ミニレポート訂正箇所

問題3-2において，12567という探索経路が出ているのに，12467と誤表記を行っていたところを訂正しました．ページ番号としては，３５・３６です．

情報工学実験B（メディア処理）
人工知能実験2020ミニレポート

学生番号:09430509

氏名:今田将也

提出日:2020年11月08日

締切日:2020年11月09日

概要（1/2）

今回の人工知能実験では，３年の3学期までに習った講義内容を元に，古典的な人工知能の典型的な問題の一つである，迷路探索問題を解くためのプログラムの作成を行う.

迷路探索に必要な処理は，適切な前提を置くことができれば，ノードの繋がったグラフ探索問題として解くことができる．本レポートでは，迷路探索に必要となる課題を大きく4つに分けて説明をする.

まず，MATLAB Octaveというソフトを用いてプログラムに落とし込むにあたり，制御文や基本的な構文についても理解をする.

また，実験を進めるにあたり必要となるスタックとキューの概念についての復習から初め，行列を利用しての無向グラフや有向グラフについての表現についても説明をしていく.

概要(2/2)

さらに今回は，深さ優先探索と幅優先探索を実装し，比較実験を通して，両者の特徴を理解することを目的として実験を進める.

実装をする探索方法は，単純な探索ではなくクローズドリストやオープンリストを用いて，探索ノードの管理も行うようにする.

そして迷路探索問題を解くためのプログラムをMATLAB　Octaveを用いて実装を行い，結果についても考察を行う.

なお，最適経路探索方式の一つである，Dijkstra's algorithmについても実装を行い，深さ・幅優先探索との比較実験を通して，その特徴についても考察を行い，結果もまとめる.

問題1

MATLAB/Octaveを用いて，次の3つの課題に取り組みなさい．

1. MATLAB/Octaveの基本的な使い方を理解しなさい．
2. スタックとキューの動作を確認し，説明しなさい．
3. 行列を利用して，グラフを表現する方法について，説明しなさい．

問題1の小問題

1. MATLAB/Octaveのソースコードを示しながら，スカラー，ベクトル（配列），行列，構造体について，簡潔に説明せよ.
2. MATLAB/Octaveにおける制御文として，while文とif文の使い方を，実例をあげながら，簡潔に説明せよ.
3. MATLAB/Octaveのソースコードを示しながら，スタックやキューの動作例を示せ.
4. スタックとキューの共通点や差異について，機能面や実装面から考察せよ.
5. MATLAB/Octaveのソースコードを示しながら，1つ以上のグラフを表現せよ.
6. 人工知能実験の課題を進めるために用いた，MATLAB/Octaveの「演算子」や「組み込み関数」の一覧を作成せよ.

1-1. MATLAB/Octaveの基本的な使い方(1/2)

スカラー

変数は，代入した時点で，暗黙的に宣言される.

- x = 10;
- y = x * 3;
- x % x=10
- y % y=10
- y = z * 3

宣言されていない変数を参照すると，エラーが表示される

配列（ベクトル）

配列の場合も，スカラーと同じように，変数に代入して利用できる

- z1 = [10 20 30] %配列

ベクトルの長さや，行列のサイズは，length や size という関数で得られる.

- length(z1)　% ans = 3
- size(z1)　% ans = 1 3

MATLABの場合，原則として**すべての変数は行列である.**

行列の添え字（インデックス）は1から始まる.

- x = [100 200 300];
- x(4) % errorになる
- x(4) = 400　% 代入はできる．大きさが3から4へ拡張される.

