問８　Boyer-Moore-Horspool 法を用いた文字列検索（データ構造及びアルゴリズム）（H27 秋・FE 午後問 8）

【解答】

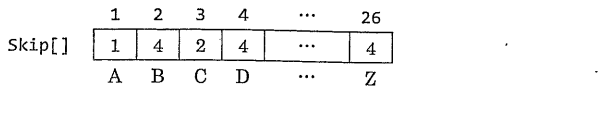
［設問１］　a－エ，b－カ
［設問２］　c－ア，d－オ，e－キ
［設問３］　f－イ

【解説】

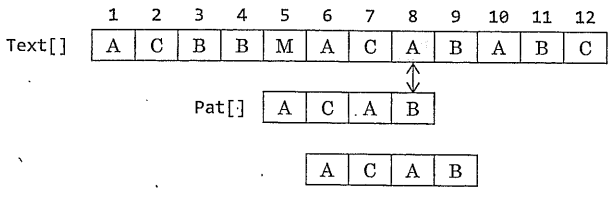
Boyer-Moore-Horspool 法（以下，BM 法）を用いた，文字列検索のアルゴリズムに関する問題である。設問は，プログラムの空欄の穴埋め，プログラムのトレース（追跡），プログラムの一部を変更した場合の動作の考察についてである。問題文には，文字列の照合手順が具体例とともに丁寧に示されており，BM 法に関する知識がなくても解答は可能である。プログラムは短く，示された例を使って，各処理が説明のどの手順に対応するのかを確認しながらトレースすれば，関数 BMMatch の処理を理解することができる。図と対応させながら問題文を確実に理解し，根気よく対応したい。

はじめに，BM 法がどのように文字列を照合するのかを確認する。BM 法は，検索文字列の末尾の文字から先頭に向かって，検索対象の文字列（以下，対象文字列）と 1 文字ずつ順に比較して照合を行う。比較した文字が一致せず，照合が失敗した場合は，検索文字列中の文字の情報を利用して，次に照合を開始する対象文字列の位置を決める。関数 BMMatch では，対象文字列を Text[]，検索文字列を Pat[]とし，文字の照合が失敗した場合の移動量は，配列 Skip[]に格納する。具体的な照合手順は，〔プログラムの説明〕の(4)に図とともに示されているが，図 1 の例を用いてポイントを説明する。

文字の照合は Pat[]の末尾の文字から行うため，最初は，対象文字列の Text[4]と検索文字列の Pat[4]の文字を比較する。Text[4]，Pat[4]の文字はともに“B”であり，一致するので比較する文字を一つ前に移動する。Text[3]の文字は“B”，Pat[3]の文字は“A”なので，照合に失敗する。そこで，次に照合を開始する対象文字列の位置を決め，Pat[]を移動する。このとき，移動量は Pat[4]の文字と比較した Text[4]の文字を基準に決める。移動量は Skip[]に格納されており，文字“A”の移動量は Skip[1]，“B”の移動量は Skip[2]，…，“Z”の移動量は Skip[26]に対応している。Text[4]の文字は“B”なので，移動量は 4 となる。検索文字列には，“B”は Pat[4]にしか存在しないため，仮に 1 文字ずつ Pat[]をずらして Text[4]と Pat[3]，Pat[2]，Pat[1]の文字を比較しても，“B”と一致する文字はない。したがって，この場合は一気に 4 文字分移動でき，次に照合を開始する対象文字列の位置は，Text[8]となる。



移動後は，同様に検索文字列の末尾の Pat[4]の文字から照合を始める。Text[8]は“A”，Pat[4]は“B”なので文字は一致しないため，Pat[]を次の照合位置まで移動する。このとき，Text[8]の文字“A”は，Pat[3]と Pat[1]に存在するが，Text[8]と Pat[3]の“A”が重なるように“A”の移動量である 1 だけ Pat[]の位置をずらす。



このように，照合に失敗した場合は，Pat[4]と比較した Text[]の文字を基準に，Skip[]に格納された，Text[]の文字に対応する移動量分だけ Pat[]を右に動かしながら照合を行う。

