

図 7.19 while ループの 8 回目のイテレーション終了後の状態

すべての頂点の color の値が BLACK になっていることが確認できます。また、すべての頂点の dist の値は、頂点 3 へ移動するときに経由する辺の数になっています。

さらに、始点である頂点 3 の vertices[3].pred を除いて、pred の値が 0 ~ 7 のいずれかの数値になっています。

■ print_path 関数 (2つの頂点間の経路の表示) の説明

46 行目～53 行目では、2 つの頂点間の経路を表示する print_path 関数を定義しています。引数として始点となる頂点の MyVertex オブジェクトを変数 src、終点となる頂点の MyVertex オブジェクトを変数 v で受け取ります。頂点 src から頂点 v への経路に含まれる頂点の識別子を表示しますが、始点から中継する頂点を列挙するのではなくて、終点となる頂点 v から始点へ向かって中継する頂点を列挙します。

図 7.20 に print_path 関数の再帰構造を示します。頂点 src から頂点 v まで移動するには頂点 v の先行頂点である v.pred を必ず通過します。そのため、print_path(vertices, src, v) を実行することは、print_path(vertices, src, vertices[v.pred]) を実行して、v.id を表示することと同じです。再帰処理を v.id と src.id が同じになるまで繰り返すことによって、頂点 src から頂点 v までの経路を表示することができます。

print_path(vertices, src, v) を実行すると、まず 47 行目の if 文で頂点 v と頂点 src が同じであるかどうかを判定します。True であれば、48 行目で頂点 v は始点なので src.id を表示して関数を終了します。False であれば、49 行目の elif 文で v.pred が None かどうかを判定します。もし、始点である頂点 src 以外の pred が None であれば、頂点 src から頂点 v へは到達不可能なことを意味します。そのため、True であれば 50 行目で経路が存在しないことの旨を表示します。47 行目と 49 行目の判定式がともに False であれば、else ブロックに入ります。52 行目と 53 行目の命令を実行し、再帰的に同様の処理を行います。

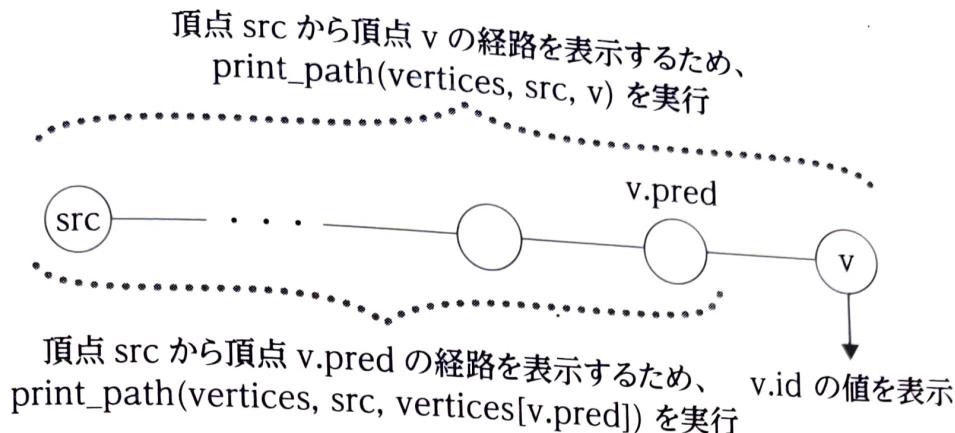


図 7.20 print_path 関数の再帰構造

たとえば、頂点 3 から頂点 7 への経路を表示したいとします。bfs 関数終了後のグラフの状態を表した図 7.19 を見てください。頂点 7 の先行頂点である vertices[7].pred の値が 0 となっています。頂点 0 の先行頂点である vertices[0].pred の値は 1 となっています。すなわち、vertices[7].pred、vertices[0].pred、vertices[1].pred、vertices[2].pred をたどっていくと、7、0、1、2、3、という順番になることがわかります。表示される順番は反対になるので、3、2、1、0、7 という順番で識別子が表示されます。

■ 幅優先探索の実装プログラムの実行

ソースコード 7.7(my_bfs.py) を実行した結果をログ 7.8 に示します。2 行目に頂点 3 から頂点 7 への経路が表示されています。print_path 関数の解説のとおりの経路が表示されていることが確認できます。

ログ 7.8 my_bfs.py プログラムの実行

```

01 $ python3 my_bfs.py
02 頂点3から頂点7への経路: 3 2 1 0 7
03 0, adj = {1, 7}, 2, 3, 1
04 1, adj = {0, 2, 6}, 2, 2, 2
05 2, adj = {1, 3, 5}, 2, 1, 3
06 3, adj = {2, 4, 5}, 2, 0, None
07 4, adj = {3, 5}, 2, 1, 3
08 5, adj = {2, 3, 4, 6}, 2, 1, 3
09 6, adj = {1, 5}, 2, 2, 5
10 7, adj = {0}, 2, 4, 0

```

これらの情報をもとに、3 行目以降は、各頂点の id と adj、color、dist、pred が表示されています。実際に探索結果を視覚化すると、図 7.21 に示すようになります。実は、幅優先探索で探索した頂点を視

覚化すると木構造になります。これを幅優先木 (breadth first tree) と呼びます。始点である頂点3を根として、階層が下に行くたびに距離 dist が 1 増えます。なお、図内での線は pred をもとに 2 つの頂点を接続しています。また、グラフがチェーンのようになっていれば、幅探索木は分岐せずに伸びていきます。

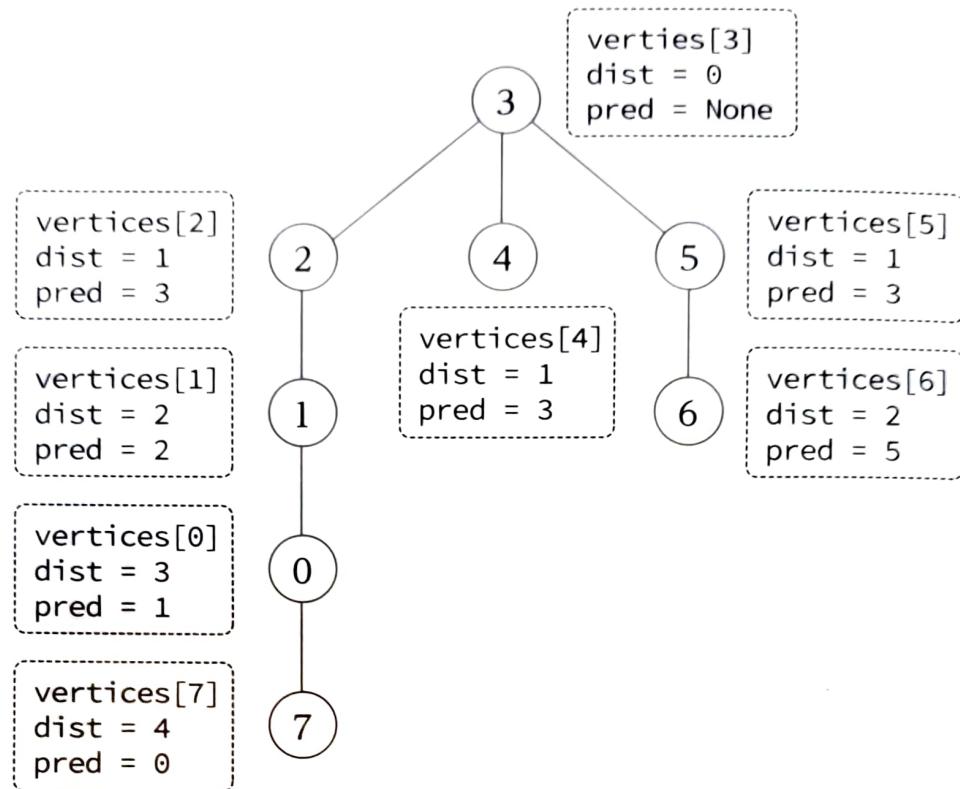


図 7.21 頂点 3 から探索した場合の幅優先木

このように幅優先木を視覚化すると始点から到達可能な各頂点への経路が明確になります。また、異なる頂点を始点として探索した場合、異なる幅優先木になります。

■ 幅優先探索の計算量

グラフ内の頂点の数を V 、辺の数を E とすると、幅優先探索の計算量は $O(V+E)$ となります。ソースコード 7.7 の 34 行目の while ループでは、各頂点を 1 度だけキューである変数 q に入れます。そのため、while ループの繰り返し回数は、頂点数と同じになるため、 $O(V)$ となります。

36 行目の for ループでは、各頂点の隣接頂点を走査しています。頂点 i と j を接続する辺があった場合、 $\text{vertices}[i].adj$ の中に頂点 j が含まれており、 $\text{vertices}[j].adj$ の中に頂点 i が含まれています。そのため、合計で for ループの繰り返し回数は $2E$ 回なので、 $O(E)$ となります。

while ループの中の for ループが含まれていますが、for ループの繰り返し回数である $O(E)$ は、while ループ全体をとおして $O(E)$ です。そのため、アルゴリズム全体の計算量が $O(V+E)$ となります。

7.6

深さ優先探索

深さ優先探索は、始点からより深く頂点をたどることによって、到達可能なすべての頂点を探索するアルゴリズムです。前節で幅優先探索との違いを説明したときに用いた図 7.8 の例を確認してください。

7.6.1 深さ優先探索の概要

深さ優先探索は再帰構造を用いて実装します。そのため、探索状態を記録するための変数が若干増えます。以下のようなクラスメンバを含んだ MyVertex クラスの定義をします。幅優先探索と比べると、dscv と cmpl が新たなクラスメンバとして定義されています。また、変数 id と adj、pred は幅優先探索と同じです。

```
class MyVertex:
    def __init__(self, id):
        self.id = id
        self.adj = set()
        self.color = WHITE
        self.dscv = INFTY
        self.cmpl = INFTY
        self.pred = None
```

探索状態を表す color は、WHITE、GRAY、BLACK の 3 つの状態のうち、いずれかの値を取ります。WHITE は未だ訪れていない頂点、GRAY はすでに訪れたが隣接頂点の探索が終了していない頂点、BLACK は隣接頂点の探索が終了した頂点を意味します。初期状態ではすべての頂点がない頂点、BLACK は隣接頂点の探索が終了した頂点を意味します。初期状態ではすべての頂点が WHITE ですが、探索終了時には始点から到達可能なすべての頂点の color が BLACK になります。

新しく追加した変数 dscv (discovered を省略して dscv) は、初めて頂点を訪れたタイムスタンプ (timestamp) を保存します。すなわち、color が WHITE から GRAY に変化するときのタイムスタンプが dscv に保存されます。タイムスタンプは 0 からはじまる整数値で、始点となる頂点から開始して、辺を経由する毎に 1 つずつ増えていきます。

一方、変数 cmpl (completed を省略して cmpl) は、その頂点の隣接頂点の走査が終了したときのタイムスタンプが保存されます。すなわち、color が GRAY から BLACK に変化するときのタイムスタンプです。

■ 頂点を訪れる順番とタイムスタンプ

前節の幅優先探索との比較で用いた図 7.8 の右側のグラフを見てください。頂点 10、5、2、12、8、14 という順番に訪れます。これにタイムスタンプの情報を加えたグラフを図 7.22 に示します。

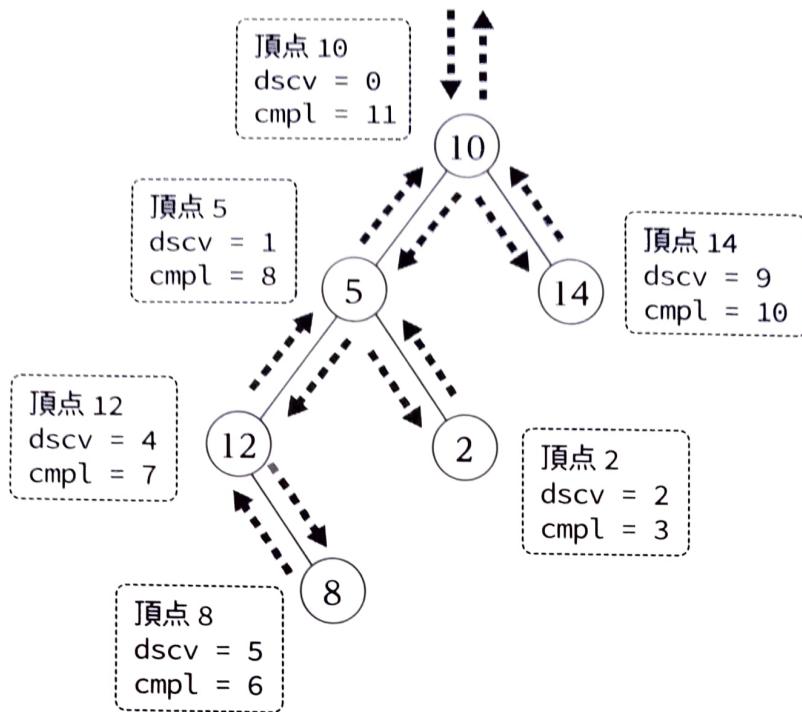


図 7.22 深さ優先探索を用いてグラフ内の頂点を訪れる順番

タイムスタンプを 0 として、頂点 10 から探索を開始します。頂点 10 の dscv は 0 です。隣接頂点が 2 つ以上ある場合は、識別子が小さい頂点から走査します。そのため、次は頂点 5 を探索します。ここで頂点 5 の dscv が 1 に設定されます。次は頂点 2 に進みます。頂点 2 の dscv は 2 です。ここで、頂点 2 はこれまでに訪れていない頂点に隣接していません。そのため、頂点 2 の探索はここで終了です。そのため、頂点 2 の cmpl が 3 に設定されて、もと来た場所の頂点 5 に戻ります。次は頂点 12 に進み、頂点 12 の dscv の値が 4 になります。以降も同様に処理が進みます。

図内の辺をたどっていくと、辺を経由した回数が合計で 10 回になります。そのため、頂点 14 の cmpl の値が 10 になります。最後に始点である頂点 10 の探索完了時間である cmpl の値が 11 になります。

■ 再帰構造の骨格

深さ優先探索では再帰構造を用いますので、その骨格を以下に示します。引数の `vertices` をグラフ内の頂点を表す `MyVertex` オブジェクトの集合、変数 `u` は参照中の頂点である `MyVertex` オブジェクト、変数 `time` はタイムスタンプとします。

```

def dfs(vertices, u, time):
    变数の更新
    for i in u.adj:
        if 隣接頂点のcolorがWHITE:
            变数の更新
            dfs(vertices, vertices[i], time)
            变数の更新
    变数の更新

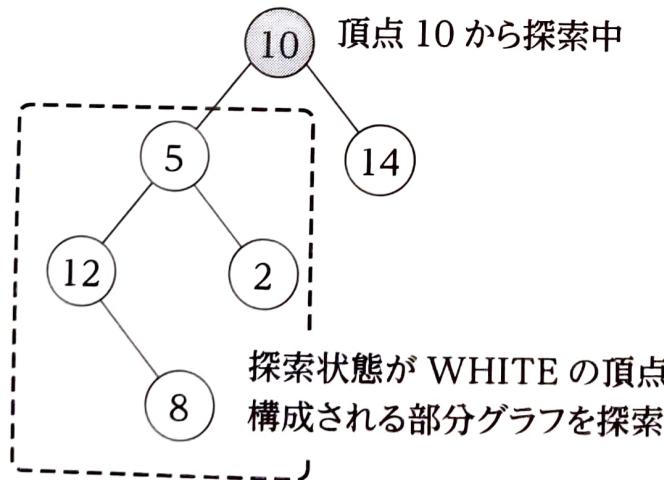
```

