

立体 pentomino のプログラム

電電公社武蔵野電気通信研究所

竹内 郁雄

1. はじめに

"GPCC 發足に類するを試みんと思ひ立ったのは、昨今の
ことではなけれど、實際その期熟するは忘れもせない、立体
pentomino 六百六拾時間たる見積りの世に同はれたる^{*)}翌日の本
年一月の拾一日ぞ、その日以来本来の業務はあり、殊には不
肖私も亦輕症の風邪に犯されたりと(中略)、難題が起ると
態とえを打棄ておき、廁上又禰上にて腦漿を絞らしことも常
であつたが(中略)、刻々拾一時間強にて全數解を求めたる
ものなり^{**)}"というわけぞ、当初二桁時間であればという期待
を越えて、一桁時間に肉迫する結果が出たのでここに報告し
よう。

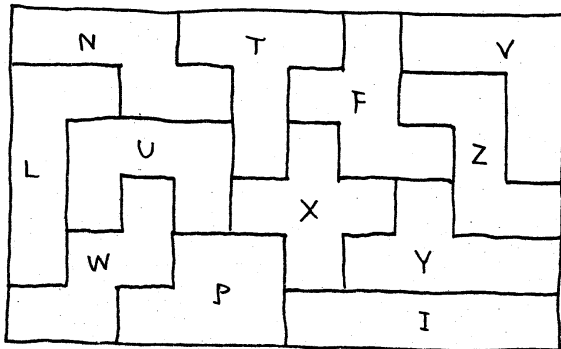
*) 航宇研の磯部氏「立体パントミノの解法について」第15回

プログラムシンポジウム 1974. 1

**) 尊敬する松浦政泰氏の「世界遊戯法大全」による。

2. 立体 pentomino について

単位立体体を5個平面上に並べると次のような12種類のピースが出る。(紙面節約のために6×10のお盆に最密充填



した形を示す.)

さて、これを3×4×5の直方体スペースに充填しようというのが、ここぞいう(狭義の)立体pentominoで

ある。これが解を持つことは古くから知られていたようであるが、この解の総数が知られたのは比較的最近である。我が国の天洋で売り出したプラスチック製の製品には、FACOMという字が箱にしたためてあるが、その横に"460通りの組合せができる3次元の立体パズル!!"と丁寧に二つもビックリマークがついている。しかし、実は解の総数は1967年にC.J. Bouwkampという人が求めた3940通りというのが正しい。使った計算機はIBM-1620でFORTRANを言語としたらしいが、そのときに要した時間は"several hundreds hours"だったそうである。ただし、若干のコメントがあって、CDC-3600上のアセンブラ・プログラムの集積を利用すれば3時間位で全数解が求まると言っている。しかし、それがその後実

現されたという話は伝わっていないようである。磯部氏はかなり一般的な構想のもと（4次元直方体などを含む）プログラムを作らしたようであるが、HITAC 5020Fで660時間かかるという見積りを出したのは、IBM-1620との単純な比較だけでもやや遅過ぎるのではないかと思われる。CDC-3600程度の機械は現在ミニコンとしてかなり出まわっているので、3時間が本当かどうかの意味も含めて、時間短縮を試みることにしよう。

3. プログラムを組む上での基本方針と構想

使用する計算機は現在かなり世に出ているpdp 11/20である。この機械の基本的な特徴を必要なだけ述べておくと

- a. レジスタが8個あり、1つはプログラム・カウンタで1つはサブルーチン等のためのスタック用レジスタである。（前者をPC, 後者をSPと呼ぶ）あとのレジスタは便宜上R0, R1, ..., R5と名付けておく。このら8個のレジスタに演算上の機能の差はないし、すべてインデックス・レジスタとして使える。
- b. 演算速度. レジスタ・レジスタで加算が $2.3\mu\text{sec}$, メモリを1回アクセスすると $1.5\mu\text{sec} \sim 3.9\mu\text{sec}$ 余分にかかる。いわゆる中速度の計算機である。

