卒業研究報告書

題目

3 次元 n クイーンの問題の解に関する研究

指導教員

石水 隆　助教

報告者

08-1-037-0149

伊藤 精一

近畿大学理工学部情報学科

平成 24 年 1 月 31 日提出

概要

本研究では、代表的な最適化問題である n クイーン問題において、クイーンの最小個数解を検証する。また、本研究では、2 次元 n クイーン問題を 3 次元に拡張した 3 次元 n クイーン問題の場合でも同様に、最小個数解について検証する。

n クイーン問題とは、n 個のクイーンを n×n のチェス盤上に、縦・横・斜めの 8 方向の直線上に 1 個のクイーンしか存在できないように配置する問題である。また、n クイーン問題最小個数解とは n×n のチェス盤上に m(≦n) 個のクイーンを配置し、新たに m+1 個目のクイーンを盤上を置くことができないような配置を解としたとき、m の値が最小の値となるものを解とする。本研究では 2 次元および 3 次元 n クイーン最小個数問題を解くアルゴリズムを提案した。本研究で提案したアルゴリズムは、バックトラック法を用いて解を得たが、膨大な時間を要した。

また、本研究では、C++言語を用いてアルゴリズムの実装を行った。作成した 2 次元 n クイーン問題プログラムは、(0,0)から x 方向、y 方向の優先順にクイーンを配置するバックトラック法により解の探索を行う。3 次元 n クイーン問題プログラムも同様に(0,0,0)から x 方向、y 方向、z 方向の優先順にクイーンを配置するバックトラック法により解の探索を行うように拡張した。

目次

1. 序論

1.1 本研究の背景

計算機を用いて解くことが困難な問題として、組み合わせ最適化問題[5]がある。組み合わせ最適化問題とは求める解に順序や分割に組み合わせ的な性質を持つ問題である。一般に、組み合わせ最適化問題は問題は、問題のサイズが大きくなると、探索範囲が指数的に増大するため、膨大な計算時間を必要とする。そのため、計算機上で組み合わせ最適化問題を解くためには、できるだけ探索範囲を狭めるように問題の性質に応じて工夫する必要がある。代表的な組み合わせ最適化問題としては、巡回セールスマン問題[6]、ナップサック問題[5]等がある。

本研究では、組み合わせ最適化問題の 1 つである *n* クイーン問題の派生問題である *n* クイーンの最小個数問題を対象とする。

1.2 *n* クイーン問題

n クイーン問題[1]は、縦・横・斜めの 8 方向に直進できるチェス駒のクイーンを、n×n マス上に縦、横、斜めの8方向の直線上に1つのクイーンしか存在しないように配置する問題であり、n≧4 の場合 n×n マス上に n 個のクイーンを配置できる。本研究では、この 2 次元 n クイーン問題を 3 次元に拡張した 3 次元 n クイーン問題[1,2,3,4]についても検証した。3 次元 n クイーン問題とは、n クイーン問題の n×n マスの盤を n×n×n の立体に拡張し、3 次元の 26 方向の直線上に 1 つのクイーンしか存在しないように配置する問題である。また、*n* クイーン問題の派生問題として、クイーンの極大配置問題および最小個数問題がある。クイーン極大配置問題とは、n×n マス上に m(≦n)個のクイーンを配置したときに、新たに m+1 個目のクイーンを盤上のどこにも置くことができないような配置を解とする。クイーン最小個数問題とは、極大配置解 m の値が最小となるものを解とする。

2 次元 n クイーン問題の既知の結果として、求められた n の最大数は n=26[9]となっている。n=26 を求めるのに要した時間はおよそ 9 ヶ月、発見された解の数はおよそ 2 京である。また、3 次元 n クイーン問題の既知の結果として、m 次元 n クイーン問題に解が存在するか否かを判別する定理[1,2,3,4]が発見されている。

