グラフとアルゴリズムとプログラムのやさしいおはなし

渡邉敏正

2021 年 6 月 13 日

第 3 回

1　グラフとは

今後いろいろな事柄を解説するときに必要となりますので、ここでグラフについてもう少し詳しい説明をしておきます。少し堅苦しい単調な話になるかもしれませんが、できるだけ皆さんにそっぽを向かれないように述べてみようと思います。

本章の最後に「ちょっと一言」で私の意見を少し書きました。それにも目を通していただければありがたいです。

グラフという用語については、数学でよく使われる関数 $y = f(x)$ のグラフなどを思い浮かべるかもしれませんが、その使用法はむしろ特殊でして、図 3 に示すような**点**（vertex: ○で表して整数で名前を付けている）の集合と点間を結ぶ線分（**辺** (edge) といい、たとえば $\{1, 2\}$ などの点の対で表す）からなる図を意味することの方が一般的です。

1.1　パスとサイクルについて

図 3 を例として使いながら説明します。点集合 V_1 と辺集合 E_1 をそれぞれ

$$V_1 = \{1, 2, 3, 4, 5, 6\},\ E_1 = \{\, \{1, 2\}, \{2, 3\}, \{3, 1\}, \{3, 4\}, \{4, 5\}, \{5, 6\}, \{6, 4\} \,\}$$

として、そのペアを $G_1 = (V_1, E_1)$ と表して、**グラフ** (graph) G_1 とよびます。

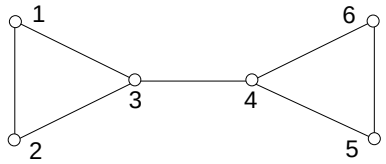

図 3　グラフ G_1

辺で結ばれている 2 点は**隣接** (adjacent) しているといいます。なお、ここでは辺について点の順序は考えないものとします：$\{u, v\}$ と $\{v, u\}$ は同じ辺と考えます。隣接する 2 点 u, v の間に 2 本以上の辺が存在する場合があります。このような辺を多重辺 (mutiple edges)、1 本の辺の場合を単純辺 (simpel edge) といいます。多重辺を含むグラフを**多重グラフ** (multiple graph)、多重辺を含まないグラフを**単純グラフ** (simple graph)

とよびます。第 1 回の図 2 で示した G_0 は多重グラフですし、今回の図 3 の G_1 は単純グラフです。以後、「グラフ」という用語は単純グラフ、多重グラフどちらにも用いることにします。グラフの図をみればどちらか区別がつきますし、用語の区別が必要ならば「単純」、「多重」を付けることにします。

隣接している点を順次重複なく辿ってできる点列を**単純なパス** (simple path) といいます。辺は重複しません。たとえば図 3 での点列 $1, 2, 3, 4$ を点 1 から 4 への（あるいは 1 と 4 を結ぶ）単純なパスという言い方をします。なお、$1, \{1,2\}, 2, \{2,3\}, 3, \{3,4\}, 4$ のように、辺を表記する場合もあります。あるいは、$1-2-3-4$ のような形式の点列で表現することもあります。また同じく図 3 で点列 $3, 1, 2, 3, 4, 5$ のように点 3 と点 5 は結びますが同じ点が重複して含まれる場合には、(「単純な」を取って）単に**パス**といいます。こちらも、辺は重複しない、という条件は含みます。単純なパスに含まれる辺の数をこのパスの**長さ** (length) といいます。通常、パスは辺を 1 本以上含むことを前提としますが、便宜上、1 点のみを長さ 0 のパスと考えることにします。特別に出発点と終点だけ同じであることを許した単純なパスを**単純なサイクル** (simple cycle)、出発点と終点が同じであるパスを**サイクル**とよびます。ただし、サイクルも単純なサイクルも 1 本以上の辺を含むこととします。パスは単純なパスを含みます。サイクルには単純なサイクルが複数個含まれることもあります。含まれる辺がすべて違うことはどの場合も同じです。イメージを確認していただくために図 4 に別の例を示しておきます。

