信州大学工学部

学士論文

マスターマインドにおける最適な戦略の再検証

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　15T2071A
氏名　中村 玲音

2019 年 2 月 27 日

目次

1　はじめに

本論文ではマスターマインドという 2 人でプレイする対戦型推理ゲームの最適戦略について論じる．

このゲームは古くから存在し，Bulls and Cows や Hit and Blow，そしてマスターマインドなど様々な名前で呼ばれている．1970 年代前半にイギリスのインヴィクタ社がマスターマインドという商品名で発売し，その後アメリカでハスブロ社から発売されるなど，世界中で発売された．また，ゲーム番組としても楽しまれてきた．1946 年からはアメリカで Twenty Questions，日本ではそれをモデルにした二十の扉というゲーム番組が放送されていた．ただし，これらは単に推理ゲームというだけで，マスターマインドとの類似点は解答者がヒントによって対象を絞り込んでいくという点である．マスターマインドとほぼ同じルールのゲーム番組としては，2011 年から 2014 年までフジテレビで放送されていた NumerOn[1] がある．

マスターマインドは，出題者と解答者と呼ばれる 2 人のプレイヤーによって行われる．まず準備として，出題者が秘密のコードというものを決め，解答者に知られないようにする．解答者は秘密のコードを推測して出題者に提示する．出題者は，解答者の推測が秘密のコードにどれだけ近いかヒントを与える．上の一組の問答を 1 手とし，その手続きを解答者が秘密のコードを当てるまで続ける．解答者にとってコードを当てるまでの手数は小さい方が良い．

解答者の振る舞いをあらかじめ記述したものを本論文では戦略と呼ぶ．戦略を決めると，秘密のコードごとにそれを当てるまでの手数が決まる．戦略の手数に関して様々な研究が行われてきている．1976 年に D. E. Knuth[2] は，最も小さい最悪手数が 5 手であることを明らかにし，それを達成する戦略を一つ与えている．1982 年に E. Neuwirth は，出題者が秘密のコードを等確率で選ぶと仮定したもとで，戦略の平均手数の最小値を求める計算方法を提案している [3]．1993 年に K. Koyama and T. W. Lai[4] は，Neuwirth が提案した方法に高速化の工夫をすることによって平均手数の最小値を実際に求め，それを達成する戦略を一つ与えている．Knuth の論文には最悪 5 手で正答を導き出す戦略が 1 つ記載されている．その具体的な導出方法については述べられていない．本研究では従来研究と異なり，秘密のコードを当てるまでの手数ではなく，秘密のコードが何であるか判明するまでの手数を考える．これは，ゲームの勝敗ではなく，情報理論的な側面への関心からである．秘密のコードが判明すれば少なくともあと 1 手で正答することができるが，その逆は必ずしも成り立たない．つまり，正答するための戦略において，正答したときにすでに秘密のコードが判明していたとは限らない．本研究では，A^* アルゴリズムに基づいて最適な戦略を見つけるプログラムを作成し，最適手数とその戦略について考察する．

2　マスターマインドのルール

この章ではまずマスターマインドのルールを説明する．次に，出題者が解答者に返すヒントについて説明する．ヒントは，戦略の重要な構造を決めている．以上のもとで戦略の最適性を定義し，以降の議論に備える．

2.1　ルール

マスターマインドのルールを説明する．ボードゲームでは 6 種類の色のピンを使うが，本論文では色の種類を 1 から 6 までの数字で表す．色 (数字) の重複を許してピンを 4 つ並べたものをコードと呼ぶ．したがってコードは全部で $6^4 = 1296$ 種類存在する．プレイヤーは出題者と解答者に分かれ以下の手順に従って対戦する．

1. 出題者は解答者に分からないように 6 色のピンの中から重複を許して 4 本選び並べる．この並べた 4 本のピン組み合わせをコードと言うこととし，出題者が選んだコードを秘密のコードと呼ぶ．
2. 解答者は秘密のコードの配置を推測して 4 本のピンを並べ出題者に提示する．推測した 4 本の並びをテストパターンと言うこととする．解答者が推測したコードを単に推測と呼ぶこともある．
3. 出題者は解答者の推測がどの程度秘密のコードに近いかをヒントというもので示す．ヒントの厳密な定義は次の節で与えるが，ボードゲームの言葉で簡単に言うとヒントは次のように決められる．ヒントは赤ピンと白ピンの数によって示される．赤ピンの数は色も位置も合っているピンの数で，白ピンの数は色だけが合っているピンの数である．
4. 示されたヒントが赤ピン 4 本のとき，つまり推測が秘密のコードと一致したとき，ゲームは終了する．このことを秘密のコードを正答すると言うこととする．正答していなければ手順 2 に戻って解答者は次の推測を作る．

2 から 3 までの手順を 1 手と数え，この手順を繰り返した回数を手数と呼ぶこととする．ゲームに勝つために，解答者は秘密のコードを正答するまでの手数を小さくしたいと考える．なぜなら，勝敗を決めるために有利になるからである．実際に勝敗をつける際は，プレイヤー同士で出題者と解答者の役割を交代してゲームを行い，秘密のコードを正答するまでの手数の小さいプレイヤーを勝ちとする．ただし，本研究では勝敗については直接扱わない．

2.2 ヒント

秘密のコード $c_1c_2c_3c_4$ と推測 $q_1q_2q_3q_4$ に対して，両者の近さを表すヒントを $h(c_1c_2c_3c_4$, $q_1q_2q_3q_4)$ と書く．秘密のコードと推測は入れ替えても同じヒントが得られる．ヒントは 2 つの整数 r，w を用いて (r,w) という形式で表される．ゲームをプレイするときは r，w はそれぞれ赤ピンと白ピンの数で示される．この r と w は次のように決まる．

1. $c_j = q_j$ を満たす j の数を r とする．
2. $c_j = i$ を満たす j の数を m_i，$q_j = i$ を満たす j の数を n_i とし，w を

$$w = \sum_{i=1}^{6} \min(m_i, n_i) - r \tag{1}$$

と定義する．例えば，$h(1123, 2413) = h(2413, 1123) = (1, 2)$ である．実際にヒントとしてとり得る値は

$$\begin{aligned}(r,w) =&(0,4),(0,3),(0,2),(0,1),(0,0),(1,3),(1,2),(1,1),(1,0),\\ &(2,2),(2,1),(2,0),(3,0),(4,0)\end{aligned}$$

の 14 通りである．以降，これら 14 個のヒントからなる集合を H と表す．

2.3 戦略の構造

解答者は出題者の提示するヒントを基に次のテストパターンを決定する．この手順が記述されたものを戦略と呼ぶ．ヒントは上記で述べた 14 通りであり戦略は基本的に 14 分木の構造をしている（図 1)．各内部ノードには秘密のコードの候補としてコードの集合が対応している．特にルートノードにはコード全体が対応している．さらに秘密のコードの候補が複数ある内部ノードに対してはテストパターンも記述されており，そのときそのノードは高々 14 個の子ノードを持ち，子ノードへの枝には 14 通りのヒントのうちのいずれかがラベルとして重複なく振られている．子ノードに対応しているコードの集合は，親ノードがもつコードのうち，テストパターンに対するヒントが枝のラベルと一致するもの全体である．

少し分かりづらいので小さい例を使って説明する．今着目している内部ノードには秘密のコードの候補として

$$C = \{2334, 2343, 2345, 2346, 2354, 2364, 2444, 2644\} \tag{2}$$

が対応しているとする．解答者はこの C の中から正しいコードを絞り込んでいく．図 2 がその戦略の例である．ヒント (4,0) に対応した子ノードに移ることによって秘密のコードを正

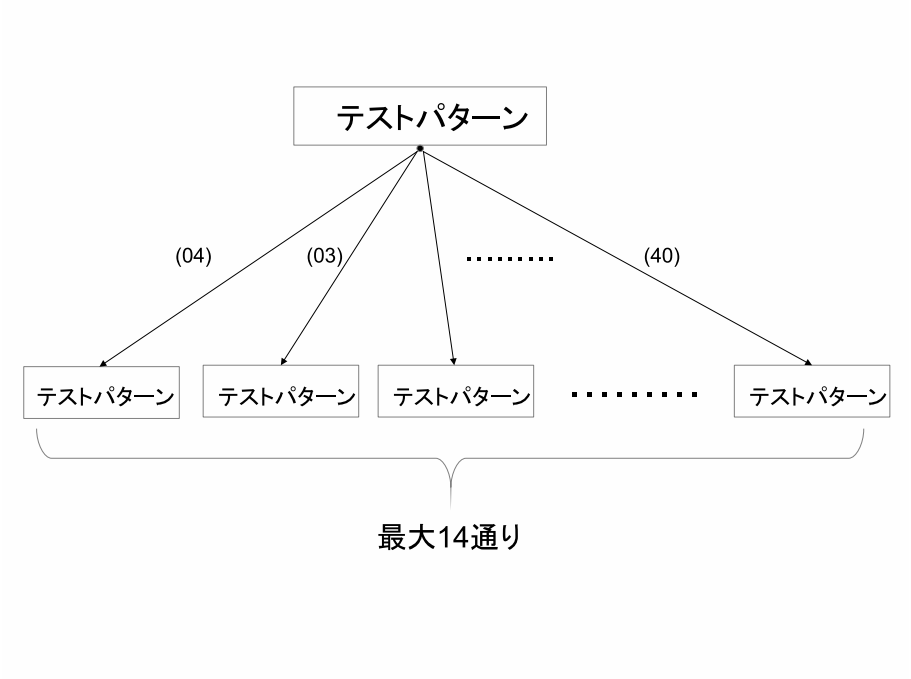

図 1　14 分木の構造図

答したことになる．たとえば出題者が式 (2) の C の中から秘密のコードとして 2345 を選んだ ときに，解答者が図 2 の戦略を使用すると次のような手順で正答に至る．解答者は，最初の推測として 1335 を出題者に提示する．出題者はヒント $h(2345, 1335) = (2, 0)$ を返してくる．すると解答者は (2,0) に対応した子ノードを参照し，次の推測 2334 を提示する．ヒントは (2,1) が返ってくるので，解答者は次に推測 2345 を提示する．出題者からヒント (4,0) が返され，解答者は正答したことが分かる．同様にして，図 2 の戦略を使用することにより，式 (2) の C の中のどれが選ばれても正答できることを実際に確認することができる．

