CS29003 Algorithms Laboratory

Assignment No: 3

Last date of submission: 10–August–2017

There are n identical items (like balls) in a bag. A game is played between two players: P (you) and C (the computer). Moves alternate between P and C. Player P makes the first move. In each move, a player removes some number of items from the bag. The number of items to be taken out in each move has to be one of $p_0, p_1, p_2, \ldots, p_{k-1}$, where k and the p_i values are known beforehand to both the players. Assume, for simplicity, that $1 \leqslant p_0 < p_1 < p_2 < \cdots < p_{k-1}$. The player who fails to make the next move loses. This happens when the number of items remaining in the bag becomes less than p_0.

Let i be the remaining number of balls in the bag. By T_i, we denote an optimal move at this point for the player who is going to make the next move. If a suitable move $j \in \{0,1,2,\ldots,k-1\}$ forces the opponent to lose, we call i a winning position of the player, and set $T_i = j$. If multiple values of j lets the player win, we choose the largest of these j values as T_i. On the contrary, if all allowed moves fail to force the opponent to a losing position, i is a losing position for the player to move next, and we set $T_i = -1$.

Game 1

Here, the pickup quantities p_i are assumed to be arbitrary (but sorted as mentioned above). Build a table $T[0 \ldots n]$ to store the T_i values as defined above. We have $T_i = -1$ for $i = 0,1,2,\ldots,p_0 - 1$. For $i \geqslant p_0$, T_i can be computed as follows. Consider a move $j \in \{0,1,2,\ldots,k-1\}$. This move is legitimate if and only if $i \geqslant p_j$. If so, making this move will leave $i - p_j$ items in the bag. If $T_{i-p_j} = -1$, then this is a losing position for the opponent. Otherwise, the opponent has an optimal move. Check for all legitimate values of j. Write a function T = *dptable*(n,k,p) to prepare and return the table T using the above dynamic-programming algorithm. Notice that your function requires $O(nk)$ time and $O(n)$ space.

Write a function *playgame1*() to play Game 1 with the computer. See *How to Operate the Black Box*.

Game 2

This game assumes that $p_j = p_0 + j$ for all $j = 0,1,2,\ldots,k-1$, that is, the allowed pickup choices are the consecutive integers from *first* $= p_0$ to *last* $= p_0 + k - 1$. In this case, you do not need the $O(nk)$-time and $O(n)$-space preprocessing for building the table T as in Game 1. Given any $i \in [0,n]$, you can calculate T_i in $O(1)$ time and using $O(1)$ space. Figure out how.

Write a function *playgame2*() to play Game 2 with the computer. This is detailed in the next section.

How to Operate the Black Box

The moves of the computer C are presented to you as a compiled black box. Download the appropriate file depending on your compiler (gcc/g++).

- Include the following lines at the beginning of your program. These are the functions defined in the black box.

```
extern int *registerme ( );
extern int makemove1 ( int );
extern int makemove2 ( int );
```

- Make a call to *registerme* to set up the parameters of the games.

```
int *A, n, k, *p;

A = registerme();
n = A[0];
k = A[1];
p = A + 2;
```

After this call returns, $A[0]$ stores n, $A[1]$ stores k, and $A[2], A[3], A[4], \ldots, A[k+1]$ store $p_0, p_1, p_2, \ldots, p_{k-1}$, respectively.

- Call *dptable*() to build the table T.
- Call *playgame1*() to play Game 1 with the computer C. Your moves are to be guided by T and the current number i of items left in the bag. So long as $i > 0$, look at T_i. If $T_i = j \geqslant 0$, call *makemove1*(j). Notice that you pass an index $j \in \{0, 1, 2, \ldots, k-1\}$ (not a p_j value). If $T_i = -1$, make a random move—you cannot win from this position. Your move lets the computer C determine its next move. After the two moves, the number of items left in the bag is returned. The function prints the moves of P and C (the values, not indices), so you do not have to print those again.
- Call *playgame2*() to play Game 2 with C. The allowed pickup choices are all the integers from *first* to *last*. Now, you are not allowed to build any table T. You can make $O(1)$ preprocessing (like the computation of *first* and *last*). Call *makemove2*(j) for supplying your next move that would be followed by a move to be made by C. As in Game 1, you pass an index j between 0 and $k-1$ (both inclusive), not $p_j = p_0 + j$, to *makemove2*. The return value is the number of items left in the bag after the moves of P and C. This function too prints the moves (values) of both P and C.
- Link the black-box code during compilation.

```
gcc/g++ -Wall myprog.c/myprog.cpp blackbox3.o
```

Sample output

```
*** Registration done.

*** Play Game 1
    n = 1764
    k = 12
    The choices are: 15 19 20 22 23 24 26 27 28 33 38 41

P-28 C-33 P-23 C-23 P-33 C-19 P-33 C-28 P-28 C-22 P-38 C-26 P-28 C-28 P-28 C-27
P-28 C-24 P-33 C-28 P-28 C-26 P-28 C-28 P-28 C-15 P-41 C-28 P-28 C-19 P-41 C-19
P-33 C-23 P-33 C-24 P-33 C-22 P-33 C-19 P-41 C-22 P-33 C-20 P-33 C-23 P-33 C-38
P-22 C-28 P-28 C-20 P-33 C-28 P-28 C-41 P-15 C-23 P-33 C-20 P-38 C-24 P-33

*** Congratulations. You have won.

*** Play Game 2
    n = 1764
    k = 12
    The choices are: 15 16 17 18 19 20 21 22 23 24 25 26

P-20 ...
...
...
...
...
...

*** Oops. You have lost.
    This is not your fault anyway. Better luck next time.
```

Submit a single C/C++ source file. Do not use global/static variables.