- c. バイトでアドレスがついている。1語は16bit である。
- d. シフト関係の命令が弱いのでビット処理を得意としない。

また、我々の使用したシステムに関してさらに付け加える
と

- e. デスク・オペレーティング・システムで28K語のメモリを持ち、他の周辺機器(MT, DISK...)もかなり完備している。
- f. アセンブラはマクロの入出力が何重にも出来る機能を持つなどかなり高度なものが用意されている。

さて、立体pentomino のプログラム作成の基本方針として
は

- (1) シフトが弱いので、bit 単位でマス目情報を保持せず
に、word 単位で保持する。つまり通常の array で直方
体を表現する。
- (2) 番号の若いマス目から、順にピースを埋めていく。磯
部氏のプログラムはピースをバラバラに入れていくのだ
が、この方法のほうが(単純なことだが)10倍位速
くなる。ピースの順番は、大体普通。
- (3) コーディング技術で時間を稼ぐ。即ち複雑なヒューリ

ステップを入れない。だから原理的にシラミつぶしプログラムとする。

(4) ステップ数, メモリ使用量については湯水の如きの方針を貫く。

(5) 対称解は最初から除くようにする。(プログラムの信頼性はすこし下がるが気にしない。)

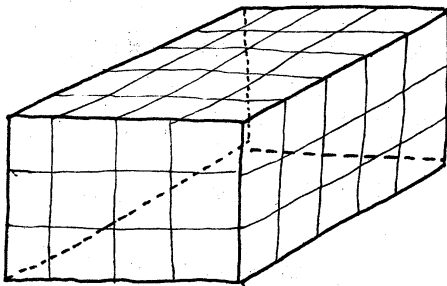
(6) (4)に関連するが, ループ管理のための overhead がもたないないので, ループを廃止しベタ書とする。またプログラム・フェッチとデータ・フェッチを一体化して時間を節約する。すなわちデータをプログラム中に本質的に埋め込んでしまう。(これによって2~4倍のスピードアップがはかれる。)

4. プログラムの基本骨格

3×4×5の直方体は次の様に表現される。(あとでアセンブラに密着した説明をするので, 予備説明としてリスティングをのせておく) array の各語はマス目か, 境界を示すためのダミーであり, 前者はそこが空いているときめ, なにかピースがつまっているときには, そのピースの名のコード(ASCII)が入れられている。後者は常に1がはいっている。次のリストはarrayの初期状態である。

12	.MACRO	TBGNR N	引数Nなるマクロ TBGNRの定義
13		.IF EQ,N	N=0なら以下を..
14		1,0,0,0,0	
15		1,0,0,0,0	
16		1,0,0,0,0	
17		.IFF	条件がF (i.e. N≠0)なら以下を..
18		1,1,1,1,1	
19		1,1,1,1,1	
20		1,1,1,1,1	
21		.IFTF	条件がTでもFでも以下を..
22		1,1,1,1,1	
23		.ENDC	条件文終り
24	.ENDM		マクロ定義終り
25	00002	TBGNR 1	
26	00052	TBGNR 0	
27	00122	TBGNR 0	
28	00172	TBGNR 0	
29	00242	TBGNR 0	
30	00312	TBGNR 0	
31	00362	TBGNR 1	

array (代表マスにTBと呼ぶ)



マス目の番号はTBから数えて何番地(8進で表示, 以後断りなき数値は8進数)であるかを示す。左図で1番奥

の3×4平面は下の(a)のように番号づけられており, その

2	4	6	10
14	16	20	22
26	30	32	34

(a)

52	54	56	60
64	66	70	72
76	100	102	104

(b)

手前の断面は左の(b)

の如くなっている。

マスが一つ手前だと

50たされた番号にな

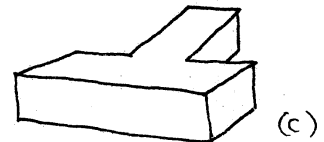
ることに注意しておこう。

ピースはあのおの何種類かの向きを持つ。これを野下氏の用語に従ってスタイルと呼ぶことにしよう。12枚のピースのスタイルで上の直方体に入るようなものは全部で160種類あ