細かい変数の更新は無視して、3行目のforループを見てください。ここで頂点uの隣接頂点を一つずつ走査します。ループカウンタがiなので、隣接する頂点のMyVertexはvertices[i]です。もし、vertices[i]のcolorがWHITE(未探索の状態)であれば、再帰的にdfs関数を呼び出します。第2引数にvertices[i]を指定します。

視覚化すると図7.23のようになります。図の左側に示すように、①頂点10を始点として、dfs(vertices, vertices[10], 0)を実行します。現在探索中の頂点は10なので、vertices[10].colorはGRAYです。3行目のforループで、隣接する頂点5をまず走査します。

変数などを更新して、②dfs(vertices, vertices[5], 1)を呼び出します。図の左側の点線で囲んだ箇所を見ると、colorがWHITEで構成される部分グラフになっていることがわかります。関数の再帰呼び出しでは、図の右側に示すように、この部分グラフに対して、頂点5から深さ優先探索を実行しているのです。

① dfs(vertices, vertices[10], 0)を実行



② 再帰的に dfs(vertices, vertices[5], 1) を実行

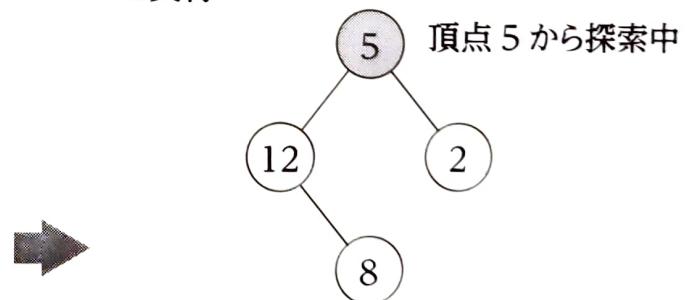


図7.23 深さ優先探索の再帰構造

点線で囲んだ部分グラフの探索がすべて終了すれば、いったん頂点10に戻ってきて、隣接頂点である頂点14の探索を行います。このように再帰構造を用いて、深さ優先探索を実装できます。

7.6.2 深さ優先探索の実装例

ソースコード 7.9 (my_dfs.py) に深さ優先探索の実装例を示します。前節と同様に 8 つの頂点と 10 個の辺から構成されるグラフを生成し、ある頂点から到達可能なすべての頂点への経路を探索するプログラムです。

ソースコード 7.9 深さ優先探索アルゴリズム

~/ohm/ch7/my_dfs.py

ソースコードの概要

2 行目～4 行目	頂点の探索状態を表すグローバル変数を定義
9 行目～24 行目	グラフ内の頂点を表す MyVertex クラスの定義
27 行目～39 行目	深さ優先探索を実行する dfs 関数の定義
42 行目～49 行目	2 つの頂点間の経路を表示する print_path 関数の定義
52 行目～54 行目	2 つの頂点を辺で接続する connect 関数の定義
56 行目～87 行目	main 関数の定義