1-1. MATLAB/Octaveの基本的な使い方(2/2)

行列

行列のスライス

- A = [10 20 30 40; 50 60 70 80; 90 100 110 120]
- A(2, 3)　% 70 と表示される
- A(:, 3)　% 30 70 110 が縦並びで表示される
- A(2, :)　% 50 60 70 80 が横並びで表示される

行列とスカラーの演算

- A = [1 2 3; 4 5 6] % [1 2 3] の下に [4 5 6] が並ぶ行列
- C = 2 % C = 2　スカラー
- Z = A * C
- % Z = 2 4 6
- %　　8 10 12 Aの全ての要素にCが掛けられている

構造体

- node = struct('id', 1, 'parent', 3);% C言語と同じように参照できる
- node.id　　% ans = 1
- node.parent　% ans = 3

構造体の配列

配列の要素として構造体を格納するイメージ

- nodes(1) = struct('id', 1, 'parent', 3);
- nodes(2) = struct('id', 2, 'parent', 5);
- nodes(2).id　　% ans = 2が出力される
- nodes(1).parent　% ans = 3が出力される

1-2. MATLAB/Octaveにおけるwhile文とif文の使い方(1/2)

if文

式を評価し，式が真 (true) であるときに一連の動作が実行される．結果が空でなく，非ゼロの要素 (論理値または実数値) のみが含まれる場合に，式は true．それ以外の場合はfalse.

else ifではなくelseifとくっつける必要があることに注意

```
n = 100;
if n > 10
    disp('n > 10')
elseif n > 5
    disp('10 > n > 5')
else
    disp('5 > n')
end   % "n > 10“
```

1-2. MATLAB/Octaveにおけるwhile文とif文の使い方(2/2)

While文

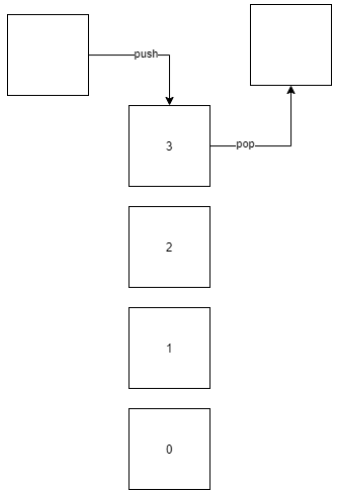
while 文は，if 文と同じように，条件式を書く．条件を満たすと end まで実行される，という点も同じ．ただし，while は end まで辿り着くと，条件式の判定行にジャンプする．

- lim = 1000;
- x = 2;
- while x <= lim
- 　x = x * 2;
- end
- x　% x = 1024

1-3.スタックやキューの動作例(1/3)

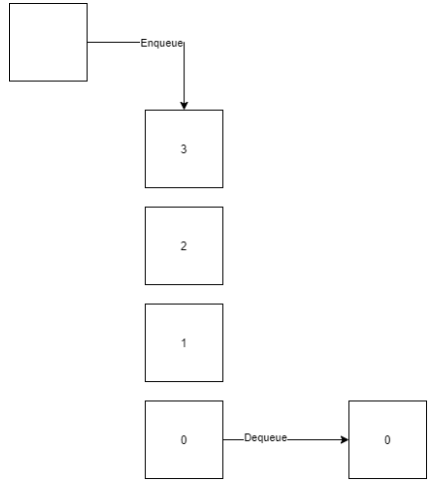
スタック

要素の挿入と削除がリストの先頭だけで行われる LIFO のデータ構造．LIFO (Last In, First Out) とは「最後に入ったものが最初に出てゆく」という意味.



キュー

キューはリストの一方の端で挿入が行われ反対の端で削除が行われる FIFO の構造．FIFO (First In, First Out) とは「最初に入ったものが最初にでてゆく」という意味.



1-3.スタックやキューの動作例(2/3)

MATLABでのスタック動作ソースコード

```
% (1) Initialize
stack_data = [];
% push
stack_data = [1 stack_data];
stack_data = [2 stack_data];
stack_data = [3 stack_data];
stack_data = [4 stack_data];
stack_data
% pop
ret = stack_data(end);
stack_data(end) = [];
ret
stack_data
```

実行結果

```
>> stack
stack_data =
   4   3   2   1
ret =  1
stack_data =
   4   3   2
```

1-3.スタックやキューの動作例(3/3)

