課題３：アルゴリズム入門（その１）

講義ノート

プログラムを設計する際，重要になるのがアルゴリズム（計算手順）である．同じ計算目標を達成するにも無数のアルゴリズムがあり，その良し悪しで計算効率は非常に（たとえば，ゆうに1兆倍以上）異なってしまう．コンピュータを活かすも殺すもアルゴリズム次第なのである．今回の課題では，そのアルゴリズムの良し悪しを体験しよう．

1. 神童ガウスの計算法

まずは，歴史上の天才数学者であるガウスの子供の頃の逸話から...（黒板で説明します.）

これは計算技とでも言うべき技巧だが，アルゴリズムの工夫には，このような計算技から非常に巧妙な方法や高度な数学に基づく方法まで，いろいろとある．また，計算技でも使い方によっては非常に有効な場合もある．今回は，その一例として，べき乗を高速に計算する計算技の威力を見ることにしよう．

べき乗とは x^n のこと．肩にのっかている数 n のことをべき数という．この計算はある意味簡単だ．x を $n-1$ 回かければよい．プログラム風に書くと

```
tmpexp = x
for t in 2..n
   tmpexp = tmpexp * x
end
ans = tmpexp
```

となる．これは乗算が n 回必要となる．この計算をもっと高速にできないだろうか？

たとえば，$n = 2048$ の場合を考えてみる．はたして，2048 回の乗算をする必要が本当にあるだろうか？[1] 実は，たった 11 回の乗算で計算できるのだ．次のようにすればよいのである．

```
tmpexp = x
for s in 1..11
   tmpexp = tmpexp * tmpexp
end
ans = tmpexp
```

このようにすると tmpexp の値は

$$x,\ x^2,\ x^4,\ x^8,\ x^{16},\ x^{32},\ x^{64},\ x^{128},\ x^{256},\ x^{512},\ x^{1024},\ x^{2048}$$

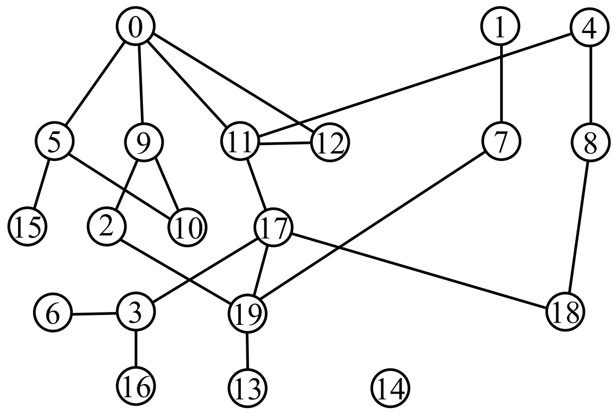
[1] もっとも，x^{2048} は異様に大きな数になってしまうので，実際に計算する必要があるかどうかは自体，疑問ではありますが...

と高速にべき数が倍々になるのである．なお，この方法は $n = 2^k$ という形のべき数にしか使えないが，もう少し工夫すれば，どんなべき数にも応用できる．けれども，今回はこれで十分なのでその先は省略する．

2. 最短経路問題

さて，今回の課題の話をしよう．今回は具体的な計算目標として最短経路問題を考える．これは，カーナビや電車の乗り換え検索など，身近な応用のある計算問題である．まず，問題の表し方から説明しよう．

入力として与えられるのは経路情報である．たとえば，電車の場合，路線ごとに駅がどのようにつながっているか，といった情報だ．道路の場合には，各交差点が，どの交差点につながっているか，といった情報である．このような情報を一般的に表わすには，グラフを使うとよい．グラフは頂点と頂点間を結ぶ辺からなる図形で，たとえば下図のような図形である．



この図は 20 個の頂点とそれらを結ぶ辺からなるグラフである．たとえば，20 個の交差点が頂点で，その間をつなぐ道が辺である．ただし，この図では 14 番の頂点が孤立している．このようなことは実際の地図ではありえないだろう．

このようなグラフは，経路図などの他にも，様々なことを抽象的に表わすためによく使われている．この例は向きの無い辺からなる無向グラフだが，辺に向きがある有向グラフもよく用いられる．グラフを用いる際には，この図のように，各頂点に番号をふって，それを使って議論することが多い．ここでもその方法を使おう．