3　判別するまでの手数と従来の研究

従来の研究では解答者が出題者の作る秘密のコードを当てるまでの手数について考えている．つまり完全に正答するまで何手かかるのかということに注目している．しかし，本論文では完全に正答できるまででではなく出題者のコードがどのようなコードなのか判別できるまでを考えている．前述したがこれはゲームの勝敗ではなく，情報理論的な側面への関心があるからである．では「判別できるまで」とはどういうことか説明する．例えば先ほどの図 2 で，1335 から 1 段下がった 2334 を見てみる．2334 と解答者が推測したとき，出題者からのヒントが

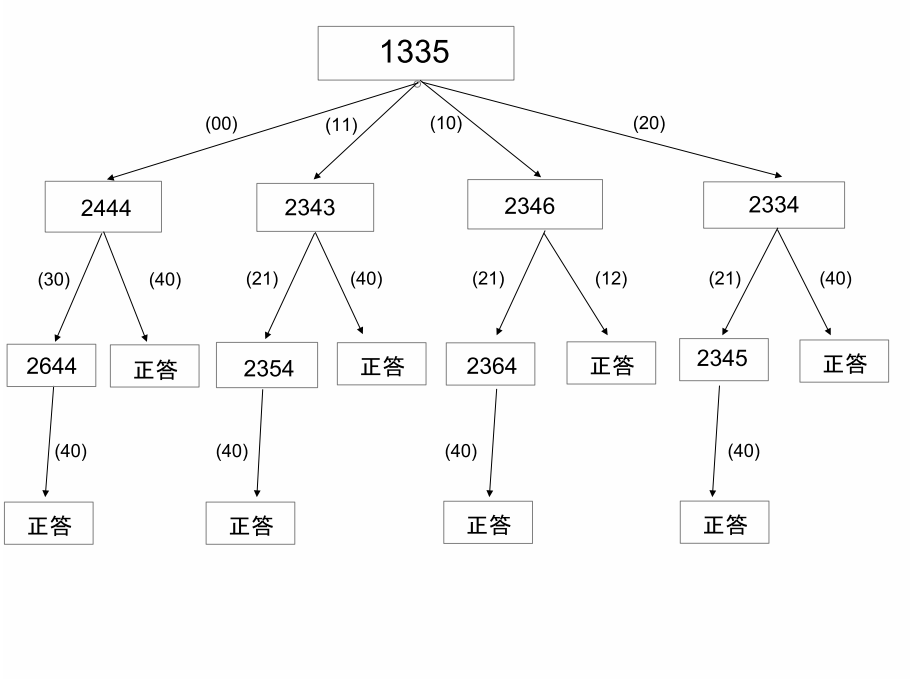

図 2　C に対する戦略の例

(4,0) の場合はもちろん 2334 が秘密のコードであることが分かるが，ヒントが (2,1) の場合も秘密のコードが 2345 であると分かる．つまり 2334 という推測に対するヒントを聞いた段階で出題者の秘密のコードを「判別」できるのである．しかし判別ができるだけで出題者に「当たりです」と言わせるにはゲームのルール上もう一度 2345 と推測しなければ正答したとは言えないのである．つまり判別さえできれば正答が何かわかるのである．本論文ではその判別できるまでについて考えている．

従来の研究において分かっていることを述べる．まず，Knuth の論文 [2] で正答するまでの最悪手数の最小値が 5 手であることが分かっており，その戦略についても記載されている．つまり正答が何か判別できるまでの最悪手数の最小値は 4 となる．次に Koyama らの論文 [4] では正答するまでの平均手数について述べられており，最悪手数については考慮せずに平均だけについて考えると，最適な戦略の平均手数は 4.341 手であることが分かっている．Knuth の論文では実際にどのようにして戦略を導き出したのかということまでは書かれていない．そこで本研究ではプログラムを用いて独自に最悪 5 手で出題者のコードを正答できる戦略を求めた．Knuth の論文では 1122 から始まる戦略が挙げられていたが他のコードから始まる戦略でも最悪手数が 5 手のものがあることが分かったので，それらの戦略の違いを検証した．また，Koyama らの論文で最小平均手数について述べられていたが，これについても検証を行った．

4　初めの推測

出題者が作ることのできるコードは 6^4 で 1296 通り存在する。解答者の推測に対して出題者が解答者に返すことのできるヒントは全部で 14 通りである．したがって最初の推測に対するヒントによって秘密のコードは高々 14 通りに分割される．例えば出題者が秘密のコードとして 4443 を選んだとする．最初の状態では解答者には何の情報もないので，解答者としては秘密のコードの候補は 1296 個存在する。ここで解答者が 1112 と推測すると出題者はヒントとして (0,0) を返してくる．このように 1112 と推測すると (0,0) とヒントが返ってくるコードは 1296 個の中に 256 個存在する．1112 と推測して (0,1) と返ってくるものは 308 個，(0,2) は 61 個等々となり，(4,0) と返ってくるものはただ 1 つ存在する．これは 1296 通りの可能性があった秘密のコードが (0,4)〜(4,0) の各ノードに振り分けられることで絞りこまれていることを意味し，(0,0) ではあと 256 個のコードを区別していけばいいということがわかる．最初の推測の選び方を変えれば秘密のコードの分類は変わる．したがって戦略を作る際，推測の候補は 1296 通り存在する．しかし，最初の推測として 1111 と 2222 を区別する必要がないのは容易に分かる．なぜなら 1111 という推測に対する分割と 2222 という推測に対する分割は，数字の 1 と 2 を入れ替えることで全く同じになるからである．すると 1111 と 2222 は実質的に同じ分割をもたらすため同じ役割を果たすことになるから 1111 という推測を候補に入れておけば 2222 という推測を候補に入れておく必要はないし，同様に 3333, 4444, 5555, 6666 も候補に入れておく必要はない．一般に言えば，実質的に同じ分割をもたらす推測がいくつか存在するならば，それらの推測のうち 1 つを残して他は候補から外すことができる．言い換えると，(0,0)〜(4,0) までの分かれる分布が同じ推測はひとつにまとめることができる．

すべてのテストパターンを考え，同じ分布はまとめたところ，最初の推測は表 1 のように 5 通りを考えれば十分であることがわかった．この表の見方としてまず，最も左の列に代表的なテストパターンが並んでいる。「代表的なテストパターン」というのは，(0,4)〜(4,0) までの分布が同じ分布，同じ分かれ方をするものをすでに出てきた分布と同じ分布としてまとめたものである．そして，一番上の列には 14 通りのヒントが並んでいる．例えば 2 行目に注目すると，

表 1　初めの推測

テストパターン	ヒント													
	04	03	02	01	00	13	12	11	10	22	21	20	30	40
1111	0	0	0	0	625	0	0	0	500	0	0	150	20	1
1112	0	0	61	308	256	0	27	156	317	3	32	123	20	1
1122	1	16	96	256	256	0	36	208	256	4	32	114	20	1
1123	2	44	222	276	81	4	84	230	182	5	40	105	20	1
1234	9	136	312	152	16	8	132	252	108	6	48	96	20	1

1111 という推測のコードに対して (4,0) というヒントを返してくるのは 15 列目を見ると「1」となっている．確かに 1111 と聞いて (4,0) と返してくる正答のコードは 1111 の 1 通りしかない．このように推測で選択肢を絞りこむ．最初の推測は上記の 5 通りのどれかを使い推測すればよいと考えられる．正答を見つけていく過程でこの最初の推測と同じようにすべてのテストパターンを調べる必要はなく，無駄な推測はせずにまとめ，効率的に絞っていくことができる．

少ない手数で秘密のコードが何か判別するためには，推測をしたとき各ヒントにテストパターンがどのように分かれるかということは非常に重要である．もし 14 通りすべてに等しくコードを分けることのできる推測があるとしたら，その推測がより細かくテストパターンを区別でき，選択肢を絞り込めるわけで，手数を小さくするにはそれが最も適した推測なのである．つまり手数を少なく正答を判別するためには解答者がその都度より多くのヒントにテストパターンが分けることができ正答を絞りやすくするような「最適な推測」を選択していくことが大切である．

5　A*アルゴリズム

マスターマインドにおける最悪手数の最小値を見つける戦略を考えるうえで A*アルゴリズムという探索アルゴリズムをプログラムの中で使用するためこれについて説明する．

5.1　A*アルゴリズムの概要

A*アルゴリズムは、 Peter, Nils, and Bertram [7] によって提案されたグラフ探索アルゴリズムの一つである．「開始ノードから現在位置に至るまでの正確な距離」と「現在位置からゴールまでの推定距離」の 2 つの距離を用いてゴールまでの最短経路を解く．推定距離を実際の値と同じかそれより小さくすることで効率よく探索できる．グラフ探索においてゴールまでの最短経路を効率的に見つけることのできるアルゴリズムである．

5.2　A*アルゴリズム

迷路の中でスタート地点からゴール地点まで歩くことを考える．早くゴールへたどり着くために A*探索アルゴリズムを用いて最短経路を計算する．

スタート地点からゴール地点までの距離の一番小さな経路が最短経路といえる．つまり、距離が小さくなるような経路を探していくと最短経路が見つかるのである．例えば今スタート地点からゴール地点までの最短経路があるとし，その最短経路上にある n という地点にいるとする．このとき、最短経路は当然ながら「スタート地点から地点 n までの最短経路」＋「地点 n からゴール地点までの最短経路」である．スタート地点から地点 n までの最短距離を $g(n)$，