1.3 本研究の目的

n クイーン問題の研究は一般的に n の最大数を求めたり、解を求めるための時間を少なくするための研究が非常に多い。また、3 次元 n クイーン問題に至っては、研究自体がほとんど行われていない。よって、本研究では、2 次元 n クイーン問題および 3 次元 n クイーン問題の各 n に対してクイーンの最小個数解を検証することにした。

1.4 本報告書の構成

本報告書では、2 章にて 2 次元 n クイーン問題および 3 次元 n クイーン問題について述べる。3 章にて本研究で作成した 2 次元 n クイーン問題および 3 次元 n クイーン問題の最小個数解を求めるプログラムのアルゴリズムについて述べる。4 章にて本研究で作成した 2 次元 n クイーン問題および 3 次元 n クイーン問題の最小個数解を求めるプログラムについて述べる。5 章にて本研究で作成したプログラムを用いた実行結果とその考察について述べる。6 章にて結論と今後の課題について述べる。

2. *n* クイーン問題

本章では、本研究で対象とする 2 次元 *n* クイーン問題および 3 次元クイーン問題、その派生問題である *n* クイーンの最小個数問題について述べる。

2.1　定義

チェスの駒の 1 つにクイーンがある。クイーンはサイズ 8×8 のチェス盤上を縦・横・斜めの 8 方向に何マスでも進むことができる。8 クイーン問題は、チェス盤上に 8 個のクイーンを配置し、どのクイーンも他のクイーンに取られるような位置に配置してはいけない問題である。n クイーン問題は、8 クイーン問題の応用問題として、サイズ n×n のチェス盤上に n 個のクイーンを配置し、どのクイーンも他のクイーンに取られるような位置に配置してはいけない問題である。n が n□4 である n クイーン問題の場合、n 個のクイーンを配置する解が複数個存在する。

また、*n* クイーン問題を 3 次元に拡大した 3 次元 n クイーン問題は、チェス盤およびクイーンの移動範囲を 3 次元に拡張したものである。3 次元 n クイーン問題のクイーンは、27 方向の方向ベクトル(x,y,z) (x,y,x={-1,0,1})の内 0 ベクトル(0,0,0)を除く 26 方向のベクトルのいずれか 1 つの方向へ、他の駒または盤橋に当たるまで移動できる駒である。図 1 に 3 次元 n クイーン問題において、座標(2,2,2)にクイーンを配置したとき、クイーンが移動できるマスを示す。図 1 の Q はクイーン、×はクイーンが移動できるマスを表す。

3 次元 *n* クイーン問題とはサイズ n×n×n の 3 次元チェス盤上で、上記の直線 26 方向に並んだマス目から成る全てのラインにおいて、各ラインにつき 1 個のクイーンしか配置できないという条件の下でクイーンをできる限り多く配置する問題である。

n クイーン最小個数問題とは、n×n のチェス盤上に m(≦n)個のクイーンを配置し、新たに m+1 個目のクイーンを盤上を置くことができないような配置を解としたとき、m の値が最小の値となるものを解とする。図 2 に 2 次元 n クイーン問題において、n＝4 の最小個数問題の解の例を示す。図 2 の Q はクイーン、×はクイーンが移動できるマスを表す。

