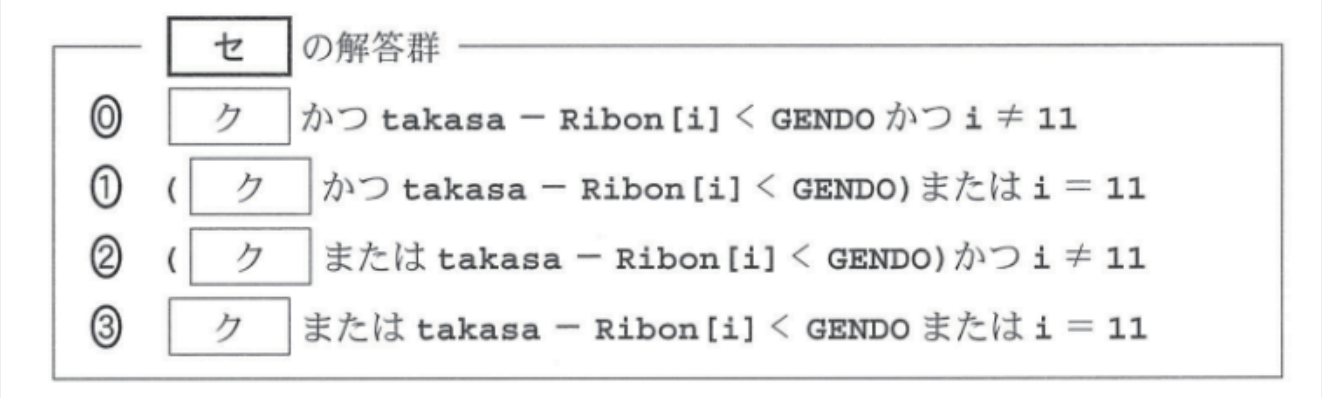
表2　図3の手続き(07)行目の直後における i，tokuten，takasaの値

この動かし方では，4までの現在の高さは「55」「53」「31」そしてリボンの高さが現在の高さより高いので次は何も変化せず「31」です．したがって【サシ】は「31」となります．得点は最初から「1」「2」「3」そして次のリボンには触れないので「3」のままです．したがって【コ】は「3」となります．【ス】はこの動かし方で最後までいったときの得点で，触るリボンは「55」「53」「31」「22」「13」「0」の6つなので「6」となります．

これで問2が終わりそうなものですが，それでは十分でないと思ったのか，続く8行でアルゴリズムの改良を取り上げています．

具体的には，条件「リボンの高さが現在の高さ以下（等しい場合も含む）なら，」を「リボンの高さが現在の高さ以下（等しい場合も含む），かつ，リボンに触れるために降りる高さがGENDO未満なら，」に直すという改良です（ただし，i=11 のときは最後なので，常に降りる）．なお，GENDOは「試しに」20とするとなっています．

この変更は，【ク】の条件の穴を置き換えればできます．その新しい穴を【セ】とし，解答群を示しています．



問2：解答群つづき

【ク】との条件の結合は「かつ」なので⓪ ①のどちらかですが，i=11 のときはともかく（内側の条件の成否にかかわらず）降りるので，【セ】は「（... 内側の条件...）または i=11」すなわち①となります．

それはいいのですが，いままで【ク】の条件が生きていたのに，今度のアルゴリズムでは i=11 のときにこの条件を無視しています．気持ち悪いのですが，実はロープ11ではリボンの高さが0mであるため，条件【ク】は常に成り立つので，前のアルゴリズムでは「または i=11」がなくてもよかった，しかし今度はGENDOの条件があるのでそうはいかない，という仕掛けです．

問2の最後はGENDOを20としたときの得点を求めます．「55」「53」で，続く31は降りる高さが GENDO 以上になるのでスキップし，「37」「37」「22」「13」で，あとは 13 より高い値が続いて，最後に「0」ですから，【ソ】は「7」となります．