地点 n からゴール地点までの最短距離を $h(n)$ と表すと，地点 n を通る最短距離 $f(n)$ は

$$f*(n) = g(n) + h(n) \tag{3}$$

のように表現できる．この場合はもともと最短経路を知っていたので，スタート地点から地点 n までの最短距離 $g(n)$，地点 n からゴール地点までの最短距離 $h(n)$ のどちらも分かっていた．よって地点 n を通る経路の最小距離は $f(n)$ ということが計算できた．次に全く初めての迷路である地点 n に置かれた場合を考える．この場合スタート地点から地点 n までの最短距離は $g(n)$ だと分かる．しかし，$h(n)$ の値は正確にはわからず推定値を使って考える．「地点 n からゴール地点までの推定最短経路」を $h^*(n)$ と表すと，地点 n を通るある最短経路の推定距離 $f^*(n)$ は，

$$f^*(n) = g(n) + h^*(n) \tag{4}$$

のように表現できる．その時点で知っている $g(n)$ と推測した $h^*(n)$ をもとに $f^*(n)$ を計算し，その値が一番小さくなる経路を選んでいくことで最短経路を求めることができる．しかし h^* があまりに見当違いな値を返すと f^* も見当違いになってしまい，f^* の値を比較しながら進んでいく手法が崩壊してしまう．そこで h^* をもっともらしい値を返す関数にすることを考える．もっともらしい推定値を与えてくれる h^* はヒューリスティック関数と呼ばれる．この場合，$h^*(n)$ の範囲を考える．まず $h^*(n)$ が最も小さいのはゴールした時であるから $h^*(n)$ が 0 以上の範囲．そして $h^*(n)$ は「地点 n からゴールまでの実際の最短経路」である $h(n)$ を超える値は取らないのでその $h(n)$ 以下の範囲．この二つを満たすような範囲を設定すると探索効率が良くなる．このような h^* を採用したときに最初に見つかった経路が最短経路であることが保証されている．もっともらしい $f^*(n)$ を求め，その $f^*(n)$ の値が一番小さくなる経路を選んで探索を進めるわけである．このようにして最短経路を求めるアルゴリズムを A*アルゴリズムという．A*アルゴリズムでは，地点 n からゴール地点までの推定最短経路を本当の最短経路以下に見積もって代入することがポイントとなる．これによって，なるべくゴールに近付く方向から探索するようになる．また，スタート地点から現在地までの経路はその都度修正していく．探索の途中で行き止まりになったり，推定距離が高くなる方向にしか進めない場面に遭遇する場合もあり，そのときは次に小さい推定距離をもつ経路を選択し，探索を続ける．ゴール地点までたどり着ける経路を見つけるとプログラムは終了する．A*アルゴリズムはこのように探索を進めることでスタート地点からゴール地点までの最短経路を導き出す探索アルゴリズムである．

6　A*アルゴリズムを応用したプログラム構造

Knuth の結果を再検証するにあたって前述した A*アルゴリズムに基づきプログラムを作成した．プログラム構造について図 3，4 を用いて説明する．

6.1　プログラム構造　最悪手数

構造体を用いて考える．まずは最悪手数の探索方法について考える．図 3 の (a) を上から見ていくと，まず最悪手数の見積もりの最大値について，これはまさに判別できるまでの最悪手数が何手必要か考えるところで，探索を進めるうちに書き換えられていくものである．最初は非常に楽観的な見積もりの値をとる．何も探索をしていない状態では $\log_{14} 1296 = 2.7158$ とする．現実的なマスターマインドのゲームにおいて 14 通りすべてに振り分けられる分布は存在しないのでこの 2.7158 はその 14 通りすべてに等しく振り分けられた場合の手数を考えているおり，非常に楽観的な推測であるといえる．プログラムの中では探索を進めていく中で最悪手数の最小値の見積もりをより正確なものに近づけていく．2 段目はその時点で解答者が選択できるテストパターンの数を表示している．この数は 4 章で述べた通り，代表的なテストパターンの数である．3 段目は 1 段下の表がリンクされている．

初めの推測を例により詳しく見ていく．図 4 を見てほしい．まず図 4 の (a) であるが，これは書き換えられる前の状態のものである．各行の 3 列目以降は推測に対するヒントを省略したものが並んでいる．本来は (0,4)〜(4,0) まで 14 個のヒントが並んでいる．1122 の 2 行目に注目すると各ヒントに対応して行が 3 分割してあり，それぞれ値が表示されている．この 3 分割の部分が先に述べた図 3(a) のような構造体になっている．このプログラムでは 3 分割の 1 段目が表す見積もりの値の大きいところから探索を進めていくこととし，1122 では (0,0) のヒントに対応する $\log_{14} 256$ が見積もりの最大値となる．その後 1111〜1234 までの最大値を比べると 1122 の見積もりが最も小さい．最悪手数がより小さくなるものを見つけたいので見積もりの最大値の小さい 1122 の (0,0) をより深く探索していく．この部分を探索するため一つ下のリンクを見ると図 4(b) のような表が出てくる．最初の推測と同様にして探索を進める．3345 では (1,1) に対応する最悪手数の見積もりが (0,4)〜(4,0) の中で最も大きい．1111〜3456 の各最悪手数の見積もりの中で最も小さいのは，この 3345 の 1.45 である．一つ下の表を探索したので上の表に戻り書き換えを行う．それが次の図 4(c) である．最悪手数の見積もりを書き換えるのだが，そのままの値を代入するのではなく，一つ下の表にいったので手数が 1 回多くなっているので 1 回プラスした $\log_{14} 46 + 1 = \log_{14}(46 \times 14)$ の値を分割した三段の一段目を書き換える．そして前と同様に最悪手数の見積もりの最大値を探す．1111〜1234 の最大値で最小のものを見つけ，そこをさらに深く調べる．このようにして徐々に探索をして

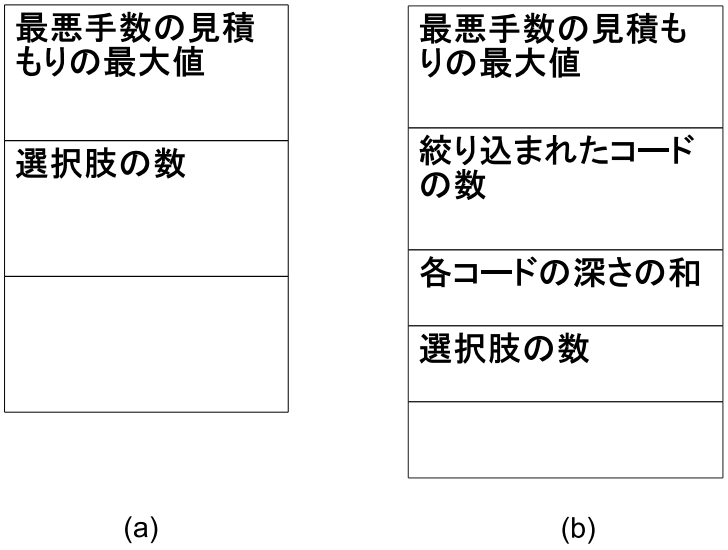

図 3　プログラムの基本構造

書き換えてを繰り返して進めていく．ヒントに対応するノードの数が 1 もしくは 0 になったらその部分はそれ以上掘り下げない．この場合戦略の末端が確定したことになる．最初に見つかった戦略は最適であり，ほかの選択肢は調べる必要がなくなる．ここで選択肢が一つに絞り込まれる．最悪手数を求めるプログラムを付録にプログラム 2 として載せた．このプログラムは 1111〜1234 のそれぞれのパターンの秘密のコードが何か判別できるまでの最悪手数を出すことができる．

6.2　プログラム構造　平均

これまでは最悪手数を最小化する戦略を考えてきたが最悪手数の最小値が 5 手のものはいくつか存在することが分かり，より良い戦略を考えていくうえで最悪手数以外の戦略に興味が出た．そのためここでは別の角度からそれぞれの戦略を考える．その基準としてそれぞれの平均

	最悪手数の見積もりの最大値	04	03	…	00	… .	40
1111	2.43						
1112	2.18						
1122	2.10	$\log_{14}1$	$\log_{14}16$		$\log_{14}256$		$\log_{14}1$
			34		12		
1123	2.13						
1234	2.17						

(a)

	最悪手数の見積もりの最大値	04	03	….	11	….	40
1111	2.10						
….	….						
3345	1.45	$\log_{14}4$	$\log_{14}38$	…	$\log_{14}46$	….	$\log_{14}1$
		21	181		205		
3456	1.55						

(b)

	最悪手数の見積もりの最大値	04	03	…	00	… .	40
1111	2.43						
1112	2.18						
1122	2.45	$\log_{14}1$	$\log_{14}16$		$\log_{14}46*14$		$\log_{14}1$
			34		12		
1123	2.13						
1234	2.17						

(c)

図 4　プログラム構造の例

値を求め，比べてみることとした．探索のプログラム構造としては最悪手数が同じならば手数の平均値が小さいほうを良い戦略とした．基本的な構造は図 3 の (b) で示している．平均は各コードの深さの和をコードの数で割ったものとなる．そのため (b) の 2 行目で絞り込まれたコードの数を記憶するための機能を追加している．これは書き換えられることはなくその時点でのコードの数を表示するものである．また，3 行目では各コードの深さの和を記憶しておける機能を追加している．基本的な動きは最悪手数を求めるときと同じで再帰的な掘り下げを繰り返して答えを導き出す．

6.3 プログラムによる検証結果

再検証を行うにあたりいくつかのプログラムを作成した．まず，出題者の秘密のコードを推測とヒントを打ち込んでいくことで当てることのできるプログラムを作成した．このプログラムでは初めに解答者は 4 章の表 1 にある「初めの推測」の中から自由に一つ目の推測をすることができ，その推測とそれに対するヒントをプログラムに打ち込む．するとプログラムが動作しその推測によってテストパターンが絞り込まれ，その絞り込まれたテストパターンが一覧となって出力される．その中から解答者は自由に推測し，その推測とそれに対するヒントを書き込むと同様に絞り込まれたテストパターンが出力される．このようにして正答を導き出すプログラムとなっている．このプログラムは「初めの推測」を考える際に使用した．このプログラムを付録 B.1 に掲載する．

次に，1111〜1234 の中のどれかが最悪手数を導き出したら終了するプログラムを作成した．この結果 1112 を初めの推測とした戦略を用いたとき，正答が何か判別できるまでの最悪手数が 4 手以内であることが分かった．つまり正答するまでの最悪手数が 5 手以内であることが分かり，実際に最悪 5 手で正答できる戦略が存在することが確認された．ただプログラムの中で 1112 が最も早く見つかっただけで他のものも 5 手で正答できる可能性があったため，1111〜1234 のそれぞれの戦略で最悪手数がいくつになるのかということを明らかにするためにプログラムを作成した．その結果最初の推測の 5 パターン (1111,1112,1122,1123,1124) のうち 1112,1122,1123,1234 の 4 つの場合で最悪 5 手で正答できることが分かった．1111 に関してはメモリ不足のためプログラムが途中で終了してしまった．[*1]