続いて，関数 BMMatch の処理を確認する。

〔プログラム〕
(行番号)

```
 1  ○整数型関数：BMMatch（文字型：Text[]，整数型：TextLen，
                          文字型：Pat[]，整数型：PatLen）
 2  ○整数型：Skip[26]，PText，PPat，PLast，I

 3  ■ I：1，I ≦ 26，1
 4  |  ・Skip[I] ← [ a ]
 5  ■
 6  ■ I：1，I ≦ PatLen － 1，1 ←――― γ
 7  |  ・Skip[Index(Pat[I])] ← [ b ]
 8  ■
 9  ・PLast ← PatLen
10  ■ PLast ≦ TextLen
11  |  ・PText ← PLast ←――――――― α
12  |  ・PPat ← PatLen
13  |  ■ Text[PText] = Pat[PPat]
14  |  |  ▲ PPat = 1 ←――――――― β
15  |  |  ・return (PText)
16  |  |  ▼
17  |  |  ・PText ← PText － 1
18  |  |  ・PPat ← PPat － 1
19  |  ■
20  |  ・PLast ← PLast ＋ Skip[Index(Text[PLast])]
21  ■
22  ・return (－1)
```

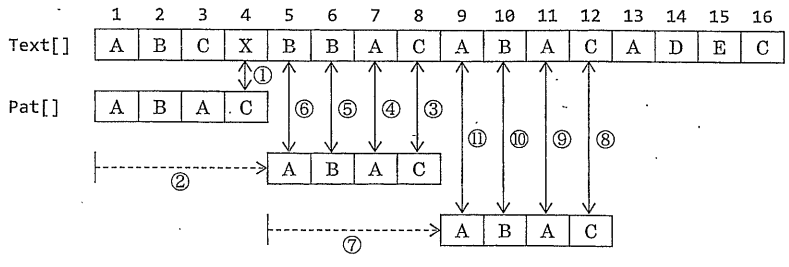
・行番号 1：関数の宣言。BMMatch は関数名，（　）の中は，関数が受け取る値（引数）を格納する変数名と型の宣言。関数名の前に記述した型は，関数の返却値の型を表す。これらの詳細は，〔関数 BMMatch の引数と返却値〕で示されている。
・行番号 2：関数内で使用する配列と変数の宣言。
・行番号 3～5：配列 Skip[]の初期化。Skip[1]～Skip[26]の移動量の初期値として，検索文字列の長さである PatLen の値を格納。
・行番号 6～8：検索文字列の Pat[1]から Pat[PatLen－1]に現れる文字に対応する移動量を格納。移動量は，その文字が，検索文字列の末尾から何文字目に現れるかを数えた文字数から 1 を引いた値となる。複数回現れる文字は，最も末尾に近い文字に対応する移動量とする。
・行番号 9：検索文字列 Pat[]の末尾の文字と比較する，対象文字列 Text[]の位置を設定。最初は PatLen 文字目となる。
・行番号 10～21：文字列の検索処理。繰返し条件の PLast≦TextLen は，検索文字列の Pat[4]と比較する対象文字列 Text[]の位置が文字列の長さ（配列の要素数の上限）を超えていないかどうかのチェック。
・行番号 11：検索文字列 Pat[]と比較する対象文字列 Text[]の位置（添字）の初期値を設定。
・行番号 12：対象文字列 Text[]と比較する検索文字列 Pat[]の位置（添字）を末尾に設定。
・行番号 13～19：対象文字列 Text[]と検索文字列 Pat[]の文字の照合処理。
　・行番号 14：Pat[]の全ての文字と一致したか（検索文字列が見つかったか）どうかの判定。
　　・行番号 15：Pat[]の全ての文字と一致した場合に実行。返却値（Pat[]の文字列と一致する文字列が始まる Text[]の位置（添字））を返し，関数の実行を終わる。
　・行番号 17：Text[]の添字を更新。比較する文字を 1 文字前に移動。
　・行番号 18：Pat[]の添字を更新。比較する文字を 1 文字前に移動。
・行番号 20：次に照合を開始する Text[]の位置（添字）を決定。
・行番号 22：検索文字列が見つからなかった場合に返却値として－1 を返し，関数の実行を終わる。

［設問１］
・空欄 a，b：空欄 a，b ともに，配列 Skip[]へ値を格納する処理である。〔プログラムの説明〕の(2)から分かるように，Skip[]には，文字列の照合が失敗した場合に，検索文字列を次に照合を開始する位置まで移動する移動量を格納する。空欄 a のある行番号 4 の処理は，Skip[]の初期化である。〔プログラムの説明〕(2)の①に「検索文字列の末尾の文字 Pat[PatLen]にだけ現れる文字と，検索文字列に現れない文字に対応する移動量は，PatLen である」と記述されており，ここでは，まず，文字“A”～“Z”に対応する Skip[1]～Skip[26]の移動量を全て PatLen で初期化する。