2 行目～4 行目のグローバル変数の宣言と 7 行目の無限大の定義、42 行目～49 行目の print_path 関数、52 行目～54 行目の connect 関数は幅優先探索と同じです。

```

01 # 状態の種類を定義
02 WHITE = 0
03 GRAY = 1
04 BLACK = 2
05
06 # 無限大の定義
07 INFTY = 2**31 - 1
08
09 class MyVertex:
10     def __init__(self, id):
11         self.id = id
12         self.adj = set()
13         self.color = WHITE
14         self.dscv = INFTY
15         self.cmpl = INFTY
16         self.pred = None
17
18     # 頂点の情報を表示
19     def __str__(self):
20         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
21
22     # 頂点を接続
23     def connect(self, v, w):
24         self.adj.add(v)
25         v.adj.add(w)
26
27     # 頂点を探索状態に設定
28     def set_color(self, c):
29         self.color = c
30
31     # 頂点を探索状態から解放
32     def clear(self):
33         self.color = WHITE
34         self.dscv = INFTY
35         self.cmpl = INFTY
36         self.pred = None
37
38     # 頂点が探索可能か確認
39     def is_gray(self):
40         return self.color == GRAY
41
42     # 頂点が探索済みか確認
43     def is_black(self):
44         return self.color == BLACK
45
46     # 頂点の隣接頂点を取得
47     def get_adj(self):
48         return self.adj
49
50     # 頂点の探索距離を取得
51     def get_dscv(self):
52         return self.dscv
53
54     # 頂点の探索済み距離を取得
55     def get_cmpl(self):
56         return self.cmpl
57
58     # 頂点の前駆頂点を取得
59     def get_pred(self):
60         return self.pred
61
62     # 頂点の情報を表示
63     def __str__(self):
64         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
65
66     # 頂点を接続
67     def connect(self, v, w):
68         self.adj.add(v)
69         v.adj.add(w)
70
71     # 頂点を探索状態に設定
72     def set_color(self, c):
73         self.color = c
74
75     # 頂点を探索状態から解放
76     def clear(self):
77         self.color = WHITE
78         self.dscv = INFTY
79         self.cmpl = INFTY
80         self.pred = None
81
82     # 頂点が探索可能か確認
83     def is_gray(self):
84         return self.color == GRAY
85
86     # 頂点が探索済みか確認
87     def is_black(self):
88         return self.color == BLACK
89
90     # 頂点の隣接頂点を取得
91     def get_adj(self):
92         return self.adj
93
94     # 頂点の探索距離を取得
95     def get_dscv(self):
96         return self.dscv
97
98     # 頂点の探索済み距離を取得
99     def get_cmpl(self):
100        return self.cmpl
101
102     # 頂点の前駆頂点を取得
103     def get_pred(self):
104        return self.pred
105
106     # 頂点の情報を表示
107     def __str__(self):
108         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
109
110     # 頂点を接続
111     def connect(self, v, w):
112         self.adj.add(v)
113         v.adj.add(w)
114
115     # 頂点を探索状態に設定
116     def set_color(self, c):
117         self.color = c
118
119     # 頂点を探索状態から解放
120     def clear(self):
121         self.color = WHITE
122         self.dscv = INFTY
123         self.cmpl = INFTY
124         self.pred = None
125
126     # 頂点が探索可能か確認
127     def is_gray(self):
128         return self.color == GRAY
129
130     # 頂点が探索済みか確認
131     def is_black(self):
132         return self.color == BLACK
133
134     # 頂点の隣接頂点を取得
135     def get_adj(self):
136         return self.adj
137
138     # 頂点の探索距離を取得
139     def get_dscv(self):
140         return self.dscv
141
142     # 頂点の探索済み距離を取得
143     def get_cmpl(self):
144         return self.cmpl
145
146     # 頂点の前駆頂点を取得
147     def get_pred(self):
148         return self.pred
149
150     # 頂点の情報を表示
151     def __str__(self):
152         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
153
154     # 頂点を接続
155     def connect(self, v, w):
156         self.adj.add(v)
157         v.adj.add(w)
158
159     # 頂点を探索状態に設定
160     def set_color(self, c):
161         self.color = c
162
163     # 頂点を探索状態から解放
164     def clear(self):
165         self.color = WHITE
166         self.dscv = INFTY
167         self.cmpl = INFTY
168         self.pred = None
169
170     # 頂点が探索可能か確認
171     def is_gray(self):
172         return self.color == GRAY
173
174     # 頂点が探索済みか確認
175     def is_black(self):
176         return self.color == BLACK
177
178     # 頂点の隣接頂点を取得
179     def get_adj(self):
180         return self.adj
181
182     # 頂点の探索距離を取得
183     def get_dscv(self):
184         return self.dscv
185
186     # 頂点の探索済み距離を取得
187     def get_cmpl(self):
188         return self.cmpl
189
190     # 頂点の前駆頂点を取得
191     def get_pred(self):
192         return self.pred
193
194     # 頂点の情報を表示
195     def __str__(self):
196         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
197
198     # 頂点を接続
199     def connect(self, v, w):
200         self.adj.add(v)
201         v.adj.add(w)
202
203     # 頂点を探索状態に設定
204     def set_color(self, c):
205         self.color = c
206
207     # 頂点を探索状態から解放
208     def clear(self):
209         self.color = WHITE
210         self.dscv = INFTY
211         self.cmpl = INFTY
212         self.pred = None
213
214     # 頂点が探索可能か確認
215     def is_gray(self):
216         return self.color == GRAY
217
218     # 頂点が探索済みか確認
219     def is_black(self):
220         return self.color == BLACK
221
222     # 頂点の隣接頂点を取得
223     def get_adj(self):
224         return self.adj
225
226     # 頂点の探索距離を取得
227     def get_dscv(self):
228         return self.dscv
229
230     # 頂点の探索済み距離を取得
231     def get_cmpl(self):
232         return self.cmpl
233
234     # 頂点の前駆頂点を取得
235     def get_pred(self):
236         return self.pred
237
238     # 頂点の情報を表示
239     def __str__(self):
240         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
241
242     # 頂点を接続
243     def connect(self, v, w):
244         self.adj.add(v)
245         v.adj.add(w)
246
247     # 頂点を探索状態に設定
248     def set_color(self, c):
249         self.color = c
250
251     # 頂点を探索状態から解放
252     def clear(self):
253         self.color = WHITE
254         self.dscv = INFTY
255         self.cmpl = INFTY
256         self.pred = None
257
258     # 頂点が探索可能か確認
259     def is_gray(self):
260         return self.color == GRAY
261
262     # 頂点が探索済みか確認
263     def is_black(self):
264         return self.color == BLACK
265
266     # 頂点の隣接頂点を取得
267     def get_adj(self):
268         return self.adj
269
270     # 頂点の探索距離を取得
271     def get_dscv(self):
272         return self.dscv
273
274     # 頂点の探索済み距離を取得
275     def get_cmpl(self):
276         return self.cmpl
277
278     # 頂点の前駆頂点を取得
279     def get_pred(self):
280         return self.pred
281
282     # 頂点の情報を表示
283     def __str__(self):
284         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
285
286     # 頂点を接続
287     def connect(self, v, w):
288         self.adj.add(v)
289         v.adj.add(w)
290
291     # 頂点を探索状態に設定
292     def set_color(self, c):
293         self.color = c
294
295     # 頂点を探索状態から解放
296     def clear(self):
297         self.color = WHITE
298         self.dscv = INFTY
299         self.cmpl = INFTY
300         self.pred = None
301
302     # 頂点が探索可能か確認
303     def is_gray(self):
304         return self.color == GRAY
305
306     # 頂点が探索済みか確認
307     def is_black(self):
308         return self.color == BLACK
309
310     # 頂点の隣接頂点を取得
311     def get_adj(self):
312         return self.adj
313
314     # 頂点の探索距離を取得
315     def get_dscv(self):
316         return self.dscv
317
318     # 頂点の探索済み距離を取得
319     def get_cmpl(self):
320         return self.cmpl
321
322     # 頂点の前駆頂点を取得
323     def get_pred(self):
324         return self.pred
325
326     # 頂点の情報を表示
327     def __str__(self):
328         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
329
330     # 頂点を接続
331     def connect(self, v, w):
332         self.adj.add(v)
333         v.adj.add(w)
334
335     # 頂点を探索状態に設定
336     def set_color(self, c):
337         self.color = c
338
339     # 頂点を探索状態から解放
340     def clear(self):
341         self.color = WHITE
342         self.dscv = INFTY
343         self.cmpl = INFTY
344         self.pred = None
345
346     # 頂点が探索可能か確認
347     def is_gray(self):
348         return self.color == GRAY
349
350     # 頂点が探索済みか確認
351     def is_black(self):
352         return self.color == BLACK
353
354     # 頂点の隣接頂点を取得
355     def get_adj(self):
356         return self.adj
357
358     # 頂点の探索距離を取得
359     def get_dscv(self):
360         return self.dscv
361
362     # 頂点の探索済み距離を取得
363     def get_cmpl(self):
364         return self.cmpl
365
366     # 頂点の前駆頂点を取得
367     def get_pred(self):
368         return self.pred
369
370     # 頂点の情報を表示
371     def __str__(self):
372         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
373
374     # 頂点を接続
375     def connect(self, v, w):
376         self.adj.add(v)
377         v.adj.add(w)
378
379     # 頂点を探索状態に設定
380     def set_color(self, c):
381         self.color = c
382
383     # 頂点を探索状態から解放
384     def clear(self):
385         self.color = WHITE
386         self.dscv = INFTY
387         self.cmpl = INFTY
388         self.pred = None
389
390     # 頂点が探索可能か確認
391     def is_gray(self):
392         return self.color == GRAY
393
394     # 頂点が探索済みか確認
395     def is_black(self):
396         return self.color == BLACK
397
398     # 頂点の隣接頂点を取得
399     def get_adj(self):
400         return self.adj
401
402     # 頂点の探索距離を取得
403     def get_dscv(self):
404         return self.dscv
405
406     # 頂点の探索済み距離を取得
407     def get_cmpl(self):
408         return self.cmpl
409
410     # 頂点の前駆頂点を取得
411     def get_pred(self):
412         return self.pred
413
414     # 頂点の情報を表示
415     def __str__(self):
416         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
417
418     # 頂点を接続
419     def connect(self, v, w):
420         self.adj.add(v)
421         v.adj.add(w)
422
423     # 頂点を探索状態に設定
424     def set_color(self, c):
425         self.color = c
426
427     # 頂点を探索状態から解放
428     def clear(self):
429         self.color = WHITE
430         self.dscv = INFTY
431         self.cmpl = INFTY
432         self.pred = None
433
434     # 頂点が探索可能か確認
435     def is_gray(self):
436         return self.color == GRAY
437
438     # 頂点が探索済みか確認
439     def is_black(self):
440         return self.color == BLACK
441
442     # 頂点の隣接頂点を取得
443     def get_adj(self):
444         return self.adj
445
446     # 頂点の探索距離を取得
447     def get_dscv(self):
448         return self.dscv
449
450     # 頂点の探索済み距離を取得
451     def get_cmpl(self):
452         return self.cmpl
453
454     # 頂点の前駆頂点を取得
455     def get_pred(self):
456         return self.pred
457
458     # 頂点の情報を表示
459     def __str__(self):
460         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
461
462     # 頂点を接続
463     def connect(self, v, w):
464         self.adj.add(v)
465         v.adj.add(w)
466
467     # 頂点を探索状態に設定
468     def set_color(self, c):
469         self.color = c
470
471     # 頂点を探索状態から解放
472     def clear(self):
473         self.color = WHITE
474         self.dscv = INFTY
475         self.cmpl = INFTY
476         self.pred = None
477
478     # 頂点が探索可能か確認
479     def is_gray(self):
480         return self.color == GRAY
481
482     # 頂点が探索済みか確認
483     def is_black(self):
484         return self.color == BLACK
485
486     # 頂点の隣接頂点を取得
487     def get_adj(self):
488         return self.adj
489
490     # 頂点の探索距離を取得
491     def get_dscv(self):
492         return self.dscv
493
494     # 頂点の探索済み距離を取得
495     def get_cmpl(self):
496         return self.cmpl
497
498     # 頂点の前駆頂点を取得
499     def get_pred(self):
500         return self.pred
501
502     # 頂点の情報を表示
503     def __str__(self):
504         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
505
506     # 頂点を接続
507     def connect(self, v, w):
508         self.adj.add(v)
509         v.adj.add(w)
510
511     # 頂点を探索状態に設定
512     def set_color(self, c):
513         self.color = c
514
515     # 頂点を探索状態から解放
516     def clear(self):
517         self.color = WHITE
518         self.dscv = INFTY
519         self.cmpl = INFTY
520         self.pred = None
521
522     # 頂点が探索可能か確認
523     def is_gray(self):
524         return self.color == GRAY
525
526     # 頂点が探索済みか確認
527     def is_black(self):
528         return self.color == BLACK
529
530     # 頂点の隣接頂点を取得
531     def get_adj(self):
532         return self.adj
533
534     # 頂点の探索距離を取得
535     def get_dscv(self):
536         return self.dscv
537
538     # 頂点の探索済み距離を取得
539     def get_cmpl(self):
540         return self.cmpl
541
542     # 頂点の前駆頂点を取得
543     def get_pred(self):
544         return self.pred
545
546     # 頂点の情報を表示
547     def __str__(self):
548         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
549
550     # 頂点を接続
551     def connect(self, v, w):
552         self.adj.add(v)
553         v.adj.add(w)
554
555     # 頂点を探索状態に設定
556     def set_color(self, c):
557         self.color = c
558
559     # 頂点を探索状態から解放
560     def clear(self):
561         self.color = WHITE
562         self.dscv = INFTY
563         self.cmpl = INFTY
564         self.pred = None
565
566     # 頂点が探索可能か確認
567     def is_gray(self):
568         return self.color == GRAY
569
570     # 頂点が探索済みか確認
571     def is_black(self):
572         return self.color == BLACK
573
574     # 頂点の隣接頂点を取得
575     def get_adj(self):
576         return self.adj
577
578     # 頂点の探索距離を取得
579     def get_dscv(self):
580         return self.dscv
581
582     # 頂点の探索済み距離を取得
583     def get_cmpl(self):
584         return self.cmpl
585
586     # 頂点の前駆頂点を取得
587     def get_pred(self):
588         return self.pred
589
590     # 頂点の情報を表示
591     def __str__(self):
592         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
593
594     # 頂点を接続
595     def connect(self, v, w):
596         self.adj.add(v)
597         v.adj.add(w)
598
599     # 頂点を探索状態に設定
600     def set_color(self, c):
601         self.color = c
602
603     # 頂点を探索状態から解放
604     def clear(self):
605         self.color = WHITE
606         self.dscv = INFTY
607         self.cmpl = INFTY
608         self.pred = None
609
610     # 頂点が探索可能か確認
611     def is_gray(self):
612         return self.color == GRAY
613
614     # 頂点が探索済みか確認
615     def is_black(self):
616         return self.color == BLACK
617
618     # 頂点の隣接頂点を取得
619     def get_adj(self):
620         return self.adj
621
622     # 頂点の探索距離を取得
623     def get_dscv(self):
624         return self.dscv
625
626     # 頂点の探索済み距離を取得
627     def get_cmpl(self):
628         return self.cmpl
629
630     # 頂点の前駆頂点を取得
631     def get_pred(self):
632         return self.pred
633
634     # 頂点の情報を表示
635     def __str__(self):
636         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
637
638     # 頂点を接続
639     def connect(self, v, w):
640         self.adj.add(v)
641         v.adj.add(w)
642
643     # 頂点を探索状態に設定
644     def set_color(self, c):
645         self.color = c
646
647     # 頂点を探索状態から解放
648     def clear(self):
649         self.color = WHITE
650         self.dscv = INFTY
651         self.cmpl = INFTY
652         self.pred = None
653
654     # 頂点が探索可能か確認
655     def is_gray(self):
656         return self.color == GRAY
657
658     # 頂点が探索済みか確認
659     def is_black(self):
660         return self.color == BLACK
661
662     # 頂点の隣接頂点を取得
663     def get_adj(self):
664         return self.adj
665
666     # 頂点の探索距離を取得
667     def get_dscv(self):
668         return self.dscv
669
670     # 頂点の探索済み距離を取得
671     def get_cmpl(self):
672         return self.cmpl
673
674     # 頂点の前駆頂点を取得
675     def get_pred(self):
676         return self.pred
677
678     # 頂点の情報を表示
679     def __str__(self):
680         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
681
682     # 頂点を接続
683     def connect(self, v, w):
684         self.adj.add(v)
685         v.adj.add(w)
686
687     # 頂点を探索状態に設定
688     def set_color(self, c):
689         self.color = c
690
691     # 頂点を探索状態から解放
692     def clear(self):
693         self.color = WHITE
694         self.dscv = INFTY
695         self.cmpl = INFTY
696         self.pred = None
697
698     # 頂点が探索可能か確認
699     def is_gray(self):
700         return self.color == GRAY
701
702     # 頂点が探索済みか確認
703     def is_black(self):
704         return self.color == BLACK
705
706     # 頂点の隣接頂点を取得
707     def get_adj(self):
708         return self.adj
709
710     # 頂点の探索距離を取得
711     def get_dscv(self):
712         return self.dscv
713
714     # 頂点の探索済み距離を取得
715     def get_cmpl(self):
716         return self.cmpl
717
718     # 頂点の前駆頂点を取得
719     def get_pred(self):
720         return self.pred
721
722     # 頂点の情報を表示
723     def __str__(self):
724         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
725
726     # 頂点を接続
727     def connect(self, v, w):
728         self.adj.add(v)
729         v.adj.add(w)
730
731     # 頂点を探索状態に設定
732     def set_color(self, c):
733         self.color = c
734
735     # 頂点を探索状態から解放
736     def clear(self):
737         self.color = WHITE
738         self.dscv = INFTY
739         self.cmpl = INFTY
740         self.pred = None
741
742     # 頂点が探索可能か確認
743     def is_gray(self):
744         return self.color == GRAY
745
746     # 頂点が探索済みか確認
747     def is_black(self):
748         return self.color == BLACK
749
750     # 頂点の隣接頂点を取得
751     def get_adj(self):
752         return self.adj
753
754     # 頂点の探索距離を取得
755     def get_dscv(self):
756         return self.dscv
757
758     # 頂点の探索済み距離を取得
759     def get_cmpl(self):
760         return self.cmpl
761
762     # 頂点の前駆頂点を取得
763     def get_pred(self):
764         return self.pred
765
766     # 頂点の情報を表示
767     def __str__(self):
768         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
769
770     # 頂点を接続
771     def connect(self, v, w):
772         self.adj.add(v)
773         v.adj.add(w)
774
775     # 頂点を探索状態に設定
776     def set_color(self, c):
777         self.color = c
778
779     # 頂点を探索状態から解放
780     def clear(self):
781         self.color = WHITE
782         self.dscv = INFTY
783         self.cmpl = INFTY
784         self.pred = None
785
786     # 頂点が探索可能か確認
787     def is_gray(self):
788         return self.color == GRAY
789
790     # 頂点が探索済みか確認
791     def is_black(self):
792         return self.color == BLACK
793
794     # 頂点の隣接頂点を取得
795     def get_adj(self):
796         return self.adj
797
798     # 頂点の探索距離を取得
799     def get_dscv(self):
800         return self.dscv
801
802     # 頂点の探索済み距離を取得
803     def get_cmpl(self):
804         return self.cmpl
805
806     # 頂点の前駆頂点を取得
807     def get_pred(self):
808         return self.pred
809
810     # 頂点の情報を表示
811     def __str__(self):
812         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
813
814     # 頂点を接続
815     def connect(self, v, w):
816         self.adj.add(v)
817         v.adj.add(w)
818
819     # 頂点を探索状態に設定
820     def set_color(self, c):
821         self.color = c
822
823     # 頂点を探索状態から解放
824     def clear(self):
825         self.color = WHITE
826         self.dscv = INFTY
827         self.cmpl = INFTY
828         self.pred = None
829
830     # 頂点が探索可能か確認
831     def is_gray(self):
832         return self.color == GRAY
833
834     # 頂点が探索済みか確認
835     def is_black(self):
836         return self.color == BLACK
837
838     # 頂点の隣接頂点を取得
839     def get_adj(self):
840         return self.adj
841
842     # 頂点の探索距離を取得
843     def get_dscv(self):
844         return self.dscv
845
846     # 頂点の探索済み距離を取得
847     def get_cmpl(self):
848         return self.cmpl
849
850     # 頂点の前駆頂点を取得
851     def get_pred(self):
852         return self.pred
853
854     # 頂点の情報を表示
855     def __str__(self):
856         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
857
858     # 頂点を接続
859     def connect(self, v, w):
860         self.adj.add(v)
861         v.adj.add(w)
862
863     # 頂点を探索状態に設定
864     def set_color(self, c):
865         self.color = c
866
867     # 頂点を探索状態から解放
868     def clear(self):
869         self.color = WHITE
870         self.dscv = INFTY
871         self.cmpl = INFTY
872         self.pred = None
873
874     # 頂点が探索可能か確認
875     def is_gray(self):
876         return self.color == GRAY
877
878     # 頂点が探索済みか確認
879     def is_black(self):
880         return self.color == BLACK
881
882     # 頂点の隣接頂点を取得
883     def get_adj(self):
884         return self.adj
885
886     # 頂点の探索距離を取得
887     def get_dscv(self):
888         return self.dscv
889
890     # 頂点の探索済み距離を取得
891     def get_cmpl(self):
892         return self.cmpl
893
894     # 頂点の前駆頂点を取得
895     def get_pred(self):
896         return self.pred
897
898     # 頂点の情報を表示
899     def __str__(self):
900         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
901
902     # 頂点を接続
903     def connect(self, v, w):
904         self.adj.add(v)
905         v.adj.add(w)
906
907     # 頂点を探索状態に設定
908     def set_color(self, c):
909         self.color = c
910
911     # 頂点を探索状態から解放
912     def clear(self):
913         self.color = WHITE
914         self.dscv = INFTY
915         self.cmpl = INFTY
916         self.pred = None
917
918     # 頂点が探索可能か確認
919     def is_gray(self):
920         return self.color == GRAY
921
922     # 頂点が探索済みか確認
923     def is_black(self):
924         return self.color == BLACK
925
926     # 頂点の隣接頂点を取得
927     def get_adj(self):
928         return self.adj
929
930     # 頂点の探索距離を取得
931     def get_dscv(self):
932         return self.dscv
933
934     # 頂点の探索済み距離を取得
935     def get_cmpl(self):
936         return self.cmpl
937
938     # 頂点の前駆頂点を取得
939     def get_pred(self):
940         return self.pred
941
942     # 頂点の情報を表示
943     def __str__(self):
944         return "id: " + str(self.id) + ", color: " + str(self.color) + ", dscv: " + str(self.dscv) + ", cmpl: " + str(self.cmpl) + ", pred: " + str(self.pred)
945
946     # 頂点を接続
947     def connect(self, v, w):
948         self.adj.add(v)
949         v.adj.add(w)
950
951     # 頂点を探索状態に設定
952     def set_color(self, c):
953         self.color = c
954
955     # 頂点を探索状態から解放
956     def clear(self):
957         self.color = WHITE
958         self.dscv = INFTY
959         self.cmpl = INFTY
960         self.pred = None
961
962     # 頂点が探索可能か確認
963     def is_gray(self):
964         return self.color == GRAY
965
966     # 頂点が探索済みか確認
967     def is_black(self):
96
```

```

62     for i in range(0, N):
63         vertices.append(MyVertex(i))
64
65     # 辺の設定
66     connect(vertices, 0, 1)
67     connect(vertices, 0, 7)
68     connect(vertices, 1, 6)
69     connect(vertices, 1, 2)
70     connect(vertices, 2, 3)
71     connect(vertices, 2, 5)
72     connect(vertices, 3, 4)
73     connect(vertices, 3, 5)
74     connect(vertices, 4, 5)
75     connect(vertices, 5, 6)
76
77     # 頂点5を始点として探索
78     dfs(vertices, vertices[5], 0)
79
80     # 始点頂点5から頂点7への経路を表示
81     print("頂点5から頂点7への経路: ", end = "")
82     print_path(vertices, vertices[5], vertices[7])
83     print("")
84
85     # 各頂点の表示
86     for i in range(0, N):
87         print(vertices[i].to_string())

```

■ main 関数の説明

56 行目～87 行目で main 関数を定義しています。幅優先探索のソースコード 7.7 (my_bfs.py) と同じグラフを生成するので、ほとんど同じです。違いは 78 行目に示す dfs 関数で探索を行う箇所だけです。また、今回は頂点 5 を始点として探索をするため、82 行目の print_path 関数への引数を変更しています。

実行結果として、頂点 5 から頂点 7 への経路を表示します。さらにグラフ内の各頂点の情報として、id と adj、color、dscv、cmpl、pred を表示します。

■ dfs 関数（軸優先探索の実行）の説明

27 行目～39 行目で幅優先探索を実行する dfs 関数を定義しています。引数として、頂点の集合を変数 vertices、始点となる頂点の MyVertex オブジェクトを変数 u、タイムスタンプを変数 time で受け取ります。

頂点 u は、初めて訪れる頂点なので、28 行目で $u.dscv$ に現在のタイムスタンプである $time$ の値を代入して、30 行目で $u.color$ を `GRAY` に設定します。また、29 行目で $time$ の値を 1 増やします。

31 行目～36 行目の `for` ループで、頂点 u に隣接する頂点を 1 つずつ走査します。前項で解説した深さ優先探索の骨格を具体的に記述したものです。32 行目で変数 v に $vertices[i]$ の `MyVertex` オブジェクトを代入します。以降の記述を完潔にするために変数 v を用います。

33 行目の `if` 文で、頂点 v が未探索 ($v.color$ が `WHITE`) かどうかを判定します。False であれば頂点 v は無視して次のイテレーションへ進みます。True であれば `if` ブロックに入ります。34 行目で頂点 v の先行頂点を u に設定します。35 行目で `dfs` 関数を再帰的に呼び出します。ここでは第 2 引数に頂点 v の `MyVertex` オブジェクト、第 3 引数に現在のタイムスタンプを指定します。頂点 v と `color` が `WHITE` のである頂点の集合から構成される部分グラフの探索をします。探索が終了すれば、36 行目に処理が戻ってきます。この時点では頂点 v から到達可能な部分グラフの頂点をすべて探し終えた状態なので、`dfs` 関数の内部で頂点 v の探索終了時間を示す $v.cmpl$ がすでに設定されています。そのため、現在のタイムスタンプを $time = v.cmpl + 1$ に更新します。ループ処理を用いて、 $u.adj$ に含まれるすべての頂点に対して同様の処理を行います。