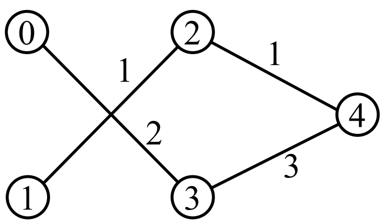
交差点のつながり方やとなり合った駅の関係を，このようなグラフでもらい，たとえば頂点 0 から頂点 7 までの経路を求めるのが経路探索問題である．一般には辺にはコストの情報も与えられていることが多い．たとえば，その辺に対応する道の距離や移動するのに必要な時間，あるいは駅間の運賃などである．単に経路を見つけるだけでなく，コストの最も低い経路を見つけるのが最短経路問題である．

我々が使う経路探索ソフトでは，出発地と目的地を与えて経路を探索するのが普通だろう．それは 2 点間経路探索問題である．ここでは，より一般的に，すべての 2 点間の最短経路をいっぺんに求めることを目標としよう．このような問題は全頂点間最短経路問題といわれている．ここ

では簡単のため，最短経路ではなく最短経路「長」を求める問題を考える．これが我々が例として使う計算問題である．

ここで「？」と思った人は偉い！グラフをどうやってコンピュータに渡すのだろうか？ と疑問に思った人である．グラフのようなものを，どうやって 0 と 1 で表わすのだろうか？やり方はいくつかあるが，ここでは隣接行列を使う方法を紹介しよう．隣接行列とは，各頂点間の辺の有無を行列で表わしたものである．頂点 i から頂点 j 間の辺の有無を ij-成分の値で表わした $n \times n$ の行列 (a_{ij}) だ．普通は，辺が有ることを $a_{ij} = 1$ で，無いことを $a_{ij} = 0$ で表わす．このように行列で表わすことができれば，グラフも数字（最終的には 0 と 1 の列）で表わすことができるだろう．

ただし，今回は各辺に距離がある場合を考える．そうした距離情報も入れた隣接行列を隣接距離行列と呼ぶことにする．各 a_{ij} は頂点 i から頂点 j への辺の距離（km）である．ちなみに，頂点 i から i は 0km なので $a_{ii} = 0$ である．辺の無いときは $a_{ij} = +\infty$ としよう．説明では見やすさを考慮してチェック印を使うことにする．たとえば，下図の左のグラフ G に対して，右の行列が G の隣接距離行列 $A_G = (a_{ij})$ である．なお，我々は無向グラフを考えているので隣接行列は対称行列となる．



$$
\begin{array}{c}
\\ 0 \\ 1 \\ 2 \\ 3 \\ 4
\end{array}
\begin{array}{c}
\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\
\begin{pmatrix}
0 & \checkmark & \checkmark & 2 & \checkmark \\
\checkmark & 0 & 1 & \checkmark & \checkmark \\
\checkmark & 1 & 0 & \checkmark & 1 \\
2 & \checkmark & \checkmark & 0 & 3 \\
\checkmark & \checkmark & 1 & 3 & 0
\end{pmatrix}
\end{array}
$$

この行列の各行を並べて表わせば，数の列となる．ただし，$+\infty$ を数で表すために -1 を使うことにする．これならばコンピュータへ入力として与えることができる．つまり，頂点数 n と，グラフ G の隣接距離行列 A_G を表わす数列が入力である．

出力も同じように行列で表わすことができる．つまり，ij-成分 d_{ij} が，頂点 i から j への最短経路長を表わしているような行列 (d_{ij}) を，答えとすればよいのだ．この行列を A_G^* と呼ぶことにしよう．なお，頂点 i, j 間に経路が無い場合には $d_{ij} = +\infty$ とする（ここでも $+\infty$ を -1 と表記する）．

このように考えると，我々の問題は次の関数の計算問題となる．

$$length(n, A_G) \;=\; A_G^*$$

ただし，A_G はグラフ G の隣接距離行列，そして $A_G^* = (d_{ij})$ の各 ij-成分 d_{ij} は，グラフ G の頂点 i から j までの最短経路長．

この関数を計算するアルゴリズムをこれから議論していく．

3. 最短経路長計算アルゴリズム

与えられたグラフを G，その隣接距離行列を $A_G = (a_{ij})$ とする．1 つ辺を通ることを「1 ステップ」とよぼう．すると，この隣接行列の ij-成分は，頂点 i から j への 1 ステップ以下での距離を表わしている．辺があれば a_{ij}km で，$i = j$ ならば $a_{ij} = 0$（つまり 0km）で到達できることを表わしている．では，もう 1 ステップ増やして，2 ステップ以下の距離を求めてみよう．つまり，辺を 2 つまで使う道のりでの距離だ．

