H24. 秋. アルゴリズム ①

## 問8　駅間の最短距離を求めるプログラム（データ構造及びアルゴリズム）（H24 秋-FE 午後問 8）

**【解答】**

a－ウ，b－ア，c－エ，d－ウ，e－エ，f－ウ，g－ウ

**【解説】**

駅間の最短距離を求めるアルゴリズムの問題である。各駅間の最短距離を求めるプログラムのトレース，計算量，メモリ使用量を減らすためのプログラム改変などについて問われている。プログラム自体は長くないので，図や表と対応付けながら説明文の内容を理解できたかどうかが正解を導くためのポイントとなる。難易度としては普通といえる。

最初に配列 **Dist** の各要素について確認する。〔プログラムの説明〕からも分かるように，副プログラム **CalcDist** に渡す配列 **Dist** は，2 駅 i，j が隣接しているときは **Dist[i][j]** 及び **Dist[j][i]** にその 2 駅間の距離，2 駅が隣接していないときは **Dist[i][j]** 及び **Dist[j][i]** に未確定を意味する 999，**Dist[i][i]**（各駅）には 0 が格納されており，図 2「配列 **Dist** の内容の変化を印字した結果」の“初期値”のようになっている。図 1「鉄道の路線例(1)」の路線情報と対応付けて確認すると，**Dist[1][2]**，**Dist[2][1]** の 18 は駅①－②間の距離であり，**Dist[1][3]**，**Dist[3][1]** の 999 は駅①と駅③が隣接していないことを示す未確定の距離である。この後，副プログラム **CalcDist** によって配列 **Dist** の内容を図 2 のように，“**Via=1**”から“**Via=5**”まで更新しながら各駅間の最短距離を求め，プログラムの実行が終わると任意の 2 駅間の最短距離が配列 **Dist** に求められていることになる。

j

| i | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 18 | 999 | 14 | 999 |
| 2 | 18 | 0 | 17 | 999 | 16 |
| 3 | 999 | 17 | 0 | 13 | 999 |
| 4 | 14 | 999 | 13 | 0 | 12 |
| 5 | 999 | 16 | 999 | 12 | 0 |

配列 **Dist** の初期値

駅①－②間の距離

駅①と駅③は隣接していないので未確定を表す 999

次にプログラムについて確認する。

〔プログラム〕

```
○副プログラム: CalcDist(整数型: N, 整数型: Dist[][])
○整数型: From, To, Via
 1  ・Print(N, Dist)
 2  ■Via: 1, Via ≦ N, 1
 3  |■From: 1, From ≦ N, 1
 4  |■To: 1, To ≦ N, 1
 5  |▲Dist[From][To] ＞ Dist[From][Via] + Dist[Via][To]
 6  |・Dist[From][To] ← Dist[From][Via] + Dist[Via][To]
 7  |▼
 8  |■
 9  |■
10  ・Print(N, Dist)
11  ■
```

プログラムは三重ループで構成され，ループはいずれも 1 から N まで繰り返す。一番外側のループ（行番号 2～11）は，変数 **Via** の値を 1 から N まで 1 ずつ変化させながら行場号 3～10 の処理を実行し，その中で，中央のループ（行番号 3～9）も，変数 **From** の値を 1 から N まで 1 ずつ変化させながら行番号 4～8 の処理を実行する。更に，一番内側のループでは変数 **To** の値を 1 から N まで 1 ずつ変化させながら，行番号 5～7 の処理を実行する。このとき，変数 N は駅数を表すので，図 1 の場合は N=5 になり，**Via**，**From**，**To** は次のように変化する。

| Via | From | To |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 1 | 3 |
| 1 | 1 | 4 |
| 1 | 1 | 5 |
| 1 | 2 | 1 |
| 1 | 2 | ⋮ |
| 1 | 2 | 5 |
| 1 | ⋮ | ⋮ |
| 1 | 4 | 1 |
| 1 | 4 | ⋮ |
| 1 | 4 | 5 |
| 1 | 5 | 1 |
| 1 | 5 | ⋮ |
| 1 | 5 | 5 |