ループを抜けると頂点 u のすべての隣接頂点を走査し終えた状態になります。そのため、37 行目で $u.color$ を `BLACK` に変更して、探索終了状態にします。38 行目の $u.cmpl = time$ という命令で探索終了時間に現在のタイムスタンプを設定して、39 行目で $time$ の値を 1 増やします。

頂点 5 を始点として、深さ優先探索を実行したときの具体例を図 7.24～7.39 に示します。まず、図 7.24 に、`dfs(vertices, vertices[5], 0)` を実行して、31 行目の `for` ループに入る前の状態を示します。図の左側がグラフ内の各頂点の状態、右側がスタックの状態です。

`main` 関数から `dfs(vertices, vertices[5], 0)` を実行したため、スタックの底に `main` 関数が格納され、その上に `dfs(vertices, vertices[5], 0)` が格納されている状態です。なお、スタックの一番上の要素が実行中の関数です。スタックによる再帰関数の管理は、第 3.4 節で解説したスタックの使用例を参照してください。

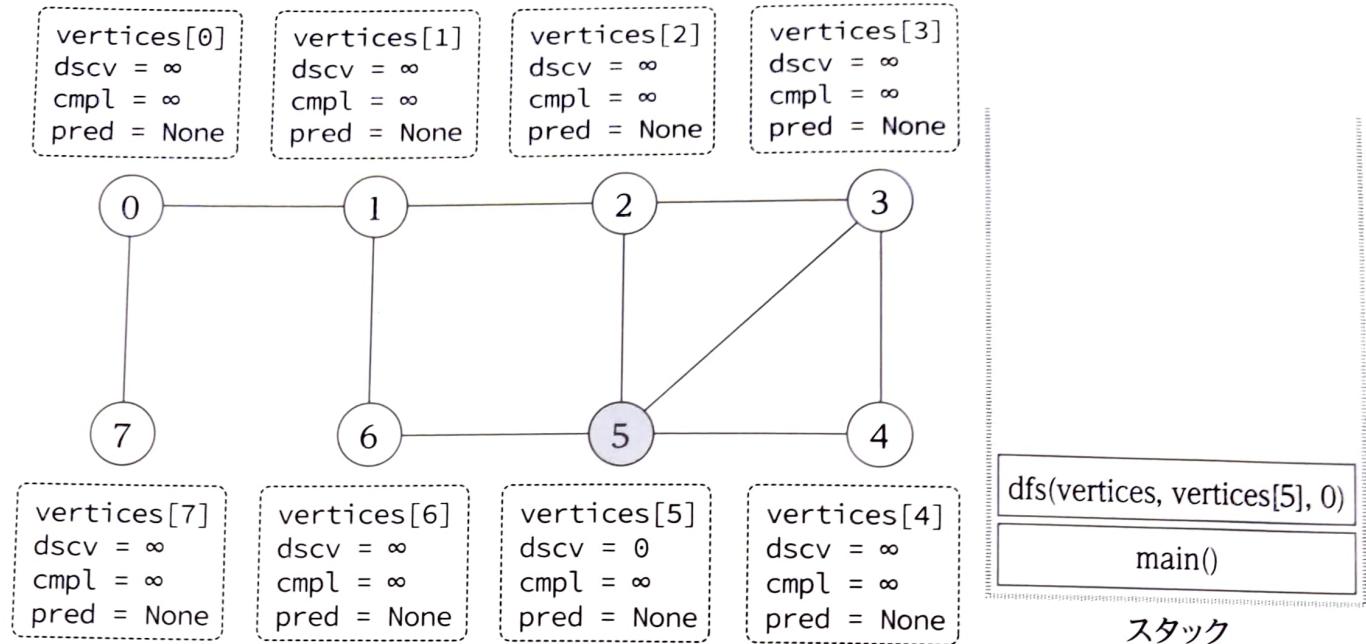


図 7.24 タイムスタンプが 0 のときのグラフの状態

31 行目～36 行目の for ループで、頂点 5 に隣接する頂点を走査します。頂点 2 は、初めて訪れる頂点なので、35 行目で再帰的に `dfs(vertices, vertices[2], 1)` を実行します。第 2 引数が頂点 2 を表す MyVertex オブジェクトです。この時点でタイムスタンプである変数 `time` の値は 1 になっているはずです。`dfs(vertices, vertices[2], 1)` を 31 行目の for ループの直前まで処理を進めたときの状態を図 7.25 に示します。vertices[2] の `color` と `dscv` と `pred` の値が更新されていることに注目してください。また、実行中の関数が `dfs(vertices, vertices[2], 1)` なので、スタックの一番上に当該関数が格納されています。

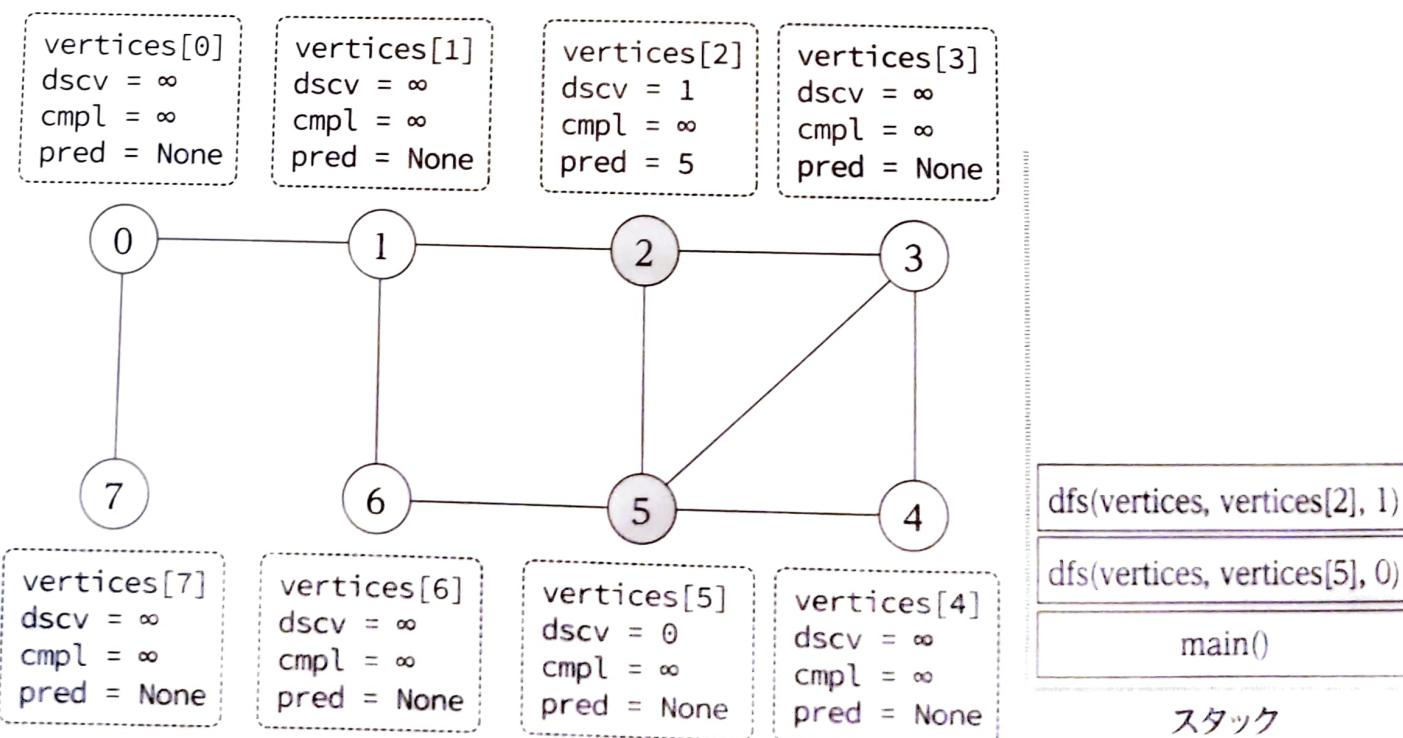


図 7.25 タイムスタンプが 1 のときのグラフの状態

同様に処理を進めていきます。次は頂点 1 を探索するために `dfs(vertices, vertices[1], 2)` が呼び出されます。そのときの状態を図 7.26 に示します。vertices[1] の変数とスタックの中身の変化に注意しながらトレースしてください。

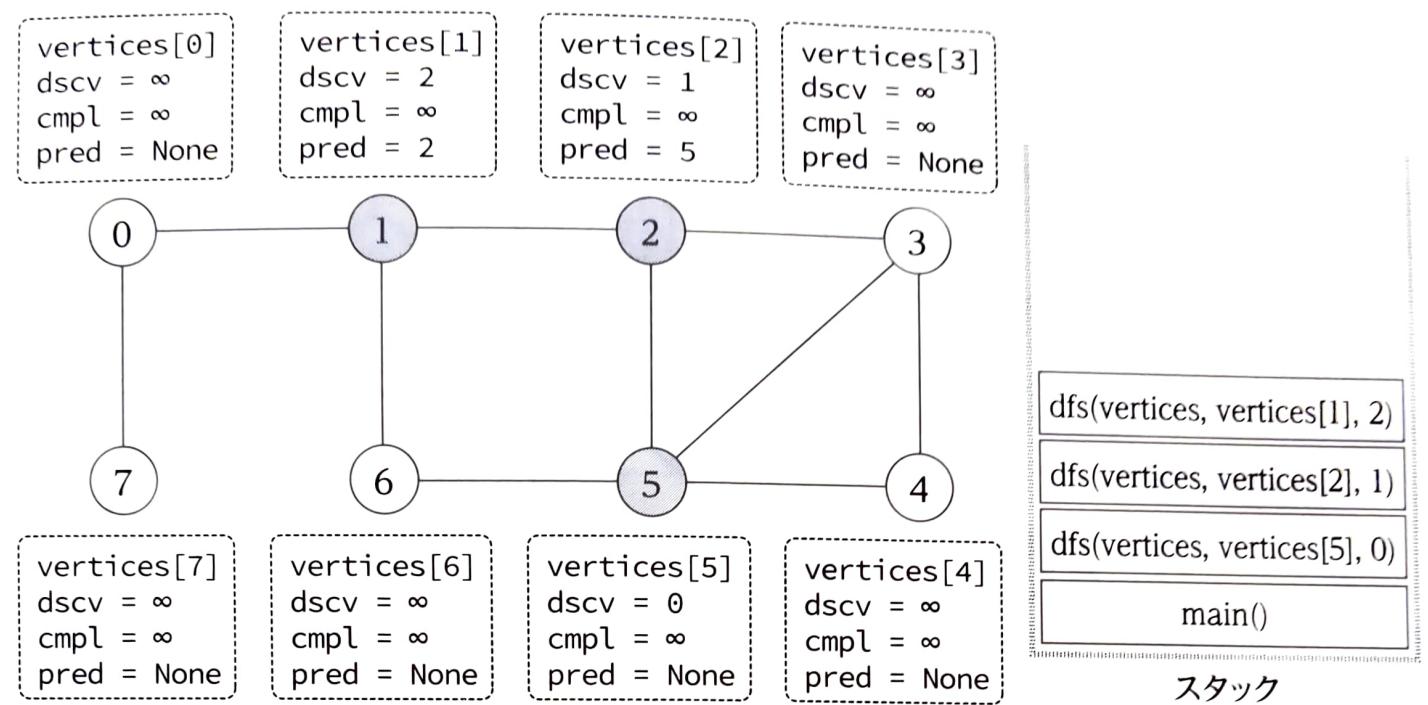


図 7.26 タイムスタンプが 2 のときのグラフの状態

頂点 1 の次は頂点 0 へ進みます。視覚化すると図 7.27 の状態になります。

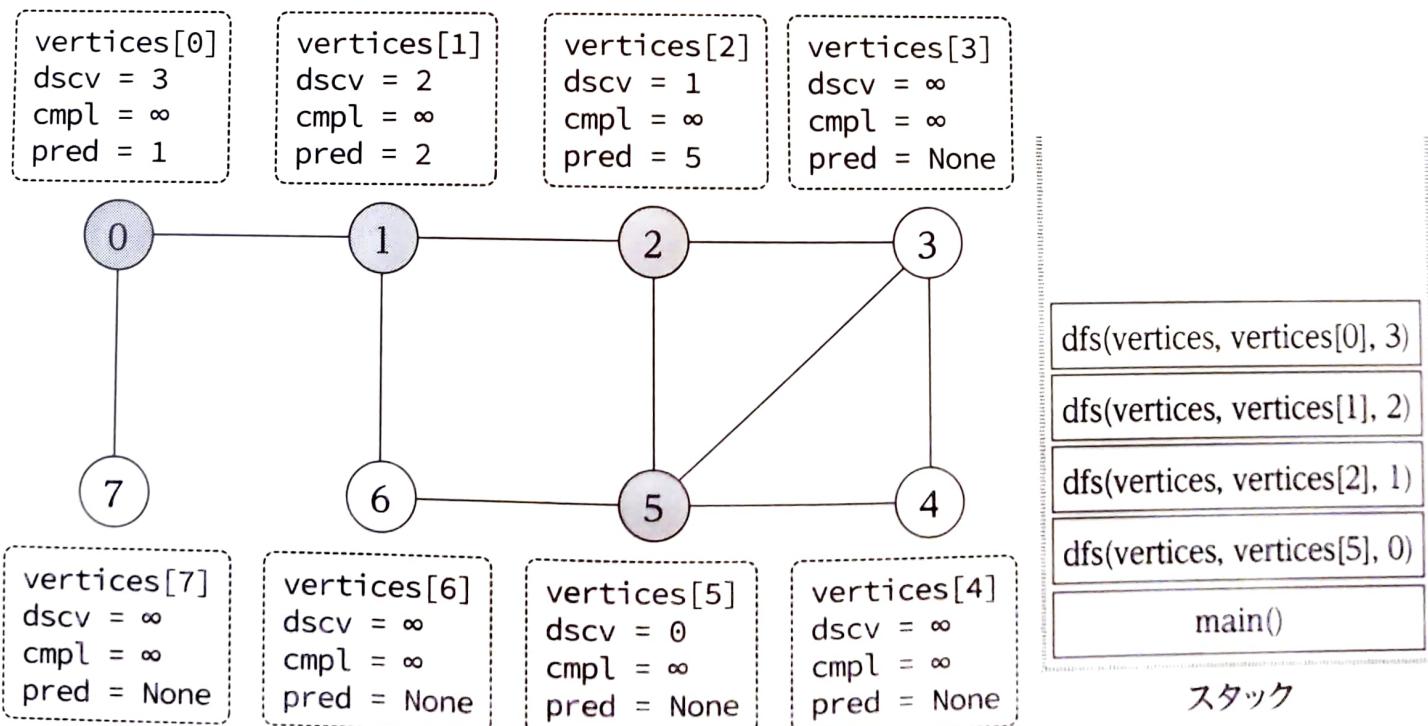


図 7.27 タイムスタンプが 3 のときのグラフの状態

さらに頂点7へ進みます。そのときの状態を図7.28に示します。いま、dfs(vertices, vertices[7], 4)を実行している状態です。グラフから読み取れるように、頂点7の隣接頂点でcolorがWHITEの頂点は存在しません。そのため、31行目のforループを抜けて、37行目へ進みます。37行目と38行目の命令で、u.colorとu.cmplの値を更新します。ここで変数uはvertices[7]を指しています。39行目で変数timeの値を更新して、関数を終了します。