たとえば，頂点 0 から 4 への最短経路（ただし，2 ステップ以下の経路）を考えてみる．その行き方は，頂点 0 から頂点 x に 1 ステップ（以下）で行き，その x から頂点 4 に 1 ステップ（以下）で行く方法のみである．たとえば，$x = 1$ だとすると，それは頂点 1 を中継点として 0 から 4 に進む道のりである．その長さは，0 から 1 に 1 ステップで行く距離 a_{01} と，さらに 1 から 4 に 1 ステップで行く距離 a_{14} の和となる．この中継点 x として $0 \sim 4$ を用いることで，すべての（ステップ 2 以下の）道のりを確認できたことになる．その中の最小値が 2 ステップ以下の最短距離である．

$$
\left.\begin{array}{ccccccc}
0 & \overset{a_{00}}{\to} & 0 & \overset{a_{04}}{\to} & 4 & \quad & a_{00} + a_{04} \\
0 & \overset{a_{01}}{\to} & 1 & \overset{a_{14}}{\to} & 4 & \quad & a_{01} + a_{14} \\
0 & \overset{a_{02}}{\to} & 2 & \overset{a_{24}}{\to} & 4 & \quad & a_{02} + a_{24} \\
0 & \overset{a_{03}}{\to} & 3 & \overset{a_{34}}{\to} & 4 & \quad & a_{03} + a_{34} \\
0 & \overset{a_{04}}{\to} & 4 & \overset{a_{44}}{\to} & 4 & \quad & a_{04} + a_{44}
\end{array}\right\} \Leftarrow \text{ この中の最小値が答え}
$$

頂点 0 から頂点 4 へ 2 ステップ以下で行く最短距離は？

この計算をプログラム風に書くと次のようになる．

```
tmp = +∞                # 無限に大きな値を暫定の最小値としておく
for k in 0..n-1
   x = a_0k + a_k4
   tmp = min(tmp, x)    # min(a,b) は a,b の小さい方の値を返す関数
end
ans = tmp
```

このプログラムでは，最終的に変数 ans に最小値が求まっているはずである．なお，どの中継点を通っても 2 ステップ以下では到達できない場合には ans $= +\infty$ となる．正確には $+\infty$ という数はないので「とっても大きな数」で代用する.（さて，いくつにすればよいでしょう？これは宿題の問題で考えることにします.）

行列 $D_G^{(2)} = (d_{ij}^{(2)})$ で，この 2 ステップ以下での i, j 間の距離を表わすことにする．上記の計算は $d_{04}^{(2)}$ の計算だったが，それをすべての i, j で行えば，行列 $D^{(2)} = (d_{ij}^{(2)})$ が求まる．その計算は次図左のようになる．

```
for i in 0..n-1
  for j in 0..n-1
    tmp = +∞
    for k in 0..n-1
      x = a_ik + a_kj
      tmp = min(tmp, x)
    end
    d_ij^(2) = tmp
  end
end
```

```
for i in 0..n-1
  for j in 0..n-1
    tmp = 0
    for k in 0..n-1
      x = t(a_ik,b_kj)      # t(u,v) = u×v
      tmp = p(tmp, x)       # p(u,v) = u+v
    end
    c_ij = tmp
  end
end
```

これはどこかで見た計算である．上図右の計算である．これは，行列 $A=(a_{ij})$ と $B=(b_{ij})$ の積 $A\times B$ となる行列 $C=(c_{ij})$ を求める計算である．その中の $*$ を $+$ に，$+$ を `min` に代えただけである．(**注意！**もう１つだけ変更箇所がある．それは，`tmp` の初期値の与え方．ここも代える必要がある.)

このように要素ごとの乗算，和算の解釈を変えた上での行列積を $\oplus$ と定義すると，

$$D_G^{(1)} \;=\; A_G, \quad D_G^{(2)} \;=\; D_G^{(1)} \oplus A_G$$

となる．その計算を

$$D_G^{(3)} \;=\; D_G^{(2)} \oplus A_G, \quad D_G^{(4)} \;=\; D_G^{(3)} \oplus A_G, \ldots$$

と行っていけば，$D_G^{(m)}$ を求めることができる．つまり，m ステップ以下道のりでの最短距離計算は，行列 A_G のべき乗を求める計算の形で表わすことができるのである．では，A_G^* には，べき数をいくつまで求めればよいのだろうか？これも今日の宿題の中で答えてもらうことにしよう．

$\boxed{\text{宿題}}$（提出 12 月 3 日；次回は演習室です！）

配布されたプリントへ回答を記入し，いつもの通り授業開始前に提出しチェックを受け，授業終了時に提出すること。