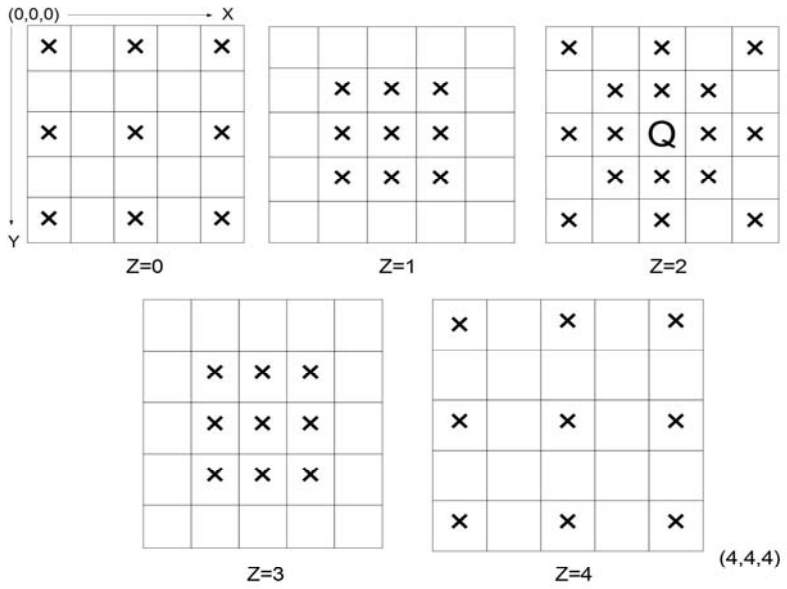

図 1：n＝5 の 3 次元 n クイーンにおいて中心座標(2,2,2)にクイーンを配置したときのクイーンの移動可能範囲

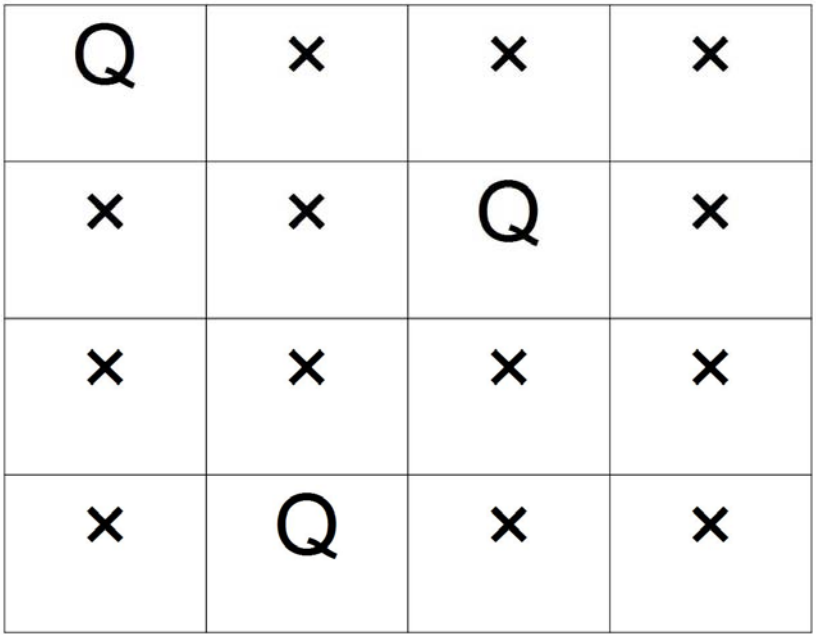

図 2：n=4 の 2 次元 n クイーンにおいて最小個数問題の解の例

2.2　解法

n クイーン問題を解くアルゴリズムとしてはバックトラック法がある。バックトラック法とはある条件を満たした解を求める時、可能性のある解を順に検証していき、その解では条件をみたさないと判断できた時点で 1 つ前の状態に戻って違う手順の解を検証していく方法である。

n クイーン問題およびその派生問題は、問題のサイズ *n* が大きくなるに従い探索範囲が指数的に広くなる。そのため、*n* クイーン問題を計算機で解く場合、大きな *n* に対しては非常に時間がかかってしまう。

3.　*n* クイーン最小個数問題を解くアルゴリズム

本章では、*n* クイーン最小個数問題を解くアルゴリズムについて述べる。本研究では、バックトラック法を用いて n クイーン最小個数問題を解く。以下に 3 次元 n クイーン問題の最小個数問題を解く基本的なアルゴリズムを示す。

1、最初の探索地点の座標を(0,0,0)として、x 方向、y 方向、z 方向の優先順に探索する。
2、x 方向、y 方向、z 方向の優先順にクイーンを設置する。
3、設置したクイーンの移動可能範囲を設置不可座標とする。
4、探索順路順にクイーン設置可能座標が存在する場合、2 と 3 の手順を繰り返す。
5、全ての探索順路の探索を終えた場合、最後に設置したクイーンを取り除き、設置されていた　の次の座標から探索を再開する。
6、1～5 の手順で結果を出し、クイーンの設置個数が最小値の場合、その結果を記録しておく。