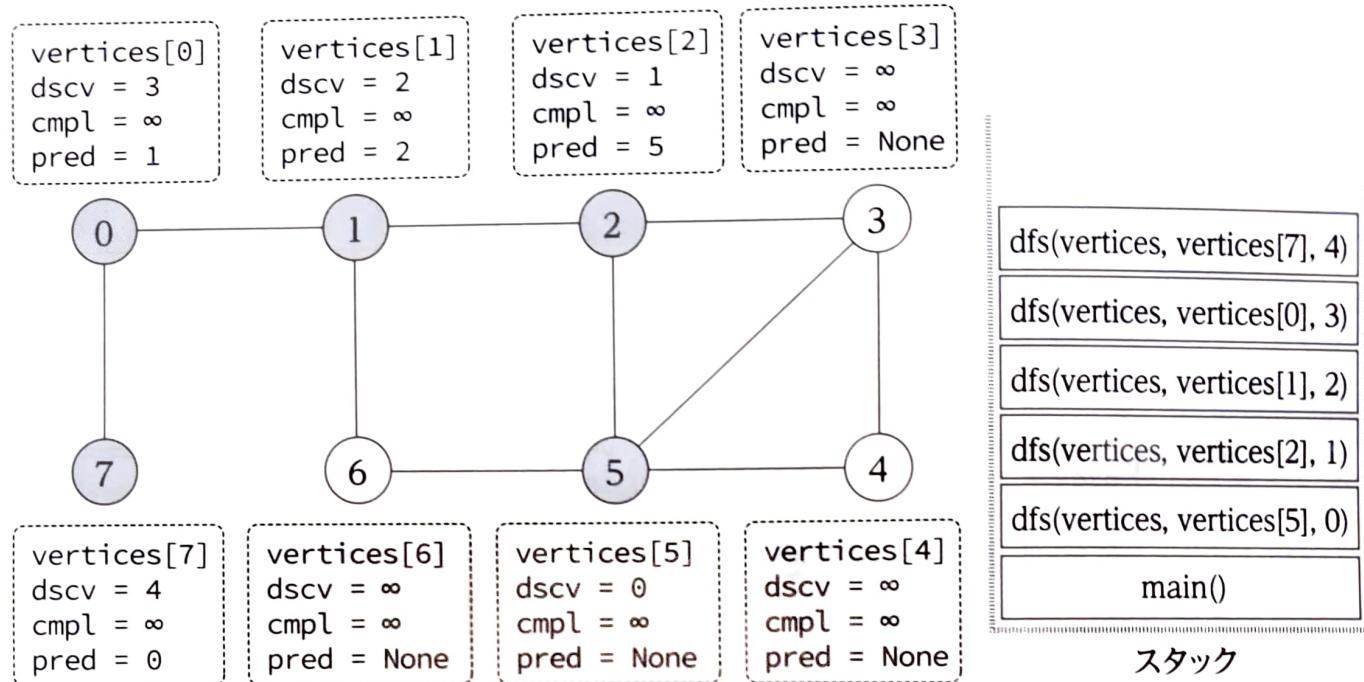


図7.28 タイムスタンプが4のときのグラフの状態

dfs(vertices, vertices[7], 4)の終了時点での状態を図7.29に示します。頂点7を表すvertices[7]のcolorがBLACKになり、cmplの値が5になります。また、関数の実行が終了したので、プログラムの制御が関数の呼び出し元へ戻します。そのため dfs(vertices, vertices[7], 4)をスタックからポップします。図の右側のスタックの一番上が、dfs(vertices, vertices[7], 4)の呼び出し元であるdfs(vertices, vertices[0], 3)になります。

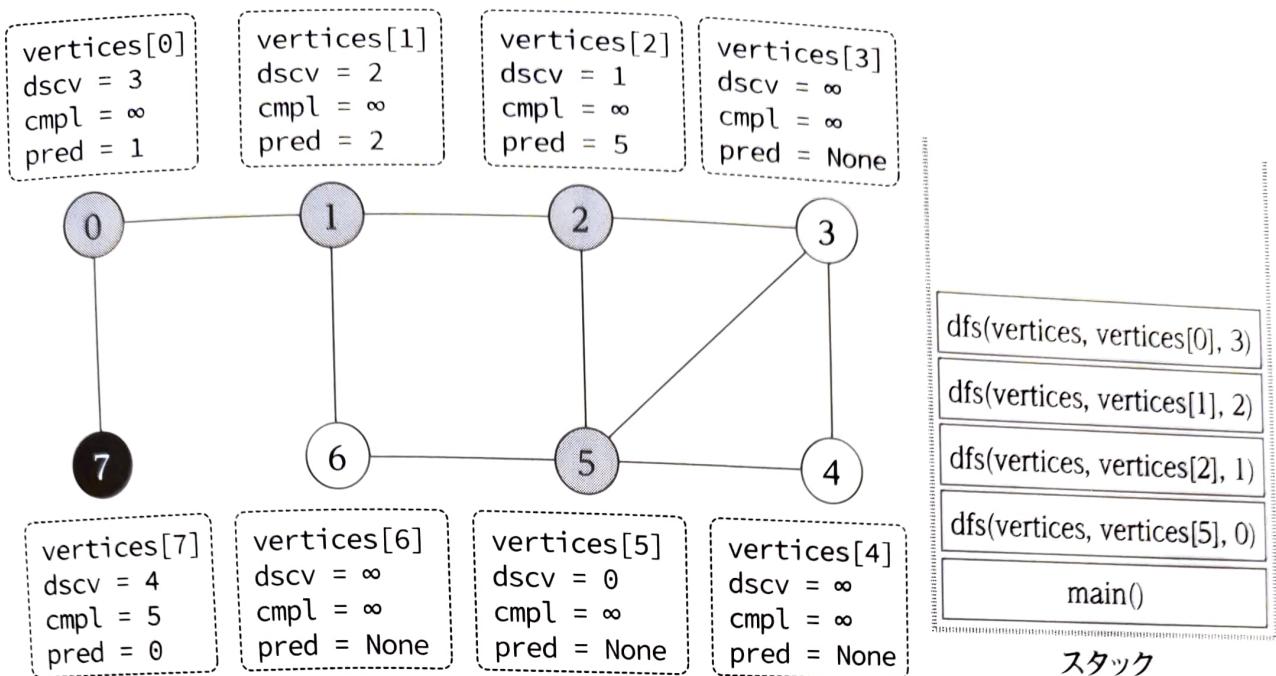


図 7.29 タイムスタンプが 5 のときのグラフの状態

dfs(vertices, vertices[7], 4) が終了すると、呼び出し元である dfs(vertices, vertices[0], 3) の 35 行目に処理が戻ってきます。変数 u は頂点 0 である vertices[0] を参照しています。これ以上走査する隣接頂点がないので、for ループを抜けます。37 行目～39 行目の処理を実行して、dfs(vertices, vertices[0], 3) を終了します。そのときの状態を図 7.30 に示します。1 つ前の図と同様に頂点 0 の color と cmpl、スタックの中身が変化していることを確認してください。

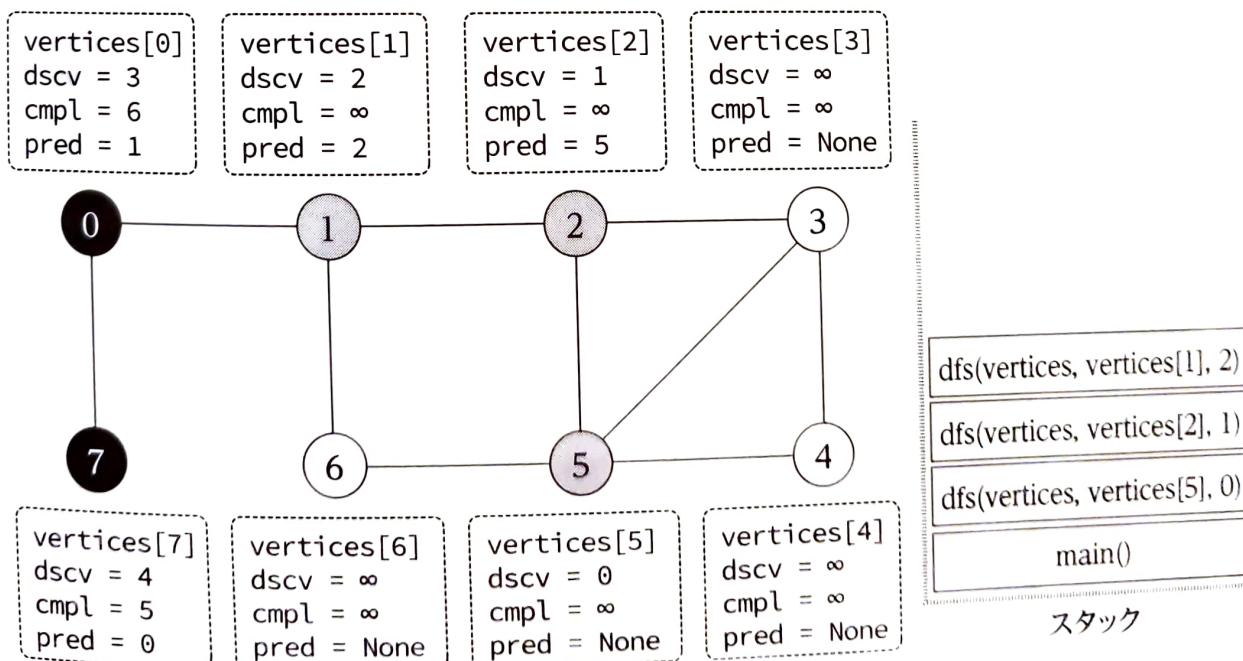


図 7.30 タイムスタンプが 6 のときのグラフの状態

処理が dfs(vertices, vertices[1], 2) の 35 行目に戻ります。36 行目で $time = v.cmpl + 1$ を実行

すると、変数 time の値が 7 になります。for ループを繰り返し、次は頂点 6 を指す vertices[6] が変数 v に格納されます。再度、再帰的に dfs(vertices, vertices[6], 7) を呼び出します。そのときの状態を図 7.31 に示します。vertices[6] の color が GRAY、dscv の値が 7、pred の値が 1 になります。また、dfs(vertices, vertices[1], 2) 内で dfs(vertices, vertices[6], 7) を呼び出すので、スタックの中身が変化しています。

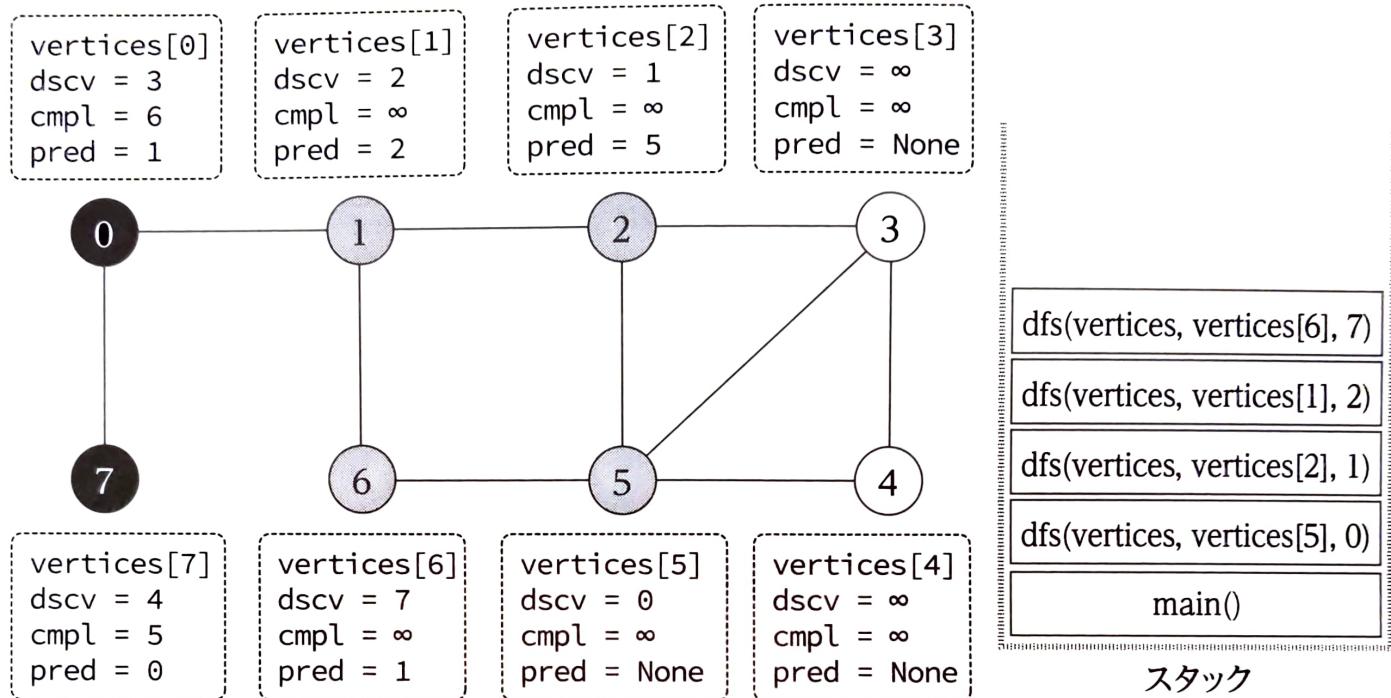


図 7.31 タイムスタンプが 7 のときのグラフの状態

この時点では、頂点 6 は color が WHITE の隣接頂点をもたないので、for ループを抜けて関数を終了します。視覚化すると図 7.32 に示します。

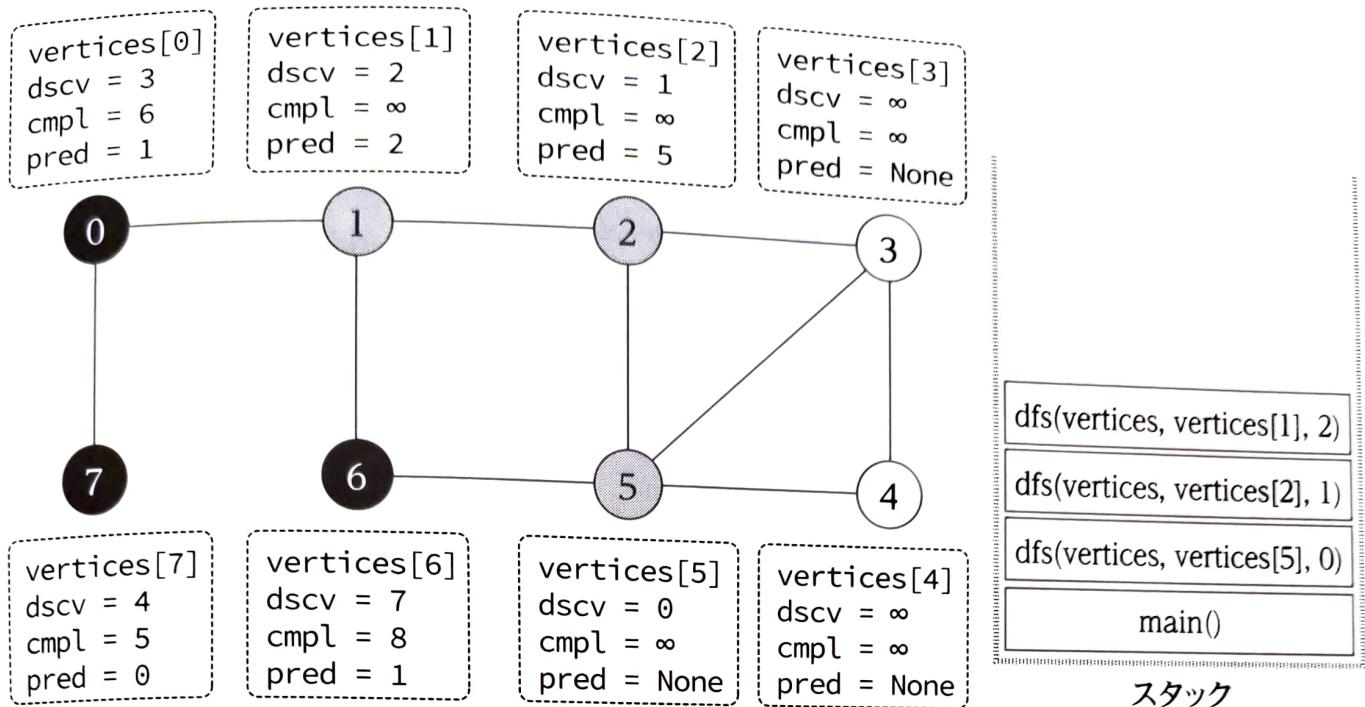


図 7.32 タイムスタンプが 8 のときのグラフの状態

以降も同様の処理を繰り返します。各頂点の変数である color、dscv、cmpl、pred とスタックの中身を確認しながら、どのようにグラフが変化するかを図 7.33 ~ 7.39 を見て確認してください。