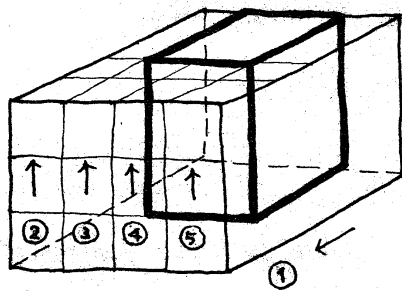
る。スタイルは、一番若い番号に入るべき単位立方体から他の4つの立方体への番号変位を書き並べることにより表現出来る。たとえば、Tというピースの(c)なるスタイルは

50, 120, 116, 122

というふうに表現出来る。番号変位の順番は都合のよいように選んでよい。



い。さて、あるスタイルを持ってきたとき、それが直方体の任意の番号のマス目を基矢として入り得るわけではない。実際、上例の(c)なるスタイルは直方体の左端、右端、前から1番目と2番目の3x4の断面のマス目を基矢として入ることが出来ない。(要するに案外入りにくい) これを各スタイルについて最初に調べておく方が得策なので、各スタイルについて valid space (VSP) なるデータを用意しておく。上の例では



左の太枠のような部分直方体が表わしたいものである。処理を速くするために、これを1語で表現する。直方体は12本のI (長さ4の

棒) からなると考えられる。この棒1本1本に1bit対応させる。またこれに直交する3x4平面が4枚あるが、このうち一番奥の(番号の若い)平面を除いたものに各々1bit対応させる。対応のさせ方は上図と次頁の図をにらんどいただけ

さて、基本チェックがOKだったとすると、次は番号変位に従ってマス目が空いている($=\phi$)か、否か($\neq\phi$)かを調べればよい。しかるに、あるピースのスタイルを調べているとき、ある番号変位のマス目が塞がっていたら、残りいくつかのスタイルのチェックを自動的にサボってもよいことが往々にしてある。たとえば、ピースTで

—  5 ϕ , 12 ϕ , 46, 44

というスタイルを調べているとき、もし5 ϕ なる番号変位のこのマス目が塞がっていたら、, , , ,

 などのスタイルは調べるまでもなくだめだということになる。これを便宜上、文脈チェックと呼ぶことにしよう。

(注意; 基本チェックも文脈チェックも枝刈りをしているわけではないことに留意されたい。) さらに5 ϕ , 12 ϕ についてはOKで、46で駄目とわかれば、次に新たに5 ϕ , 12 ϕ について調べる必要もないことに注意しよう。すなわち、あるスタイルで途中で失敗すれば、次のスタイルの途中から調べれば十分なこともある。このような文脈チェックを可能な限りとり入れた速度の向上が期待が出来る。

ここで1つのスタイルに対するチェックを書き下してみることにしてしよう。R1には当座のマス目のビットパターンがはい

っており、 $R0$ には当座調べているマス目の番地 ($TB + \text{番号}$)
がはいっており、 $R5$ にはピースを実際に埋め込むサブルーチ
ンの番地がはいっている。(後述の番地をレジスタに持っていると
命令語が短縮できる。) ; はコメントの意。

; スタイル・チェック入口

- ① BIT $R1$, the VSP ; 基本チェック。BIT は二つのオ
ペランドの AND をとって条件コードを立て
るだけの命令。
- ② BNE 上が失敗したときに飛びべきスタイル・チェック
; BNE $\equiv$ Branch if Not Equal (to zero)
- ③ TST 番号変位₁($R0$) ; TST $\equiv$ TeST. オペランドに関
する条件コードを立てるだけの命令。
オペランド X (レジスタ) の意味は、
レジスタの内容 + X が実効番地であ
る。
- ④ BNE このが失敗したときの飛び先。
- ⑤ TST 番号変位₂($R0$)
- ⑥ BNE このが失敗したときの飛び先。
- ⑦ TST 番号変位₃($R0$)
- ⑧ BNE このが失敗したときの飛び先。
- ⑨ TST 番号変位₄($R0$)

- ⑩ BNE このが失敗したときの飛び先
- ⑪ JSR PC, (R5) ; このは、全部のチェックにパスしたときに実行される。JSR $\equiv$ Jump SubRoutine.
pdp 11 ではサブルーチンジャンプと戻り先が自動的にスタックされる。つまり、次の代替スタイル・チェックの番地が自動的に保存される。

各スタイルにつき以上11ステップ(10進), 16語(10進)のプログラム・セグメントが対応する。これを全部手を書くのは大変だから、マクロを使う。左程の下のスタイルはマクロで

STYLE VSP2, 1, 5φ, 12φ, 46, 44, 7φ, 1φφ, 13, 13.