MATLABでのキュー動作ソースコード

```
% (1) Initialize
q_data = [];
% push
q_data = [1 q_data];
q_data = [2 q_data];
q_data = [3 q_data];
q_data = [4 q_data];
q_data
% pop
ret = q_data(1);
q_data(1) = [];
ret
q_data
```

実行結果

```
>> queue
q_data =
   4   3   2   1
ret =  4
q_data =
   3   2   1
```

1-4.スタックとキューの共通点や差異について(機能面)

共通点

- スタックもキューも以下のような機能を持つデータ構造のこと
  - 要素 x をデータ構造に追加する
  - データ構造から要素を取り出す

差異

- スタックはデータ構造に入っている要素のうち、最後に push した要素を取り出す
- キューデータ構造に入っている要素のうち、最初に push した要素を取り出す

1-4.スタックとキューの共通点や差異について(実装面)

共通点

- どちらも要素を取り出す，要素を追加するという実装を行う必要がある

差異

- スタックは要素を取り出す際は一番最後に追加されたものの位置を記憶しておき，取り出し，要素数を1減らす
- キューは要素を取り出す際に一番最初に追加されたものを取り出すため，先頭の要素を記憶しておき，取り出し，要素数を1減らす

1-5.グラフを表現(1/2)

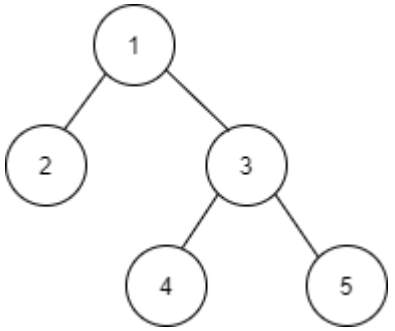
グラフを表現するには隣接行列を表現すれば良い.

隣接行列（Adjacency matrix）とは，「あるノードと別のあるノードが接続しているか否か」という情報を表現した行列．あるノードとあるノードの接続に注目し，それらが接続している場合は1，接続されていない場合は0の要素を持つ.

- 無向グラフでは，各ノード間を互いに行き来できるように1が指定された行列を定義すれば良い
- 有効グラフでは，上から下にのみ遷移しうる木構造であれば目的とは違う方向には進めないように0が指定された行列を定義すれば良い

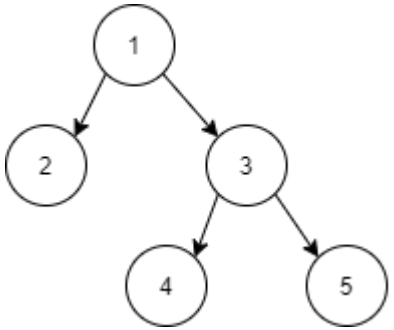
1-5.グラフを表現(2/2)

無向グラフの表現例




```
A = [
      0 1 1 0 0; % 1 -> 2 or 3
      1 0 0 0 0; % 2 -> 1
      1 0 0 1 1;
      0 0 1 0 0;
      0 0 1 0 0
    ];
```

有向グラフの表現例




```
A = [
      0 1 1 0 0; % 1 -> 2 or 3
      0 0 0 0 0; % 2 -> NULL
      0 0 0 1 1; % 3 -> 4 or 5
      0 0 0 0 0; % 4 -> NULL
      0 0 0 0 0  % 5 -> NULL
    ];
```

1-6.「演算子」や「組み込み関数」の一覧(1/2)

基本的な算術演算

- 加算 +
- 減算 -
- 乗算 .*
- 行列乗算　*
- 配列の右除算　./
- 要素単位のべき乗　.^
- 行列のべき乗　^
- 除算後の剰余 (モジュロ演算) mod
- 最も近い小数または整数への丸め round

関係演算子

- 等価性の判定 ==
- 以上かどうかの判別 >=
- より大きいかどうかの判別 >
- 以下かどうかの判別 <=
- 未満かどうかの判別 <
- 不等価の判定 ~=
- 配列の等価性を判別 isequal

1-6.「演算子」や「組み込み関数」の一覧(2/2)

