Problem C: DuLL

Source file: `dull.{c, cpp, java}`
Input file: `dull.in`

In Windows, a DLL (or dynamic link library) is a file that contains a collection of pre-compiled functions that can be loaded into a program at runtime. The two primary benefits of DLLs are (1) only one copy of a DLL is needed in memory, regardless of how many different programs are using it at the same time, and (2) since they are separate from programs, DLLs can be upgraded independently, without having to recompile the programs that use them. (DLLs have their problems, too, but we'll ignore those for now.) Your job is to calculate the maximum memory usage when running a series of programs together with the DLLs they need.

The DLLs in our system are not very exciting. These dull DLLs (or DuLLs) each require a fixed amount of memory which never changes as long as the DuLL is in memory. Similarly, each program has its own fixed memory requirements which never change as long as the program is executing. Each program also requires certain DuLLs to be in memory the entire time the program is executing. Therefore, the only time the amount of memory required changes is when a new program is executed, or a currently running program exits. When a new program begins execution, all DuLLs required by that program that must be loaded into memory if they are not there already. When a currently running program exits, all DuLLs that are no longer needed by any currently running programs are removed from memory.

Remember, there will never be more than one copy of a specific DuLL in memory at any given time. However, it is possible for multiple instances of the same program to be running at the same time. In this case each instance of the program would require its own memory; however, the instances still share DuLLs in the same way two unrelated programs would.

Input: The input consists of at least one data set, followed by a line containing only 0.

The first line of a data set contains three space separated integers *N P S*, where *N* is the number of DuLLs available, $1 \le N \le 20$, *P* is the number of programs which can be executed, $1 \le P \le 9$, and *S* is the number of state transitions recorded, $1 \le S \le 32$.

The next line contains exactly *N* space separated integers representing the sizes in bytes of each of the DuLLs, $1 \le size \le 1{,}000$. Each DuLL is implicitly labeled with a letter: 'A', 'B', 'C', …, possibly extending to 'T'. Therefore the first integer is the size of 'A', the second integer is the size of 'B', and so on.

The next *P* lines contain information about each of the programs, one program per line. Each line contains a single integer representing the size of the program in bytes, $1 \le size \le 1{,}000$, followed by 1 to *N* characters representing the DuLLs required by that program. There will be a single space between the size of the program and the DuLL labels, but no spaces between the labels themselves. The order of the labels is insignificant and therefore undefined, but they will all be valid DuLL labels, and no label will occur more than once. Each program is implicitly labeled with an integer: 1, 2, 3, … possibly extending to 9.

The final line of the data set will contain *S* space separated integers. Each integer will either be a positive number *q*, $1 \le q \le P$, indicating that a new execution of program *q* has begun, or else it will be a negative number $-q$, $1 \le q \le P$, indicating that a single execution of program *q* has completed. The transitions are given in the order they occurred. Each is a valid program number; if it is a negative number $-q$ then there will always be at least one instance of program *q* running.

Output: There is one line of output for each data set, containing only the maximum amount of memory required throughout the execution of the data set.

Example input:	**Example output:**
2 2 3 500 600 100 A 200 B 2 1 2 5 4 8 100 400 200 500 300 250 AC 360 ACE 120 AB 40 DE 2 3 4 -3 1 2 -2 1 0	1600 2110