# problem2

February 13, 2024

[1]:
```python
import numpy as np
import scipy
import os
from scipy.sparse import diags, dia_matrix
```

## 0.1 a. Matrix multiplication

[2]:
```python
A =np.arange(1,21)
A = A.reshape((5,4))
# A = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16],[17,18,19,20]])
print(A.shape, A)

B = np.arange(1,13)
B = B.reshape((3,4)).T
# B = np.array([[1,5,9],[2,6,10],[3,7,11],[4,8,12]])
print(B.shape, B)
```

```
(5, 4) [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]
(4, 3) [[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]
```

[3]:
```python
def mat_multiply(A,B):
    return np.dot(A,B)

print('Matrix multiplication',mat_multiply(A,B))
```

```
Matrix multiplication [[ 30  70 110]
 [ 70 174 278]
 [110 278 446]
 [150 382 614]
 [190 486 782]]
```

## 0.2 b. Sparse matrix-vector multiplication

[4]:
```
def generate_sparse_matrix(n):
    print("N is",n)
    vector_x = np.random.randint(5,size=(n)) # generates number in an array␣
 ↪from 0 to 5
    diagonals = [np.ones((n-1)), np.ones((n-1)) * -1, np.zeros((n-1))]
    sparse_mat_A = diags(diagonals, offsets=[0, 1,2], shape=(n-1,n)).toarray()
    return sparse_mat_A @ vector_x.T


print(generate_sparse_matrix(7))
```

```
N is 7
[-2.  2. -4.  2. -1.  0.]
```

The worst case time complexity (in Big O notation) of multiplying a matrix A of dimension Rn×n with a dense vector v  Rn = O(n^2) = Mulitplication & addition operation

What if matrix A is sparse, denote the number of non-zero elements by nnz(A) = O(nnz(A) + n)

## 0.3 c. Manipulating text using python, data structures

[5]:
```
file_name = 'data_example.txt'

file_content_arr = []




def unique_words_with_count(file_name):
    current_directory = os.getcwd() # cost = 1, time = 1
    file_path = os.path.join(current_directory, file_name)
    dict_unique_words_with_count = {}

    try:
        with open(file_path, 'r') as file:
        # Read the entire contents of the file
            file_contents = file.read()
            file_content_arr = file_contents.lower().split(' ')
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except Exception as e:
        print(f"An error occurred: {e}")

    # print(file_content_arr)
    for i in file_content_arr:
        if i in dict_unique_words_with_count:
            dict_unique_words_with_count[i] = dict_unique_words_with_count[i] +␣
 ↪1
```

```
        else:
            dict_unique_words_with_count[i] = 1

    return dict_unique_words_with_count


print('Count of unique words', unique_words_with_count(file_name))
```

```
Count of unique words {'this': 5, 'film': 4, 'took': 1, 'me': 1, 'by': 1,
'surprise.': 1, 'i': 5, 'make': 2, 'it': 5, 'a': 7, 'habit': 1, 'of': 6,
'finding': 1, 'out': 1, 'as': 3, 'little': 1, 'possible': 1, 'about': 2,
'films': 1, 'before': 1, 'attending': 1, 'because': 1, 'trailers': 1, 'and': 10,
'reviews': 1, 'provide': 1, 'spoiler': 1, 'after': 1, 'spoiler.': 1, 'all': 1,
'knew': 2, 'upon': 1, 'entering': 1, 'the': 8, 'theater': 1, 'is': 4, 'that': 4,
'was': 5, 'documentary': 2, 'long': 2, 'married': 1, 'couple.': 1, 'filmmaker':
2, 'doug': 2, 'block': 1, 'decided': 1, 'to': 5, 'record': 1, 'his': 2,
'parents': 1, '"for': 1, 'posterity"': 1, 'at': 1, 'beginning': 2, 'we': 1,
'are': 1, 'treated': 1, 'requisite': 1, 'interviews': 1, 'with': 3, 'parents,':
1, 'outspoken': 1, 'mother': 1, 'mina,': 1, 'less': 1, 'than': 1, 'forthcoming':
1, 'dad,': 1, 'mike.': 1, 'immediately': 1, 'found': 1, 'couple': 1,
'interesting': 1, 'had': 1, 'no': 1, 'idea': 1, 'where': 2, '(mike': 1, '&': 1,
"mina's": 1, 'son': 1, 'doug)': 1, 'going': 2, 'take': 1, 'us.': 1, 'matter': 1,
'fact,': 1, 'doubt': 1, 'himself': 1, 'he': 1, 'this!': 1, 'life': 2, 'takes':
1, 'unexpected': 1, 'twists': 1, 'turns': 1, 'beautifully': 1, 'expressive': 1,
'follows': 1, 'journey.': 1, 'difficult': 1, 'verbalize': 1, 'just': 1, 'how':
1, 'moved': 1, 'story': 1, 'unique': 1, 'way': 1, 'in': 1, 'which': 1, 'told.':
1, 'absolutely': 1, 'riveting': 1, 'from': 1, 'end': 1, 'really': 1, 'must-see':
1, 'even': 2, 'if': 1, 'you': 3, "aren't": 1, 'fan': 1, 'genre.': 1, 'will': 1,
'think': 1, 'your': 1, 'own': 1, 'might': 1, 'evoke': 1, 'memories': 1,
'thought': 1, 'were': 1, 'forgotten.': 1, '"51': 1, 'birch': 1, 'street"': 1,
'one': 1, 'those': 1, 'rare': 1, 'filmgoing': 1, 'experiences': 1, 'makes': 1,
'deep': 1, 'impression': 1, 'never': 1, 'leaves': 1, 'you.': 1}
```

1. I have used array and dictionary to solve this problem. As, time complexity (is not time required to execute the function), the measure of runtime of an algorithm (number of basic operations involved, time it takes to execute the operation for a given input. suppose your function takes input n, and that function is simply print operations, no matter what the value of n is, it's always one operation & is fixed. But in case of loop with n, as the number of n increases, operations inside loop also increase.) will change as the input changes.

2. Time complexity of above function is O(n + m), where n = size of array and m = how big the file size is.
3. Space complexity is O(nm)

## 0.4 d. Recursion with bookkeeping in python

[6]:
```python
# [0,1,1,2,3,5,8,13,....]
def fibonacci(n):
    if n < 0:
        return 'Please, give positive number only'
    elif n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(38))
```

```
39088169
```

# This will have n^2 complexity for time & space. O(n^2).

[7]:
```python
def fibonacci_reduced_complexity(n, dict ={}):
    if n in dict:
        return dict[n]
    if n < 0:
        return 'Please, give positive number only'
    elif n == 0:
        # dict[n] = 0
        return 0
    elif n == 1 or n == 2:
        dict[n] = 1
        return 1
    else:
        dict[n] = fibonacci_reduced_complexity(n-1, dict) +␣
 ↪fibonacci_reduced_complexity(n-2, dict)
    return dict[n]

print(fibonacci_reduced_complexity(10))
```

```
55
```

## 1 This will have 2n complexity for time & space. O(2n) = O(n).

```
[ ]:
```