次に，それぞれの平均値を求めた．4 つの平均値は以下の表 3 にまとめている．この表 3 から 1123 が最も平均値が低く最も最適な戦略といえる．Knuth の最適戦略は，正答までの最悪手数を最小化したものであるので単純に比較はできないが，最初のテストパターンは 1122 であり平均手数は 4.478 である．また，平均値について述べている Koyama らの論文で最悪手数は考えず単に平均が最も低い戦略として挙げられているものがあるが，その最初のテストパ

[*1] 1112,1122,1123,1234 を調べるのに必要だったメモリはそれぞれ 1.32GB, 1.04GB, 7.94GB ,5.12GB であった．1111 については 28.4GB を使用したところでプログラムが終了した．

表 2　各コードの平均値

コード	平均値
1112	3.692130
1122	3.625000
1123	3.564815
1234	3.594136

ターンは 1123 で平均値は 4.341 手である．最悪手数を考慮した場合もしない場合も 1123 が平均手数を下げる推測であることがわかる．付録 B.2 に最悪手数と平均を出すプログラムが掲載する．

7　まとめ

本研究では，マスターマインドというゲームに関して，秘密のコードが何か判別できるまでの手数が最悪何手必要なのかということに関して研究を行った．プログラムを用いて秘密のコードが何か判別できるまで最悪 4 手，つまり完全に正答するまでの最悪手数が 5 手になる戦略を複数見つけることができた．複数存在する戦略の中で平均値が最も少ないものがより良い戦略とし，1123 から始まる戦略が最も平均値が低く，最悪手数が小さくなる戦略であることが分かった．

謝辞

本研究に当たり，細かく指導してくださった指導教員の西新幹彦准教授に感謝の意を表する．

参考文献

[1]「ヌメロン-フジテレビ」， https:\\www.fujitv.co.jp\b_hp\numer0n\，2019 年 1 月閲覧

[2] D. E. Knuth,“The Computer as Master Mind”, J. Recreational Mathematics, Vol. 9(1), pp.1-6, 1976-77.

[3] E. Neuwirth, “Some Strategies for Mastermind”, Zeitschrift fur Operations Research, 26, pp. 257-278,1982

[4] K. Koyama and T. W. Lai, “An Optimal Master Mind Strategy”, Journal of Recreational Mathematics, pp.251-256, 1993.

[5] 荒井航，「マスターマインドにおける最悪手数を最小化する戦略とそれに対する堅牢な出

題」，信州大学工学部卒業論文（指導教員：西新幹彦），2014 年 3 月.

[6] 荒井航，「マスターマインドにおける最適な戦略の数え上げに向けて」，信州大学大学院理工学研究科修士論文（指導教員：西新幹彦），2015 年 3 月

[7] Peter E. Hart and Nils J. Nilsson and Bertram Raphael , “A Formal Basis for the Heuristic Determination of Minimal Cost Paths”, IEEE Transactions on Systems Science and Cybernetics 4, pp100-107, 1968

[8] 「(A-star：エースター)探索アルゴリズムの原理」, https://piyajk.com/archives/162, 2019 年 1 月閲覧

付録 A　最適な戦略

1123 を最初のテストパターンとした戦略である．戦略の使用方法を説明する．解答者はまず一行目を見る．一行目は以下のようになっている．

[000] 1123:[001],[002],[012],[065],[119];[140],[141],[162],[212];[253],[255],[265];[290],1123.

戦略の各行は以下のような構造となっている．

[ラベル] テストパターン:

(0,4),(0,3),(0,2),(0,1),(0,0);(1,3),(1,2),(1,1),(1,0);(2,2),(2,1),(2,0);(3,0);(4,0)

コロン以降にはヒントが記載されており，解答者が次に行動するべきことが，ヒントごとに分けられて記されている．

最初にラベル [000] の行に記載されている 1123 のテストパターンを解答する．判定により得られたヒントの場所には以下のことが記載されているので，その指示に従う． 例えば 1123 と解答し，ヒントが (0,4) となった場合，戦略を見ると [001] と明記されている．解答者は，次に [001] の行に移り，そこに明記されているテストパターンを解答する． このように，[xxx] の場合，xxx と同じ数字のラベルの行に記載されているテストパターンを解答する．

次にラベル [001] の行にを見てみる．ヒントが (3,0) の場合，戦略を見ると 2311 と明記されている．解答者は，次に 2311 と解答すると正答となる． ヒントが (0,0) の場合，戦略を見ると , と明記されている．この場合，次に解答するテストパ ターンは存在しないので，出題者の提示したヒントが間違っている，もしくは解答者が参照する場所を間違えている．

[000] 1123:[001],[002],[015],[064],[132];[153],[154],[178],[236];[277],[280],[294];[324],1123.
[001] 1211:,,,,;,,,;,2311,;3211,.
[002] 1111:,,,,;,,,[003];,,[011];,.
[003] 2345:,[004],[005],,;,[006],[007],;,[008],[009];[010],.
[004] 1235:3512,3412,,,;3251,3214,,;5231,4231,;,.
[005] 1261:,3612,,,;,3216,3212,;,6231,3231;3261,.
[006] 1412:,3241,2531,,;,2431,3215,;,4312,5312;,.
[007] 1116:,,2631,2231,;,,6312,3312;,,;,.
[008] 1112:,,2351,,;,,2314,;,,;,.
[009] 1116:,,2361,2331,;,,,2312;,,2316;,.
[010] 1112:,,2341,,;,,2315,;,,;,.
[011] 1214:,,,,;2411,[012],[013],;4211,[014],;,.
[012] 1345:,3411,2511,2611,;,4311,,;,,;,.
[013] 3513:,,,,;,5311,6311,;,3311,3611;3511,.
[014] 2115:,,,,;5211,6211,,;,2211,;,.
[015] 1234:,[016],[025],[036],,;,[037],[045],,;[053],[057];[061],.
[016] 2425:,[017],[018],[019],,;,[020],[021],[022];,[023],[024];2415,.
[017] 1541:,,4352,,;,4512,5342,;,,3542;,.
[018] 3246:4362,4612,4351,,;6342,4342,5341,;3642,3342,3541;,.
[019] 3641:4316,,,,;4361,,,;6341,4341,;3341,3641.
[020] 1115:,,2541,3452,2342;,,5412,;,,;,.
[021] 1462:2641,2346,4315,,;,,3451,;6412,4412,3442;3462,.

[022] 1116:,,3461,3441,;,,,;,,3416;,.
[023] 1112:,,2451,2345,;,,,;,,2412;,.
[024] 1116:,,2461,2441,;,,,3415;,,2416;,.
[025] 5325:3552,[026],[027],[028],[029];,[030],[031],[032];[033],[034],[035];5315,.
[026] 1162:,,2551,3551,;,,3652,;,,3562;,.
[027] 2251:,6512,3516,3662,;2512,2516,3561,;,2561,3651;2651,.
[028] 2126:,6612,3661,4511,;2612,2661,3616,;,2616,;,.
[029] 1414:,,,,;,4611,,;4411,6411,;,.
[030] 2365:,,5512,,;6352,3352,3515,;2356,2352,2515;,.
[031] 2626:,6362,3362,6351,3351;,2362,5612,3615;,2366,2615;,.
[032] 1336:,,,5411,;3361,6361,,;3316,6316,;,.
[033] 1112:,,,2355,;,,,5352;,,;,.
[034] 1115:,,5351,5362,;,,,2365;,,;,.
[035] 1116:,,5361,,;,,6315,3315;,,5316;,.
[036] 1515:,,,,;,5611,6611,;5511,6511,;,.
[037] 2451:4215,[038],[039],,;[040],[041],[042],;4251,[043],[044];,.
[038] 1536:,5314,3245,4212,;,3514,4216,;,,4532;,.
[039] 4622:,3246,6314,3314,;,3242,3614,4314;,,4332;4632,.
[040] 1112:,,5241,,;,,2514,;,,;,.
[041] 1261:,2614,,5432,;,,,4531;,6241,4241;4261,.
[042] 6442:,2364,4631,4331,;,2344,3414,;,4432,3432;6432,.
[043] 1144:,,5431,2435,;,2414,2241,2354;,,;,.
[044] 3136:,,,,;,6431,4431,2432;,3431,2436;,.
[045] 1552:5215,[046],[047],[048],;5251,[049],[050],[051];,5532,[052];,.
[046] 1116:,,5261,,;,,6215,2215;,,5216;,.
[047] 2612:,6261,3265,5331,;2261,6216,5631,2335;2216,,2635;,.
[048] 1333:,,,3266,;,3631,6631,2636;3331,6331,2336;,.
[049] 1561:,,5212,3255,;,6251,2251,2535;,,5531;,.
[050] 2533:,3256,,6212,;5332,5632,,2212;,3531,6531;2536,.
[051] 2333:,,3262,,;,3632,6632,;3332,6332,2632;2332,.
[052] 2133:,,3252,,;,3532,6532,;,,2532;,.
[053] 1115:,,,[054],[055];,,5214,[056];,,;,.
[054] 1131:,,,3254,;,,,2534;,,;,.
[055] 2633:,3264,3244,,;,4236,4232,;,2334,2434;2634,.
[056] 2116:,,,4235,;,6214,4214,;,,2214;,.
[057] 1155:,,,[058],[059];,,5235,[060];,,;,.
[058] 1112:,,,5236,;,,,5232;,,;,.
[059] 2166:,,6232,3232,;,6236,3236,2232;,,2236;,.
[060] 2116:,,6235,3235,;,,,2235;,,;,.
[061] 1156:,,,[062],[063];,,,;,,;,.
[062] 1115:,,,5234,6234;,,,;,,;,.
[063] 2133:,,,,;,3234,4234,;,,2234;,.
[064] 1244:,[065],[068],[075],[085];,[094],[100],[109];,[121],[124];[130],.
[065] 1111:,,,,[066];,,,[067];,,;,.
[066] 1115:,,,4452,4462;,,,;,,;,.
[067] 1115:,,4451,4461,;,,,4416;,,4415;,.
[068] 1155:,,[069],[070],[071];,[072],[073],[074];,,2455;,.
[069] 1416:,4561,,,;,5461,,;,4516,;5416,.
[070] 1416:,4661,4562,,;,6461,5462,;,4616,;6416,.
[071] 1622:,,2466,4436,;,2462,6462,;,,4662;,.