マクロ名 37φφ7 につけた名前. 番号変位 文脈チェック情報 基本チェック飛び先情報

と書かれる。基本チェック飛び先情報は、基本チェックに失敗したときに、次のスタイル・チェックをいくつか省略するかに関するものである。^(うし3に) 同じVSPのスタイルがn個並べばこれはnになる。文脈チェック情報は4つの番号変位に対応して4つある。末桁はスタイル・チェックのどの部分に飛び先を表わし、それを除いた残りの桁は基本チェック情報に同じ意味。たとえば上の例では、番号変位5φで失敗すれば、次に続くスタイル・チェック6つを省略して7つ目に、12φで失

敗すれば8つ目に, 46で失敗すれば1つ目 (i.e. 次) のスタイル・チェックの3番目の番号変位のチェックに依べという意味である。(実はこのあとにもっとたくさん省略できるチェックがあるかも知れぬが, 条件分岐命令は $\pm 128_{10}$ 語のスキップしか出来ないという制約があったりして高々7φ, 1φφで抑さえられる.)

このように出来る限り省略を行なうのであるが, 実際これを手作業でやったためかなり大変であった. たとえば, 省略を伸ばすために次のようなことをする.

i	VSP _i	変位 _{i₁} , 変位 _{i₂} , 変位 _{i₃} , 変位 _{i₄}
j	VSP _j	変位 _{j₁} , ...
k	VSP _k	変位 _{k₁} , ...
l	VSP _l	変位 _{l₁} , ...
m	VSP _m	変位 _{m₁} ,

というようにスタイル・チェックが並んでいて, 変位_{i₁} = 変位_{j₁} = 変位_{k₁} であり, $VSP_i \subseteq VSP_m$, $VSP_i \cap VSP_l = \emptyset$ (空) なら変位_{i₁} で失敗したとき, j, k なるスタイル・チェックを省略するのはもちろん, $VSP_i \cap VSP_l = \emptyset$ (空) により l も省略し (変位_{i₁} を調べるためには VSP_i のチェックにパスしているはず), さらに VSP_m の基本チェックも省略するのである. ただし, この省略が連鎖するととんでもないことになる. だから推移条

