CS 59000-NMC, 6 September

Please answer the following questions. You may not use any outside references or technology. Justify and explain all answers. This quiz is for my own evaluation, so that I can provide better instruction in the course.

Question

Let $\boldsymbol{A}$ be a binary matrix. Suppose this matrix is composed of mostly ones and that the zeros are stored with a compressed-sparse row data structure. Write down an efficient algorithm to compute $\mathbf{y} = \boldsymbol{A}\mathbf{x}$ give the compressed sparse row data structure for $\boldsymbol{A}$'s zeros in the arrays `pointer` and `columns`.

Solution

The key insight is that the zero pattern tells us what to *exclude* from the matrix-vector product, rather than what to include. Consider that if $\boldsymbol{A}$ was composed entirely of ones, then:

$$[\boldsymbol{A}\mathbf{x}]_i = \sum x_i$$

Consequently, if we set $\alpha = \sum x_i$, then

$$\boldsymbol{A}\mathbf{x} = \alpha\mathbf{e}$$

where $\mathbf{e}$ is the vector of all ones.

Once we have this property, let $\boldsymbol{O}$ be the matrix of all ones. Now consider $\boldsymbol{A}$ from the problem

$$\boldsymbol{A} = \boldsymbol{O} - \boldsymbol{B}$$

where $\boldsymbol{B}$ is a *sparse matrix*with the *zero pattern* from $\boldsymbol{A}$ and each entry is a one.

Consequently, we can just compute

$$\boldsymbol{A}\mathbf{x} = \alpha\mathbf{e} - \boldsymbol{B}\mathbf{x}$$

where $\boldsymbol{B}\mathbf{x}$ is just a standard matrix-vector product.

```
function sparse_zero_matvec(n,pointers,columns,x)
""" Compute a mat-vec with a mostly one matrix, with a sparse zero pattern.

This function will multiply a binary matrix $A$, which is all
ones except for a sparse pattern of zeros, by a vector $x$.

Here the arrays are zero indexed.

@param n the dimension of the matrix
@param pointers the array of pointers for a CSR pattern of zeros in the matrix
@param columns the array of columns for the CSR pattern of zeros
@param an array with the values of x
@return an array such
"""

alpha = 0
```

```
for xi in x: # assumes x implements an interable interface
  alpha += xi

y = [alpha for _ in xrange(n)] # initialize y
for i in xrange(n):
  change = 0
  for nzi in xrange(pointers[i],pointers[i+1]):
    col = columns[nzi]
    change += x[nzi]

  y[i] -= change # adjust the value

return y
```