[072] 1415:,4551,,,;,5451,,;,4515,;5415,.
[073] 1415:,4651,4552,,;,6451,5452,;,4615,;6415,.
[074] 1652:,2465,,4435,;,2456,,;,6452,2452;4652,.
[075] 5562:2655,2656,[076],[077],;2556,[078],[079],[080];[081],[082],[083];[084],5562.
[076] 3456:4635,,6615,,;6435,4355,6651,;4356,,;3455,3456.
[077] 3435:,,,,6616;,4336,4636,;4335,,6436;3436,3435.
[078] 6151:,5615,2665,,;,6515,2652,2555;,5651,6652;6551,.
[079] 4536:3465,,,,;4365,5435,5616,2666;5436,,6516;4535,4536.
[080] 1314:,,3466,6661,;,,4366,;,,;,.
[081] 1515:,,5652,,;,,6552,;,,2565;,.
[082] 1116:,,,,2552;,,,2566;,,5516;,.
[083] 2515:,,5661,6662,;,5551,6561,2662;,,;5515,.
[084] 1622:,,5561,,;,2562,6562,5552;,,5662;,.
[085] 1355:,5536,[086],[087],;,[088],[089],[090];,[091],[092];[093],.
[086] 1566:,,,,;,5636,,;,6536,3536;3566,.
[087] 1133:,,,3666,;,,3636,6636;,,;,.
[088] 3516:,5635,,,;,6535,5535,;,3565,3535;3556,.
[089] 3356:,6635,,,;3635,3665,,;5336,5366,;3656,.
[090] 1333:,,,,;,,3366,6366;,3336,6336;,.
[091] 1156:,,5365,5335,;,,3655,3555;,,5356;,.
[092] 1665:,6356,3356,,;,,6335,3335;,6365,3365;,.
[093] 3116:,,6355,5355,;,,,3355;,,;,.
[094] 1115:,,[095],[096],[097];,,[098],[099];,,;,.
[095] 1411:,,,,;,,4541,;,,5441;,.
[096] 1464:4641,,4542,,;6441,4441,5442,;,,2454;,.
[097] 2416:,4642,,,;,6442,4442,;,2464,2442;2446,.
[098] 1411:,,,,;,,4514,;,,5414;,.
[099] 1416:,,,,;,4614,,2445;,6414,4414;,.
[100] 1455:[101],[102],[103],[104],;,[105],[106],[107];,4255,[108];,.
[101] 1112:,,,5541,;,,,5514;,,;,.
[102] 1514:,5641,,,;,6541,5542,;,5614,;6514,.
[103] 2516:,5642,6641,,;,6542,6614,4534;,2564,2542;2546,.
[104] 2264:,6642,4346,,;2642,2646,4634,4334;4262,4266,4364;2664,.
[105] 1114:,,,2545,;,,,2554;,,;,.
[106] 2654:4265,,4345,,;4256,4252,5434,;2645,,4354;,2654.
[107] 3116:,,6434,4434,;,,3464,3434;,,3446;,.
[108] 1114:,,,3445,;,,,3454;,,;,.
[109] 3455:5534,[110],[111],[112],[113];[114],[115],[116],[117];[118],[119],[120];,.
[110] 1634:,5346,,,;,5364,,;,6534,5334;5634,.
[111] 1343:,,6634,,;,6334,6364,;,,6346;,.
[112] 1112:,,,5266,;,,,5262;,,;,.
[113] 1222:,,,,;,,2266,6266;,2262,6262;,.
[114] 1114:,,,5345,;,,,5354;,,;,.
[115] 3534:,6345,,,;,6354,,;,3546,;3564,3534.
[116] 3346:,,,5265,5252;3634,3664,,5256;3364,3334,;3646,3346.
[117] 1656:,6265,2265,,;,,6252,2252;,6256,2256;,.
[118] 1114:,,,3545,;,,,3554;,,;,.
[119] 1146:,,3654,3354,;,,3645,3345;,,;,.
[120] 2116:,,6255,5255,;,,,2255;,,;,.
[121] 1114:,,,[122],;,,,[123];,,;,.

[122] 1152:,,4245,4246,;,,,4242;,,;,.
[123] 1445:,4254,4264,,;,2544,2644,;,,2444;,.
[124] 1152:,,[125],[126],[127];,,[128],[129];,,;,.
[125] 2146:,5264,,,;,6245,5245,;,5246,2245;,.
[126] 2326:,,6264,3544,;,2264,6246,5344;,2246,;,.
[127] 3416:,6344,4344,,;,3644,3344,;,,3444;,.
[128] 2155:,,5242,,;,5254,6254,;,,2254;,.
[129] 1116:,,,6242,2242;,,,;,,;,.
[130] 1115:,,,5244,[131];,,,;,,;,.
[131] 2116:,,6244,4244,;,,,2244;,,;,.
[132] 1114:,,,[133],[142];,,,[147];,,;,.
[133] 1455:,5546,[134],[135],;,[136],[137],[138];,[139],[140];[141],.
[134] 1566:,,,,;,5646,,;,6546,4546;4566,.
[135] 1144:,,,4666,;,,4646,6646;,,;,.
[136] 4516:,5645,,,;,6545,5545,;,4565,4545;4556,.
[137] 4456:,6645,,,;4645,4665,,;5446,5466,;4656,.
[138] 1444:,,,,;,,4466,6466;,4446,6446;,.
[139] 1156:,,5465,5445,;,,4655,4555;,,5456;,.
[140] 1665:,6456,4456,,;,,6445,4445;,6465,4465;,.
[141] 4116:,,6455,5455,;,,,4455;,,;,.
[142] 1155:,,5566,[143],6666;,,[144],[145];,,[146];,.
[143] 1511:,,,5666,;,,,6566;,,;,.
[144] 1656:,6565,5565,,;,5665,,;,6556,5556;5656,.
[145] 1115:,,,6656,;,,,6665;,,;,.
[146] 1566:,6655,5655,,;,,6555,5555;,,;,.
[147] 1451:,,[148],[149],;,,[150],[151];,,[152];,.
[148] 4566:,5644,,,;5664,6544,5544,;6564,5564,4544;4564,.
[149] 1444:,,,,;,,4664,6664;,4644,6644;,.
[150] 4656:,5464,5444,,;,6554,5554,;6654,5654,4554;4654,.
[151] 1166:,,,6444,4444;,,6464,4464;,,;,.
[152] 4116:,,6454,5454,;,,,4454;,,;,.
[153] 1231:3112,,,,;1312,,,;2131,,;,1231.
[154] 1112:,,[155],,;,[158],[159],;[166],[167],[168];[175],.
[155] 1145:,,[156],[157],;,,,;,,;,.
[156] 1314:,3421,3521,,;,4321,5321,;,,;,.
[157] 2316:3621,3221,,,;6321,3321,,;,2321,;,.
[158] 1245:,2151,2161,,;,2141,,;,1251,1261;1241,.
[159] 1224:,2413,[160],[161],;,[162],[163],[164];,,[165];1234,.
[160] 2335:,,,3141,;,2513,2613,4131;,2313,2136;2135,.
[161] 3153:,,,,;,5131,6131,;,3131,3161;3151,.
[162] 1114:,,,,;,,4213,2213;,,2134;,.
[163] 3235:,,,1341,;,5213,6213,1431;,3213,;,.
[164] 1353:,,,,;,1531,1631,;,1331,1361;1351,.
[165] 1115:,,,,;,,,1236;,,1235;,.
[166] 1211:,,,,;,,,;2111,,;,1211.
[167] 1245:,2114,2116,3111,;,2115,,1311;,1214,1216;1215,.
[168] 1224:,[169],[170],[171],;,[172],[173],[174];,1232,1314;,.
[169] 1134:,,,,;,3142,,;,4132,2132;,.
[170] 3153:,,,,;,5132,6132,;,3132,3162;3152,.
[171] 1115:,,,,;,,,;,,3116;3115,.