件に常に気を配ったかなり疲れる作業となる。

さて一つのピースのスタイル・チェックはまとめて並べられるが、その先頭に次のような小さなプログラムがつく。

piece 名 : JMP 次のピース ; 次のピースがなければ
label backtrack用 routineへ。

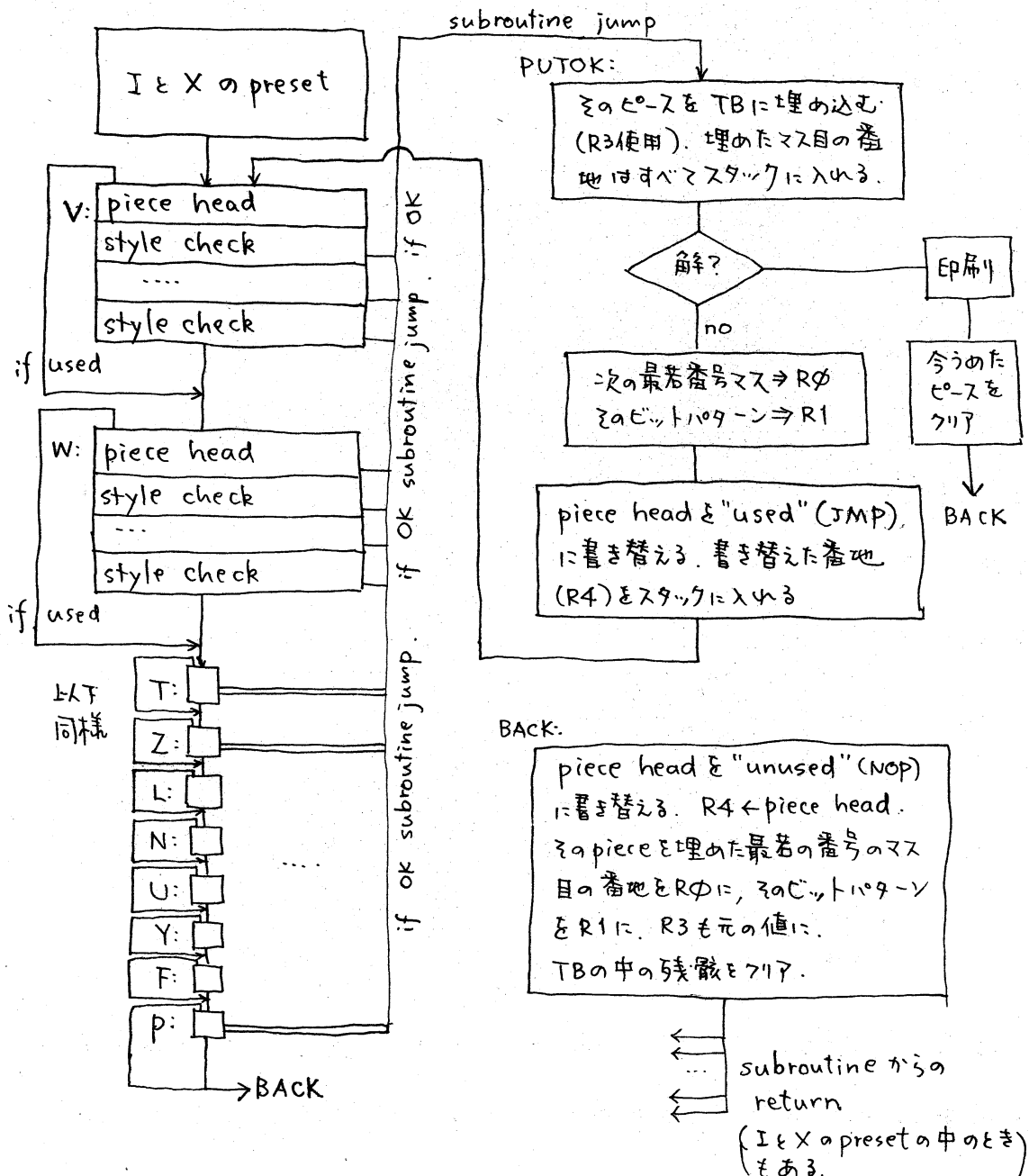
MOV piece名のASCIIコード, R3
; スタイル・チェックをする
間 R3にはピースの名を保持
させておく。

MOV piece名[label], R4
; このプログラムの一番最初
の命令を書き換えるための
番地を R4 に保持しておく。

最初のJMP命令はそのピースがすでに使われたかどうかで書き換えられている。すでにそのピースを使ったのであればもうそのピースを調べる必要はないから、ここは上の通りのJMP命令である。もしまだ使っていないのであれば、以下のステップとスタイル・チェックの列を実行しなければならなから、これは等価的に No. Operation の命令にかえられる。

(実際には BR +4, BR≡Branch, +≡location counter, 上のJMPは2語命令であるから、1語スキップせよという命令)