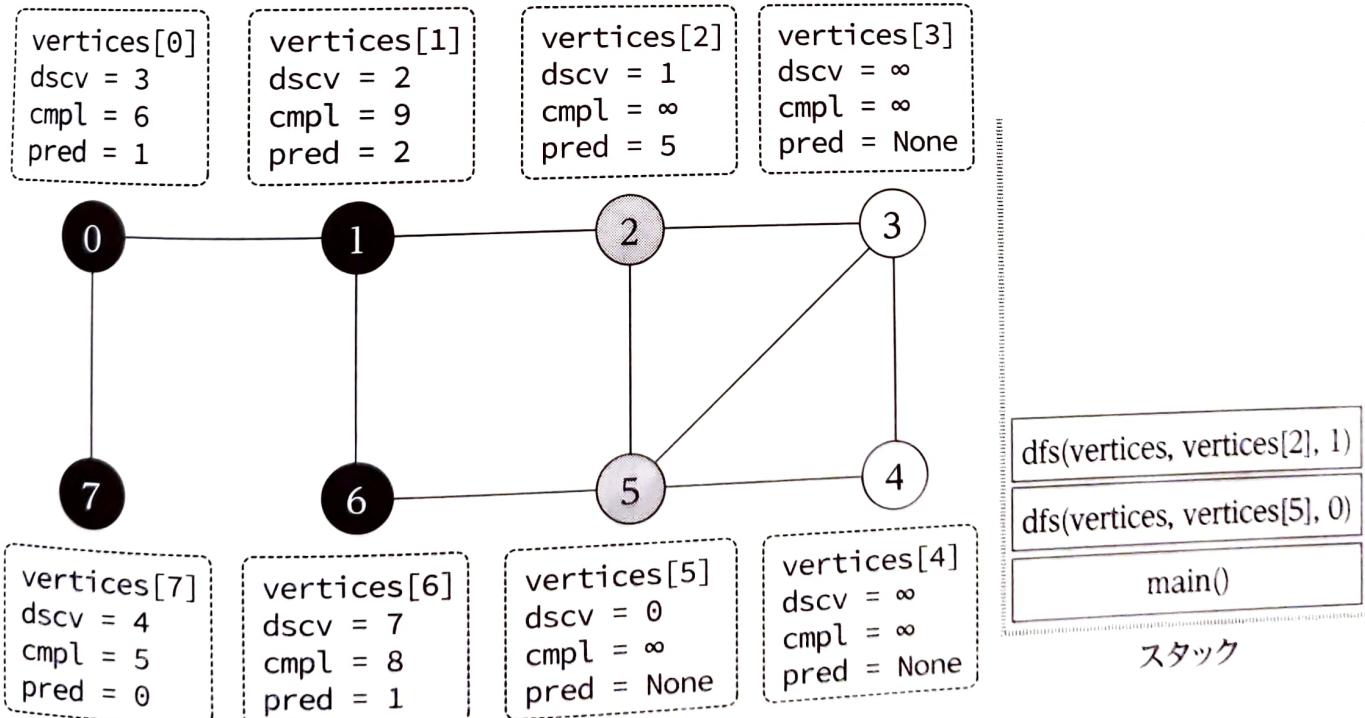


図 7.33 タイムスタンプが 9 のときのグラフの状態

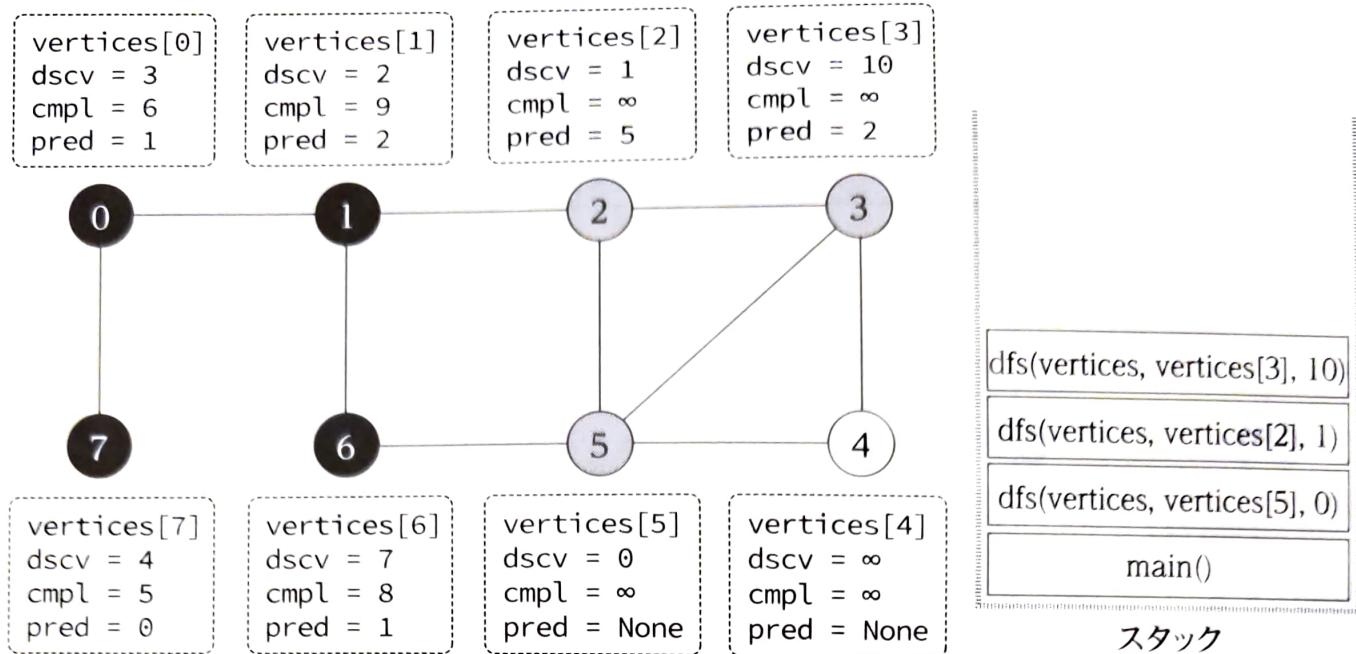


図 7.34 タイムスタンプが 10 のときのグラフの状態

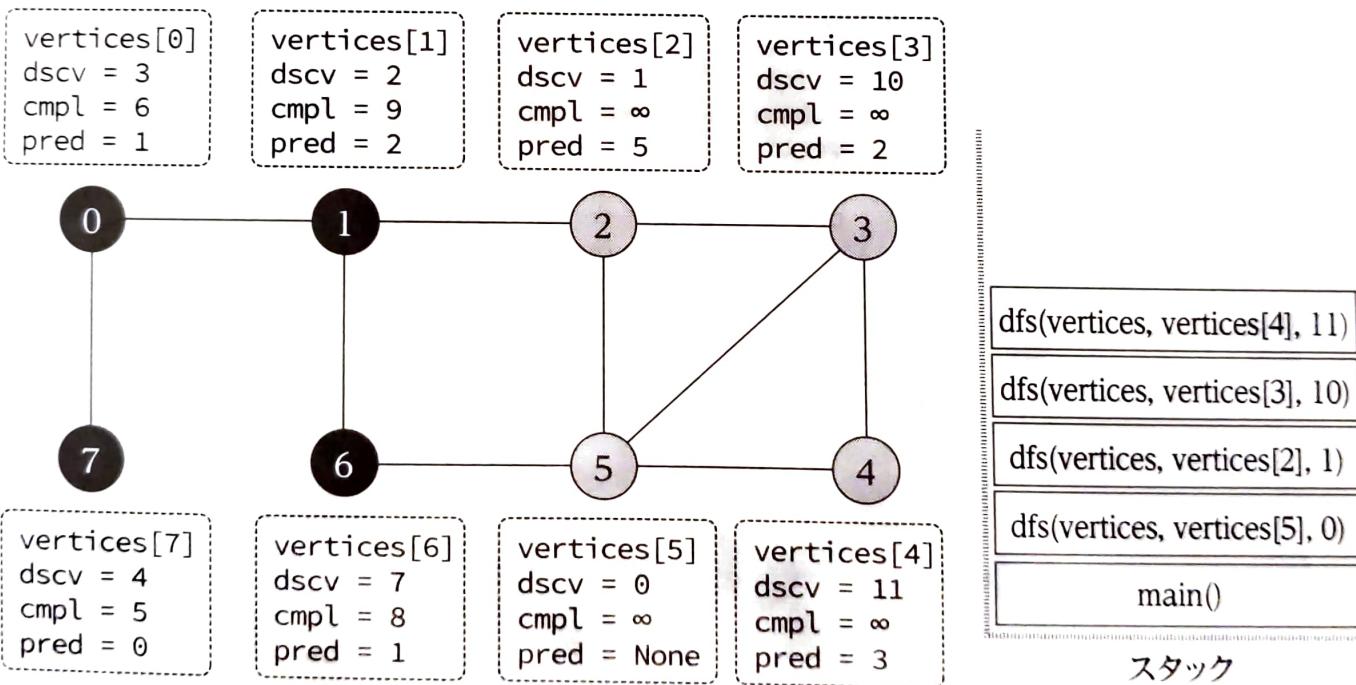


図 7.35 タイムスタンプが 11 のときのグラフの状態

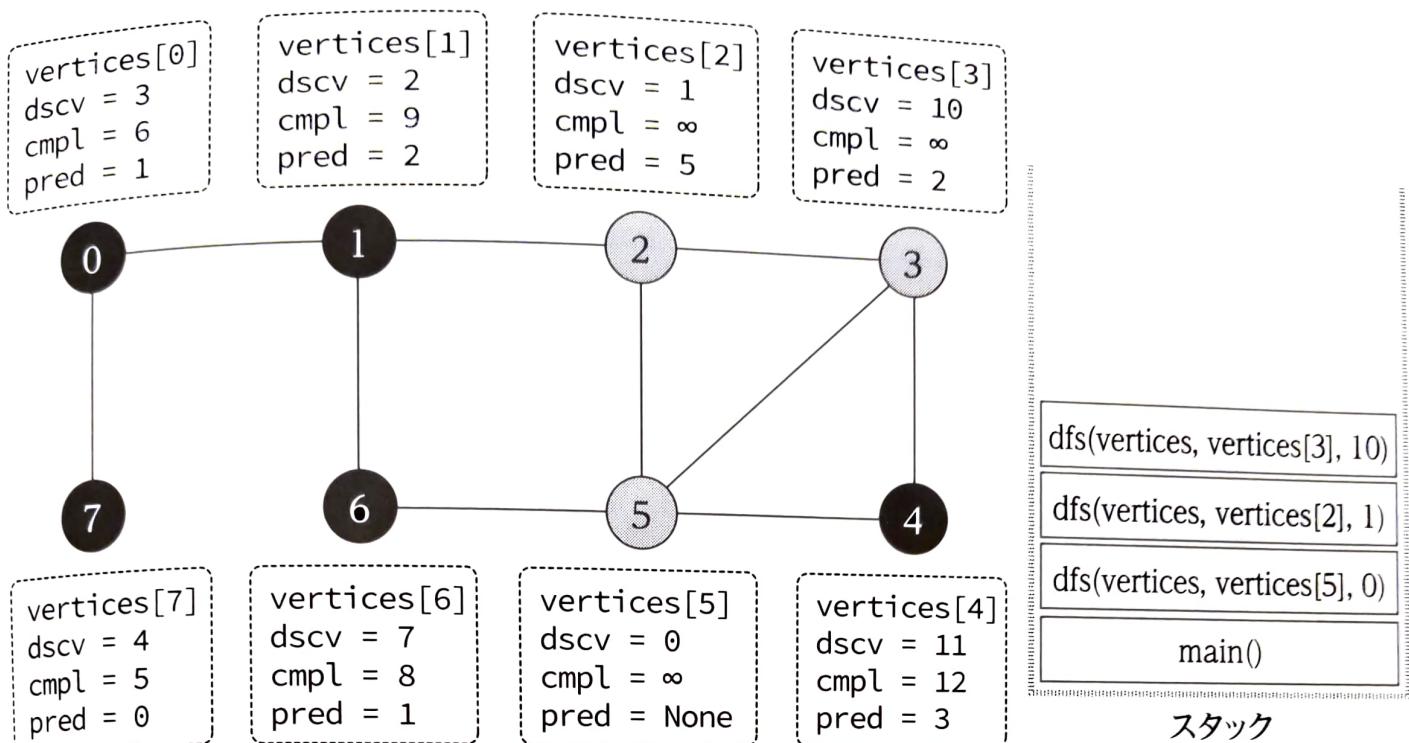


図 7.36 タイムスタンプが 12 のときのグラフの状態

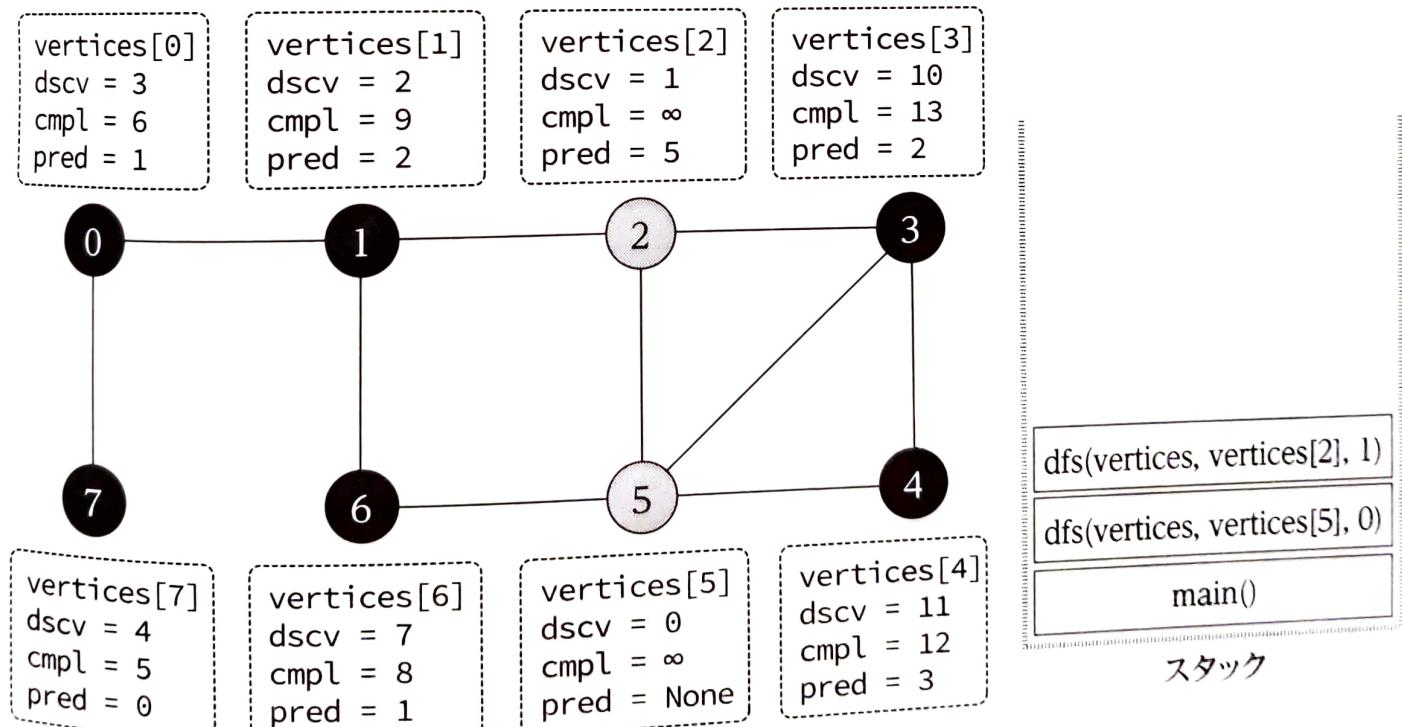


図 7.37 タイムスタンプが 13 のときのグラフの状態

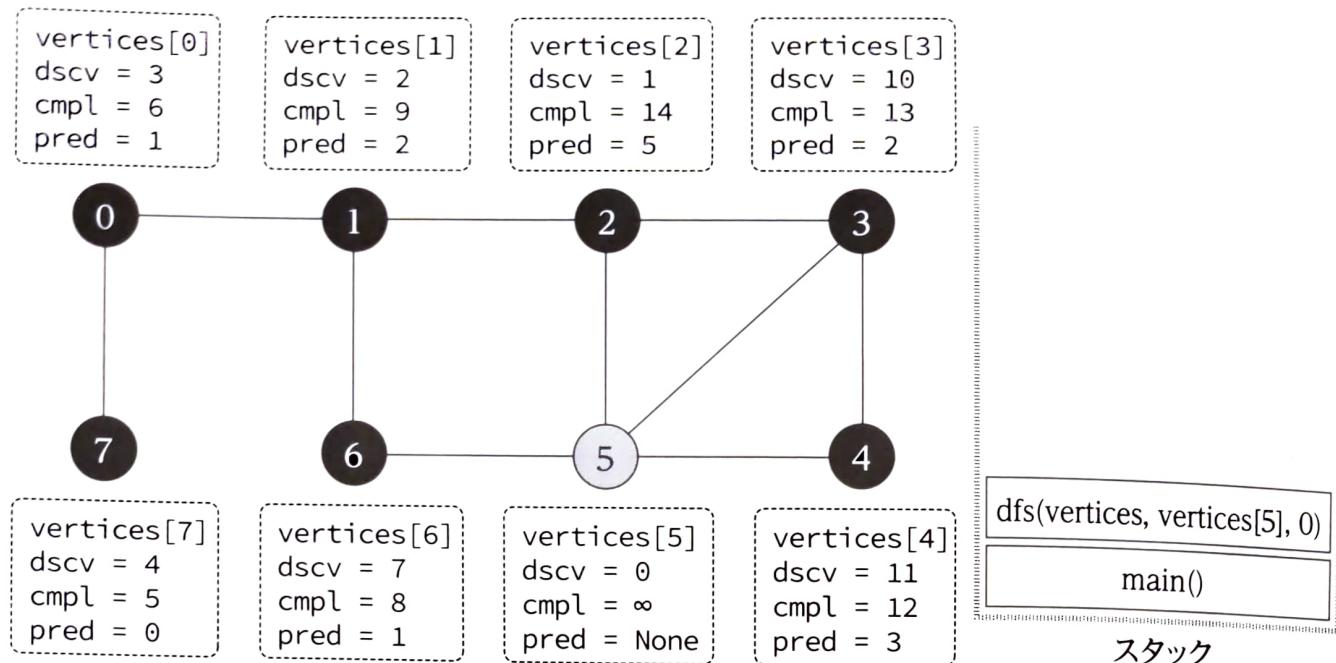


図 7.38 タイムスタンプが 14 のときのグラフの状態

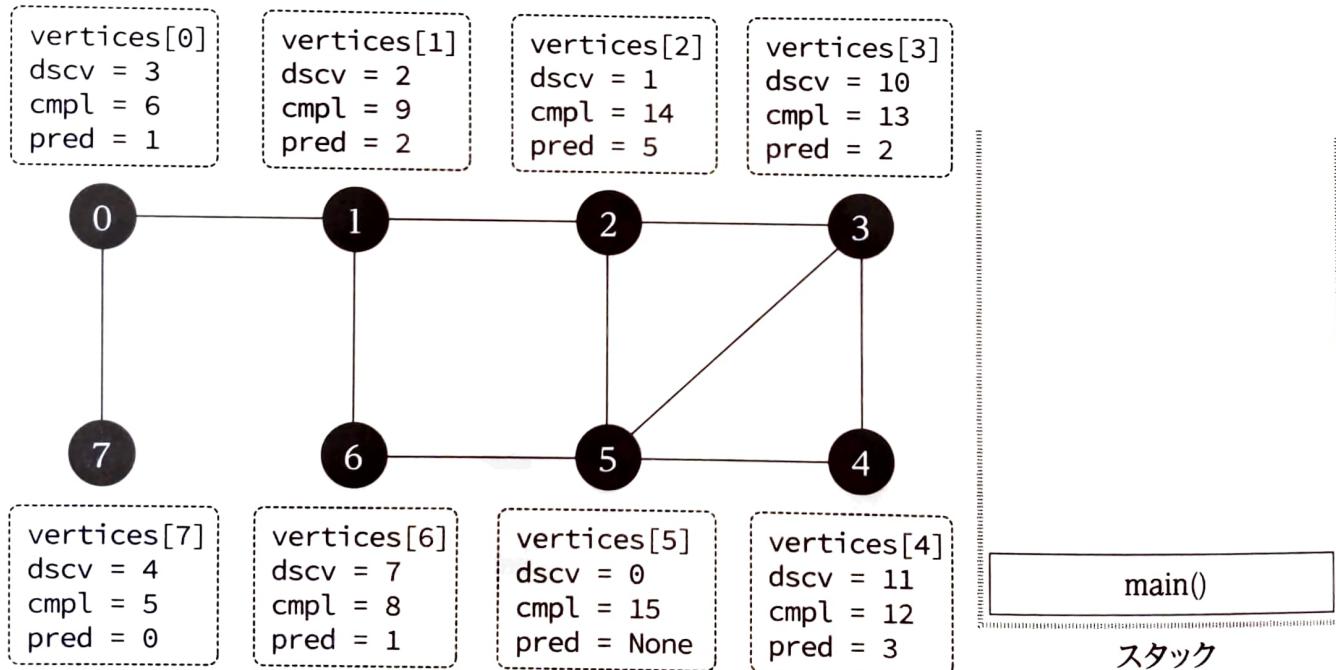


図 7.39 タイムスタンプが 15 のときのグラフの状態

到達可能なすべての頂点を探索し、color が BLACK になれば探索は終了です。始点である頂点 5 から到達可能なすべての頂点の dscv と cmpl と pred の値が設定されます。

■ 深さ優先探索の実装プログラムの実行

ソースコード 7.9 (`my_dfs.py`) を実行した結果を **ログ 7.10** に示します。2 行目に頂点 5 から頂点 7 までの経路が表示されています。図 7.39 のグラフと見比べると、正しく経路が探索できてい

ることが確認できます。3行目以降は各頂点の情報が表示されています。**図 7.39** 内で示した変数の値と同じです。

ログ 7.10 my_dfs.py プログラムの実行

```

01 $ python3 my_dfs.py
02 頂点5から頂点7への経路: 5 2 1 0 7
03 0, adj = {1, 7}, 2, 3, 6, 1
04 1, adj = {0, 2, 6}, 2, 2, 9, 2
05 2, adj = {1, 3, 5}, 2, 1, 14, 5
06 3, adj = {2, 4, 5}, 2, 10, 13, 2
07 4, adj = {3, 5}, 2, 11, 12, 3
08 5, adj = {2, 3, 4, 6}, 2, 0, 15, None
09 6, adj = {1, 5}, 2, 7, 8, 1