4.　*n* クイーン最小個数問題を解くプログラム

本章では、*n* クイーン最小個数問題を解くアルゴリズムを実装した C++言語プログラムについ

て述べる。
付録に本研究で作成した 3 次元 n クイーン問題の C++言語プログラムを示す。

・int Q…探索途中で配置したクイーンの数
・int Qmin…判明している Q の最小個数解
・clear()…void 型、チェス盤をリセットする
・display()…void 型、チェス盤を表示する
・check()…void 型、全てのコマの可動範囲を-1 にする
・search()…bool 型、空いている座標を数える
・start(int a,int b,int c)…void 型、空いている座標から 3 次元 n クイーンの解を求める
・main()…void 型、プログラムを実行し、結果を表示する

5. 結果および考察

付録に示したプログラムを用いて本研究で得られた各 n に対する 2 次元および 3 次元 n クイーン最小値個数解の m の値と探索に要した時間を表 1 および表 2 に示す。表 1 および表 2 から示されるように 2 次元 n クイーン問題および 3 次元 n クイーン問題は n が増加するに応じて探索時間は指数的に増加する。本研究では、2 次元 n クイーン問題は n=13、3 次元 n クイーン問題では n＝6 の時、探索時間に膨大な時間を要し、解を示すことが出来なかった。また、2 次元 n クイーン問題は、n の値が小さい場合、n の値が 1 増えたときに、急に m の値が増えることがないことが示される。2 次元 *n* クイーン最小配置問題において *m* の急激な増加が起こらない理由としては、2 次元 n クイーンの探索範囲は n の二乗のため、n の数字が小さい場合は探索範囲が大きく増加しないからではないかと考える。3 次元 n クイーン問題は、得られた解が少ないため m の値について有意なデータは得らなかった。探索途中の現時点での解だが、n＝6 の時に m＝10 と示されており、n=5 の時の m＝6 と比べて m の値の差が大きく増加している。3 次元 n クイーン問題の探索範囲は n の三乗のため、探索範囲の増加が大きい。この事により 3 次元 n クイーン問題は、n の値が増えるに応じて m の値が大きく増えるのではないかと予想される。

表 1：2 次元 n クイーン最小個数解の m の値

n	1	2	3	4	5	6	7
m	1	1	1	3	3	4	4
探索時間	0 秒	0 秒	0 秒	0 秒	0.01 秒	0.01 秒	0.1 秒
n	8	9	10	11	12	13	14
m	4	5	5	5	7	*7	
探索時間	0.6 秒	5.6 秒	60 秒	621 秒	2 時間		

*探索途中の現時点での解

表 2：3 次元 n クイーン最小個数解の m の値

n	1	2	3	4	5	6
m	1	1	1	4	6	*10
探索時間	0 秒	0 秒	0.01 秒	0.5 秒	1918 秒	

*探索途中の現時点での解

6. 結論および今後の課題

本研究では 2 次元および 3 次元 n クイーン最小個数問題を解くアルゴリズムを提案した。本研究で提案したアルゴリズムはバックトラック法により解を求めた。今回作成したプログラムは、バックトラック法を用いたため、結果を出すために莫大な時間を要し、3 次元 n クイーン問題に関しては満足のいく結果が得られなかった。今後の課題として探索方法を改良することにより計算時間の短縮、また、それにより、より大きな n の解を出す事と n と m との関係性を検証することが挙げられる。

謝辞

本研究および本報告書を作成するにあたって、石水隆先生には御指導、御支援していただき、大変お世話になりました。ありがとうございました。また、同研究室の皆にも大変お世話になりました。誠に感謝申し上げます。

参考文献

[1] 岡田章三：m 次元 n クイーン問題, 岐阜高専紀要 第 37 号, pp13－16, (2002).

