

情報工学科 レポート

実験演習記録			判定・指示		
	年月日時	共同作業者			
1	2019/04/14				
2					
3					
4					
レポート提出記録					
	提出年月日	期限年月日			
初	2019/04/29	2019/04/29			
再					
科目名		テーマ担当教員	学年	学期	単位
情報工学実験2		藤田先生	3	前期	2
テーマ番号 テーマ名		学籍番号 氏 名			
基本的な探索アルゴリズム		16268062 SALIC ERTUGRUL			

1. はじめに

本レポートでは、情報工学実験 2 の第 1 回目のレポートとして、基本的な検索アルゴリズムについての課題に取り組み、そのプログラムの作成・実装および考察について述べる。全ての課題では、**プログラム言語**として **PYTHON** を使用する。

2. 目的

本レポートでは、知識を用いない検索の実行、そして、幅優先検索と深さ優先検索を用いた迷路抜けアルゴリズムの実装を目的とする。

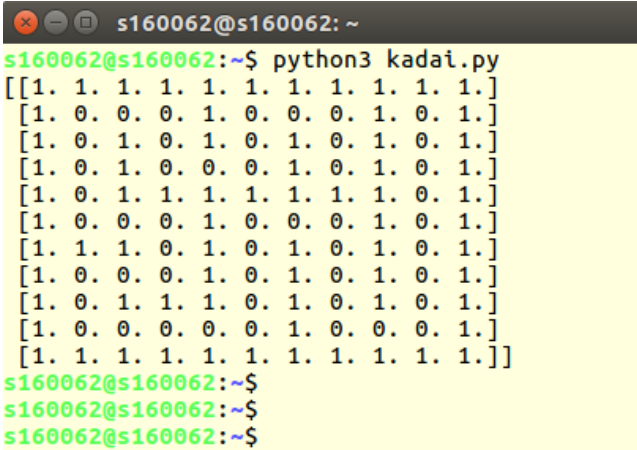
3. 課題 1

本課題では迷路ファイルを読み込み、表示するプログラムを作成する。プログラム言語は PYTHON を使用する。

まずファイルを指定し、指定したファイルから行ごとに読み込みを行い、読み込んだ値を `lines` というリスト型の変数に代入する。Python のライブラリである `numpy` を用いて行列を作成し、その行列に読み込み用のリストである `lines` から値を格納する。そして `print` 関数で `matrix` 行列を表示させる。なお、ファイル読み込みの際に起き得るエラーの対策のために例外処理 (`try-except`) を使用する。エラーが出た場合は“ファイル読み込みにエラーが出た！”と出力し、`sys.exit` でプログラムを終了させる。作成したコードを次に示す。

```
1. try: #ファイル読み込みを行う
2.     f = open("map1010.txt")
3.     lines = f.readlines()
4.     matrix = np.zeros((11, 11))
5.     for i in range(11):
6.         for j in range(11):
7.             matrix[i, j] = lines[i][j]
8.
9. except IOError: #エラーが出たのであればエラーメッセージを出力する
10.     print("ファイル読み込みにエラーが出た！")
11.     sys.exit()
12. finally:
13.     f.close()
14.
15. print(matrix)
```

そのプログラムの実行結果を図 1 に示す.



```
s160062@s160062: ~  
s160062@s160062:~$ python3 kadai.py  
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]  
 [1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1.]  
 [1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]  
 [1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 1.]  
 [1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1.]  
 [1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1.]  
 [1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 1.]  
 [1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1.]  
 [1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1.]  
 [1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1.]  
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]  
s160062@s160062:~$  
s160062@s160062:~$  
s160062@s160062:~$
```

図 1 課題 1 の実行結果 (map1010.txt の読込+出力)

図 1 の結果では, 1 が壁を 0 が進めるマスを表している. 初期状態は行列の要素[1, 1]の「0」である.

4. 課題 2

本課題では, 迷路抜け問題において, 現状態から次に進めるマスを求める next() 関数を作成する.

本課題では検索アルゴリズムにオープンリストとクローズリストを使用する. オープンリストは発見済みでこれから検索するノードを管理するリストである. クローズリストは, 既に検索が終了したノードを管理するリストである.

next 関数では, 現状態から進めるマスの中, クローズリストに含まれないマスを求める. そのために入力値として現状態とクローズリストを設定する.