10 7, adj = {0}, 2, 4, 5, 0

```

幅優先探索と同様に、深さ優先探索の探索結果を視覚化すると**図 7.40** に示す木構造になります。これを**深さ優先木** (depth first tree) と呼びます。深さ優先木で確認すると、始点である頂点 5 から各頂点への経路が視覚的にも明確になります。

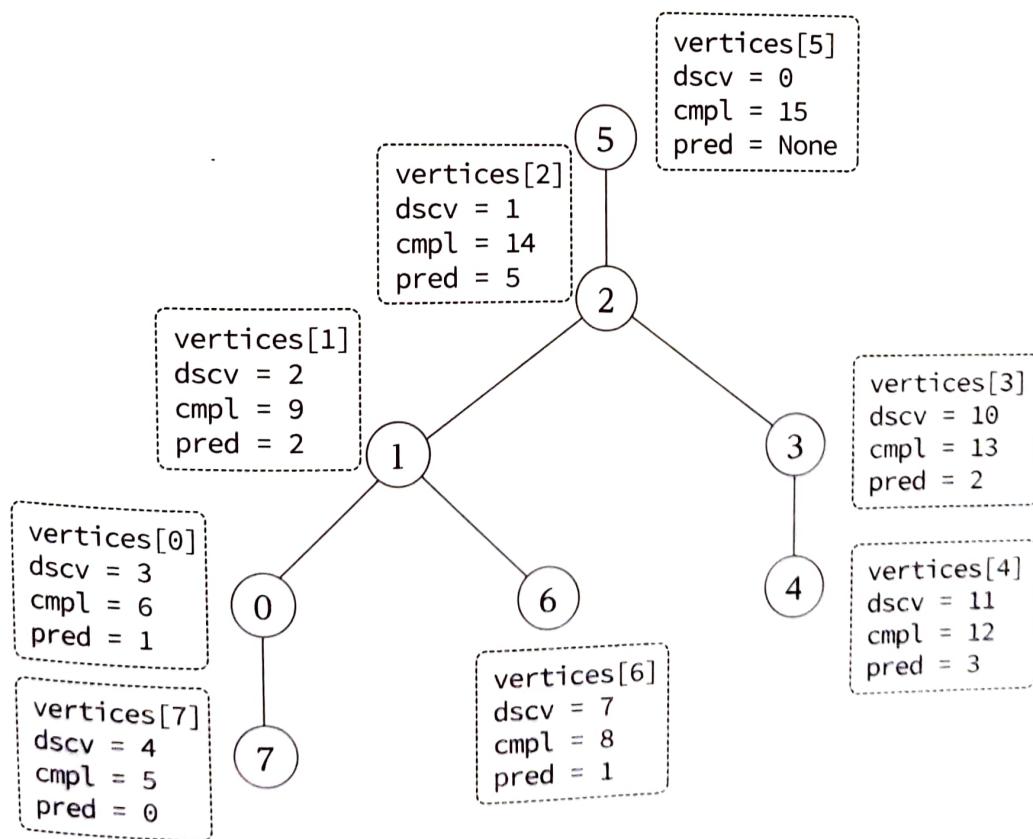


図 7.40 頂点 5 から探索したときの深さ優先木

深さ優先探索の計算量は $O(V+E)$ です。理由は幅優先探索とほとんど同じなので割愛します。

7.7 ダイクストラ法

ダイクストラ法 (dijkstra) は、グラフ内のある頂点から到達可能なすべての頂点への最短経路 (shortest path) を探索するアルゴリズムです。これまでの例では、すべての辺は同じ重みをもつ、と仮定していました。このようなグラフを重みなしグラフ (unweighted_graph) と呼びます。実際には辺に重みはありますが、すべての辺が同じ重みなので、「重みを考慮する必要はないグラフ」という意味です。

重みなしグラフでは、すべての辺が同じ重みをもつため、ある頂点から他の頂点へたどり着くまでに経由する辺の数が一番少ない経路が最短経路となります。幅優先探索を適応すれば最短経路を探索できます。

しかし、多くのグラフ構造では、各辺はそれぞれの重みをもちます。これを辺の重み (edge weight) と呼びます。このように辺に重みがあるグラフを**重み付きグラフ (weighted graph)** と呼びます。重み付きグラフでは、ある頂点から他の頂点へ経路のうち、重みが一番小さい経路が最短経路となります。ダイクストラ法は重み付きグラフにおいて、最短経路を求めるアルゴリズムなのです。もちろん、重みなしグラフにダイクストラ法を適応することも可能です。

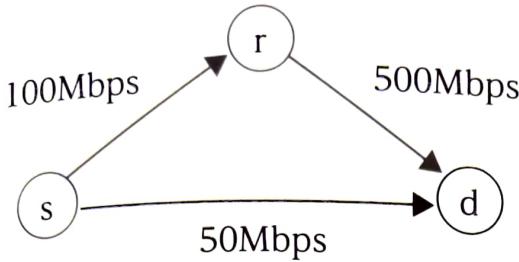
7.7.1 距離の定義

ダイクストラ法が適応できる条件として、負の値をもつ辺がグラフに存在しないことです。その前にアルゴリズムで重要な概念である**距離 (distance)**について説明します。

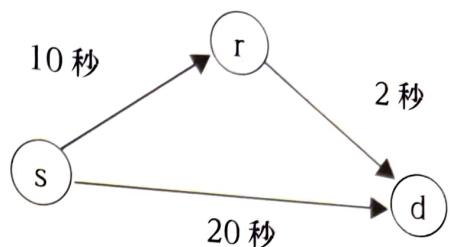
頂点 u と頂点 v を接続する辺の重みを $w(u, v)$ と表記します。1つ以上の辺を経由して到達可能な2つの頂点の距離をどのように定義するか考えてみましょう。