組み込み関数の一例

- length　最大の配列の次元の長さ
- size　配列サイズ
- ndims　配列の次元数
- numel　配列の要素数
- isempty　空かどうか
- ismember　その要素が配列内にあるか

問題1のまとめ

MATLAB/Octaveを用いて，次の3つの課題に取り組みなさい．

1. MATLAB/Octaveの基本的な使い方を理解しなさい．
2. スタックとキューの動作を確認し，説明しなさい．
3. 行列を利用して，グラフを表現する方法について，説明しなさい．

まとめ

- MATLABの基本的な使い方を整理しつつ，演算子を用いて，キューとスタックをMATLABで実現した．
- MATLABでは配列の扱いが簡単で，すぐに実装できた
- グラフの表現も行列を用いることで，有向グラフと無向グラフを実装できることを確認した

問題2

グラフ探索における“深さ優先探索”と“幅優先探索”を，MATLAB/Octaveで実装しなさい．さらに，その探索方式や実装について，問題1の説明と関連付けて，考察しなさい．

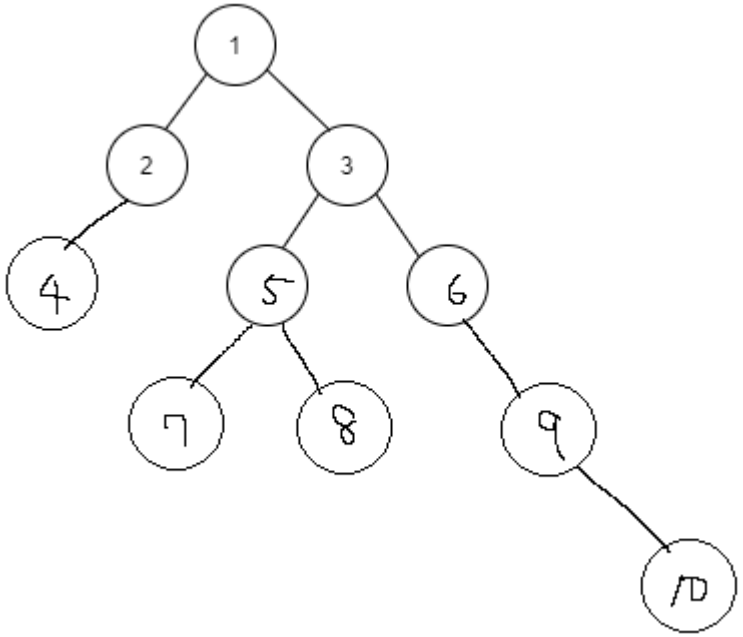
問題2の小問題

1. 深さ有線探索と幅優先探索のソースコードを，それぞれ示しなさい．
2. 探索対象となるグラフを1つ以上用意し，探索結果を示しなさい．
3. 実装や結果の，共通点や差異に注目して，二つの探索方式の，実装面と機能面について考察せよ．

2-1幅優先探索ソースコード

探索をするグラフは以下のグラフである.

スタートは1，ゴール地点は10である.




```
A = [ 0 1 1 0 0 0 0 0 0 0;
      1 0 0 1 0 0 0 0 0 0;
      1 0 0 0 1 1 0 0 0 0;
      0 1 0 0 0 0 0 0 0 0;
      0 0 1 0 0 0 1 1 0 0;
      0 0 1 0 0 0 0 0 1 0;
      0 0 0 0 1 0 0 0 0 0;
      0 0 0 0 1 0 0 0 0 0;
      0 0 0 0 0 1 0 0 0 1;
      0 0 0 0 0 0 0 0 1 0;
];
initial_node = 1;
target_node  = 10;
```