現状態を指す変数が genzaichi 変数であり, 2 個の要素を含むリストである. その 1 番目の要素が行列の行を, 2 番目の要素が列を指している. クローズリストもリスト型の変数である.

move_list が進めるマスを入れるリストであり, next 関数の返却値でもある.

next 関数では, 現状態の周りのマスを調べ, 調べるマスがもし進めるマスであれば(0 であれば)そしてクローズリストに含まれなければ move_list に入れる処理を行う.

Python 言語で作成したコードを次に示す.

```

1. def next(genzaichi, CL):
2.     # クローズリスト (CL) と現在地の座標 (現状態: genzaichi=[x,y]) を引数としている.
3.     # また, 次の進めるマスの座標 (リスト型) のリストを返却値としている.
4.     move_list = [] # リストの定義
5.     x = genzaichi[0] # x 座標
6.     y = genzaichi[1] # y 座標
7.
8.     for i in (x - 1, x + 1): # 現在位置の上のマスと下のマスを調べる
9.         # クローズリストに存在しなければmove_listに入れる
10.        if matrix[i,y] == 0 and ([i,y] not in CL):
11.            move_list.append([i, y])
12.        for j in (y - 1, y + 1): # 現在位置の右と左のマスを調べる
13.            # クローズリストに存在しなければmove_listに入れる
14.            if matrix[x, j] == 0 and ([x, j] not in CL):
15.                move_list.append([x, j])
16.        return move_list # 次に進めるマスのリストを返却する
17.
18.

```

5. 課題 3

課題 3 では幅優先検索を行うプログラムを作成する. 配布されたテスト用のデータ (map1010.txt) において, スタートからゴールの経路を正しく検索できることを確認する. そのために図 2 のアルゴリズムを用いる.

OL: オープンリスト, *CL*: クローズドリスト, *x*: 現在のノード
next(x, L): *CL* に含まれない *x* の子ノードリストを返す関数

Step 1: $OL := \{a\}$, $CL = \{ \}$
Step 2: $OL = \{ \}$ ならば *false* を返して終了
Step 3: *x* を *OL* から取り出す
Step 4: *x* がゴール状態ならば *True* を返して終了
Step 5: *x* を *CL* を追加
Step 6: *OL* に *next(x, CL)* の返却値を追加
Step 7: goto Step 2

図 2 基本となる検索アルゴリズム

幅優先検索では同じ深さのノードを調べ終わってから次の深さに進むという処理を行う。そのためにオープンリストの構造がキュー構造となる。したがって、課題3ではキューを用いる。

図2のSTEP1はオープンリストに初期状態である[1,1]を入れ、クローズリストを空き状態にすることであった。それを habayusen という関数の中で行う。しかしそこでオープンリストをキュー構造にする。その際に python の queue ライブラリを用いる。その後、検索を開始する。検索を search 関数内で行う。habayusen 関数を次に示す。

```
1. def habayusen():
2.     # STEP-1: OL={1,1}, CL={}
3.     # オープンリスト（これから探索するマスのリスト）をキューにして、幅優先で探索を行う
4.     olist = queue.Queue()
5.     olist.put([1, 1]) # 初期状態：現在地をオープンリストに追加
6.     clist = [] # 初期状態：探索はまだ行っていない
7.     search(olist, clist) # 探索開始
8.     print("幅優先 Close List の中身:")
9.     print(clist)
10.
```

次に search 関数について解説する。search 関数は図2 基本となる検索アルゴリズムのSTEP2～7に該当する処理を行う関数である。

search 関数ではまず、オープンリストが空きかどうかの判定を行う。もし空きであればエラーを出しプログラムを終了させる。これはSTEP2である。

オープンリストが空きでなければ、オープンリストの先頭の要素（すなわち現状態）を取り出す。これはSTEP3である。

取り出したノードがゴール状態であれば “*****ゴールに到着した！！！” と出力してプログラムをreturnする。これはSTEP4である。もしそのノードがゴール状態でなければクローズリストに追加する。(STEP5) また、そのノードから進めるマスを next 関数で求め、オープンリストに追加する。これはSTEP6になる。