[172] 1131:,,,,;,,1342,;,,1432;,.
[173] 1353:,,3114,,;,1532,1632,;,1332,1362;1352,.
[174] 1115:,,,,;,,,;,,1316;1315,.
[175] 1145:,,,,;,[176],[177],;,,,;,.
[176] 1214:,,,,;4112,5112,,;1412,1512,;,.
[177] 1212:,,,,;,,,,;2112,6112,;1612,1212.
[178] 1114:,,[179],[182],[190];,[198],[202],[211];[221],[222],[225];[233],.
[179] 1115:,,[180],[181],;,,,;,,;,.
[180] 1411:,,,,;,,4521,;,,5421;,.
[181] 2416:4621,4221,,,;6421,4421,,;,2421,;,.
[182] 1256:[183],[184],[185],[186],;,[187],[188],[189];,,4253;,.
[183] 1511:,,,,;,,5621,;,,6521;,.
[184] 1522:,,,,;,2621,6621,;2521,5521,;,.
[185] 1415:,,2543,2643,;,,4325,2463;,,3425;,.
[186] 1343:,,3422,,;,2433,4322,;,,2443;2343,.
[187] 1115:,,5221,6221,;,,,;,,;,.
[188] 3263:,4326,,,;,3426,2453,2221;,6243,5243;4263,.
[189] 2133:,,,,;,3243,4243,;,4233,2243;,.
[190] 2553:5325,[191],[192],,;3525,[193],[194],;5253,[195],[196];[197],2553.
[191] 3265:5326,5322,,,;6325,,,;3625,3325,;3225,.
[192] 3266:,6322,,,;6326,3622,3322,;3626,3326,3222;3226,.
[193] 1232:,2325,3526,,;,3522,5263,;,,5233;,.
[194] 1336:,3263,6263,,;,6233,3233,2322;,,2326;,.
[195] 1116:,,,6253,3253;,,,;,,;,.
[196] 1266:,,2633,2333,;,2663,2363,2233;,,2263;,.
[197] 1256:,2563,2533,,;,2653,2353,;,,2253;,.
[198] 1144:,,,,;,[199],,;[200],[201],;,.
[199] 1215:,4151,4161,,;,1451,1461,;,,;,.
[200] 1211:,,,,;,,4141,;,,1441;,.
[201] 1215:,5141,6141,,;,1541,1641,;,,;,.
[202] 1245:4152,[203],[204],6161,;[205],[206],[207],1661;[208],[209],1346;[210],1245.
[203] 1411:,,4162,,;,,4513,;,,5413;,.
[204] 4366:,6413,3413,6151,5151;,4613,4413,5161;,4136,4313;,.
[205] 1141:,,,,;,,1452,;,,5142;,.
[206] 2116:,1462,,,;,6142,4142,4135;,,2142;2146,.
[207] 1255:,,,3146,;,,1561,1436;,1551,1651;,.
[208] 1112:,,,,;,,2145,;,,1542;,.
[209] 1216:,,,3145,;,1642,1442,1435;,,;,.
[210] 1112:,,,,;,,1246,1345;,,1242;,.
[211] 1335:3513,[212],[213],[214],;5313,[215],[216],[217];3135,[218],[219];[220],1335.
[212] 1515:,,3156,,;,,5613,3613;,5513,6513;,.
[213] 2256:,5162,3524,6613,;,6152,5152,3166;,2152,,2156,.
[214] 1226:2162,6162,,,;,2166,3624,3424;,,3224;,.
[215] 1165:,,6313,3313,;,5136,3136,;,,3155;3165,.
[216] 1256:2165,,5324,3324,;1562,2155,6136,;1652,1552,;1252,1256.
[217] 2126:1262,1662,,,;,1266,6324,4324;,,2324;,.
[218] 1156:,,,,;,6135,5135,;,1536,;1356,.
[219] 1136:,,,,;,,1265,1255;,1366,;1636,.
[220] 1136:,,,,;,1365,1355,;,1635,1535;1336,.
[221] 1211:,,,,;,,,;,4111,;1411,.

[222] 1111:,,,,;,,,;,,[223];[224],.
[223] 1215:,,,,;,,4116,;,4115,1416;1415,.
[224] 1215:,,,,;,5111,6111,;,1511,1611;,.
[225] 1215:,2154,[226],[227],;,5116,[228],[229];,[230],[231];[232],.
[226] 1356:,5134,2164,2144,;,3154,,;,,;,.
[227] 3146:,,,,;6134,4134,,;3164,3134,;3144,.
[228] 1134:,,,,;,,6116,;,1354,;1534,.
[229] 1346:,,,,;1634,1434,,;1364,1334,;1344,.
[230] 1116:,,,,;,,,1254;,6115,5115;1516,.
[231] 1116:,,,,;,,1264,1244;,,;1616,.
[232] 1116:,,,,;,,,;,1615,1515;,.
[233] 1115:,,,,;,,,;,[234],[235];,.
[234] 1211:,,,,;,,5114,;,,1514;,.
[235] 1216:,,,,;,6114,4114,;,1614,1414;,.
[236] 1124:,,,[237],[244];,,[250],[259];,,[269];,.
[237] 3455:5543,[238],[239],,;4553,[240],[241],;5453,[242],[243];3453,.
[238] 4536:5643,5343,,,;6543,,,;4563,4543,;4533,.
[239] 4636:,6343,,,;6643,4363,4343,;4663,4643,4333;4633,.
[240] 3446:,4653,4353,,;,5463,5433,;,5443,3543;,.
[241] 6436:,3643,3343,,;,4463,4443,;6463,6443,4433;6433,.
[242] 1116:,,,6453,4453;,,,;,,;,.
[243] 1136:,,3463,3443,;,,,3433;,,;,.
[244] 1351:,,[245],[246],;,,[247],[248];,,[249];,.
[245] 3566:,5633,,,;5663,6533,5533,;6563,5563,3533;3563,.
[246] 1333:,,,,;,,3663,6663:,3633,6633;,.
[247] 3656:,5363,5333,,;,6553,5553,;6653,5653,3553;3653,.
[248] 1166:,,,6333,3333;,,6363,3363;,,;,.
[249] 3116:,,6353,5353,;,,,3353;,,;,.
[250] 1445:,4156,[251],[252],;,[253],[254],[255];4145,[256],[257];[258],1445.
[251] 1116:,,,,4522;,,,4526;,,4166;,.
[252] 1166:,,,4622,4222;,,4626,4226;,,;,.
[253] 1155:,,,,;,,5146,4146;,,4165;4155,.
[254] 4265:5426,5422,6146,,;,4426,4422,;4625,4525,;4225,.
[255] 1166:,,,6422,2422;,,6426,2426;,,;,.
[256] 1156:,,,4425,;,6145,5145,;,1546,;1456,.
[257] 2146:,6425,5425,,;,1466,2425,;,1646,;,.
[258] 1146:,,,,;,1465,1455,;,1645,1545;1446,.
[259] 1255:,[260],[261],[262],;,[263],[264],[265];,[266],[267];[268],.
[260] 1112:,,,5526,;,,,5522;,,;,.
[261] 1566:,,5622,,;,5626,6522,2522;5166,6526,2526;,.
[262] 1222:,,,6166,;,,2626,6626;,2622,6622;,.
[263] 2526:,,5165,,;,5625,5156,;,6525,5525;2525,.
[264] 1661:,6156,,5226,5222;,6165,6625,2625;,1566,;,.
[265] 1166:,,,6222,2222;,,6226,2226;,,;1666,.
[266] 1156:,,,5225,;,1565,,;,6155,5155;1556,.
[267] 1116:,,,6225,2225;,,1665,;,,1656;,.
[268] 1116:,,,,;,,1655,1555;,,;,.
[269] 1455:,[270],[271],[272],;5154,[273],[274],[275];1554,1654,[276];1454,.
[270] 1116:,,,,5524;,,5164,5144;,,;,.
[271] 4562:5624,5224,6144,,;6524,2524,6164,4144;,4524,4164;,.

[272] 2616:,,6224,4224,;,6624,4624,2224;,,2624;,.
[273] 1156:,,,,;,1564,1544,;,6154,4154;,.
[274] 1141:,,,5424,;,,1664,;,,1644;,.
[275] 2116:,,6424,4424,;,,,2424;,,;,.
[276] 1116:,,,,;,,1464,1444;,,;,.
[277] 1112:,,,,;,[278],,;,[279],,;1132,.
[278] 1211:,,,,;,3121,,;,1321,;,.
[279] 1211:,,,,;,2113,,;,,;1213,.
[280] 1111:,,,,;,,,[281];,,[286];[293],.
[281] 1124:,,,,;,[282],[283],;,,[284];[285],.
[282] 1211:,,2143,,;,,,;,,1243;,.
[283] 1235:2153,2163,,,;,2133,,;1253,1263,;1233,.
[284] 1352:3125,3126,,,;,3122,,;1325,1326,;1322,.
[285] 1211:,,3124,,;,,1324,;,,;,.
[286] 1214:4121,[287],,,,;[288],[289],[290],;,[291],[292];,.
[287] 2115:,,,,;5121,6121,,;,2121,;,.
[288] 1112:,,,,;,1421,,;,,;1142,.
[289] 1152:,,,,;1521,1621,4113,;,,,;1162,1152.
[290] 3135:,,,,;,5113,6113,;,3113,1136;1135,.
[291] 1113:,,,,;,,1221,;,1134,;1413,.
[292] 1315:,,,,;,,,;1513,1613,;1313,.
[293] 1112:,,,,;,,,;,1131,;,1112.
[294] 1114:,,,[295],[298];,,[302],[307];1141,[316],[317];[321],1114.
[295] 1156:,,,[296],[297];,,,;,,;,.
[296] 1415:,,4523,4623,;,,5423,6423;,,;,.
[297] 2424:,,,,;,4223,4323,;,4423,3423;2423,.
[298] 1255:,5523,[299],[300],;,,5223,[301];,,;,.
[299] 2516:,5623,5323,,;,6523,3523,;,,2523;,.
[300] 2333:,,,,;,3623,6623,;3323,6323,2623;2323,.
[301] 2116:,,6223,3223,;,,,2223;,,;,.
[302] 1215:,,[303],[304],;,4125,[305],[306];,1425,;,.
[303] 1142:,,,,;,4126,4153,;,4122,5143;,.
[304] 3146:,,,,;4163,4133,,;6143,4143,;3143,.
[305] 1142:,,,,;,1426,1453,;,1422,1543;,.
[306] 1346:,,,,;1463,1433,,;1643,1443,;1343,.
[307] 1225:,,[308],[309],;5122,[310],[311],[312];[313],[314],1626;[315],1225.
[308] 3156:,,,,;5163,5133,,;6153,5153,;3153,.
[309] 1166:,,,,;,,6133,3133;,6163,3163;,.
[310] 1156:,,,,;,,6122,2122;,5126,2126;,.
[311] 1356:,,,,;1563,1533,6126,;1653,1553,;1353,.
[312] 1166:,,,,;,,1633,1333;,1663,1363;,.
[313] 1112:,,,,;,,2125,;,,1522;,.
[314] 1155:,,,,;,,1526,1622;,5125,6125;,.
[315] 1156:,,,,;,1625,1525,1222;,,1226;,.
[316] 1115:,,,,;,,,;1151,1161,1146;1145,.
[317] 1156:,,,,;,,[318],[319];1165,,;[320],1156.
[318] 1215:,5124,6124,,;,1524,1624,;,,;,.
[319] 1212:,2124,4124,,;,,1424,;,1224,;,.
[320] 1115:,,,,;,,,;,,1166;1155,.
[321] 1111:,,,,;,,,;,,[322];[323],1111.

[322] 1145:,,,,;,,,;1154,1164,;1144,.
[323] 1115:,,,,;,,,;,,;1116,1115.
[324] 1245:,[325],[326],,;,[327],[328],[329];,1125,[330];,.
[325] 1114:,,,,;,,4123,5123;,,;,.
[326] 2116:,,,,;,6123,3123,;,,2123;,.
[327] 1114:,,,,;,,1423,1523;,,;1124,.
[328] 1162:,,,,;,1623,1323,;1126,1121,1153;1122,.
[329] 1116:,,,,;,,,;,1163,1133;1113,.
[330] 1111:,,,,;,,,1223;,,1143;,.