2-1幅優先・深さ優先探索ソースコード

```
% 初期状態をオープンリストに入れる
open_list = [ initial_node ];
% クローズドリストを空にする
closed_list = [];

while ~isempty(open_list)

  cur_node = open_list(1)
  open_list(1) = [];
  closed_list = [closed_list cur_node]

  if cur_node == target_node
    disp('I got a Target node!')
    break;
  end

  hyp_nodes = find(A(cur_node, :) > 0)

  % closed_list実装により，展開済みのノードを再びオープンリストに格納しないようにする
  if ~isempty(closed_list)
    hyp_nodes = hyp_nodes(~ismember(hyp_nodes, closed_list));
  end

  open_list = [ open_list hyp_nodes ]

end
```

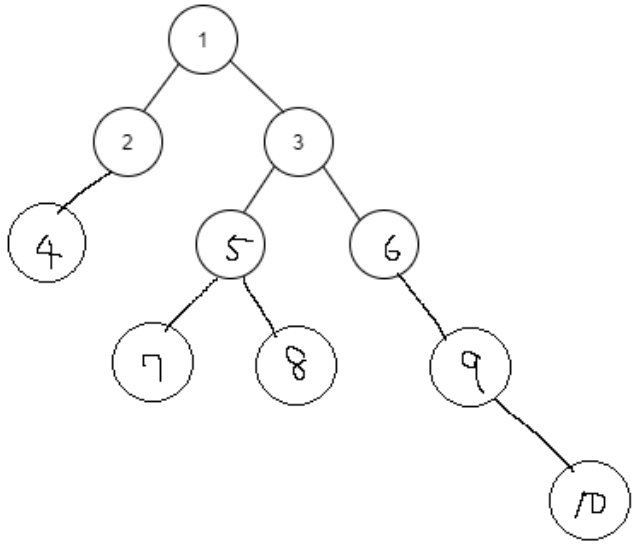
左に示したのは幅優先探索のソースである．

深さ優先にするには，最後の行の，open_listの末尾に追記している動作を，先頭に追記するように変更を加えれば良い．

2-2探索結果

探索に用いたグラフを再度示す.

また右図のうち，上が幅優先探索の探索順，下が深さ優先探索の探索順である




```
open_list =  10
cur_node =  10
closed_list =

    1    2    3    4    5    6    7    8    9   10

I got a Target node!
```

```
open_list =  10
cur_node =  10
closed_list =

    1    2    4    3    5    7    8    6    9   10

I got a Target node!
```

2-3二つの探索方式の共通点

共通点

- オープンリストとクローズドリストを用いた
- 初期ノードと目標ノードを示した
- どちらもしらみつぶし的にすべてのノードを調べている
- 目標が見つからない場合探索が止まる（target_nodeが存在しない値の場合，オープンリストが空になる）

2-3二つの探索方式の差異

差異

- 幅優先はオープンリストの配列の末尾に追加する
  - キューを使う
  - 浅い順番から横に広く探索が行われていた
- 深さ優先はオープンリストの配列の先頭に追加する
  - スタックを使う
  - とりあえず先に繋がっているノードを優先的に調べていた

問題2のまとめ

グラフ探索における“深さ優先探索”と“幅優先探索”を，MATLAB/Octaveで実装しなさい．さらに，その探索方式や実装について，問題1の説明と関連付けて，考察しなさい．

まとめ

- 幅優先と深さ優先をOctaveで実装を行い，その結果のソースコードを示した．
- 探索対象となるグラフを用意し，その探索結果を記載した．
- 両者の探索方法の共通点や差異を，実装面や機能面に着目しその違いについて理解を深めた

問題 3（優先発展♫）

最適経路探索として Dijkstra’s algorithm の実装をおこない，考察を述べよ.