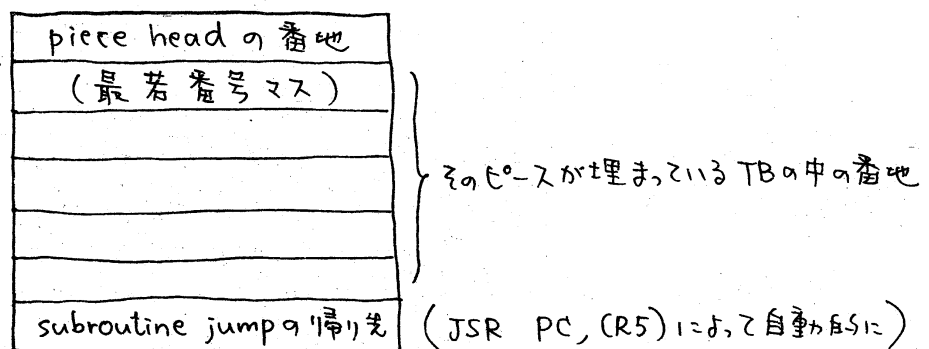
さて以上見たようにこれで、いわゆるスタイル表現用情報、used piece 情報、next style 情報などは完全にプログラム中に埋め込まれたことになる。プログラムの全構成を模式的に示すと（ただし対称解に属する部分は省く）



この図からわかるようにプログラムは大きな再帰的プログラムとなっている。(I と X の preset の部分ははっきりかからなかったが、これとほぼ同じで、BACK routine は他のピースと共用) pdp 11 の自動スタックの機能、あるいはレジスタをインデックスとして使ったときの自動インデックス増減機能は、このプログラムの性能に大きく寄与している。たとえばピースをクリアする部分は

```
CLR  @(sp)+
CLR  @(sp)+
CLR  @(sp)+
CLR  @(sp)+
CLR  @(sp)+
```

と本当に5ステップ(5語)ですんでしまう。なおスタックは一本で、1つのピースがはいると次のようなエレメントがスタックの上にのせられることになる。このスタックへのアクセスはSP レジスタを使う。



また、PUTOK routine で TB のマスと埋めるときに番号変位の情報は subroutine jump の帰先から最初の4エックプログラム

ラムをアクセスして知る。これが全部で 17_{10} ステップ(21_{10} 語)かかるのはいたしかたない。ちなみに PUTOK routine は全部で、 28_{10} ステップで中に次の最若番号マスを捜す2ステップのループがある。BACK routineは 11_{10} ステップである。

次にスタイル・チェックのマクロと piece head のマクロをリスト通りに示しておく。

```

.MACRO STYLE VSP, VSPBR, L1, L2, L3, L4, L1BR, L2BR, L3BR, L4BR
    BIT R1, #VSP
    BNE VSPBR*40-4+.
    BRCNT3=0
.MACRO CHECK LN, LNBR
    TST LN(R0)
    BRCNT1=LNBR/10
    BRCNT2=40*BRCNT1-12-BRCNT3+.
    BRCNT1=-BRCNT1*10+LNBR
    BNE BRCNT2+<BRCNT1*6>
    BRCNT3=BRCNT3+6
.ENDM
    CHECK L1, L1BR
    CHECK L2, L2BR
    CHECK L3, L3BR
    CHECK L4, L4BR
    JSR PC, (R5)
.ENDM

.MACRO PIECE IDENT, NEXTP
IDENT: JMP NEXTP
    MOV #IDENT, R3
    MOV #IDENT, R4
.ENDM

```

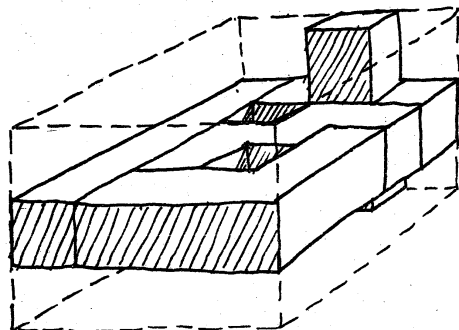
=は諸ゆる代入擬似命令で、アセンブル時に実行される。

STYLE の定義ではマクロの静的な入れ子が使われている。次にピースLについてのチェック・プログラムを示す。このように書いてみるとデータとプログラムが一体化している様子がよくわかる。