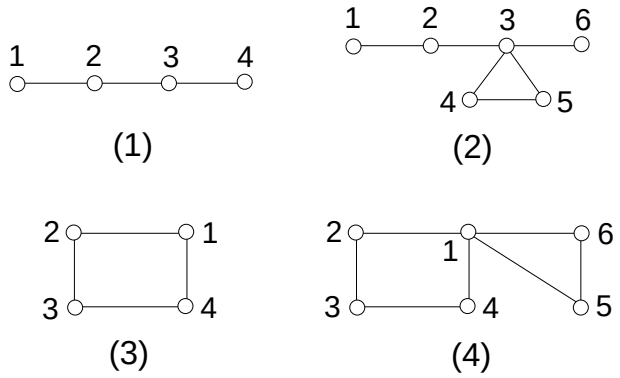

図 4　パスとサイクルの例　(1) 1 から 4 への単純なパス；(2) 1 から 6 へのパス；(3) 単純なサイクル；(4) サイクル

用語についての補足　パスやサイクルをはじめとしてグラフに関する用語は世界共通という訳ではありません。むしろ同じ用語が人によってあるいは状況によって少し違う意味で用いられることがよく見られます。ですから、使われている用語の意味をよく確かめるようにしましょう。ここでは、耳慣れない用語はできるだけ増やさないことを意識して選びました。「パス」と「単純なパス」、「サイクル」と「単純なサイクル」、それぞれの違いに少しだけ注意をしてください。「単純な」が付かないときは、「同じ点が複数回出てもいい」ということを意味します。一方、「単純な」が付くと「全て違う点である」という条件が追加されます。どちらも、辺はすべて異なる、というのは同じです。

どのような 2 点間にもそれらを結ぶ単純なパスがあるとき、グラフは**連結** (connected) であるといいます。G_1 は連結です。特別な場合として、1 点のみからなるグラフは連結です (長さ 0 のパスで自分自身と連結しています)。一方、G_1 から辺 $\{3,4\}$ を削除する（両端点は残し、線分のみ除きます）と図 5 のグラフ G_1' となります。G_1' は連結ではありません。以後、G_1' は**非連結** (disconnected) である、という言い方をします。

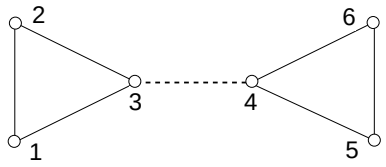

図 5　グラフ G_1'

1.2　グラフの描画について

図 3、図 6、図 7、図 8 はすべてグラフ G_1 の**描画**（drawing: 紙面などの平面上に書いたもの）です。点はどこに配置してもよいし、辺は両端点を結べばどのように描いてもいいです。ただし、途中で点を通過することは禁止することとし、点を通過しない限り辺が交わるように描画しても差し支えありません。さて、皆さんはこれらの 4 つのグラフが同じものだとすぐに分かるでしょうか。

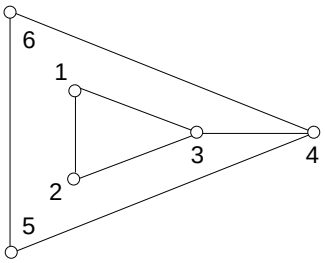

図 6　G_1 の別描画（2）

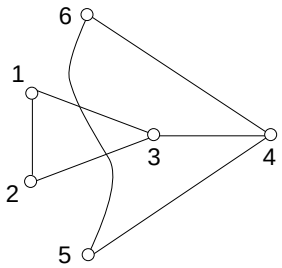

図 7　G_1 の別描画（3）

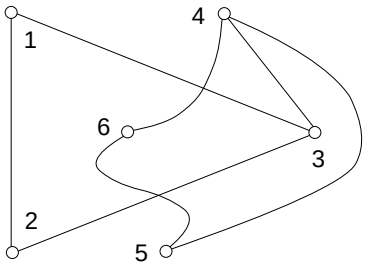

図 8　G_1 の別描画（4）

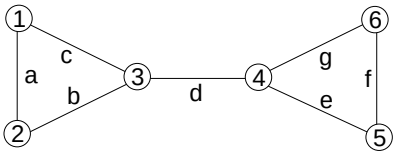

図 9　グラフ G_1（点の名前を内部に記載し、辺にラベル付け）

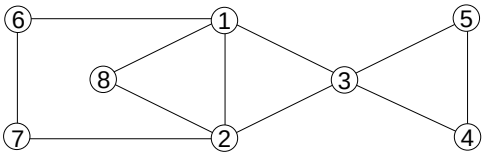

図 10　グラフ G_2

1.3 点の次数について

さて、これからの説明の都合で図 3 の G_1 の各辺にアルファベットで名前を付けることにします。たとえば、辺 $a = \{1, 2\}$ というように a〜g と名前を付けて、図 9 に再び掲載しておきます。これについて考えていきます。G_1 は点数 $|V_1| = 6$、辺数 $|E_1| = 7$ です。

