Course Authored By:

K.Srinivasan
Personal website(research): https://sites.google.com/site/kuja27/
NeuronRain GitHub and SourceForge Documentation: http://neuronrain-documentation.readthedocs.io/

##
##

This is a non-linearly organized, continually updated set of course notes on miscellaneous topics in Graduate/Doctoral level
Computer Science and Machine Learning and supplements NeuronRain AsFer Design Notes in:

NeuronRain Enterprise Version Design Documents:

AsFer Machine Learning - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
NeuronRain Research Version Design Documents:

AsFer Machine Learning - https://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/AstroInferDesign.txt
----
----
9 March 2017

----
759. (THEORY and FEATURE) RNN and GRU - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
Recurrent Neural Network - Long Term Short Term Memory:

Traditional neural networks have a threshold function which outputs 1 or 0 based on a threshold. But they don't preserve state information
over points in time. For example, if there is a requirement that next state depends on present state and an input, usual neural network
cannot satisfy it. Recurrent Neural Networks fill this void through ability to feedback while traditional neural network is feedforward.
LongTerm-ShortTerm memory Recurrent Neural Networks are defined with schematic below:

```
      Forget gate --------------> * <------------- + ---------------> *
-------------->
                            |---------------->|            /|\
                                         /|\          |
                                          |                 |
```

```
      Cell gate ----------------> * --------------->|             |
                      /|\                               |
                       |                          |
      Input gate -----------------|                     |
                                                  |
      Output gate ------------------------------------------------------
```

It has four gates: Forget gate, Cell gate, Input gate and Output gate and has a recurrence/feedback as shown in first line between Forget, Cell and Input gates. * is per-element product and + is per-element sum of vectors.

Recurrent Neural Network - Gated Recurrent Unit:

A slight variation of RNN LSTM diagram previously is RNN Gated Recurrent Unit (GRU). It lacks an output gate and merges the functionality of input gates into two gates - reset and update - as drawn in schematic below:

```
h(t-1)-------------------> * --------------> Ct ----------------> *
-------------------> h(t)
      |              /\                    /\                   /\
      |              |              |              |
      |     reset    * <----------------x------------------->* update
      |             /|\                                /|\
      |              |                                  |
      V------------------|---------------------------------------|
```

with equations:
      ut = sigmoid(Vu*xt + Wu*h(t-1) + bu)
      rt = sigmoid(Vr*xt + Wr*h(t-1) + br)
      Ct = tanh(Vc*xt + Wc*(h(t-1) * rt))
      ht = (1-ut)*Ct + ut*h(t-1)
where ut = update gate,
      rt = reset gate,
      Ct = cell gate
      ht = state at time t
      Vu,Vr,Vc,Wu,Wr,Wc are weight vectors.

Both above have been implemented in:
https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/DeepLearning_LSTMRecurrentNeuralNetwork.py
https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/DeepLearning_GRURecurrentNeuralNetwork.py

----
Mathematical Puzzles of Sam Loyd (selected and edited by Martin Gardner) - Puzzle 18 - What is the most economical form of a tank designed to hold 1000 cubic feet? - 26 January 2018

A Plumber wanted to estimate the lowest possible cost of a copper tank to hold 1000 cubic feet. Copper costs $1 per square foot. Problem is to determine most economical dimensions of the rectangular tank of capacity 1000 cubic feet. Trivial solution of 10 feet * 10 feet * 10 feet = 1000 cubic feet tank costs $500 of copper surface (100 in bottom + 4*100 in sides). Another solution which costs less than $500 for copper surfacing is expected.

Plumber's problem has applications in packing/knapsack algorithms which minimize the cost of packing items in least volume. This is also equivalent to storing set of 1000 elements in a 3 dimensional array (cube) subject to minimizing the objective function xy + 2z(x+y) and constraint xyz =

1000 for array indices x,y,z.