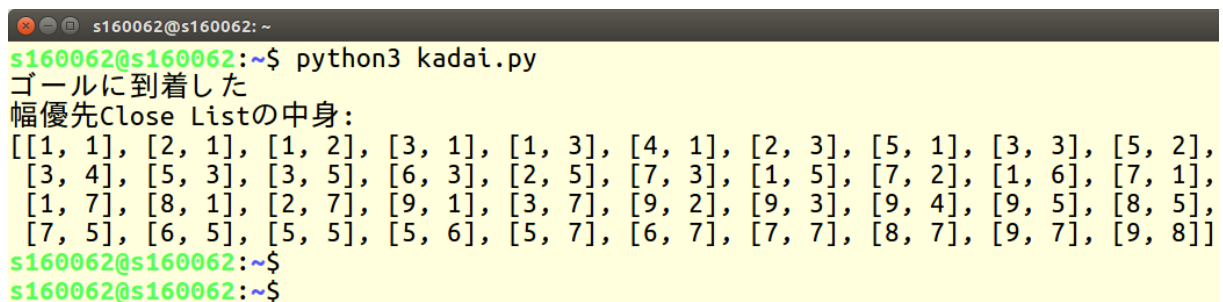
最後にSTEP2に戻る。つまり再帰関数を用いて、search 関数を再び呼び出す。Search 関数のコードを次に示す。

```

1. def search(OL, CL):
2.     # STEP-2: OL={}ならば false を返して終了
3.     if OL.empty():
4.         print("Error: Openlist is empty.¥n")
5.         sys.exit() # プログラム終了
6.
7.     # STEP-3: x を OL から取り出す
8.     else:
9.         genzaichi = OL.get()
10.        # STEP-4: x がゴール状態ならば True を返して終了
11.        if genzaichi[0] == 9 and genzaichi[1] == 9:
12.            print("*****ゴールに到着した!!!*****")
13.            return
14.        # STEP-5: x を CL に追加
15.        else:
16.            CL.append(genzaichi)
17.            # STEP-6: OL に next(genzaichi, CL) の返却値を追加
18.            move_list = next(genzaichi, CL)
19.            for i in move_list:
20.                OL.put(i)
21.
22.        # STEP-7: Step 2 に戻る
23.        search(OL, CL)

```

課題 3 の実行結果を図 3 に示す.



```

s160062@s160062: ~
s160062@s160062:~$ python3 kadai.py
ゴールに到着した
幅優先Close Listの中身:
[[1, 1], [2, 1], [1, 2], [3, 1], [1, 3], [4, 1], [2, 3], [5, 1], [3, 3], [5, 2],
[3, 4], [5, 3], [3, 5], [6, 3], [2, 5], [7, 3], [1, 5], [7, 2], [1, 6], [7, 1],
[1, 7], [8, 1], [2, 7], [9, 1], [3, 7], [9, 2], [9, 3], [9, 4], [9, 5], [8, 5],
[7, 5], [6, 5], [5, 5], [5, 6], [5, 7], [6, 7], [7, 7], [8, 7], [9, 7], [9, 8]]
s160062@s160062:~$
s160062@s160062:~$

```

図 3 課題 3 の実行結果 (幅優先検索の結果)

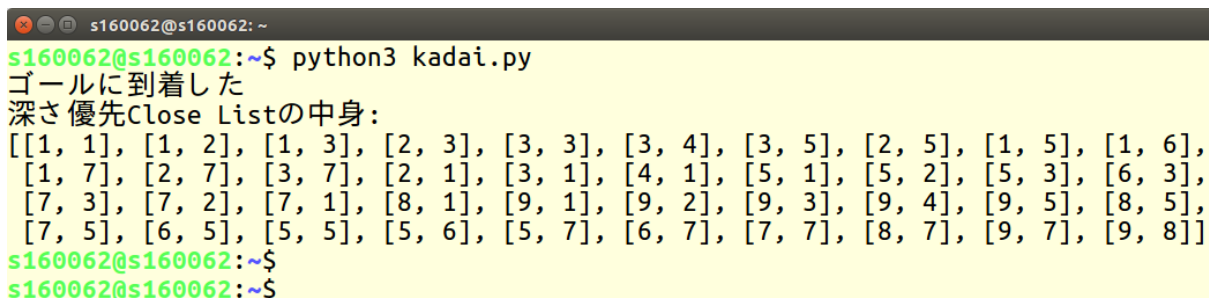
6. 課題 4

課題 4 では深さ優先検索を行う。深さ優先探索では進めるかぎり掘り進め、行き詰まった時に次の枝を調べるという処理を行う。そのためオープンリストの構造がスタックとなる。