その後，空欄 b のある行番号 7 の処理で，〔プログラムの説明〕(2)の②に記述されているように「検索文字列の Pat[1]から Pat[PatLen－1]に現れる文字に対応する移動量は，その文字が，検索文字列の末尾から何文字目に現れるかを数えた文字数から 1 を引いた値」に再設定する。ただし，複数回現れる文字は，最も末尾に近い文字に対応する移動量となる。末尾から数えた文字数から 1 を引いた値は，PatLen が 4 の場合，Pat[1]の文字は 3，Pat[2]の文字は 2，Pat[3]の文字は 1 となる。Pat[]の添字は変数 I であり，I は 1 から始まり，PatLen－1 まで 1 ずつ増えながら変化するため，移動量は PatLen から I を引くとよいことが分かる。

したがって，空欄 a には（エ）の「PatLen」が，空欄 b は（カ）の「PatLen － I」が入る。なお，Skip[Index(Pat[I])]は，Index(Pat[I])の部分が，Pat[I]の文字に対応する整数（“A”は 1，“B”は 2，…，“Z”は 26）を返すため，Pat[I]が“A”の場合は Skip[1]に，“B”の場合は Skip[2]に移動量を格納できる。

［設問２］
・空欄 c，d，e：与えられた対象文字列と検索文字列を用いて文字列の照合の過程を図にすると，次のようになる。なお，Text[]に格納されている文字の移動量は，“A”が 1，“B”が 2，“C”と残りの文字は全て 4 である。



① Text[4]と Pat[4]を比較する。Text[4]の“X”と Pat[4]の“C”は異なる文字である。
② ①で検索文字列の末尾の文字 Pat[4]と比較した Text[4]の“X”に対応する移動量である 4 だけ Pat[]を右に移動する。
③ Text[8]と Pat[4]を比較する。Text[8]と Pat[4]は同じ文字“C”である。
④ Text[7]と Pat[3]を比較する。Text[7]と Pat[3]は同じ文字“A”である。
⑤ Text[6]と Pat[2]を比較する。Text[6]と Pat[2]は同じ文字“B”である。
⑥ Text[5]と Pat[1]を比較する。Text[5]の“B”と Pat[1]の“A”は異なる文字である。
⑦ ③で検索文字列の末尾の文字 Pat[4]と比較した Text[8]を基準に，Text[8]の文字“C”に対応する移動量である 4 だけ Pat[]を右に移動する。
⑧ Text[12]と Pat[4]を比較する。Text[12]と Pat[4]は同じ文字“C”である。

⑨ Text[11]と Pat[3]を比較する。Text[11]と Pat[3]は同じ文字“A”である。
⑩ Text[10]と Pat[2]を比較する。Text[10]と Pat[2]は同じ文字“B”である。
⑪ Text[9]と Pat[1]を比較する。Text[9]と Pat[1]は同じ文字“A”である。

⑧～⑪の比較で，対象文字列 Text[]の連続した一部分が検索文字列 Pat[]に完全に一致したので，検索処理は終了する。

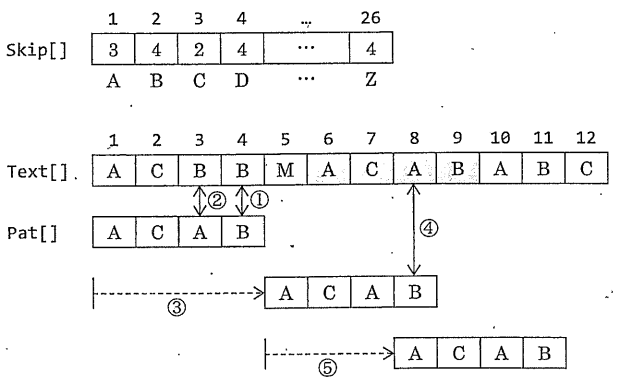
続いて，**α**（行番号 11），**β**（行番号 14）の処理を確認する。**α**の処理は，行番号 10～21 の繰返し処理の中にあり，変数 PText に値を設定する処理である。行番号 13 の繰返しの条件から，PText は配列 Text[]の添字であることが分かる。行番号 13～19 の処理は文字の照合処理であり，**α**の処理は，Pat[4]の文字と比較する Text[]の文字位置を設定していることが理解できる。**α**の処理は，繰返し処理に入った直後と繰返しの都度，実行される。繰返し処理に入った直後に設定される PText の値は 4 であり，繰返しの都度設定される PText の値は，行番号 20 で更新された，次に照合を開始する対象文字列の位置である。行番号 10～21 の繰返し処理は，Text[]に Pat[]と比較する文字が残っている間，行うが，Pat[]の文字列が Text[]内の連続した一部分と完全に一致すると，行番号 14 の条件が成立し，関数 BMMatch は終了する。トレース結果からも分かるように，Pat[4]と Text[]の文字を比較するのは 3 回であり，Pat[4]と比較する Text[]の位置を表す PText の値を設定するαの処理は 3 回実行することになる。したがって，空欄 c は（ア）が正解である。

