CSE 30
Programming Abstractions: Python
Lab Assignment 2

In this assignment you will re-create your Subset.py program from pa1, this time using a different method for representing subsets. You will again use the list type to represent k-element subsets of $\{1, 2, 3, \ldots, n\}$, but this time with a different encoding. As an example, let $n = 8$ and $k = 4$, and consider the 4-element subset $\{2, 4, 5, 7\}$ of the 8-element set $\{1, 2, 3, 4, 5, 6, 7, 8\}$. In pa1 we represented this subset by the following *bit-list* of length 9.

$$\begin{array}{rl} \text{indices:} & 0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8 \\ B = & [\,*,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0] \end{array}$$

As before, $*$ stands for whatever is placed in index 0 (which is not used). In this assignment, we will represent the same subset by the *element-list* $L = [2, 4, 5, 7]$.

Each encoding has its own advantages. The bit-list implicitly carries the length of the set $\{1, 2, 3, \ldots, n\}$, i.e., $n = \text{len}(B) - 1$. As a result, it was not necessary to pass n as an argument to the recursive function printSubsets(). It will be necessary to do so in this project. The bit-list representation also establishes a theoretical link between the problem of finding all k-subsets of an n-set, and finding all bit-strings of length n whose number of 1s is exactly k. As one learns in Discrete Mathematics, the two problems are actually one and the same.

The element-list is a more direct and intuitive structure to represent a set, and is arguably more Pythonic. One might ask whether the *set* type in Python would be even more natural. Using *set* instead of *list* is appropriate only if one does not care about the order in which elements are printed. It is not possible to dictate or predict this order when using *set*, since it is an implicitly unordered container object. This is because of the way in which the *set* type is represented internally. We do care about this order (since we want everyone's output files to be identical), and hence we use *list*.

The two required functions to_string(L) and printSubsets(L, n, k, i) must be altered accordingly. Function to_string() is much easier this time around, since all you have to do is take the string representation of a list, replace the brackets '[' and ']' by set braces '{' and '}', and return the resulting string. The other function printSubsets(), is also easy to alter. Instead of flipping the i^{th} bit from 0 to 1, just call append() to add i to L. Instead of flipping the i^{th} bit from 1 back to 0, call pop() to delete the last element from L. (See the Python documentation if you are unfamiliar with either of these list operations.)

Use the following stubs to create these two functions, including the given doc strings.

```
def to_string(L):
    """
    Returns the string representation of the subset of {1, 2, ..., n}
    encoded by the list L.
    """



# end
```

```
def printSubsets(L, n, k, i):
    """
    Prints all subsets of {1, 2, 3, ..., n} of the form (S union T), where S
    is the subset of {1, 2, 3, ..., (i-1)} encoded by the list L, and T is a
    k-element subset of {i, (i+1), ..., n}.
    """


# end
```

Program output for this project will emulate exactly that of pa1, except in the case where the user provides wrong or inconsistent command line inputs. In pa1, the specified response was to simply print nothing and quit. Output for this project will be as indicated in the session quoted below. All error messages will be written to stderr.

```
$ python3 Subset.py foo 3
cannot parse 'foo' as int
Usage: python3 Subset.py n k (where 0<=k<=n)
$ python3 Subset.py 6 bar
cannot parse 'bar' as int
Usage: python3 Subset.py n k (where 0<=k<=n)
$ python3 Subset.py
Usage: python3 Subset.py n k (where 0<=k<=n)
$ python3 Subset.py 6 3 8
Usage: python3 Subset.py n k (where 0<=k<=n)
$ python3 Subset.py 3 6
Usage: python3 Subset.py n k (where 0<=k<=n)
$ python3 Subset.py 6 3
{1, 2, 3}
{1, 2, 4}
{1, 2, 5}
{1, 2, 6}
{1, 3, 4}
{1, 3, 5}
{1, 3, 6}
{1, 4, 5}
{1, 4, 6}
{1, 5, 6}
{2, 3, 4}
{2, 3, 5}
{2, 3, 6}
{2, 4, 5}
{2, 4, 6}
{2, 5, 6}
{3, 4, 5}
{3, 4, 6}
{3, 5, 6}
{4, 5, 6}
$ python3 Subset.py 6 0
{ }
$
```

As you can see from above, if anything other than 3 command line arguments are provided (Subset.py and two other strings), then a usage message is printed. If the two strings after Subset.py cannot be parsed as

ints, then a message resembling "`cannot parse 'foo' as int`" will be printed, where '`foo`' referrs to the first of the two strings that cannot be so parsed. If the two strings after Subset.py can be parsed as ints (n followed by k), but do not satisfy $0 \leq k \leq n$, then the usage message will be printed. It is recommended that you write a function called usage() that prints a message to the stderr output stream, then ends the program. Of course the file Subset.py should follow the template provided at

https://classes.soe.ucsc.edu/cse030/Spring21/Examples/GeneralTemplate.py

Given that you've already solved this problem in pa1, lab2 should be very easy. Nevertheless, it will take some time to complete, so please do not wait until the day before the deadline to begin. Start early, formulate well composed questions, then post them on Piazza and ask them in office hours.