深さ優先検索を行うために fukasayusen という関数を定義する。その関数では図 2 の基本となる検索アルゴリズムの STEP1 に該当する処理を行う。つまり、オープンリストに初期状態である [1, 1] を入れ、クローズリストを空き状態にする。オープンリストをスタック構造にするために python の queue ライブラリを用い、olist (オープンリスト) を Lifo.Queue として定義する。Lifo は最後に入った用語が最初に出るという意味でありスタック構造を表している。検索をするのに search 関数を使用するが、search 関数は節 5 の課題 3 において述べているので今回は省略する。深さ優先検索を行うために作成した関数である fukasayusen 関数の中身を次に示す。

```
1. def fukasayusen():
2.     # STEP-1: OL={1,1}, CL={}
3.     olist = queue.LifoQueue() # 深さ優先ではオープンリストをスタックにする
    (LIFO=最後に入れたのが最初に出る)
4.     olist.put([1, 1]) # 初期状態：現在地をオープンリストに追加
5.     clist = [] # 初期状態：探索はまだ行っていない
6.     search(olist, clist) # 探索開始
7.     print("深さ優先 Close List の中身:")
8.     print(clist)
```

またその実行結果を図 4 に示す。



```
s160062@s160062: ~
s160062@s160062:~$ python3 kadai.py
ゴールに到着した
深さ優先Close Listの中身:
[[1, 1], [1, 2], [1, 3], [2, 3], [3, 3], [3, 4], [3, 5], [2, 5], [1, 5], [1, 6],
[1, 7], [2, 7], [3, 7], [2, 1], [3, 1], [4, 1], [5, 1], [5, 2], [5, 3], [6, 3],
[7, 3], [7, 2], [7, 1], [8, 1], [9, 1], [9, 2], [9, 3], [9, 4], [9, 5], [8, 5],
[7, 5], [6, 5], [5, 5], [5, 6], [5, 7], [6, 7], [7, 7], [8, 7], [9, 7], [9, 8]]
s160062@s160062:~$
s160062@s160062:~$
```

図 4 課題 4 深さ優先探索の実行結果

7. 課題 5

課題 5 では、100 個の実験用迷路データを使って、幅優先検索と深さ優先検索の計算量とメモリ量を比較するプログラムを作成する。本課題での計算量は、ゴールまでに next 関数が呼ばれた回数であり、メモリ量は、オープンリストに保存された座標情報の最大サイズである。

課題 5 で作成するプログラムは、まず、1 個のマップを読み込み、その個のマップの行列を作成し、その行列を用いて検索を行い、次のマップを読み取るという再帰的な処理を行う。本プログラムではその処理を file_select という関数で行うことにする。その関数の内容を次に示す。

```

1. def file_select(): # マップを選んで、そのマップにおいて検索を行う関数
2.     global file_number # 現在のマップを記録する変数
3.     if file_number < 100:
4.         try: # ファイル読み込みを行う
5.             f = open("map" + str(file_number))
6.             lines = f.readlines()
7.             for i in range(101):
8.                 for j in range(101):
9.                     matrix[i, j] = lines[i][j]
10.        except IOError: # エラーが出たらエラーメッセージを出力する
11.            print("ファイル読み込みにエラーが出た！")
12.            sys.exit()
13.        finally:
14.            f.close()
15.            habayusen()
16.            fukasayusen()
17.            file_number += 1 # マップのナンバーを記録する変数を1増やす
18.            file_select() # 全てのマップ検索が終わるまで再帰
19.    else:
20.        return

```

file_number が現在のマップのナンバーを指す変数である。マップを読み取り、その内容を101x101の行列に格納する。その行列を用いて幅優先と深さ優先それぞれの検索を行う。次のマップのナンバーを指定して（file_number を1つ増やして）から再帰を行う（file_select() を再度呼び出す）。エラー対策のために try-except の構造で関数を作成する。

本課題で作成するプログラムは課題4のプログラムに類似するところが多くあるので、相違しているところのみについて解説する。

幅優先と深さ優先で検索を行う際に図2の基本となるアルゴリズムに従って検索を行う。なお、計算量とメモリ量を求めるためにSTEP6を次のように編集する。

```

1.     # STEP-6: OL に next(genzaichi, CL)の返却値を追加
2.     move_list = next(genzaichi, CL)
3.     next_count[select_yusen, file_number] += 1
4.
5.     for i in move_list:
6.         OL.put(i)
7.         if OL.qsize() > olist_maxsize[select_yusen, file_number]:
8.             olist_maxsize[select_yusen, file_number] = OL.qsize()