問2について，振り返ってみましょう．【ク〜ケ】はプログラムの穴埋めで，問1のプログラムとの違いをきちんと分かっている必要があります．【コ〜シ】は変数の途中の値ですが，どちらかと言えばアルゴリズムの問題です．【ス】は問1で言えば【キ】のように，このアルゴリズムで最後まで動かしたときの得点が計算できるか見ます．

そしてこの先，アルゴリズムの改良版が登場し，【セ】でそれをコーディングする穴埋め，【ソ】で最後まで動かしたときの得点を尋ねています．1つの問の中にアルゴリズムが2つというのは，かなりハードな感じがしますし，【セ】の穴埋めの内容が複雑なので一層そう思われます．

問3：動的計画法

問3に進んで，いよいよ動的計画法が出てきます（この仰々しい名称は出していませんが）．動的計画法をかいつまんで言うと，「大きさNの問題を解くのに，配列を用意して，大きさ1〜N-1の問題に対する解を全部覚えおくことで，たやすく最良解を得る手法」です（正確には配列が2次元以上の場合も有り得ますが，そこは省略）．では見てみましょう．

Aさんはこのゲームで獲得可能な最高得点を求めるため，次の手順を考えた。まず，1本目からi本目のロープまでに限定して考え，しかもi本目のロープのリボンには必ず触れることとする。このとき獲得可能な最高得点を**Kokomade[i]**点とする。

i＝1のとき，このゲームでは1本目のロープのリボンの高さが55 mでスタート地点と一致しているので，**Kokomade[1]**の値は1である。i≧2のときは，iより小さいすべてのt(ただしt≧1)について**Kokomade[t]**が求まっていれば，次のように考えて**Kokomade[i]**を求めることができる。

【Kokomade[i]の求め方】 t本目のロープのリボンに触れて，かつi本目のロープのリボンにも触れることができる条件は，［ タ ］であるが，その条件を満たすすべてのtの中で，**Kokomade[t]**の値が最も大きいものを選ぶと，「i本目のロープのリボンに触れる一つ前に触れておくべきリボン」が定まる。すると，その**Kokomade[t]**の値にもとづいて，**Kokomade[i]**の値が定まる。

iの値を2から順次増やしていけば，すべてのiについて**Kokomade[i]**の値を求めることができる。最後のロープのリボンの高さは0 mなのでこのリボンには必ず触れることができることを考えると，**Kokomade[11]**がこのゲームで獲得可能な最高得点となる。この手順を考えたAさんが作成した手続きが図4である。

問3：問題文

この「i=1 のとき，Kokomade[i] は簡単に求まる」「i ≧ 2 のとき，Kokomade[t](1 ≦ t ＜ i) が求まっていれば，これこれの方法で Kokomade[i] が求

まる」という，数学的帰納法によく似た説明が，動的計画法のパターンです．初めての方はかなり戸惑うと思いますが，落ち着いて理解してください．幸い，動的計画法はプログラムを作る段階では，比較的簡単なことが多いです．

では，文章とプログラムの両方に共通の，【タ】の穴を考えます．t 本目のループのリボンに触れて，かつ，それより後の i 本目のループのリボンに触れることのできる条件は「後者のリボンの高さが前者より高くない（同じか低い）」ですから，【タ】は「Ribon[t] ≧ Ribon[i]」つまり解答群の③になります．

```
(01)  Kokomade[1]←1
(02)  i を 2 から 11 まで 1 ずつ増やしながら，
(03)  |  saikou←0
(04)  |  t を 1 から i − 1 まで 1 ずつ増やしながら，
(05)  |  |  もし [ タ ] かつ saikou < Kokomade[t] ならば
(06)  |  |  |  saikou←Kokomade[t]
(07)  |  |  を実行する
(08)  |  を繰り返す
(09)  |  Kokomade[i]←[ チ ]
(10)  を繰り返す
(11)  「獲得可能な最高得点は」と Kokomade[11] と「点」を表示する
```

図4　獲得可能な最高得点を求める手続き

コードを見てみると，まず Kokomade[1] ← 1とした後，i を2から11まで1ずつ増やしながら Kokomade[i] を求めていきます．この順なら，i 番目を求めるときには，1〜i-1番目はもう求まった状態にあり，動的計画法の条件が保たれることになります．