付録 B　ソースコード

B.1　プログラム 1

出題者の秘密のコードを推測とヒントを打ち込んでいくことで当てることのできるプログラムを作成した．このプログラムでは初めに解答者は 4 章の表 1 にある「初めの推測」の中から自由に一つ目の推測をすることができ，その推測とそれに対するヒントをプログラムに打ち込む．するとプログラムが動作しその推測によってテストパターンが絞り込まれ，その絞り込まれたテストパターンが一覧となって出力される．その中から解答者は自由に推測し，その推測とそれに対するヒントを書き込むと同様に絞り込まれたテストパターンが出力される．このようにして正答を導き出すプログラムとなっている．

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define CODELEN 4
5  #define DIGITS 6
6  #define NUM_CODES (6*6*6*6)
7  #define NUM_RESP ((CODELEN + 1)*(CODELEN + 1))
8
9  #define PROFLISTSIZE NUM_CODES
10
11 struct proflist {
12     int id;
13     int max;
14     int count[NUM_RESP];
15 } proflist[PROFLISTSIZE];
16 int num_prof;
17
18 char *strcode(char *str, int id)
19 {
20     for (int i = 0; i < CODELEN; i++, id /= DIGITS) {
21     str[CODELEN - 1 - i] = '1' + id % DIGITS;
22     }
23     return(str);
24 }
25
26 int response(char *code, char *test)
27 {
28     char cc[CODELEN], ct[CODELEN];
29     int bh = 0, wh = 0;
30
31     for (int i = 0; i < CODELEN; i++) {
32      if (code[i] == test[i]) {
33       bh++;
34           cc[i] = ct[i] = 1;
35        }
36        else {
37           cc[i] = ct[i] = 0;
```

```
38          }
39      }
40      for (int i = 0; i < CODELEN; i++) {
41          if (cc[i]) continue;
42          for (int j = 0; j < CODELEN; j++) {
43              if (ct[j]) continue;
44              if (code[i] == test[j]) {
45                  wh++;
46                  ct[j] = 1;
47                  break;
48              }
49          }
50      }
51      return(bh * (CODELEN + 1) + wh);
52  }
53
54  void
55  rgstprof(int k, int count[])
56  {
57      int i, j;
58      int max;
59      for (i = 0; i < num_prof; i++) {
60          for (j = 0; j < NUM_RESP; j++) {
61              if (count[j] != proflist[i].count[j]) break;
62              //までと(00)~(44)のものと比較してちがうところがあればこの文を抜ける proflistfor
63              //次の段にどやっていくかわからない？
64          }
65          if (j == NUM_RESP) break; // 同じものが既にある
66      }
67      if (i < num_prof) return; // 同じものは登録しない
68
69                                  // 登録する
70      if (num_prof == PROFLISTSIZE) {
71          printf("[%d]\n", __LINE__);
72          return;
73      }
74      proflist[num_prof].id = k;
75      max = 0;
76      j = 0;
77      for (i = 0; i < NUM_RESP; i++) {
78  //      printf("%d ", count[i]);
79          proflist[num_prof].count[i] = count[i];
80          if (proflist[num_prof].count[i] > max) {
81              max = proflist[num_prof].count[i];
82              j = i;
83          }
84      }
85      proflist[num_prof].max = max;
86      num_prof++;
87  //  printf("-> max:%d [%d]\n", max, j);
88
89      return;
90  }
91
92  void
93  makeproflist(int flag[])
94  {
95      char code[CODELEN + 1];
96      char test[CODELEN + 1];
97      int resp;
98
```

```
 99      code[CODELEN] = '\0';
100      test[CODELEN] = '\0';
101      num_prof = 0;
102
103      for (int k = 0; k < NUM_CODES; k++) {
104          int count[NUM_RESP];
105          strcode(test, k);
106          //上に行ってに代入しテストk=1が1111となるように数値を作るstrcode
107          for (int j = 0; j < NUM_RESP; j++) {
108              count[j] = 0;
109              //全てがゼロにリセットされる(00)~(44)
110          }
111          for (int i = 0; i < NUM_CODES; i++) {
112              if (flag[i] == 0) continue;
113              strcode(code, i);
114              //上に行ってに代入しコード=1が1111となるように数値を作るstrcodei
115              resp = response(code, test);
116              //上ので計算されるresponce
117              count[resp]++;
118              //    (の数値が入ってくる  returnのときは(00)が0になりreturncounntに1プラスされる{0}
119          }
120          rgstprof(k, count);
121      }
122      return;
123  }
124
125  int
126  flagselect(int flag[], int id, int resp)
127  {
128      char code[CODELEN];
129      char test[CODELEN];
130      int n = 0;
131
132      strcode(test, id);
133      for (int i = 0; i < NUM_CODES; i++){
134          if (flag[i] == 0) continue;
135          strcode(code, i);
136          if (response(code, test) != resp){
137              flag[i] = 0;
138          } else {
139              n++;
140          }
141      }
142      return(n);
143  }
144
145
146  int
147  int2id(int i_test)
148  {
149      int id = 0;
150      int mul = 1;
151
152      for (int i = 0; i < CODELEN; i++) {
153          id += ((i_test % 10) - 1) * mul;
154          i_test /= 10;
155          mul *= DIGITS;
156      }
157      return(id);
158  }
159
```

```
160  int
161  int2resp(int i_resp)
162  {
163      int resp;
164
165      resp = i_resp % 10;
166      i_resp /= 10;
167      resp += (i_resp % 10) * (CODELEN + 1);
168      return(resp);
169  }
170
171  int
172  interact(int flag[])
173  {
174      int i_test, i_resp;
175      int n;
176
177      scanf("␣%d␣%d", &i_test, &i_resp);
178      n = flagselect(flag, int2id(i_test), int2resp(i_resp));
179      printf("残り%個ですd\n", n);
180      return(n);
181  }
182
183  void
184  printprof()
185  {
186      char test[CODELEN + 1];
187
188      for (int i = 0; i < num_prof; i++) {
189          strcode(test, proflist[i].id);
190          test[CODELEN] = '\0';
191          printf("%s(%d):", test, proflist[i].max);
192          for (int j = 0; j < NUM_RESP; j++) {
193              int bh = j / (CODELEN + 1);
194              int wh = j % (CODELEN + 1);
195              printf("␣(%d%d)=%d", bh, wh, proflist[i].count[j]);
196          }
197          printf("\n");
198      }
199      return;
200  }
201
202  int
203  main()
204  {
205      int flag[NUM_CODES];
206      int n;
207
208      for (int i = 0; i < NUM_CODES; i++) {
209          flag[i] = 1;
210      }
211      makeproflist(flag);
212      printprof();
213
214      while ((n = interact(flag)) > 1) {
215          makeproflist(flag);
216          printprof();
217      }
218      if (n == 0) {
219          printf("該当するコードはありません\n");
220      } else {
```

```
221             int i;
222             char code[CODELEN + 1];
223             for (i = 0; i < NUM_CODES; i++){
224                 if (flag[i] == 1) break;
225             }
226             strcode(code, i);
227             code[CODELEN] = '\0';
228             printf("コード %s が特定されました\n", code);
229         }
230 
231         return(0);
232     }
```

B.2 プログラム 2

最悪手数と平均を出すプログラムを作成した．1112,1122,1123,1234 のどれか一つを最初の推測とした戦略の最悪手数とその平均手数を導き出すことのできるプログラムである．

```
1  
2  #include <stdio.h>
3  #include <stdlib.h>
4  
5  #define CODELEN 4
6  #define DIGITS 6
7  #define NUM_CODES (6*6*6*6)
8  #define NUM_RESP 14
9  //#define MAX_MEMORY 10000000 // android max
10 #define MAX_MEMORY 1000000
11 
12 struct strategy_node {
13     int n_survivors;
14     int est_worst;
15     int est_average;
16     int n_options;
17     int finish;
18     struct search_node *p;
19 };
20 
21 struct search_node {
22     int test_id;
23     struct strategy_node strategy[NUM_RESP];
24 };
25 
26 int survivor[NUM_CODES];
27 int memory = 0;
28 int max_memory = 0;
29 
30 void *realloc_s(void *__ptr, size_t __byte_count, int line)
31 {
32     void *p = realloc(__ptr, __byte_count);
33     if (p == NULL) {
34         printf("[%d] out of memory\n", line);
35         exit(1);
36     }
37     return(p);
38 }
39 
40 char *strcode(char *str, int id)
41 {
42     for (int i = 0; i < CODELEN; i++, id /= DIGITS) {
```

```
43          str[CODELEN - 1 - i] = '1' + id % DIGITS;
44      }
45      return(str);
46  }
47
48  int response(char *code, char *test)
49  {
50      int cc[CODELEN], ct[CODELEN];
51      int bh = 0, wh = 0;
52
53      for (int i = 0; i < CODELEN; i++) {
54          if (code[i] == test[i]) {
55              bh++;
56              cc[i] = ct[i] = 1;
57          }
58          else {
59              cc[i] = ct[i] = 0;
60          }
61      }
62      for (int i = 0; i < CODELEN; i++) {
63          if (cc[i]) continue;
64          for (int j = 0; j < CODELEN; j++) {
65              if (ct[j]) continue;
66              if (code[i] == test[j]) {
67                  wh++;
68                  ct[j] = 1;
69                  break;
70              }
71          }
72      }
73      return(bh * (CODELEN + 1) + wh);
74  }
75
76  int respindex(int resp)
77  {
78      static int index_table[] = {
79           4,  3,  2,  1,  0,
80           8,  7,  6,  5, -1,
81          11, 10,  9, -1, -1,
82          12, -1, -1, -1, -1,
83          13, -1, -1, -1, -1 };
84      int index;
85
86      if (resp > 24 || resp < 0) {
87          resp = 24;
88      }
89      index = index_table[resp];
90      if (index < 0) {
91          printf("[%d]program error\n", __LINE__);
92          exit(1);
93      }
94      return(index);
95  }
96
97  int estimateaverage(int n)
98  {
99      int log14 = 0;
100
101     for (int i = 1; i < n; i *= 14) {
102         log14++;
103     }
```