| Via | From | To |
|---|---|---|
| 2 | 1 | 1 |
| 2 | 1 | ⋮ |
| 2 | 1 | 5 |
| 2 | ⋮ | ⋮ |
| 2 | 5 | 1 |
| 2 | 5 | ⋮ |
| 2 | 5 | 5 |
| ⋮ | ⋮ | ⋮ |
| 5 | 1 | 1 |
| 5 | 1 | ⋮ |
| 5 | 1 | 5 |
| 5 | ⋮ | ⋮ |
| 5 | 5 | 1 |
| 5 | 5 | ⋮ |
| 5 | 5 | 5 |

行番号 5，6 にある **Dist[From][To]**，**Dist[From][Via]+Dist[Via][To]** という記述から，変数 **Form** と **To** は距離を求める 2 駅の番号，変数 **Via** は駅 **From** から駅 **To** へ向かう際に経由する駅の番号を表すことが読み取れる。したがって，副プログラム **CalcDist** は，経由する駅を①から⑤まで変化させながら，駅①から順に，任意の駅（**From**，**To**）間の現時点の最短距離（**Dist[From][To]**）と基準となる駅（**Via**）を経由した場合の 2 駅間の距離（**Dist[From][Via]+Dist[Via][To]**）を比較し（行番号 5），最短距離を更新していく（行番号 6）ことが分かる。ちなみに，**Via** というのは，「～経由で」というような意味の英語である。

［設問］

・空欄 a，b：空欄 a は駅①－③間の距離，空欄 b は駅③－⑤間の距離が入る部分である。この表は **Via=2** のものなので，空欄 a には，駅②を経由したときの駅①－③間の距離が入る。**Via=1** のときの配列 **Dist** の内容を見ると，同時点の駅①－③間の距離（**Dist[1][3]**）には 999 が格納されている。この値は，駅②を経由する，駅①－②間の距離（**Dist[1][2]**＝18）と駅②－③間の距離（**Dist[2][3]**＝17）の和である **35** よりも大きいため，駅①－③間の距離は **35** に更新される。同様に，空欄 b に入る駅③－⑤間の距離も，**Via=1** のときの 999 から，駅③－②間の距離（**Dist[3][2]**＝17）と駅②－⑤間の距離（**Dist[2][5]**＝16）の和である **33** に更新される。したがって，空欄 a は（ウ），空欄 b は（ア）が正解である。

・空欄 c，d：計算量には時間計算量と領域計算量がある。時間計算量は処理にどの程度の時間が必要であるかを示し，領域計算量は処理にどの程度の記憶領域が必要であるかを示す。通常，計算量という場合には，時間計算量を指す場合が多く $O$ 記法（オーダ記法）を用いて表記される。例えば，$O(N)$は処理時間（命令の実行数）がデータの件数 N に比例することを，$O(1)$はデータの件数に関係なく処理時間が一定であること示す。$O$ 記法はおおよその傾向を表現するためのものであるので，N の係数や定数は無視し，処理の回数が 2N＋10 のような場合には $O(N)$と表記する。また，項は最高次以外は無視し，$N^2+2N+1$ のような場合には $O(N^2)$と表記する。

副プログラム **CalcDist** は，前述のように三重ループで構成されている。そして，行番号 4～8 のループは処理を N 回繰り返すのでオーダは $O(N)$となる。次に，行番号 3～9 のループも処理を N 回繰り返すが，その中で行番号 4～8 が N 回実行されるので，ここまでの二重ループで処理回数は $N\times N=N^2$ となりオーダは $O(N^2)$となる。そして，この副プログラムの計算量を決める，行番号 2～11 の繰返し処理も N 回実行されるが，更に，その中で $N^2$ 回の処理が実行されるので，最終的に三重ループの処理回数は $N\times N^2=N^3$ となりオーダは $O(N^3)$となる。よって，**CalcDist** の計算量のオーダは $O(N^3)$である。次に，メモリ使用量のオーダは最も容量を必要とする 2 次元配列 **Dist** について考えればよく，配列のサイズは $N\times N=N^2$ となるのでメモリ使用量のオーダは $O(N^2)$となる。したがって，空欄 c は（エ），空欄 d は（ウ）が正解である。