たとえば、コンピュータネットワークをグラフ化した例を図 7.41 に示します。図 7.41(a) には、3 つの端末があり、それぞれ**送信端末 s** (source の略)、**目的端末 d** (destination の略)、**中継端末 r** (relay の略) とします。端末 s と端末 d を結ぶ回線の転送速度を 50Mbps、端末 s と端末 r を結ぶ回線の転送速度を 100Mbps、端末 r と端末 d を結ぶ回線の転送速度を 500Mbps とします。すなわち、 $w(s, d) = 10$ 、 $w(s, r) = 100$ 、 $w(r, d) = 500$ です。辺の重みが転送速度なので、当然ながら大きい値をもつ辺のほうが良いです。では、2 点間の距離はどうなるでしょうか？ $s \rightarrow r \rightarrow d$ という経路でデータを転送する場合は $w(s, r)$ がボトルネックとなるため、実際のパフォーマンスは経路の中でボトルネックとなる辺の値が一番大きい経路が一番良い経路（最短経路）と定義できます。

図 7.41(a) では、中継端末 r を経由する経路である $s \rightarrow r \rightarrow d$ が最短経路となります。



(a) データの転送速度を用いた距離



(b) データの転送時間を用いた距離

図 7.41 距離の定義

この方法は、コンピュータネットワークにおいて最適な経路を決めるための最も直感的な方法です。しかし、これではダイクストラ法は適応できません。ダイクストラ法を適応するには、各辺の重みをコストとしてみなし、距離が**単調増加**（減らないという意味で、英語では **non-decreasing** と呼ぶこともある）するようにしなければなりません。

次は違う方法で距離を定義して、図 7.41(b) に示すようにコンピュータネットワークをグラフ化します。今回は辺の重みを 1,000Mbits のデータを転送するために必要な時間と定義しました。具体的には $w(s, d) = 20$ 、 $w(s, r) = 10$ 、 $w(r, d) = 2$ となります。時間なので、小さい値をもつ辺が良いと判断できます。そして、2 点間の距離を経由する辺の重みの合計値とします。この場合、 $s \rightarrow d$ という経路よりも、 $s \rightarrow r \rightarrow d$ という経路のほうがデータ転送に要する時間の合計値が小さくなります。

このように辺の重みと距離を定義すれば、負の値を含む辺は存在しなくなり、2 つの頂点間の距離が辺を経由する毎に単調増加するため、ダイクストラ法が適応可能となります。実際のアプリケーションやコンピュータシステムを抽象化して、グラフ構造を生成するときは、注意して距離を定義しなければいけません。

本書では、辺の重みを正の整数値として定義し、2 つの頂点間の距離は経由する辺の重みの合計値とします。

7.7.2 ダイクストラ法の概要

ダイクストラ法を適応するために頂点を表す MyVertex クラスを以下のように定義します。識別子 self.id と始点からの距離 self.dist、先行頂点 self.pred は幅優先探索と同様です。ただし、dist は隣接頂点を表す self.adj の初期化方法が波括弧 (brace) に変わって経路内の辺の重みの合計です。隣接頂点を表す self.adj の初期化方法が波括弧 (brace) に変わったことに気づくと思います。あとで簡単に解説しますが、これは辞書型という型です。

```
class MyVertex:
    def __init__(self, id):
        self.id = id
        self.adj = {} # dic型
        self.dist = INFY
        self.pred = None
```

ダイクストラ法の骨格は以下の擬似コードのとおりです。引数として、頂点を表す MyVertex オブジェクトの集合を変数 vertices、始点となる頂点の MyVertex オブジェクトを変数 src で受け取ります。

```
def dijkstra(vertices, src):
    # 初期化
    src.dist = 0
    q = [] # 優先度付きキュー
    for u in vertices:
        u を q にエンキューしヒープ化

    # 探索
    while len(q) > 0:
        q をデキューして u に格納
        for i in u.adj.keys():
            リラックス処理と必要に応じて q をヒープ化
```

幅優先探索と似ているようで似ていません。3行目で src.dist を 0 で初期化します。始点なので距離が 0 となります。4行目の変数 q を空のリストで初期化し、5行目～6行目で vertices 内のすべての要素を変数 q にエンキューします。ここで変数 q は優先度付きキューとして用いますので、ヒープの性質を維持するようにエンキューします。具体的には heapq モジュールを使用しますが、別途解説します。

9行目の while ループで、優先度付きキューから頂点を 1 つずつ取り出して、隣接頂点を走査します。隣接頂点の走査対象が u.adj.keys() となっていたり、リラックス処理という言葉が出てきますが、これらも後ほど解説します。ここでは要素を距離 dist の値をもとに優先度付きキューに格納された頂点を 1 つずつ取り出し、取り出した頂点の隣接頂点を走査し、距離が短い経路が見つかったら隣接端末の dist を更新する、と理解してください。

上記の骨格から、ダイクストラ法では、各頂点と各辺を 1 度ずつ処理することがわかると思います。それでは辞書型とリラックス処理、優先度付きキュー、heapq モジュールの使い方について、それぞれ解説します。

■ 辞書型を用いた隣接頂点の管理

第 7.5 節と第 7.6 節では、クラスメンバ adj を **set 型の変数** として、隣接する頂点の識別子を記録していました。しかし、この方法では重みを保存できません。

隣接行列を用いれば、隣接する頂点とその辺の重みを簡単に管理できます。多くのアルゴリズムの図書では隣接行列を用いています。しかし、隣接行列を用いた場合、隣接頂点の探索で時間がかかるのでお勧めしません。

そこで、本書では**辞書型 (dictionary type)** と呼ばれる型を用います。略して dict 型と呼ぶこと

もあります。辞書型は、キーとそれに対応する値のペアを格納するためのデータ構造です。学問的には連想配列 (associative array) と呼びます。また、Java や C++ では Map という名前で、同じような機能をもつコレクションクラスが提供されています。

辞書型変数の宣言はリストに似ています。初期化時に角括弧 (bracket) の代わりに波括弧 (brace) を用います。たとえば、以下のソースコードを見てください。

```
v = MyVertex(0)
v.adj = {"1": "10", "2": 5}
```

MyVertex のインスタンスを生成し、変数 v を初期化します。v.id は 0 です。2 行目では、dict 型の変数 adj に隣接頂点の識別子をキー、距離を値として初期化します。書式は、"キー": "距離" です。キーと値をそれぞれダブルクオートで囲み、コロンで分離します。これをペアとして dict 型に格納します。各ペアは、リストの要素と同様にカンマで区切れます。

v.adj が含む要素から、 $w(0, 1) = 10$ 、 $w(0, 2) = 5$ ということがわかります。このように dict 型を用いることによって、各頂点の隣接頂点と辺の重みを管理できます。

また、空の辞書を生成する場合は、波括弧だけを記述します。要素を挿入する場合は、`v.adj[キー] = 値`、という書式です。以下のように記述すると、変数 v.adj の中身が上記のソースコードの抜粋と同じになります。

```
v = MyVertex(0)
v.adj = {}
v.adj[1] = 10
v.adj[2] = 5
```

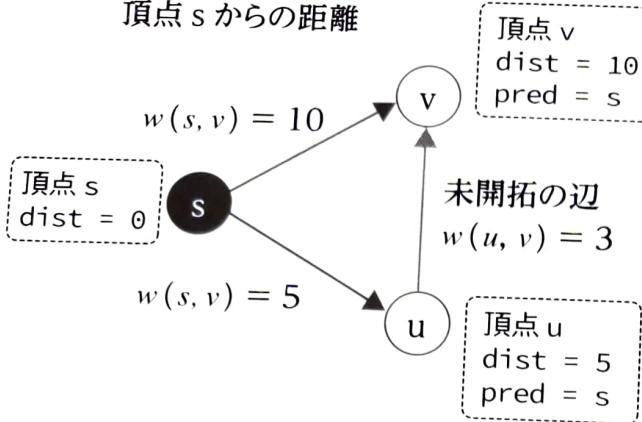
■ リラックス処理

リラックス処理 (relaxation process) とは、始点である頂点 s から頂点 v までの経路について、すでに発見された経路よりも良い経路が見つかったときに頂点 v への距離 dist と先行頂点 pred を更新する処理です。

たとえば、図 7.42(a) に示すグラフを見てください。3 つの頂点 s と u と v がグラフに含まれており、頂点 s から v、頂点 s から u、頂点 u から v の辺が存在します。辺の重みは、それぞれ $w(s, u) = 5$ 、 $w(s, v) = 10$ 、 $w(u, v) = 3$ とします。説明の簡略化のために、ここでは有向グラフを用いて説明します。すでに頂点 s の隣接頂点が走査された状態であり、頂点 v の dist が 10、頂点 u の dist が 5 です。

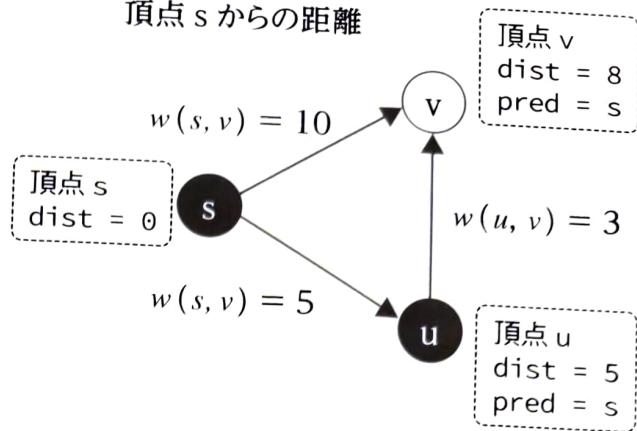
次に頂点 u の隣接頂点を走査します。 $w(u, v) = 3$ なので、 $s \rightarrow u \rightarrow v$ という経路を通った場合、dist が 8 になります。すでに発見されている $s \rightarrow v$ という経路よりも良い経路が見つかったので、頂点 v の dist と pred を更新する必要があります。更新後の状態を図 7.42(b) に示すように、頂点 v の dist と pred の値はそれぞれ 8 と u になります。

頂点 s からの距離



(a) 頂点 s の隣接頂点を走査

頂点 s からの距離



(b) 頂点 u の隣接頂点を走査

図 7.42 リラックス処理の例

リラックス処理の擬似コードは以下のとおりです。関数名を relax とします。引数として 2 つの MyVetex オブジェクトを変数 u と v で受け取ります。頂点 u は、現在隣接する頂点を走査中の頂点です。頂点 s は登場しませんが、頂点 u と v は図 7.42(b) と同じ状況です。すでに u.dist と v.dist の値はなんらかの値が設定されています。もし、頂点 v への経路が 1 つも発見されていなければ、v.dist は初期値として ∞ が設定されています。

```
def relax(u, v):
    if v.dist > u.dist + w(u, v):
        v.dist = u.dist + w(u, v)
        v.pred = u.id
```

擬似コードを見てのとおり、頂点 u を経由して頂点 v に到達する経路のほうが、すでに発見された経路よりも良い経路 (dist の値が小さい) であれば、v.dist と v.pred を更新します。

Column

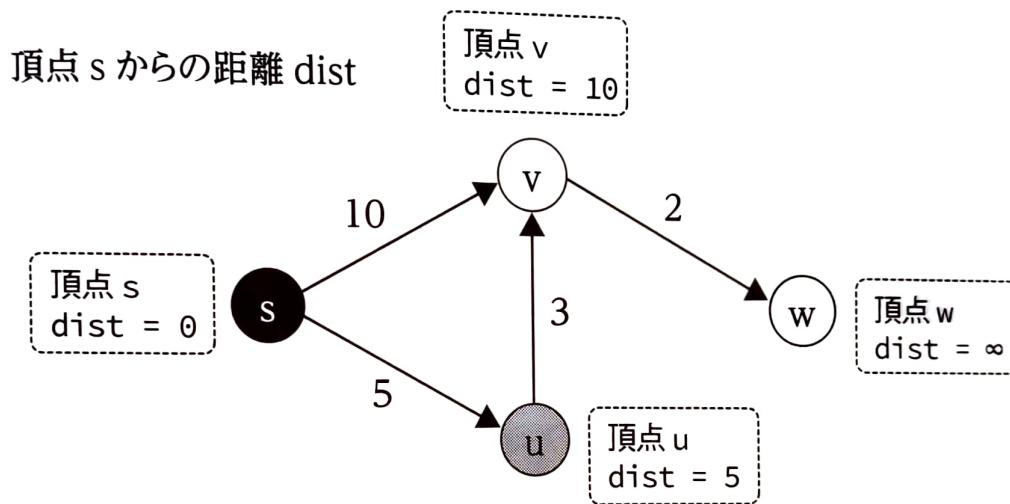
リラックス (relax) という単語について

良い経路に更新する処理を「リラックス (relax)」と呼ぶことについて疑問に思うかもしれません。歴史的な背景があります。数学において relax という単語は、制限を緩和するときに使用します。頂点 v へ繋がる辺の集合 (図 7.42 では 2 つ) があり、この中の 1 つが頂点 v への最短経路に含まれます。1 回の relax 関数の処理で、最短経路に含まれる辺となる要素の候補を 1 つ減らすことができます。この処理を「制限を取り除く」と解釈できるため、ダイクストラ法では relax という単語が使用されます。

■ 優先度付きキューの使用

ダイクストラ法では、各頂点の隣接頂点を1度ずつ走査します。始点から近い順番に頂点を探索しますが、具体的にどういう基準で頂点を選ぶかが重要となります。ダイクストラ法では、各頂点の探索状態を管理するため**優先度付きキュー**を用います。優先度付きキューなので、第6.4節で解説したヒープを使用します。

まず、図7.43に示すグラフを見てください。有向グラフは4つの頂点s、u、v、wで構成されます。矢印で示した4つの辺があり、辺の重みはそれぞれ $w(s, u) = 5$ 、 $w(s, v) = 10$ 、 $w(u, v) = 3$ 、 $w(v, w) = 2$ です。図は頂点sの隣接端末の走査を終えた時点の状態です。そのため、頂点u.distとv.distの値はそれぞれ5と10です。一方、w.distは初期値から変更されていないので、 ∞ のままでです。また、ここではひとまず先行頂点predは無視してください。



- ・頂点 s はすでに探索済み
- ・頂点 u が未探索の頂点の中で一番 dist の値が小さい

図7.43 有向グラフの頂点 s の隣接頂点を走査

頂点sの隣接頂点を走査し終えた状態なので、頂点sを指す円形を黒色で塗りつぶしています。頂点uの円形を灰色で塗りつぶしていますが、未探索の頂点の中で一番 dist の値が小さいのが頂点u、という意味です。

ダイクストラ法では、distの値をもとに優先度を決めます。優先度の必要性を説明するために、優先度を付けなければどういった問題が発生するか見てみます。図7.44に示すように、頂点uより先に頂点vの隣接頂点を走査したとしましょう。頂点wのdistが更新されて値が12になります。また、頂点vの円形を黒色で塗りつぶします。