```


`next_count(2,100)` は numpy ライブラリで作られた行列であり、`next_count[0]` は幅優先探索の計算量の配列を、`next_count[1]` は深さ優先探索の計算量を表している。STEP6 の上のコードでは、`next` 関数が呼び出された度に `next_count[select_yusen, file_number]` を 1 増やす。`select_yusen` は幅優先か深さ優先かを選ぶ変数であり、0 か 1 である。`file_number` はマップの番号であり、0~99 のどれかである。また、`olist_maxsize` が存在しており、それはメモリ量、すなわちオープンリストの最大要素数である。

課題 5 では計算量とメモリ量を求めるが、出力としてはそれぞれの平均値、分散値、最悪値と最良値も求める。そのために次の `average` (平均値)、`variance` (分散地) 関数を作成する。最悪値と最良値を求めるのに python で使用できる `max` と `min` 関数を用いる。

```
1. def average(data, temp=0): # 平均値
2.     for i in range(100):
3.         temp += data[i]
4.     return temp / 100
5. def variance(data, temp=0): # 分散地
6.     avrg = average(data) #平均値を計算
7.     for i in range(100):
8.         temp += (data[i] - avrg) ** 2
9.     return temp / 100
```

また、求めた平均値・分散地・最悪値・最良値を出力する。出力を行うコードを次に示す。

```
1. print("¥n 計算量の平均値¥n")
2.     "幅優先  :" + str(average(next_count[0])) + "¥n"
3.     "深さ優先:" + str(average(next_count[1])) + "¥n"
4.
5. print("計算量の分散値¥n")
6.     "幅優先  :" + str(variance(next_count[0])) + "¥n"
7.     "深さ優先:" + str(variance(next_count[1])) + "¥n")
8.
9. print("計算量の最悪値¥n")
10.    "幅優先  :" + str(max(next_count[0])) + "¥n"
11.    "深さ優先:" + str(max(next_count[1])) + "¥n")
12.
13. print("計算量の最良値¥n")
14.    "幅優先  :" + str(min(next_count[0])) + "¥n"
15.    "深さ優先:" + str(min(next_count[1])) + "¥n")
16.
17. print("情報量の平均値¥n")
```

```

18.         "幅優先  :" + str(average(olist_maxsize[0])) + "\n"
19.         "深さ優先:" + str(average(olist_maxsize[1])) + "\n"
20.
21.     print("情報量の分散値\n")
22.         "幅優先  :" + str(variance(olist_maxsize[0])) + "\n"
23.         "深さ優先:" + str(variance(olist_maxsize[1])) + "\n"
24.
25.     print("情報量の最悪値\n")
26.         "幅優先  :" + str(max(olist_maxsize[0])) + "\n"
27.         "深さ優先:" + str(max(olist_maxsize[1])) + "\n"
28.
29.     print("情報量の最良値\n")
30.         "幅優先  :" + str(min(olist_maxsize[0])) + "\n"
31.         "深さ優先:" + str(min(olist_maxsize[1]))

```

課題 5 の実行結果を図 5 に示す。(図は節 8. 考察に貼り付けている.)

8. 考察

課題 1 の考察

課題 1 では指定のファイルから読み込みを行い、その読み込んだデータを表示するプログラムを作成した。図 1 はそのプログラムの実行結果であるが、図 1 より、プログラムが正しく動作したことが分かる。

課題 2 の考察

課題 2 では入力値が現状態とクローズリスト、返却値が次に進めるマス目のリストである next 関数を作成した。課題 2 の next 関数が図 2 の基本となる検索アルゴリズムの基本であり、課題 3・4 および 5 で使用する重要な関数である。なぜならば、クローズリストと一緒に基本検索アルゴリズムの柱であるオープンリストが next 関数の返却値で作成されている。課題 2 に取り組むことによって、検索の処理がどのように行われるのかについて考え、プログラムでその考えを実現させることができた。

課題 3 の考察

課題 3 では habayusen と search 関数で幅優先検索を実装した。その結果は図 3 に示している。また課題 3 では図 2 の基本となるアルゴリズムを使用しており、そして課題 2 で作成した next 関数を用いた。幅優先検索がキューの構造で検索をすることであるので、本課題ではオープンリストをキュー構造にした。プログラム言語としては python を使用したので、キューの使用にあたって python の queue クラスを用いた。大学の 1 年次から勉強してきた C 言語でキュー構造を作る必要があるが、python ではその必要がなく、用意されているクラスを用いることが可能である。本課題の実装によって、プログラム言語の相違点を実際に見ることができた。