そのループの内側の(03)-(08)のところで，t=1, 2, ...,i-i かつ【タ】の条件が成り立つ Kokomade[t] の最大値を求めて，変数 saikouに入れています．i 番目のループではその値にこのループのリボンに触った1点を加えますから，【チ】は「saikou+1」すなわち①になります．

タ の解答群

⓪ Ribon[t] < Ribon[i]　　① Ribon[t] > Ribon[i]

② Ribon[t] ≦ Ribon[i]　　③ Ribon[t] ≧ Ribon[i]

チ の解答群

⓪ saikou　　① saikou + 1

② Kokomade[i − 1] + 1　　③ Kokomade[i − 1] + saikou

問3：解答群

このように理屈は分かりにくくてもコードは比較的単純なので，それで終わりにはしにくいからでしょうか，続いて，i=3, 4, 5 のときの Kokomade[i] と，最終的

な最高得点つまり Kokomade[11] を尋ねています.

手続きの流れを確認するため，表3を用意して，図4の(09)行目の直後における i，Kokomade[i] の値を記録した。i ＝2のときは，2より小さい t は1のみであり，しかもこのとき条件［ タ ］を満たす。つまり1本目と2本目の両方のロープのリボンに触れることができるので，Kokomade[2] の値は2となる。同様に記録を続けると，i ＝3，4，5のときの Kokomade[i] の値はそれぞれ［ ツ ］，［ テ ］，［ ト ］となる。手続きを実行し，このゲームで獲得可能な最高得点は［ ナ ］点であることがわかった。

問3：問題文つづき

i	1	2	3	4	5	6	7	8	9	10	11
Kokomade[i]	1	2	ツ	テ	ト						ナ

表4　図4の手続き(09)行目の直後における i，Kokomade[i] の値

表1を見ながら判断してください．ループ3はループ2から来られますから，【ツ】はループ2の Kokomade[i] に1を加えた「3」．ループ4はループ3からは来られず，ループ2からなので，【テ】も「3」．ループ5はループ4から来られるので，【ト】は「4」．ループ6はループ5からで5，ループ7はループ6からで6．この先がリボンの高さが高くなるので注意が必要です．ループ8はループ6からで6，ループ9はループ5からで5，あとはループ10はループ8からで7（得点が高いもの

を選ぶこと！），そしてロープ11すなわち【ナ】はロープ10からで「8」となります．

問3について，振り返ってみましょう．この問はとにかく，解法が分かりにくいのですが，誘導にしたがって【タ〜チ】の穴を埋めるのは解法を完全にマスターしなくてもできそうです．続く【ツ〜ト】も，そこまでの最良値ですから，解法をマスターしていなくてもわりあいできます．【ナ】は解法が分かっている人向けですが，それらしい値を入れて正解する人もいるかもしれません．

そして，あまり手間でなく最適解が求められる，というのも動的計画法を採用した恩恵です．動的計画法を使わずに一般に最適解を求めようとすると，各ロープについて，そのロープで降りる場合と降りずにパスする場合の両方を考えることになり（ただしそれより前のロープの選択によってはパスしか選択できない場合もある），プログラムも大変ですし，その実行時間も多くなります．

動かし方を求める「トレースバック」

問3には，解法の理解を前提としなくても多くの問に正答できてしまうという難点（これは，解答者にとってはやさしくなっているという点で「慈悲」）があります．そして，もう1つの不満は，プログラムを作る目的が「どのようにキャラクタを動かしたらよいか検討」することのはずなのに，問3のプログラムでは動かし方

が分からない，という点です．

実は，動的計画法では，解法を求めている最中は「1〜Nに対する最適値を同時に求めている」ため，個別の動かし方は分かりません．動かし方を知るには，各 i に対する「どこからが一番優れていたか」（前節で「ループXはループYから来られる」と言っていたもの）の情報を覚えておき（これをトレースバック情報と言います），Nからこの情報を遡ることが必要です．