・空欄 e：配列 **Dist** は 2 駅間の距離を格納するためのもので，駅①から駅②までの距離が **Dist[1][2]**，駅②から駅①までの距離が **Dist[2][1]** に，それぞれ格納される。しかし，駅①から駅②と駅②から駅①については，進行方向は逆ではあるが距離は変わらない。下図の配列 **Dist** からも分かるように，右上と左下の値が対称となるため，2 駅間の距離は一方向（右上）だけを求めればよい。更に，**Dist[1][1]**，**Dist[2][2]** などの各駅を表す部分の距離は 0 であるため，この部分も対象外とすることができる。そして，駅①から駅②，③，④，⑤，駅②から駅③，④，⑤，駅③から駅④，⑤，駅④から駅⑤までの距離を求めるように処理を変更すれば，選択処理の実行回数を約半分に減らすことができる。したがって，空欄 e は変数 **To** の初期値を **From** の次の駅の番号（**From+1**）から始める（エ）が正解である。

| From ＼ To | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 18 | 999 | 14 | 999 |
| 2 | 18 | 0 | 17 | 999 | 16 |
| 3 | 999 | 17 | 0 | 13 | 999 |
| 4 | 14 | 999 | 13 | 0 | 12 |
| 5 | 999 | 16 | 999 | 12 | 0 |

配列 **Dist**（初期値の場合）

・空欄 f：各駅間の最短距離を求める方法の(4)に，2 駅間の関係には，両駅が属する区間が異なる場合と両駅が属する区間が同じ場合の 2 通りがあり，それぞれの場合で距離の求め方が違うということが記述されている。両駅が属する区間が異なる場合の最短距離は **D1**，**D2**，**D3**，**D4** のうちの最小値である。一方，両駅が同じ区間に属する場合の距離は，その区間によって直接行く距離と，異なる区間を経由して行く距離の 2 通りが考えられる。

空欄 f のあるアルゴリズムで 2 駅間の最短距離 **Res** を求める式を見ると，空欄 f が真の場合は，**Res←min(abs(ToKL[i]-ToKL[j]),D1,D2,D3,D4)**，偽の場合は，**Res←min(D1,D2,D3,D4)** となっている。したがって，真の場合が，両駅が同じ区間に属するとき，偽の場合が，両駅が属する区間が異なるときであることが分かる。そして，この二つの処理のどちらを実行すべきかは，両駅が同じ区間かどうかの判定結果によって決まるが，その駅が属する区間の区間名は駅情報表の **Sec** に格納されているので，**Sec[i]=Sec[j]** であれば，両駅は同じ区間に属することになる。したがって，空欄 f は（ウ）が正解である。なお，問題に示されている駅情報表は次のようになっている。

| 駅番号 | Sec | KL | ToKL | KH | ToKH |
|---|---|---|---|---|---|
| 1 | “1” | 1 | 0 | 1 | 0 |
| 2 | “2” | 2 | 0 | 2 | 0 |
| 3 | “3” | 3 | 0 | 3 | 0 |
| 4 | “4” | 4 | 0 | 4 | 0 |
| 5 | “B” | 1 | 15 | 3 | 30 |
| 6 | “B” | 1 | 25 | 3 | 20 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

**Sec[5]** の値の“B”は，駅⑤が区間 B に属することを表し，**KL[5]** の値の 1，**KH[5]** の値の 3 は，駅⑤が属する区間 B の両端の駅が，それぞれ駅①と駅③であることを表している。また，**ToKL[5]** の値の 15 は，駅⑤から駅①までの距離が 15，**ToKH** の値の 30 は，駅⑤から区間の終端である駅③までの距離が 30 であることを表している。したがって，同じ区間に属している駅⑤と駅⑥について，直接駅⑤から駅⑥へ行く場合の距離は，**abs (ToKL[5]-ToKL[6])** とすれば求めることができる。