辺 $a = \{1, 2\}$ とその両端点 $1, 2$ に着目してみましょう。点と点の関係として、1 と 2 が隣接している、という言い方は既に紹介しましたが、点と辺の関係としては、点 1 には辺 a が**接続** (incident) している、と言います。他の点についても同様の言い方です。さらに、点 1 には辺 a と c が接続していると言います。点 1 に接続している辺の総数を点 1 の**次数** (degree) とよび、$d(1)$ あるいは $d_{G_1}(1)$ と記号で表します。(G_1 は単純グラフですので、点 1 の次数は隣接点の個数に等しくなります。多重グラフの場合は等しくなるとは限りません。) $d_{G_1}(1)$ の下添字 G_1 は対象のグラフを明示するために付ける場合がありますが、どのグラフかわかっている場合には省略します。辺が接続していない点の次数は 0 で、**孤立点** (isolated vertex) とよびます。特別な場合ですが、一つの点 v から v への辺を**自己ループ** (self-loop) とよび、辺は 1 本と勘定し v の次数は 2 だけ増やします。1 本の自己ループは一つの単純なサイクルをなします。点 v での複数の自己ループを合わせると v を複数回通過するサイクルとなります。ところで、2 点以上を含むグラフが連結であれば、孤立点はありません。ですから、

2 点以上を含む連結グラフでは各点の次数は 1 以上、つまりどの点にも 1 本以上の辺が接続している

ということになります。このことは頭の隅に置いておいてください。

さて、G_1 については次数は以下のようになり、合わせてその次数の総和も示しましょう。

$$d(1) = d(2) = d(5) = d(6) = 2,\ d(3) = d(4) = 3,\ \text{次数総和} = 14 = 2 \times |E_1|$$

さらに、図 10 に別のグラフ $G_2 = (V_2, E_2)$ を示します。ただし

$$V_2 = \{1, 2, 3, 4, 5, 6, 7, 8\},$$
$$E_2 = \{\{1, 2\}, \{2, 3\}, \{3, 1\}, \{3, 4\}, \{4, 5\}, \{5, 3\}, \{1, 6\}, \{6, 7\}, \{7, 2\}, \{1, 8\}, \{2, 8\}\}$$

であり、G_2 は連結です。G_2 は点数 $|V_2| = 8$、辺数 $|E_2| = 11$ です。同様に G_2 について、各点の次数とその総和を示してみます。

$$d(4) = d(5) = d(6) = d(7) = d(8) = 2,\ d(1) = d(2) = d(3) = 4,\ \text{次数総和} = 22 = 2 \times |E_2|$$

ここで何か気が付かれたでしょうか。どちらの場合も次数総和が辺数の 2 倍に等しくなっています。(ですから、総和は偶数です。)

さて、これは偶然でしょうか。あるいは、常に次数総和は辺数の 2 倍に等しいのでしょうか。考えてみてください。

> **中高生、大学 1〜2 年の皆さんへ**　考えて（たとえ途中で止まったとしても）自分なりの結論を用意して以下の説明に進むようにすると、少しずつ考える力がついてくると思います。そのような取り組みをすると、最後まで辿りつけなかったとしても、なぜそこで止まったのか、他にどういう考え方があったのか、などがはっきりと見えて、ものごとを考える力をつけていくことにつながります。途中で進めなくなったときは悔しい気持ちもありますね。

実は、この等式はグラフに対して常に成り立ちます。なぜそうなるのか、G_1 を例としてできるだけ分かりやすく説明したいと思います。いろいろな考え方があると思いますが、1 つの考え方を図に示すことをやってみます。

図 11〜図 18 をご覧ください。図 11 に示すように G_1 の点のみを置いておき、そこから辺を a から順に g まで 1 本ずつ付け加えていき、各辺を付け加えたときに点の次数がどのようになるかを順を追って示したものです。図 12 では、a が加わったことで、両端点 1 と 2 の次数が 1 ずつ増えたことを示しています。(a) は a を付加したことを示し、(1) はその辺の追加により次数が 1 だけ増えたことを表します。このとき追加される値 1 はそれまでの追加分とは別です。そのことを示すために、たとえば図 13 では $1+(1)$ と表記しています。a の追加で増えた次数の増加分 1 と b の追加で増えた次数の増加分 1 は別物という意味です。以下、c, d, e, f, g と追加して次数の変化を追ってみて下さい。

辺 1 本につき次数増加分の 1 が 2 つずつありますので、置かれた 1 の個数の合計は $2 \times$ 辺数 $|E_1|$ です。一方、図 18 からも分かるように、最終的にすべての辺を置き終わったときに各点に置かれた 1 の個数が点の次数となり、これらの合計が次数総和です。以上が、次数総和が $2 \times |E_1|$ に等しくなる理由です。