問題3の小問題

1. MATLAB/Octaveを用いた，Dijkstra’s algorithm の実装を示しなさい.
2. 迷路探索の結果として，1つ以上の最適経路探索の，探索結果を示しなさい.（注：自作プログラムによって”結果”を得た，という事実がわかるような説明は必須とする.）
3. 探索結果に対して，考察せよ.（注：少なくとも「探索結果は想定通りか？」という観点での考察は必須とする.）
4. 様々なグラフを，Dijkstra’s algorithm で探索し，さらなる考察を加える

3-1. MATLAB/Octaveを用いた，Dijkstra's algorithm の実装を示しなさい(1/3)

- まずは，迷路問題を解くためのグラフに，コストの概念を導入した
  - 各要素の数値をコストとする
  - コストが小さいノードは経由しやすい
  - コストが大きいノードは経由しにくくなる
  - コスト0のみ「コスト0で接続されている」ではなく，「コスト0は，接続されていない」という意味
- コストに対応できるようにopen_listに入れるノードの構造体を拡張

```
open_list = [ struct('id', initial_node, 'parent', 0, 'cost', 0) ];
```

- 親ノードcur_nodeから，コストを引き継ぐ処理を追加

```
hyp_node_costs = A(cur_node.id, hyp_node_ids) + cur_node.cost;    % <-- new! [2a]
hyp_node_parents = repmat(cur_node.id, size(hyp_node_ids));
hyp_nodes = struct('id', num2cell(hyp_node_ids), ...
                   'parent', num2cell(hyp_node_parents), ...
                   'cost', num2cell(hyp_node_costs));         % <-- update! [2b]
```

3-1. MATLAB/Octaveを用いた，Dijkstra's algorithm の実装を示しなさい(2/3)

- オープンリストは，U-Priority Queue として実装
  - U-Priority Queueは追加するときにソート順を維持する

```
open_list = [open_list hyp_nodes];
[~, idx] = sort([open_list.cost]);              % <-- new! 値，何番目かが帰ってくる
open_list = open_list(idx);                     % <-- new!
[~, idx] = unique([open_list.id], 'first');     % <-- new!
open_list = open_list(sort(idx));               % <-- new!
```

- オープンリストでの重複のチェックは不要になったので，該当するソースコードをコメントアウト
- オープンリストに追加する際，hyp_node（探索候補の配列）空だと挿入する必要がないため，hyp_nodesを更新する前に事前検証処理を追加

```
% Validate: hyp_node_ids  <-- new
if isempty(hyp_node_ids)
  disp('No hypothesis: skip updating open_list')
  continue
end

PrepareNodes
  hyp_node_costs = A(cur_node.id, hyp_node_ids) + cur_node.cost;    % <-- new! [2a]
```

3-1. MATLAB/Octaveを用いた，Dijkstra’s algorithm の実装を示しなさい(3/3)

- 終了条件も構造体のid要素で判別

```
if cur_node.id == target_node    % <-- update!!
    disp('I got Target node!')
    break
end
```

- cur_nodeが，構造体になったため，curnode.idに置き換え

```
hyp_node_ids = find(A(cur_node.id, :) > 0);  % <-- update!!

if ~isempty(open_list)
    hyp_node_ids = hyp_node_ids(~ismember(hyp_node_ids, [open_list.id]))  % <-- update!!
end
```