実際に問3のプログラムを直してみましょう．トレースバック情報も配列に入れる必要があるので，これを Koko_mae[i] という名前にします．(01)と(02)の間に次のものを入れます．

```
Koko_mae[1] ← 0
```

これは「ループ1の前はない」に相当します．続いて，(06)と(07)の間に次のものを入れます．

```
|  |  |  Koko_mae[i] ← t
```

つまり saikou を更新したときに，そのループ番号を Koko_mae[i] に覚えます．

そして，(11)の後にトレースバック情報をたどって表示する次のものを加えます．

```
i ← 11
i が 0 でない間、
 |「ロープ」と i と「が高さ」と Ribon[i] と「m」を表示する
 |  i ← Koko_mae[i]
を繰り返す
```

表1の状況に対して，この拡張されたプログラムを実行すると，次の出力となります．トレース「バック」の名前どおり，最後から逆順に表示されていきます．

```
獲得可能な最高得点は 8 点
ロープ 11 が高さ 0m
ロープ 10 が高さ 16m
ロープ 8 が高さ 19m
ロープ 6 が高さ 22m
ロープ 5 が高さ 37m
ロープ 4 が高さ 37m
ロープ 2 が高さ 53m
ロープ 1 が高さ 55m
```

分量の問題からトレースバックまでは入れなかったと思われますが，結果の分か

りやすさまで考えるなら，入れてほしかった気はします．

まとめ：動的計画法は学ぶべき？

そういうわけで，この問題は「ロープ飛び移りゲーム」について，問1で意外な動かし方，問2で普通の動かし方，問3で動的計画法による最適な動かし方を扱い，どの問でもアルゴリズム，プログラム，具体的な解を扱うようにできています．難易度は「題材にしては」やさしく設定されているように思います．

この問題について一番悩むのは，何といっても「動的計画法を学ぶか（生徒に学ばせるか）否か」でしょう．次の2説があります．

- 難しくて悩むだけなので，学ばない．この問題の問3のように，知らなくてもその場で理解できるように配慮されているし，完全に理解しなければいけない問題はあったとしても配点は少しだろう．
- いろいろ役に立つ場面もあるので，頑張って学ぶ．そうすれば，この問題のようなものに出会ったときに安心していられる．マスターするのに少しかかっても，価値がある．

「いろいろ役に立つ」の例として，たとえばさまざまな「プロコン（プログラミング・コンテスト）」があります．プロコンでは，問題を解く時の計算量（計算時

間）に制約を設けるのが普通なので，あるレベル以上の問題では動的計画法が必須となります．

とは言え，プロコン等はそれこそ「好きな人の」ものですから，普通に情報の授業を受けている人に常にお勧め，とは言えません．結局，ここの問3の解説を見て「わかった，もっと学んで見たい」と思った人はどうぞ，ということでしょうか（たとえば文献2）には動的計画法も含めてさまざまな手法が解説されています）．

明確な結論がないのは申し訳ないですが，この問題をきっかけとして，動的計画法というものがある，ということを頭の隅にとどめていただければ幸いです．

参考文献

1）情報関係基礎アーカイブ，情報処理学会 情報入試委員会，
https://sites.google.com/a.ipsj.or.jp/ipsjjn/resources/JHK
2）秋葉，岩田，北川：プログラミングコンテストチャレンジブック［第2版］，マイナビ (2012).

（2023年3月24日受付）
（2023年6月5日note公開）

■**久野　靖**（正会員）

電気通信大学 特命教授，筑波大学 名誉教授．理学博士．プログラミング言語，ユーザインタフェース，情報教育に興味を持つ．

情報処理学会ジュニア会員へのお誘い

小中高校生，高専生本科～専攻科1年，大学学部１～3年生の皆さんは，情報処理学会に**無料で入会**できます．**会員になると有料記事の閲覧，情報処理を学べるさまざまなイベントにお得に参加できる等のメリットがあります**．ぜひ，入会をご検討ください．入会は**こちら**から！