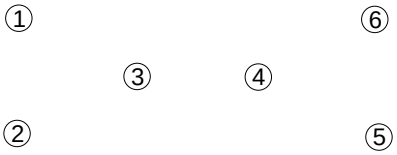

図 11　グラフ G_1 の点のみ表示

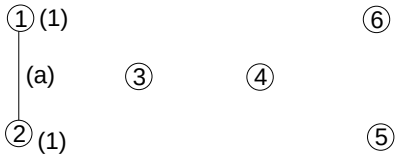

図 12　辺 a を追加

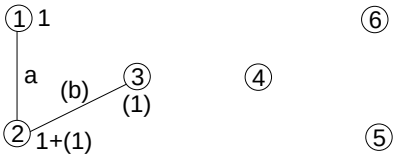

図 13　辺 a, b を追加

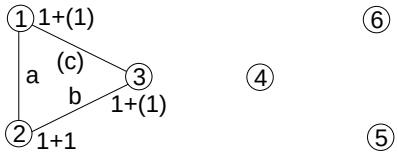

図 14　辺 a, b, c を追加

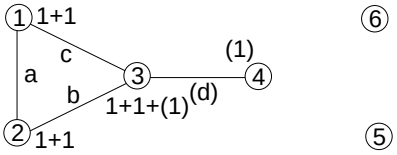

図 15　辺 a, b, c, d を追加

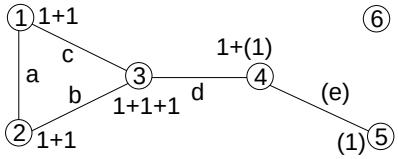

図 16　辺 a, b, c, d, e を追加

ちょっと一言　40 年以上も前のことですが、市内の私立高校と公立高校に在籍していた知り合いの生徒二人に上記の次数総和について同じ問題をクイズとして尋ねたことがあります。共に有名校で成績も悪くない生徒でした。一人は考えること自体をしませんでした。もう一人は少し考えてから「習ったことがないのでどう考えたらいいかわからん」と言いました。二人とも、その当時の授業では教えられていない形式、内容だったことが大きな壁だったようです。対応の仕方は違えど同じような理由でした

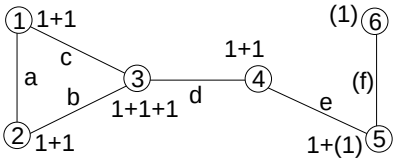

図 17　辺 a, b, c, d, e, f を追加

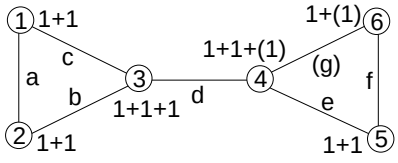

図 18　辺 a, b, c, d, e, f, g を追加（G_1 となる）

し、当然のことだったのかもしれません。ただ私にはその記憶がありましたので、皆さん対してくどいとは思いましたが、高校生に説明するイメージで解説を書いてみました。ものごとを考える、という訓練がその後高校でどのように行われているのか、私には興味があります。

コンピュータプログラミングは IT（情報技術）あるいは ICT（情報通信技術）の基盤となる不可欠な技術です。(以後、「コンピュータ」は省略して単に「プログラミング」と言うことにします。) また、IT&ICT 分野を支える人材に必須の能力でもあります。しかし、日本ではプログラミング力を備えた人材の不足が深刻で、その人材育成が叫ばれて久しいのですがなかなか成果が見えていないようです。国家の重大問題と言っても過言ではないのですが、長くなりますのでここでは詳しいことには触れません。**プログラミング** (programming) は、一言で言えば、

> コンピュータに物事を処理させるための**プログラム**（program: 具体的な操作系列で構成された詳細な指令書；専用の特殊な用語で記述する）を作ること

です。その人材育成には

- 未知の事象や現象などを取り扱いの対象や課題とし、それを「問題」として認識してその解明、解決に挑戦する
- 問題に含まれる様々な要因、項目を具体的に抽出し、これらのつながり、順序、強弱などの関連性を明確にする
- 明らかになった要因、項目の関連性に基づいて、問題の解明、解決のための具体的な方針、方法を決める
- 具体的な方針、方法に沿って、誤動作なく迅速に動作し、外からの攻撃に対して頑強な（影響を受けにくい）プログラムを作成する

などの視点からの教育が求められます。今後の IT&ICT 人材を育成するという観点から言えば、中高生や大学 1, 2 年など若い人に対する上記のような視点からの教育は重要です。さて、現状ではどうでしょうか。

ちょっとした思いつきとキーボード操作で簡単なプログラミングによってゲームなどを作ってみる、といった遊びの要素はプログラミングへの「誘い」としての意義は十分にあります。一方、別な視点から言うと、中途半端なやり方であるいは我流である程度プログラミングができるようになった若い人の矯正に、それまでの「自分はできる」という意識が大きな障害となることが多々あるということです。ある時点で将来を見据えて本物を教えて身に付けさせることが重要になってくると思います。これは何もプログラミングに限った話ではなく、たとえばスポーツなどでもよく見受けられることです。