[2] 岡田章三：m 次元 n クイーン問題に関する計算例と予測, 岐阜高専紀要 第 38 号, pp11－14, (2003).

[3] 岡田章三：m 次元 n クイーン問題に関する研究, 岐阜高専紀要 第 39 号, pp7－9, (2004).

[4] 岡田章三：m 次元 n クイーン問題に関する報告, 岐阜高専紀要 第 40 号, pp1－3, (2005).

[5] 谷萩隆嗣,高速アルゴリズムと並列信号処理, コロナ社, 2000.

[6] 巡回セールスマン問題と GA, STUDIO-K, http://www.geocities.jp/studio_k32/tsp/index.html

[7] Jeff Somers's N Queens Solutions, http://www.jsomers.com/nqueen_demo/nqueens.html

[8] 吉瀬謙二, N-Queens Homepage in Japanese, 電気通信大学, 2004,http://www.arch.cs.titech.ac.jp/~kise/nq/

[9] Queen@TUD, Technische Universitat Dresden, 2009, http://Queens.inf.tu-dresden.de/

[10] 萩野谷一二, “NQueen 問題への新しいアプローチ(部分解合成法)について,” 情報処理学会報告書, Vol.2011-GI-26, No.11, 2011.

付録

本研究で作成したプログラムのソースファイルを以下に示す。

1.　2次元nクイーンの最小個数問題を解くプログラム

```
/* バックトラック法を用いて2次元nクイーン問題の最小個数解を出すプログラム */

#include<iostream>
#include<time.h>

#define N 4// チェス盤のサイズ

using namespace std;

int pos[N][N];
// 二次元のチェス盤。サイズはN×N。
// xが二次元上の左右、yが二次元上の上下。
long int countA=0;      //      解の数を数える(ユニーク解、バリエーション解全てを含む)
long int countB=0;
int Q=0;        // 探索現在のクイーンの数
int QMin=0;     // 判明している最小のQの個数

/**
 * 盤をリセットする関数
 */
void clear(){
        for(int i=0;i<N;i++){
                for(int j=0;j<N;j++){
                        pos[i][j]=0;
                }
        }
}

/**
 * 盤を表示する関数
 */
void display(){//
        for(int i=0;i<N;i++){
                for(int j=0;j<N;j++){
                        if(pos[i][j]==1){       // コマがあるマス
                                cout << "◎";
                        }else if(pos[i][j]==0){  // コマがないマス
                                cout << "□";
                        }else{  // コマを置くことができないマス
                                cout << "×";
                        }
                }
                cout << endl;
```

```
	}
	cout << endl;
}

/**
 * 全てのコマの可動範囲を-1 にする関数
 */
void check(){
	for(int i=0;i<N;i++){	// コマのないマスをクリア
		for(int j=0;j<N;j++){
			if(pos[i][j]!=1) pos[i][j]=0;
		}
	}

	for(int i=0;i<N;i++){	// コマのあるマスの可動範囲を-1 に
		for(int j=0;j<N;j++){
			if(pos[i][j]==1){
				//横
				for(int x=0;x<N;x++) {
					if(pos[i][x] != 1) {
						pos[i][x] = -1;
					}
				}

	//縦
	for(int y=0;y<N;y++) {
		if(pos[y][j] != 1) {
			pos[y][j] = -1;
		}
	}

	//左斜め上
	for(int m=0;m<N;m++) {
		if(pos[i-m][j-m] != 1 && i-m>=0 && j-m>=0) {
			pos[i-m][j-m] = -1;
		}
	}

	//左斜め下
	for(int n=0;n<N;n++) {
		if(pos[i+n][j-n] != 1 && i+n<N && j-n>=0) {
			pos[i+n][j-n] = -1;
		}
	}

	//右斜め上
	for(int o=0;o<N;o++) {
		if(pos[i-o][j+o] != 1 && i-o>=0 && j+o<N) {
```

```
                        pos[i-o][j+o] = -1;
                }
        }

        //右斜め下
        for(int p=0;p<N;p++) {
                if(pos[i+p][j+p] != 1 && i+p<N && j+p<N) {
                        pos[i+p][j+p] = -1;