H24. 秋. アルゴリズム ①

## 問8　駅間の最短距離を求めるプログラム（データ構造及びアルゴリズム）（H24 秋-FE 午後問 8）

**【解答】**

a－ウ，b－ア，c－エ，d－ウ，e－エ，f－ウ，g－ウ

**【解説】**

駅間の最短距離を求めるアルゴリズムの問題である。各駅間の最短距離を求めるプログラムのトレース，計算量，メモリ使用量を減らすためのプログラム改変などについて問われている。プログラム自体は長くないので，図や表と対応付けながら説明文の内容を理解できたかどうかが正解を導くためのポイントとなる。難易度としては普通といえる。

最初に配列 **Dist** の各要素について確認する。〔プログラムの説明〕からも分かるように，副プログラム **CalcDist** に渡す配列 **Dist** は，2 駅 i，j が隣接しているときは **Dist[i][j]** 及び **Dist[j][i]** にその 2 駅間の距離，2 駅が隣接していないときは **Dist[i][j]** 及び **Dist[j][i]** に未確定を意味する 999，**Dist[i][i]**（各駅）には 0 が格納されており，図 2「配列 **Dist** の内容の変化を印字した結果」の“初期値”のようになっている。図 1「鉄道の路線例(1)」の路線情報と対応付けて確認すると，**Dist[1][2]**，**Dist[2][1]** の 18 は駅①－②間の距離であり，**Dist[1][3]**，**Dist[3][1]** の 999 は駅①と駅③が隣接していないことを示す未確定の距離である。この後，副プログラム **CalcDist** によって配列 **Dist** の内容を図 2 のように，“**Via=1**”から“**Via=5**”まで更新しながら各駅間の最短距離を求め，プログラムの実行が終わると任意の 2 駅間の最短距離が配列 **Dist** に求められていることになる。

j

| i | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 18 | 999 | 14 | 999 |
| 2 | 18 | 0 | 17 | 999 | 16 |
| 3 | 999 | 17 | 0 | 13 | 999 |
| 4 | 14 | 999 | 13 | 0 | 12 |
| 5 | 999 | 16 | 999 | 12 | 0 |

配列 **Dist** の初期値

駅①－②間の距離

駅①と駅③は隣接していないので未確定を表す 999

次にプログラムについて確認する。

〔プログラム〕

```
○副プログラム: CalcDist(整数型: N, 整数型: Dist[][])
○整数型: From, To, Via
 1  ・Print(N, Dist)
 2  ■Via: 1, Via ≦ N, 1
 3  |■From: 1, From ≦ N, 1
 4  |■To: 1, To ≦ N, 1
 5  |▲Dist[From][To] ＞ Dist[From][Via] + Dist[Via][To]
 6  |・Dist[From][To] ← Dist[From][Via] + Dist[Via][To]
 7  |▼
 8  |■
 9  |■
10  ・Print(N, Dist)
11  ■
```

プログラムは三重ループで構成され，ループはいずれも 1 から N まで繰り返す。一番外側のループ（行番号 2～11）は，変数 **Via** の値を 1 から N まで 1 ずつ変化させながら行場号 3～10 の処理を実行し，その中で，中央のループ（行番号 3～9）も，変数 **From** の値を 1 から N まで 1 ずつ変化させながら行番号 4～8 の処理を実行する。更に，一番内側のループでは変数 **To** の値を 1 から N まで 1 ずつ変化させながら，行番号 5～7 の処理を実行する。このとき，変数 N は駅数を表すので，図 1 の場合は N=5 になり，**Via**，**From**，**To** は次のように変化する。

| Via | From | To |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 1 | 3 |
| 1 | 1 | 4 |
| 1 | 1 | 5 |
| 1 | 2 | 1 |
| 1 | 2 | ⋮ |
| 1 | 2 | 5 |
| 1 | ⋮ | ⋮ |
| 1 | 4 | 1 |
| 1 | 4 | ⋮ |
| 1 | 4 | 5 |
| 1 | 5 | 1 |
| 1 | 5 | ⋮ |
| 1 | 5 | 5 |