402	4024	PIECE	L, N
403	4040	STYLE	PPE, 2, 2, 4, 6, 50, 50, 40, 40, 14
404	4100	STYLE	PPE, 1, 2, 4, 6, 56, 40, 30, 30, 14
405	4140	STYLE	PPI, 2, 2, 4, 6, 20, 30, 20, 20, 14
406	4200	STYLE	PPI, 1, 2, 4, 6, 12, 20, 10, 10, 10
407	4240	STYLE	PPG, 1, 2, 52, 122, 172, 10, 10, 10, 10
408	4300	STYLE	PPF, 1, 50, 120, 170, 166, 60, 60, 60, 14
409	4340	STYLE	PPG, 2, 50, 120, 170, 172, 50, 50, 50, 14
410	4400	STYLE	PPG, 1, 50, 120, 170, 2, 40, 40, 40, 14
411	4440	STYLE	PPH, 2, 50, 120, 170, 202, 30, 30, 30, 14
412	4500	STYLE	PPH, 1, 50, 120, 170, 12, 20, 20, 20, 14
413	4540	STYLE	PPR, 1, 50, 120, 170, 156, 10, 10, 10, 10
414	4600	STYLE	PPH, 1, 12, 62, 132, 202, 30, 12, 12, 12
415	4640	STYLE	PPI, 1, 12, 14, 16, 20, 20, 12, 12, 12
416	4700	STYLE	PPJ, 1, 12, 10, 6, 4, 10, 10, 10, 10
417	4740	STYLE	PPD, 1, 50, 46, 44, 42, 20, 20, 20, 20
418	5000	STYLE	PPE, 1, 50, 52, 54, 56, 10, 10, 10, 10

5. 対称解を如何に省くか.

実はこの問題が一番大変であり、最後まで論理的虫（プログラム以前の虫）のといへなかったところである。4で見たようにプログラムの基本的な骨格はかなりスッキリしているが、対称解の追放は、かなり無駄な贅肉を呼び込んだようである。IとXに加えてV, さらに必要な場合はW, さらに必要な場合はY（総計5枚）の位置に制限を加えないといけないというのは、いかに立体であるとはいえ大変である。（思うに12個のピースが平面的であることがその最大の原因である。）



左のようにI, X, V, Wを決めても上下の対称解が有り得る（実際にはない）

対称解を追放する方法としては、まずI, X以外のV,

Wのプリセット（この場合にV, Wのピース・チェックはやらない。）し、そこから残った最初のピース（WあるいはT）のVSPを狭くしてやる方法をとる。ここでは詳しく述べないが基本骨格を変形しないようにするため、全体としてかなり"とってつけた"ようなプログラム・セグメントになっている。

6. 結果

まずプログラムのサイズについて述べる。アセンブラのステップ数は formatting のための空ステップやマクロを含むので、実際の語数を記す。以下の数値は10進。

種々の初期的疑似命令	50 steps	0 words
初期化テーブル	81 steps	430 words
EとXのプリセット	174 steps	1793 words

(内マクロ定義 111 steps)

残りのピースのチェック	175 steps	2564 words
PUTOK, BACK, COUNT 等	42 steps	59 words
対称解追放関係	169 steps	1500 words
印刷	53 steps	237 words

と全体としてかなり大きいプログラムであるがステップ数は小さい。

所要時間は印刷とディスクに出して11時間17分30秒ぐらい。
ディスクI/Oの時間は約3~4分と思われる。これをラインプ
リンタに打たせると約30分位かかるようである。解はほぼ10
秒に1個の割合で出てくるようであり、10分位の間隔でみる
とほとんど一樣な速度と言える。

虫取りの都合で文脈チェックの情報を全部ゆにした(すな
わち、失敗したら次のスタイルの最初から調べる。)、時間は
約1.4~1.5倍かかったので、これ自身は苦勞の割には効果が
あがらなかつたらしい。ただ、基本チェックの命令が

BIT R1, the VSP

のときと

BIT BITTB-TB(R0), the VSP

のとき(毎回マス目番地をたよりにビットパターンをインデ
ックス修飾して持ち出す)とで、1時間時間の差が出たの
には驚いた。両者の時間差は2.7 μ secだから、このチェックが
10億回以上実行されていることがわかる。

結局、端から詰めていくことで約10倍、基本チェックで約
2倍、文脈チェックで1.5倍、プログラムのベタ書きで約2
倍速度が向上したと私は予想している。ちなみに、実習生の
人(プログラム未経験)に作ってもらったプログラムは基本
的に磯部氏の手法に近いのであるが一つの解が出るまで4時