                }
        }
}
                }

        }}

/**
 * (a,b)から空きマスを数える関数
 * @return 空きマスが存在した時にfalseを返す
 * @return 空きマスが存在しない時にtrueを返す
 */
bool search(int a,int b) {
        bool first=true;
        for(int i=0;i<N;i++){
                for(int j=0;j<N;j++){
                                if(first){
                                        i=i+a;
                                        j=j+b;
                                        first=false;
                                }
                                if(pos[i][j]==0) return false;
                }

        }
        return true;
}

/**
 * 再帰的呼び出しで(a,b)からクイーンを設置し、2D-N-Queenの解を求める関数
 */
void start(int a,int b){
        bool first=true;
        for(int i=0;i<N;i++){
                for(int j=0;j<N;j++){
                                if(first){
                                        i=i+a;
                                        j=j+b;
                                        first=false;
                                }
```

```
                                if(pos[i][j]==0){
                                        pos[i][j]=1;     // 座標にコマを置く
                                        Q++;
                                        check();// そのコマの可動範囲を-1 に
                                        if((QMin==0||Q<QMin)&&search(0,0)) {
                                                QMin=Q;
                                                display();
                                                countA = 1;
                                                countB = 0;
                                        }
                                        else if(QMin==Q) {
                                                countA++;
                                                if (countA==100000)
                                                        countB++;
                                        }
                                        start(i,j);
                                        pos[i][j]=0;     // 座標のコマを除外
                                        Q--;
                                        check();
                        }
                }
        }
  }

  void main(){
        clock_t startC,endC;      // 計算時間測定用変数
        startC = clock();
        clear();
        start(0,0);
        endC = clock();
        cout << "2D-N-Queen 問題(N=" << N << ")において解の存在する最小の Q の個数は" << QMin
<< "個" << endl;
        cout << "ユニーク解、バリエーション解全てを含む解(パターン)は" << countB << countA <<
"個" << endl;
        cout << "計算時間は" << (long double)(endC-startC)/CLOCKS_PER_SEC << "秒" << endl;
        cout << "何か入力して終了してください。" << endl;
        int x;
        cin >> x;
        exit(0);
  }
```