| Via | From | To |
|---|---|---|
| 2 | 1 | 1 |
| 2 | 1 | ⋮ |
| 2 | 1 | 5 |
| 2 | ⋮ | ⋮ |
| 2 | 5 | 1 |
| 2 | 5 | ⋮ |
| 2 | 5 | 5 |
| ⋮ | ⋮ | ⋮ |
| 5 | 1 | 1 |
| 5 | 1 | ⋮ |
| 5 | 1 | 5 |
| 5 | ⋮ | ⋮ |
| 5 | 5 | 1 |
| 5 | 5 | ⋮ |
| 5 | 5 | 5 |

行番号 5，6 にある **Dist[From][To]**，**Dist[From][Via]+Dist[Via][To]** という記述から，変数 **Form** と **To** は距離を求める 2 駅の番号，変数 **Via** は駅 **From** から駅 **To** へ向かう際に経由する駅の番号を表すことが読み取れる。したがって，副プログラム **CalcDist** は，経由する駅を①から⑤まで変化させながら，駅①から順に，任意の駅（**From**，**To**）間の現時点の最短距離（**Dist[From][To]**）と基準となる駅（**Via**）を経由した場合の 2 駅間の距離（**Dist[From][Via]+Dist[Via][To]**）を比較し（行番号 5），最短距離を更新していく（行番号 6）ことが分かる。ちなみに，**Via** というのは，「～経由で」というような意味の英語である。

［設問］

・空欄 a，b：空欄 a は駅①－③間の距離，空欄 b は駅③－⑤間の距離が入る部分である。この表は **Via=2** のものなので，空欄 a には，駅②を経由したときの駅①－③間の距離が入る。**Via=1** のときの配列 **Dist** の内容を見ると，同時点の駅①－③間の距離（**Dist[1][3]**）には 999 が格納されている。この値は，駅②を経由する，駅①－②間の距離（**Dist[1][2]**＝18）と駅②－③間の距離（**Dist[2][3]**＝17）の和である **35** よりも大きいため，駅①－③間の距離は **35** に更新される。同様に，空欄 b に入る駅③－⑤間の距離も，**Via=1** のときの 999 から，駅③－②間の距離（**Dist[3][2]**＝17）と駅②－⑤間の距離（**Dist[2][5]**＝16）の和である **33** に更新される。したがって，空欄 a は（ウ），空欄 b は（ア）が正解である。

・空欄 c，d：計算量には時間計算量と領域計算量がある。時間計算量は処理にどの程度の時間が必要であるかを示し，領域計算量は処理にどの程度の記憶領域が必要であるかを示す。通常，計算量という場合には，時間計算量を指す場合が多く $O$ 記法（オーダ記法）を用いて表記される。例えば，$O(N)$は処理時間（命令の実行数）がデータの件数 N に比例することを，$O(1)$はデータの件数に関係なく処理時間が一定であること示す。$O$ 記法はおおよその傾向を表現するためのものであるので，N の係数や定数は無視し，処理の回数が 2N＋10 のような場合には $O(N)$と表記する。また，項は最高次以外は無視し，$N^2+2N+1$ のような場合には $O(N^2)$と表記する。

副プログラム **CalcDist** は，前述のように三重ループで構成されている。そして，行番号 4～8 のループは処理を N 回繰り返すのでオーダは $O(N)$となる。次に，行番号 3～9 のループも処理を N 回繰り返すが，その中で行番号 4～8 が N 回実行されるので，ここまでの二重ループで処理回数は $N\times N=N^2$ となりオーダは $O(N^2)$となる。そして，この副プログラムの計算量を決める，行番号 2～11 の繰返し処理も N 回実行されるが，更に，その中で $N^2$ 回の処理が実行されるので，最終的に三重ループの処理回数は $N\times N^2=N^3$ となりオーダは $O(N^3)$となる。よって，**CalcDist** の計算量のオーダは $O(N^3)$である。次に，メモリ使用量のオーダは最も容量を必要とする 2 次元配列 **Dist** について考えればよく，配列のサイズは $N\times N=N^2$ となるのでメモリ使用量のオーダは $O(N^2)$となる。したがって，空欄 c は（エ），空欄 d は（ウ）が正解である。

