各 a_i, b_j $(1 \leqq i \leqq n,\ 1 \leqq j \leqq n)$ が 1 から n の整数からなる LIST

$A = (a_1, a_2, \cdots, a_n)$

$B = (b_1, b_2, \cdots, b_n)$

から, 同様な LIST

$C = (c_1, c_2, \cdots, c_n)$

を作る. ただし, C は, 指定された整数 $1 \leqq k \leqq n$ に対して, 数 $a_1, \cdots, a_k$ は B と同じ位置にあり, 数 $a_{k+1}, \cdots, a_n$ はこの順序で現れるようなリストである.

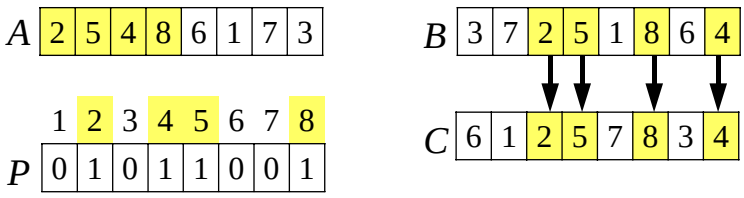
このような LIST C を作るため, 次のような操作をしていたのでは, $O(n)$ で作れない.

- 各 a_i $(1 \leqq i \leqq k)$ が LIST B のどの位置にあるかを, B を前から順に調べる.
- 各 b_j $(1 \leqq j \leqq n)$ が a_1 から a_k の中の数であるか否かを, A を前から順に調べる.

なぜなら, このような操作には $O(kn)$ の計算量が必要となり, 例えば, $k = n/2$ のときにはこの計算量は $O(n^2)$ である.

そこで, LIST A の各要素が 1 から n の整数であることを利用して, 数 $1 \leqq i \leqq n$ が a_1 から a_k の中の数であるか否かを示す配列 $P[1..n]$ を用意しよう. すなわち, 各 $P[i]$ $(1 \leqq i \leqq n)$ は, 数 i が, A において前から k 番目以内にあれば 1, そうでなければ 0 とする. 明らかに, このような配列 $P[1..n]$ は, 全 $P[i]$ $(1 \leqq i \leqq n)$ を 0 とした後, 各数 a_i $(1 \leqq i \leqq k)$ に対して, $P[a_i] := 1$ としていけばよいから, $O(n+k)$ の計算量でできる.

このような配列 $P[1..n]$ ができれば, LIST B を前から順に調べながら, LIST C を作ることができる. すなわち, 各 b_j $(1 \leqq j \leqq n)$ に対して, $P[b_j] = 1$ であれば, b_j を LIST C に *Insert* し, さもなくば, a_{k+1} から a_n の数を *Insert* していけばよい. この操作は, 次のように書ける. なお, 下図は図 2.10 の具体例に対する配列 $P[1..n]$ の値である.



```
void  Crossover(  n, k, A, B, C  )
        入力 :  n  :     整数
                k  :     整数 (1≦k≦n)
                A  :     LIST (値呼び)
                B  :     LIST (値呼び)
        出力 :  C  :     LIST (名前呼び)
int  P[1..n] ;                  /*  0-1 配列. 局所的変数  */
int  i, j ;                     /*  整数. 局所的変数  */
position  p, q ;                /*  LIST の要素を調べるための局所的変数  */
{
        for ( 各 1 ≦ i ≦ n )    P[i]  :=  0  ;
        p  :=  First( A )  ;
        i  :=  1  ;
        while (  i ≦ k  ) {
                P[ Retrieve( p, A ) ]  :=  1  ;
                p  :=  Next( p, A )  ;
                i  :=  i + 1 ;
        }                                   /*  上記の操作で  P[1..n]  ができた  */
                                            /*  p  は  a_{k+1}  の  position  を示している  */
        q  :=  Makenull( C )  ;             /*  C  を空リストにする  */
        q  :=  First( B )  ;                /*  以下で, B  を前から順に調べる  */
        while (  q  ≠  Tail( B )  ) {
                j  :=  Retrieve( q, B ) ;
                if (  P[ j ] = 0  ) {
                        Insert( Retrieve( p, A ), Tail( C ), C ) ;
                        p  :=  Next( p, A )  ;
                }
                else        Insert( j, Tail( C ), C ) ;
                q  :=  Next( q, B )  ;
        }
} /*  Crossover  */
```

上記の手続き *Crossover* の計算量が $O(n)$ であることは明らかであろう. 第 1 章を読み終えたのであれば, それが確認できなければならない.