間かかった。

7. 展望

当初の予想はなんとか2桁時間達成が目標であったが、意外に記録が伸びた。多分5時間位まではいくと予想されるが、そのためには相当強力な heuristics が要求されるだろう。ここで述べたプログラムはシラミつぶしであるが非常に高速な Tree Traverse を行なっているので、今後の heuristic program との比較基準とすることが出来ると思う。

ビット処理による高速化のねらいは、アーキテクチャに多大に依存するが、なかなか思う通りの計算機はないものである。

謝辞

虫トリに多大な助言をいただいた池野信一室長、奥乃博氏に多大な感謝をします。

解の一部分 (key board 上に出たもの)

注: 印刷の7007"ラダ少くも果に作, た9で, 計算機内でTBの
イマージとこの印刷の間には, 何き相違がある.

IIIII LLLN TTUU	IIIII LLLN TTLY	IIIII LLLN PPPL	IIIII LLLN LPPY
WXFZ2 VFLLN VTFFU	WXUUN VFZ2N VTPPY	WXUUN VFZ2N VPPTV	WXUUN VFZ2N VPPTV
XXXZN VVVVN VTUUU	XXXUN FFFZV VTPPY	XXXUN FFFZV VTTTY	XXXUN FFFZV VTTTY
WXZ2N WMPPP VMMPP	WXUUN WMFZ2 VMMPP	WXUUN WMFZ2 VMMTY	WXUUN WMFZ2 VMMTY

IIIII LZ2PN LLLY	IIIII UUNNL NNNZY	IIIII VVVVP FFFLP	IIIII LLLF TTTL
WXTPN VZTPN VUTUY	WXPPP VUFTL VZ2ZY	WXTTT VNNNP VFLLP	WXMPF VVWME VTZMM
XXXPN Z2TPY WUUUY	XXXPP UFTL VZFYV	XXXTZ NNZ2Z VFZLP	XXXPP VVVVF VT22Z
MXFFN WMTFF VMMFY	MXFTL WMTL VMMTY	MXUTU WUUUU VMWLL	UXUPP UUNNN VNNNZ

IIIII PPPVM TTTFM	IIIII UTUYL UUUFP	IIIII TTLY NLLL	IIIII LLLN LVVVV
WXZ2M VPPVM VTFFF	WXZ2M VTZYL VZ2PP	MXVPU MTVU NNWV	WXUUV MPPPN TTTF
XXXZM NNNV VTNNF	XXXMM TTLY VNNV	XXXFF MTVU MPPPP	XXXUV MPPPN MTFFF
UXU2Z UUUYL WLLL	FXWML FFFYL VFNNN	ZXFFU Z2ZYU MNZPP	ZXUUV Z2ZNV MTZFY

IIIII VVVYL NLLL	IIIII TTZ2 NVVVV	IIIII TTZ2 FVVVN	IIIII Z2TT NVVVV
MXVPU MVVTU NNWV	MXVPU MTVZU NNWV	MXUUU MTUZU FFFVN	MXVPU MZVTU NNWV
XXXFF MTITU MNPP	XXXFF MTZ2U MNPP	XXXPP MTZ2N MFFVN	XXXFF MZ2TU MNPP
ZXFFU Z2ZTU MNZPP	LXFFU LLLLU MNWV	LXPPP LLLLN MYYV	LXFFU LLLLU MNWV

IIIII Z2VPF NVVVV	IIIII Z2TT FVVVN	IIIII VVVVT NVUUU	IIIII TTTF2 LVVVV
MXVTF MZVPP NNWU	MXUUU MZUTU FFFVN	MXFZ2 MFFFT NNUFU	MXPPP MTZ2Z LLLV
XXXTF MZ2PF MNVUF	XXXPP MZ2TN MFFVN	XXXZT MPPPT MNPT	XXXPP MTZNN MNNNV
LXTIT LLLF MNVUU	LXPPP LLLLN MYYV	LXZ2V LLLLV MNWV	FXUUU FFFUV MFFUV