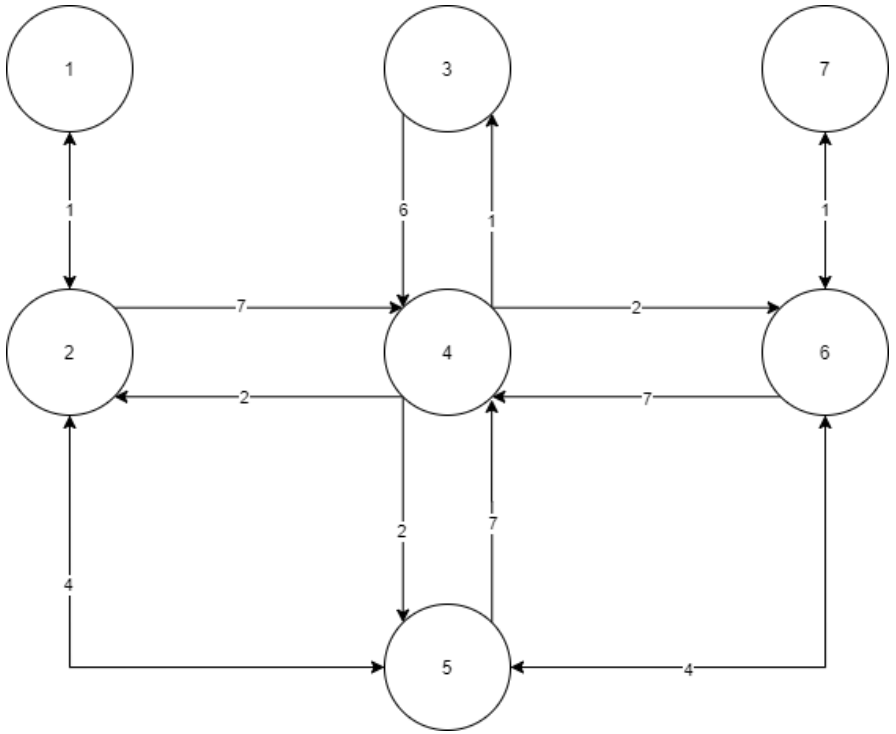
- open_list close_listが空のときの処理を記載

```
% closed_list実装により，展開済みのノードを再びオープンリストに格納しないようにする
if ~isempty(closed_list)
  hyp_node_ids = hyp_node_ids(~ismember(hyp_node_ids, [closed_list.id]));
end

% DeleteNodesInList: remove nodes in open_list
%    **ToDo:** This process will be omitted after 'AI-2'
%if ~isempty(open_list)
%    hyp_nodes = hyp_nodes(~ismember(hyp_nodes, [open_list.id]))
%end
```

3-2.迷路探索の結果として，1つ以上の最適経路探索の，探索結果を示しなさい．(1/2)

探索対象として，講義資料にあった有向グラフを用いた．以下その図である．



実験の設定

✓全部で，7個のノードからなる図である．

✓入るとき，出る時のそれぞれに重みを持たせた有向グラフになっている．

✓初期ノードは1，目標ノードは7で設定をした

```
A = [0 1 0 0 0 0 0;
     1 0 0 7 4 0 0;
     0 0 0 6 0 0 0;
     0 2 1 0 2 2 0;
     0 4 0 7 0 4 0;
     0 0 0 7 4 0 1;
     0 0 0 0 0 1 0;
];
initial_node = 1;
target_node  = 7;
```

3-2.迷路探索の結果として，1つ以上の最適経路探索の，探索結果を示しなさい．(2/2)

結果は以下のようになった．

```
DEBUG_closed_list =

     1     2     5     4     6     3     7
     0     1     2     2     5     4     6
     0     1     5     8     9     9    10

I got Target node!
route =

   1   2   5   6   7
```

【実験結果】

Back-trackingをして経路を示した．

探索した結果として，

1-2-5-6-7のノードを通る結果が表示されていることがわかり，最適な探索が行えているように見える．

3-3.探索結果に対して，考察せよ.

【想定】

- 前述のグラフでは，重みの小さいところを通っていくのが最適な探索経路と言える.
- 目視では，1-2-5-6-7が最適な経路のように見える

【結果】

- 前述の通り，1-2-5-6-7のノードを通る結果が表示されていることがわかった.
- 想定通りの探索が行われていた.

問題3のまとめ

最適経路探索として Dijkstra’s algorithm の実装をおこない，考察を述べよ.

まとめ

- MATLAB/Octaveを用いた，Dijkstra’s algorithm の実装をし，その概要を理解した
- 迷路探索の結果として，1つ以上の最適経路探索の，探索結果を示し，その動作内容を見た
- 探索結果に対して，考察をし，想定とどうだったかについて述べた.

問題４

ゴールまでの経路出力を可能とした“深さ優先探索”と“幅優先探索”を実装し，考察を述べよ.