```
104         return(n * log14);
105     }
106
107     int
108     makeprof(int prof[NUM_RESP], int test_id)
109     {
110         char code[CODELEN];
111         char test[CODELEN];
112         int count = 0;
113
114         for (int i = 0; i < NUM_RESP; i++) {
115             prof[i] = 0;
116         }
117         strcode(test, test_id);
118         for (int i = 0; i < NUM_CODES; i++) {
119             if (!survivor[i]) continue;
120             strcode(code, i);
121             prof[respindex(response(code, test))]++;
122         }
123         for (int i = 0; i < NUM_RESP; i++) {
124             if (prof[i] > 0) {
125                 count++;
126             }
127         }
128         return(count);
129     }
130
131     struct search_node *
132         searchprof(struct strategy_node *stgy, int prof[NUM_RESP])
133     {
134         for (int i = 0; i < stgy->n_options; i++) {
135             int j;
136             for (j = 0; j < NUM_RESP; j++) {
137                 if (prof[j] != stgy->p[i].strategy[j].n_survivors) break;
138             }
139             if (j >= NUM_RESP) {
140                 return(stgy->p + i);
141             }
142         }
143         return(NULL);
144     }
145
146     int
147     addprof(struct strategy_node *stgy, int test_id, int prof[NUM_RESP])
148     {
149         //  if (memory >= MAX_MEMORY){
150         //      printf("[%d]out of memory\n", __LINE__);
151         //      exit(1);
152         //  }
153         stgy->p = realloc_s(stgy->p, sizeof(struct search_node) * (stgy->n_options + 1), __LINE__);
154         if (++memory > max_memory) {
155             max_memory = memory;
156         }
157         for (int i = 0; i < NUM_RESP; i++) {
158             stgy->p[stgy->n_options].strategy[i].n_survivors = prof[i];
159             stgy->p[stgy->n_options].strategy[i].est_worst = prof[i];
160             stgy->p[stgy->n_options].strategy[i].est_average = estimateaverage(prof[i]);
161             stgy->p[stgy->n_options].strategy[i].n_options = 0;
162             stgy->p[stgy->n_options].strategy[i].finish = (prof[i] < 2) ? 1 : 0;
163             stgy->p[stgy->n_options].strategy[i].p = NULL;
164         }
```

```
165     stgy->p[stgy->n_options].test_id = test_id;
166     stgy->n_options++;
167     return(stgy->n_options);
168 }
169
170 void
171 printsearchnode(struct strategy_node *stgy)
172 {
173     for (int i = 0; i < stgy->n_options; i++) {
174         printf("%d:", stgy->p[i].test_id);
175         for (int j = 0; j < NUM_RESP; j++) {
176             printf(" %d", stgy->p[i].strategy[j].n_survivors);
177         }
178         putchar('\n');
179     }
180     printf("%d %d %d\n", stgy->est_worst, stgy->est_average, stgy->n_options);
181     return;
182 }
183
184 int
185 makesearchnode(struct strategy_node *stgy)
186 {
187     int prof[NUM_RESP];
188
189     for (int i = 0; i < NUM_CODES; i++) {
190         if (makeprof(prof, i) < 2) continue;
191         if (searchprof(stgy, prof)) continue;
192         addprof(stgy, i, prof);
193     }
194     return(stgy->n_options);
195 }
196
197 int
198 selectsurvivors(int test_id, int respidx)
199 {
200     char code[CODELEN];
201     char test[CODELEN];
202     int n_sur = 0;
203
204     strcode(test, test_id);
205     for (int i = 0; i < NUM_CODES; i++) {
206         if (!survivor[i]) continue;
207         if (respindex(response(strcode(code, i), test)) == respidx) {
208             n_sur++;
209         }
210         else {
211             survivor[i] = 0;
212         }
213     }
214     return(n_sur);
215 }
216
217 int searchstgy(struct strategy_node *stgy);
218
219 int
220 godownstairs(struct search_node *p)
221 {
222     int respidx = -1;
223     int max_wst = 0;
224     int max_avg = 0;
225     int wst, avg;
```

```
226
227     for (int i = 0; i < NUM_RESP; i++) {
228         if (p->strategy[i].finish == 1) continue;
229         wst = p->strategy[i].est_worst;
230         avg = p->strategy[i].est_average;
231         if (wst > max_wst || (wst == max_wst && avg > max_avg)) {
232             max_wst = wst;
233             max_avg = avg;
234             respidx = i;
235         }
236     }
237     if (respidx < 0) {
238         return(-1); // node completed
239     }
240     selectsurvivors(p->test_id, respidx);
241     searchstgy(p->strategy + respidx);
242     return(0);
243 }
244
245 void
246 freestrategy(struct strategy_node *stgy, int except)
247 {
248     for (int i = 0; i < stgy->n_options; i++) {
249         if (i == except) continue;
250         for (int j = 0; j < NUM_RESP; j++) {
251             if (stgy->p[i].strategy[j].p) {
252                 freestrategy(stgy->p[i].strategy + j, -1);
253             }
254         }
255     }
256     if (except < 0) {
257         free(stgy->p);
258         memory -= stgy->n_options;
259         stgy->n_options = 0;
260         stgy->p = NULL;
261     }
262     return;
263 }
264
265 int
266 countworst(struct search_node  *p)
267 {
268     int wst = 0;
269
270     for (int i = 0; i < NUM_RESP; i++) {
271         if (p->strategy[i].est_worst > wst) {
272             wst = p->strategy[i].est_worst;
273         }
274     }
275     return(wst);
276 }
277
278 int
279 countaverage(struct search_node *p)
280 {
281     int avg = 0;
282
283     for (int i = 0; i < NUM_RESP; i++) {
284         avg += p->strategy[i].est_average;
285     }
286     return(avg);
```

```
287 }
288
289 int
290 determine(struct strategy_node *stgy, int sel)
291 {
292     freestrategy(stgy, sel);
293     stgy->p[0] = stgy->p[sel];
294     stgy->p = realloc_s(stgy->p, sizeof(struct search_node), __LINE__);
295     memory -= stgy->n_options - 1;
296     stgy->est_worst = countworst(stgy->p) * NUM_RESP;
297     stgy->est_average = countaverage(stgy->p) + stgy->n_survivors;
298     stgy->n_options = 1;
299     stgy->finish = 1;
300     return(stgy->est_worst);
301 }
302
303 void
304 showstrategy(struct strategy_node *stgy)
305 {
306     char code[CODELEN + 1];
307
308     code[CODELEN] = '\0';
309     strcode(code, stgy->p->test_id);
310     printf("%s(", code);
311     for (int i = 0; i < NUM_RESP; i++) {
312         if (stgy->p->strategy[i].n_survivors == 0) {
313             //putchar('*');
314         }
315         else if (stgy->p->strategy[i].n_survivors == 1) {
316             putchar('*');
317         }
318         else {
319             showstrategy(stgy->p->strategy + i);
320         }
321         putchar((i < NUM_RESP - 1) ? ',' : ')');
322     }
323     return;
324 }
325
326 int
327 bestsearchnode(struct strategy_node *stgy)
328 {
329     int sel;
330
331     stgy->est_worst = (unsigned int)-1 >> 1;
332     stgy->est_average = (unsigned int)-1 >> 1;
333     for (int i = 0; i < stgy->n_options; i++) {
334         int wst = countworst(stgy->p + i);
335         int avg = countaverage(stgy->p + i);
336         if (wst < stgy->est_worst || (wst == stgy->est_worst && avg < stgy->est_average)) {
337             stgy->est_worst = wst;
338             stgy->est_average = avg;
339             sel = i;
340         }
341     }
342     stgy->est_worst *= NUM_RESP;
343     stgy->est_average += stgy->n_survivors;
344     return(sel);
345 }
346
347 int
```

```
348 searchstgy(struct strategy_node *stgy)
349 {
350     int sel;
351
352     if (stgy->n_options == 0) {
353         makesearchnode(stgy);
354         bestsearchnode(stgy);
355         return(stgy->est_worst);
356     }
357
358     sel = bestsearchnode(stgy);
359     if (godownstairs(stgy->p + sel) < 0) {
360         printf("mem:%d ", memory);
361         determine(stgy, sel);
362         showstrategy(stgy);
363         putchar('\n');
364     }
365     else {
366         bestsearchnode(stgy);
367     }
368     return(stgy->est_worst);
369 }
370
371 int
372 optimalstrategy(struct strategy_node *stgy)
373 {
374     int target[NUM_CODES];
375
376     for (int i = 0; i < NUM_CODES; i++) {
377         target[i] = survivor[i];
378     }
379     while (stgy->finish == 0) {
380 //printf("[%d]\n", __LINE__);
381         for (int i = 0; i < NUM_CODES; i++) {
382             survivor[i] = target[i];
383         }
384         searchstgy(stgy);
385     }
386     return(0);
387 }
388
389 int
390 main()
391 {
392     struct strategy_node stgy;
393
394     for (int i = 0; i < NUM_CODES; i++) {
395         survivor[i] = 1;
396     }
397
398     //  stgy.n_survivors = selectsurvivors(7, 4);
399     stgy.n_survivors = NUM_CODES;
400     stgy.est_worst = stgy.n_survivors;
401     stgy.est_average = estimateaverage(stgy.n_survivors);
402     stgy.n_options = 0;
403     stgy.finish = 0;
404     stgy.p = NULL;
405
406     makesearchnode(&stgy);
407     determine(&stgy, 2);
408     stgy.finish = 0;
```

```
409
410         for (int i = 0; i < NUM_CODES; i++) {
411             survivor[i] = 1;
412         }
413
414         optimalstrategy(&stgy);
415
416         printf("[%d]mem:%d worst:log 14(%d) avg:%f\n", __LINE__,
417             memory, stgy.est_worst, (double)stgy.est_average / stgy.n_survivors);
418         showstrategy(&stgy);
419
420         freestrategy(&stgy, -1);
421         printf("\nmem:%d max:%dMB\n", memory, (int)((max_memory * sizeof(struct search_node)) >> 20
422
423         puts("Ok");
424         return(0);
425     }
```