続いて，**β**の処理を考える。**β**の処理は PPat の値が 1 かどうかを判定している。行番号 13 の繰返しの条件から，PPat は Pat[]の添字であることが分かる。PPat の初期値は PatLen であり，行番号 13～19 の繰返し処理で 1 ずつ減る。**β**の処理が実行されるのは，Text[PText]＝Pat[PPat]が成り立つ（比較した Text[]の文字と Pat[]の文字が一致した）場合である。トレースの図で確認すると，処理を終了するまでに，配列内の文字が一致するのは 7 回である。したがって，空欄 d は（オ）が正解である。

最後に関数 BMMatch の返却値であるが，Pat[]の文字列が Text[]中に見つかった場合，行番号 15 の処理で値を返却する。返却する値は PText の値であり，PText の値は，Pat[]の文字と比較する都度，1 ずつ減っていく。Pat[]の文字列と完全に一致する文字列が Text[]内に見つかった場合の値は，Pat[1]の文字と一致する文字がある Text[]の添字を返却することになる。トレース結果を確認すると，Pat[]と一致する文字列は Text[9]から始まっており，返却値は 9 となる。したがって，空欄 e は（キ）が正解である。

［設問 3］

・空欄 f：まず，**γ**の処理を変更することによって，図 1 の例で示された検索文字列に含まれる“A”，“C”の移動量がどのように変わるのかを確認する。変更前の処理では，移動量は検索文字列の先頭から算出するため，複数回現れる文字は，最も末尾に近い文字に対応する移動量となる。一方，変更後の処理では，PatLen＝4 なので，I は 3，2，1 と変化し，Pat[3]，Pat[2]，Pat[1]の順に移動量を格納することになり，複数回現れる文字は，最も先頭に近い文字に対応する移動量となる。その結果，文字“A”，“B”，“C”に対応する移動量は，次の図の Skip[]のようになる。これによって，Pat[4]と比較した Text[]と同じ文字が，Pat[]中に複数回現れる場合，照合が失敗すると，最も末尾に近い文字を Pat[4]と比較した Text[]の位置まで移動するのではなく，最も先頭に近い文字を Text[]の位置まで移動することになる。このとき，本来比較すべき文字を比較せずに通り越してしまうため，「対象文字列に，検索文字列が含まれているのに，（検索文字列を見つけることができずに，）－1 を返す場合がある」。したがって，（イ）が入る。なお，図 1 の例を使ってトレースすると次の図のようになる。また，（ア）については，行番号 14 の判定で，検索文字が全て一致したかどうかを確認しているため，発生することはない。



① Text[4]と Pat[4]を比較する。Text[4]と Pat[4]は同じ文字“B”である。
② Text[3]と Pat[3]を比較する。Text[3]の“B”と Pat[3]の“A”は異なる文字である。
③ ①で検索文字列の末尾の文字 Pat[4]と比較した Text[4]の“B”を基準に，“B”に対応する移動量である 4 だけ Pat[]を右に移動する。
④ Text[8]と Pat[4]を比較する。Text[8]の“A”と Pat[4]の“B”は異なる文字である。
⑤ ④で検索文字列の末尾の文字 Pat[4]と比較した Text[8]の“A”に対応する移動量である 3 だけ Pat[]を右に移動する。

この結果，Text[6]～Text[9]に存在する文字列“ACAB”を通り越してしまう。