問題4の小問題

- 深さ優先探索や幅優先探索に対してどのように実装をしたか
- 幅優先探索あるいは深さ優先探索（または，その両方）による経路探索をおこない，その結果と考察を示す．

4-1深さ優先探索や幅優先探索に対してどのように実装をしたか

- 元はダイクストラ法による構造体を用いたものを使用
- オープンリストをソートする必要がないためその記述箇所を消去
- 展開済みのノードを再びオープンリストに格納しないようにするための処理を復帰
- コストが記されているが，深さ優先，幅優先も考慮できないため，実験に用いた行列に何も変化は加えていない．

4-2.幅優先探索

【結果】

１－２－４－６－7を通る探索の結果になった.

```
DEBUG_closed_list =

     1     2     4     5     3     6     7
     0     1     2     2     4     4     6
     0     1     8     5     9    10    11

I got Target node!
route =

    1    2    4    6    7
```

【考察】

幅優先では最適な経路での探索はできない.

コストを検討しない探索方法であるからと考える

4-2.深さ優先探索

【結果】

１－２－４－６－7を通る，幅優先探索の結果と同様になった.

```
DEBUG_closed_list =

     1     2     4     5     3     6     7
     0     1     2     2     4     4     6
     0     1     8     5     9    10    11

I got Target node!
route =

   1   2   4   6   7
```

【考察】

深さ優先探索も幅優先探索と同じ結果であったため，最適な経路での探索はできないといえる

4-2 なぜ2つの探索結果が違うのか

深さ優先探索，幅優先探索ともにある地点からある地点までの最短経路を求めるアルゴリズムで有るといえる

- 両者とも各地点（ノード）間を結ぶ道の条件（コスト）は一定であるか，異なったとしていても考慮ができない

対して，Dijkstra’s algorithmは各地点（ノード）間を結ぶ道の条件（コスト）が一定でなくてもそれを考慮することができるアルゴリズムであるから

- 今回のような，入る時と出るときとでコストが異なるような迷路探索の場合でも有向グラフとして扱うことで考慮することが可能

したがって，そもそものアルゴリズムの特性から，結果が異なる．

しかし，なぜ深さ優先探索，幅優先探索が同じなのかは実験できていない．

問題4のまとめ

ゴールまでの経路出力を可能とした“深さ優先探索”と“幅優先探索”を実装し，考察を述べよ.

まとめ

- 深さ優先探索や幅優先探索に対してどのように実装をしたかを述べた.
- 幅優先探索あるいは深さ優先探索（または，その両方）による経路探索をおこない，その結果と考察を示し，幅優先探索と深さ優先探索では，ダイクストラ法と同様の結果は得られないことがわかった.

まとめ（2/2）

全部で4つの大問題を通し迷路探索について実験を行った.

まず，今回で初めて触れたMATLAB Octaveの基本的な操作について，どのような関数があるか，どのような演算子があるかを調べ，理解を行ったあと，スカラー値の扱い方，制御文の記述方法や行列の表現方法について理解をした．その後，その行列を用いて，どのように無向グラフと有向グラフを表すかの違いについても実験を行い，スタックのキューの実装面および機能面の違いについてもソースコードを示しながら説明をした.

また，幅優先探索，深さ優先探索についても実際に探索対象となるグラフを用意し，それを解くためのプログラムについても両者とも実装を行った後，結果を示した．そこから2つの探索方式の共通点および差異についてもオープンリストとクローズドリストの内容を比較しながら考察を行った．実際にクローズドリストに格納されている順番が異なっていたことがわかった.

まとめ（1/2）

そして，Dijkstra’s algorithmについて，プログラムでの実装を示した．その後，コストが異なるような有向グラフを用意して，実際に解かせた結果を示した．実験前の想定を示し，その結果と照らし合わせた考察をした．

さらに，幅優先探索，深さ優先の経路も示しながら，Dijkstra’s algorithmと同様の迷路を解かせた結果の考察についても述べた．深さ優先と幅優先は同じ結果だったが，Dijkstra’s algorithmとは異なった結果となった．なぜ2つのアルゴリズムと異なったかの考察についても述べたが，深さ優先と幅優先が同じになったことについては疑問が残るため，ノードの多いグラフを用いての比較が必要だった．