また課題 3 ではプログラムを作成する際に図 2 のステップでプログラムを作成した。プログラムを予めステップごとに分けることによって、プログラム作成（コード化）が簡単になるという

ことを実感した。

幅優先検索で検索を行った結果が図 3 となっている。図 3 より、プログラムでゴールに到着できることを確認できる。

課題 4 の考察

課題 4 では fukasayusen と search 関数によって深さ優先検索を行った。その結果が図 4 である。図 4 より、検索でゴールに到着したことが確認できる。

課題 3 の検索を habayusen+search と 2 つに分けていた。habayusen 関数内でオープンリストをキュー構造にして、search 関数で検索を行うという処理を行った。そのように処理を 2 つに分ける（モジュール化）によって、深さ優先検索のコードを書くことが非常に用意になった。なぜならば、同じ search 関数を使用するので、オープンリストの構造をキューからスタックに変えるだけでプログラムが作成できた。モジュール化によってプログラムの作成や計画が用意になることが確認できた。

図 3 は幅優先検索の結果で、課題 4 は深さ優先検索の結果である。それぞれにクローズリストが出力されている。クローズリストの状態より、それぞれの検索の進み方やゴールに到着するまでの経路が分かり、またそれらについて考えることができる。

課題 5 の考察

課題 5 では幅優先検索と深さ優先検索のそれぞれの計算量とメモリ量を比較するプログラムを作成した。そのために配布された 100 個のデータからマップを読み込み、そのマップにおいて検索を行った。その結果を図 5 に示す。

```
@s160062: ~/kadai5
s160062@s160062:~$ cd kadai5
s160062@s160062:~/kadai5$ python3 kadai5.py

計算量の平均値
幅優先    : 3498.47
深さ優先  : 2589.94

計算量の分散値
幅優先    : 603127.7291
深さ優先  : 525807.55640000001

計算量の最悪値
幅優先    : 4994.0
深さ優先  : 4232.0

計算量の最良値
幅優先    : 1737.0
深さ優先  : 868.0

情報量の平均値
幅優先    : 10.4
深さ優先  : 31.85

情報量の分散値
幅優先    : 4.82
深さ優先  : 80.547500000000001

情報量の最悪値
幅優先    : 17.0
深さ優先  : 51.0

情報量の最良値
幅優先    : 6.0
深さ優先  : 15.0
s160062@s160062:~/kadai5$ █
```

図 5 課題 5 の実行結果

計算量の平均値を比較すると、深さ優先検索の平均値が幅優先検索の平均値より低く、計算量の点では優れているように見える。計算量では平均値や分散、最悪値や最良値どれを見ても深さ優先検索のほうが優れているように見える。

メモリ量の点で比較したところ、幅優先検索のほうが倍ぐらい優れていることに気づく。データの平均値や分散、最悪値や最良値どれに関しても幅優先検索が深さ優先検索の2~3倍ぐらいのパフォーマンスを出している。

上記より、深さ優先探索の利点としては、幅優先探索に比べて行う計算数が多くならない傾向があると考えられる。しかし、深さ優先検索では深さが優先されるため、解が初期状態に近いところにある時でも、発見するために無駄な探索を行ってしまうこともある。つまり検索対象のマップが非常に深い場合は、使用するメモリ量も非常に多くなる。一方の幅優先検索の利点としては、解が初期状態に近いところにある時に速く解を発見できるということが挙げられる。しかし、対象のマップがたくさんの分岐を持っている場合はメモリ量が増えると考えられ、処理が遅くなると思われる。計算量も深さ優先検索より大きいと、分岐の多いグラフでは用いないほうが良いと考えられる。

以上より、どちらが効率の良い検索方法なのかということは、対象の検索マップの性質によると思われる。分岐の多さより深さの大きいグラフでは幅優先検索を、深さより分岐の多さの大きいグラフでは深さ優先検索を用いることが効率的ではないかと考えられる。

課題1から5まで、オープンリストとクローズリストの管理について、また、幅優先検索と深さ優先検索について勉強し、そしてプログラムを実際に作成して、実行した。本実験では、基本的な検索アルゴリズムに関して知識を深めることができ、演習することもできた。

9. 参考文献

[1] 藤田桂英, 情報工学実験2 講義資料, 東京農工大学情報工学科3年次科目, 情報工学実験2, 2019年4月14日