・空欄 e：配列 **Dist** は 2 駅間の距離を格納するためのもので，駅①から駅②までの距離が **Dist[1][2]**，駅②から駅①までの距離が **Dist[2][1]** に，それぞれ格納される。しかし，駅①から駅②と駅②から駅①については，進行方向は逆ではあるが距離は変わらない。下図の配列 **Dist** からも分かるように，右上と左下の値が対称となるため，2 駅間の距離は一方向（右上）だけを求めればよい。更に，**Dist[1][1]**，**Dist[2][2]** などの各駅を表す部分の距離は 0 であるため，この部分も対象外とすることができる。そして，駅①から駅②，③，④，⑤，駅②から駅③，④，⑤，駅③から駅④，⑤，駅④から駅⑤までの距離を求めるように処理を変更すれば，選択処理の実行回数を約半分に減らすことができる。したがって，空欄 e は変数 **To** の初期値を **From** の次の駅の番号（**From+1**）から始める（エ）が正解である。

| From ＼ To | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 18 | 999 | 14 | 999 |
| 2 | 18 | 0 | 17 | 999 | 16 |
| 3 | 999 | 17 | 0 | 13 | 999 |
| 4 | 14 | 999 | 13 | 0 | 12 |
| 5 | 999 | 16 | 999 | 12 | 0 |

配列 **Dist**（初期値の場合）

・空欄 f：各駅間の最短距離を求める方法の(4)に，2 駅間の関係には，両駅が属する区間が異なる場合と両駅が属する区間が同じ場合の 2 通りがあり，それぞれの場合で距離の求め方が違うということが記述されている。両駅が属する区間が異なる場合の最短距離は **D1**，**D2**，**D3**，**D4** のうちの最小値である。一方，両駅が同じ区間に属する場合の距離は，その区間によって直接行く距離と，異なる区間を経由して行く距離の 2 通りが考えられる。

空欄 f のあるアルゴリズムで 2 駅間の最短距離 **Res** を求める式を見ると，空欄 f が真の場合は，**Res←min(abs(ToKL[i]-ToKL[j]),D1,D2,D3,D4)**，偽の場合は，**Res←min(D1,D2,D3,D4)** となっている。したがって，真の場合が，両駅が同じ区間に属するとき，偽の場合が，両駅が属する区間が異なるときであることが分かる。そして，この二つの処理のどちらを実行すべきかは，両駅が同じ区間かどうかの判定結果によって決まるが，その駅が属する区間の区間名は駅情報表の **Sec** に格納されているので，**Sec[i]=Sec[j]** であれば，両駅は同じ区間に属することになる。したがって，空欄 f は（ウ）が正解である。なお，問題に示されている駅情報表は次のようになっている。

| 駅番号 | Sec | KL | ToKL | KH | ToKH |
|---|---|---|---|---|---|
| 1 | “1” | 1 | 0 | 1 | 0 |
| 2 | “2” | 2 | 0 | 2 | 0 |
| 3 | “3” | 3 | 0 | 3 | 0 |
| 4 | “4” | 4 | 0 | 4 | 0 |
| 5 | “B” | 1 | 15 | 3 | 30 |
| 6 | “B” | 1 | 25 | 3 | 20 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

**Sec[5]** の値の“B”は，駅⑤が区間 B に属することを表し，**KL[5]** の値の 1，**KH[5]** の値の 3 は，駅⑤が属する区間 B の両端の駅が，それぞれ駅①と駅③であることを表している。また，**ToKL[5]** の値の 15 は，駅⑤から駅①までの距離が 15，**ToKH** の値の 30 は，駅⑤から区間の終端である駅③までの距離が 30 であることを表している。したがって，同じ区間に属している駅⑤と駅⑥について，直接駅⑤から駅⑥へ行く場合の距離は，**abs (ToKL[5]-ToKL[6])** とすれば求めることができる。