2. 3 次元 n クイーンの最小個数問題を解くプログラム

```
/* バックトラック法を用いて 3 次元 n クイーン問題の最小個数解を出すプログラム */

#include<iostream>
#include<time.h>
```

```
#define N 4// チェス盤のサイズ

using namespace std;

int pos[N][N][N];
// 三次元のチェス盤。サイズはN×N×N。最上段、二次元上の左上を0としてpos[z][y][x]。
// xが二次元上の左右、yが二次元上の上下、zが三次元の深さ。
/*
(例)N=3ならば
最上段(z=0)
(0,0,0)(0,0,1)(0,0,2)
(0,1,0)(0,1,1)(0,1,2)
(0,2,0)(0,2,1)(0,2,2)
z=1
(1,0,0)(1,0,1)(1,0,2)
(1,1,0)(1,1,1)(1,1,2)
(1,2,0)(1,2,1)(1,2,2)
最下段(z=2)
(2,0,0)(2,0,1)(2,0,2)
(2,1,0)(2,1,1)(2,1,2)
(2,2,0)(2,2,1)(2,2,2)
*/
long int countA=0;      //      解の数を数える(ユニーク解、バリエーション解全てを含む)
long int countB=0;
int Q=0;        // 現在のクイーンの数
int QMin=0;     // 判明している最小のQの個数

/**
 * 盤をリセットする関数
 */
void clear(){
        for(int i=0;i<N;i++){
                for(int j=0;j<N;j++){
                        for(int k=0;k<N;k++){
                                pos[i][j][k]=0;
                        }
                }
        }
}

/**
 * 盤を表示する関数
 */
void display(){
        for(int i=0;i<N;i++){
                for(int j=0;j<N;j++){
                        for(int k=0;k<N;k++){
```

```
                                if(pos[i][j][k]==1){    // コマがあるマス
                                        cout << "◎";
                                }else if(pos[i][j][k]==0){      // コマがないマス
                                        cout << "□";
                                }else{  // コマを置くことができないマス
                                        cout << "×";
                                }
                        }
                        cout << endl;
                }
                cout << endl;
        }
        cout << "＊＊＊＊＊＊＊＊＊＊" << endl << endl;
}

/**
 * 全てのコマの可動範囲を-1 にする関数
 */
void check(){
        for(int i=0;i<N;i++){   // コマのないマスをクリア
                for(int j=0;j<N;j++){
                        for(int k=0;k<N;k++){
                                if(pos[i][j][k]!=1)pos[i][j][k]=0;
                        }
                }
        }
                for(int i=0;i<N;i++){   // コマのあるマスの可動範囲を-1 に
                for(int j=0;j<N;j++){
                        for(int k=0;k<N;k++){
                                if(pos[i][j][k]==1){
                                        for(int l=0;l<N;l++){
                                                // R ルークの可動範囲。立方体の面方向。
                                                if(pos[i][j][l]!=1)pos[i][j][l]=-1;
        // x 軸方向
                                                if(pos[i][l][k]!=1)pos[i][l][k]=-1;
        // y 軸方向
                                                if(pos[l][j][k]!=1)pos[l][j][k]=-1;
        // z 軸方向
                                                // B ビショップの可動範囲。立方体の辺方
向。
                                                if(j-l>=0       &&      k-l>=0       &&
pos[i][j-l][k-l]!=1)pos[i][j-l][k-l]=-1; // 二次元左上方向。(x,0,0)方向。
                                                if(j-l>=0       &&      k+l<N        &&
pos[i][j-l][k+l]!=1)pos[i][j-l][k+l]=-1; // 二次元右上方向。(x,0,N-1)方向。
                                                if(j+l<N        &&      k+l<N        &&
pos[i][j+l][k+l]!=1)pos[i][j+l][k+l]=-1; // 二次元右下方向。(x,N-1,N-1)方向。
                                                if(j+l<N        &&      k-l>=0       &&
```

```
pos[i][j+1][k-1]!=1)pos[i][j+1][k-1]=-1; // 二次元左下方向。(x,N-1,0)方向。
                                                    if(i-1>=0       &&       j-1>=0       &&
pos[i-1][j-1][k]!=1)pos[i-1][j-1][k]=-1; // (0,0,x)方向。
                                                    if(i-1>=0       &&       k+1<N        &&
pos[i-1][j][k+1]!=1)pos[i-1][j][k+1]=-1; // (0,x,N-1)方向。
                                                    if(i-1>=0       &&       j+1<N        &&
pos[i-1][j+1][k]!=1)pos[i-1][j+1][k]=-1;         // (0,N-1,x)方向。
                                                    if(i-1>=0       &&       k-1>=0       &&
pos[i-1][j][k-1]!=1)pos[i-1][j][k-1]=-1; // (0,x,0)方向。
                                                    if(i+1<N        &&       j-1>=0       &&
pos[i+1][j-1][k]!=1)pos[i+1][j-1][k]=-1;         // (N-1,0,x)方向。
                                                    if(i+1<N        &&       k+1<N        &&
pos[i+1][j][k+1]!=1)pos[i+1][j][k+1]=-1; // (N-1,x,N-1)方向。
                                                    if(i+1<N        &&       j+1<N        &&
pos[i+1][j+1][k]!=1)pos[i+1][j+1][k]=-1; // (N-1,N-1,x)方向。
                                                    if(i+1<N        &&       k-1>=0       &&
pos[i+1][j][k-1]!=1)pos[i+1][j][k-1]=-1;         // (N-1,x,0)方向。
                                                    // U ユニコーンの可動範囲。立方体の頂点
方向。
                                                    if(i-1>=0   &&   j-1>=0   &&   k-1>=0   &&
pos[i-1][j-1][k-1]!=1)pos[i-1][j-1][k-1]=-1; // (0,0,0)方向。
                                                    if(i-1>=0   &&   j-1>=0   &&   k+1<N   &&
pos[i-1][j-1][k+1]!=1)pos[i-1][j-1][k+1]=-1; // (0,0,N-1)方向。
                                                    if(i-1>=0   &&   j+1<N   &&   k+1<N   &&
pos[i-1][j+1][k+1]!=1)pos[i-1][j+1][k+1]=-1; // (0,N-1,N-1)方向。
                                                    if(i-1>=0   &&   j+1<N   &&   k-1>=0   &&
pos[i-1][j+1][k-1]!=1)pos[i-1][j+1][k-1]=-1; // (0,N-1,0)方向。
                                                    if(i+1<N   &&   j-1>=0   &&   k-1>=0   &&
pos[i+1][j-1][k-1]!=1)pos[i+1][j-1][k-1]=-1; // (N-1,0,0)方向。
                                                    if(i+1<N   &&   j-1>=0   &&   k+1<N   &&
pos[i+1][j-1][k+1]!=1)pos[i+1][j-1][k+1]=-1; // (N-1,0,N-1)方向。
                                                    if(i+1<N   &&   j+1<N   &&   k+1<N   &&
pos[i+1][j+1][k+1]!=1)pos[i+1][j+1][k+1]=-1; // (N-1,N-1,N-1)方向。
                                                    if(i+1<N   &&   j+1<N   &&   k-1>=0   &&
pos[i+1][j+1][k-1]!=1)pos[i+1][j+1][k-1]=-1; // (N-1,N-1,0)方向。
                                                }
                                            }
                                        }
                                    }
                                }
                            }
}

/**
 * 空きマスを数える関数
 * @return 空きマスが存在した時に false を返す
 * @return 空きマスが存在しない時に true を返す
 */
```

```
bool search() {
	for(int	i=0;i<N;i++)	for(int	j=0;j<N;j++)	for(int	k=0;k<N;k++)
if(pos[i][j][k]==0)  return false;
	return true;
}

/**
 * 再帰的呼び出しで(a,b,c)からクイーンを設置し、3D-N-Queen の解を求める関数
 */
void start(int a,int b,int c){
	bool first=true;
	for(int i=0;i<N;i++){
		for(int j=0;j<N;j++){
			for(int k=0;k<N;k++){
				if(first){
					i=i+a;
					j=j+b;
					k=k+c;
					first=false;
				}
				if(pos[i][j][k]==0){
					pos[i][j][k]=1; // 座標にコマを置く
					Q++;
					check();	// そのコマの可動範囲を-1 に
					if((QMin==0||Q<QMin)&&search()) {
						QMin=Q;
						display();
						countA = 1;
						countB = 0;
					}
					else if(QMin==Q) {
						countA++;
						if (countA==100000)
							countB++;
					}
					start(i,j,k);
					pos[i][j][k]=0; // 座標のコマを除外
					Q--;
					check();
				}
			}
		}
	}
}
void main(){
	clock_t startC,endC;	// 計算時間測定用変数
	startC = clock();
```

```
        clear();
        start(0,0,0);
        endC = clock();
        cout << "3D-N-Queen問題(N=" << N << ")において解の存在する最小のQの個数は" << QMin
<< "個" << endl;
        cout << "ユニーク解、バリエーション解全てを含む解(パターン)は" << countB <<
countA << "個" << endl;
        cout << "計算時間は" << (long double)(endC-startC)/CLOCKS_PER_SEC << "秒" << endl;
        cout << "何か入力して終了してください。" << endl;
        int x;
        cin >> x;
        exit(0);
}
```