This problem can be cast into a (Multi)Linear Program formulation - sums of products
Let l,b,h be the length,breadth and height of the tank.

The objective cost function for copper plating the surface to be minimized is:

    l*b + 2*h*l + 2*h*b = cost

subject to constraint:

    l*b*h = 1000

Objective function can be rewritten as:

    1000/h + 2h(l + b) = cost

Solving multilinear programs is non-trivial requiring reformulation and linearization creating a new LP(RL algorithms).

----
1. (l+b) is a constant:

If (l+b) = sum of sides of rectangles is fixed to be a constant elementary calculus can solve this:
first derivative of cost function is equated to zero:

    d(cost)/dh = -1000/h^2 + 2(l+b) = 0
    2(l+b) = 1000/h^2
    h^2 = 1000/[2(l+b)]
    h = 22.36068/sqrt(l+b)

[=> lbh = lb*22.36068/sqrt(l+b) = 1000, lb/sqrt(l+b) = 1000/22.36068 = 44.72]

Second derivative of cost function is positive, implying a local minima. Thus if height of the tank h is inversely related to square root of sum of length and breadth of bottom rectangle as h = 22.36/sqrt(l+b), cost of copper plating is minimized. If bottom is a square, l = b and
h = 22.36/(1.414*sqrt(l)) = 15.811/sqrt(l).

=> lbh = ll*15.811/sqrt(l) = 15.811*l*sqrt(l) = 1000
=> l^1.5 = 1000/15.811 = 63.247
=> 1.5 log l = log 63.247
=> l = 15.874
=> h = 15.811/sqrt(15.874) = 3.968

Dimensions of the tank of least copper plating cost by local mimima = 15.874 * 15.874 * 3.968
Cost = 500 which is not less than 10 * 10 * 10.

----
2. Bottom is a square and is a function of h:

Bottom is square : l=b=kh
Cost function: kh*kh + 2h*kh + 2h*kh = k^2*h^2 + 4k*h^2
Cost = (k^2 + 4k) * h^2
Volume = lbh = kh*kh*h = k^2*h^3 = 1000
=> h^3 = 1000/(k^2)

Cost = (k^2 + 4k) * (1000)^0.66/k^1.33) = (k^2 + 4k)/k^1.33 * 95.49926
Cost = (k^0.66 + 4k^(-0.33)) * 95.49926
d(Cost)/dk = 0.66*k^(-0.33) - 1.33*k^(-1.33) = 0
=> minima at k = 2.

Dimensions are 2h*2h*h and
Cost is 12h^2 = 476.21 for h=6.2966

Book Solution:

If bottom is a square of side = 2h for height h, economical cost is attained.
=> 2h*2h*h = 4h^3 = 1000
=> h = 6.2996
Cost = 4h^2 + 2h(4h) = 12h^2 = 12*(6.2996)^2 = 476.21

Reference:

Mathematical Puzzles of SAM LOYD - Selected and Edited by MARTIN GARDNER

Catalan Numbers - How many squares and lattice paths are in a grid e.g 4 * 4 - Puzzle 142 - Puzzles To Puzzle You -
Shakuntala Devi - 26 January 2018

In a grid of 4 * 4, number of possible squares are obtained by moving a sliding 2 dimensional square window left-right, top-down as in
algorithm below:

```
      for square sliding window size w*w
      {
            slide window top-down
            {
                  slide window left-right
                  number_of_squares += 1
            }
            w = w+1
      }
```

Sliding window square increases in size from 1*1 to 4*4.

```
Number of squares of size 1*1 = 16 = 4*4
Number of squares of size 2*2 = 9  = 3*3
Number of squares of size 3*3 = 4  = 2*2
Number of squares of size 4*4 = 1  = 1*1
                  -----------
                  Total = 30
                  -----------
```

Generic series is = 1 + 2^2 + 3^2 + ... + n^2

Number of lattice paths in the grid which lead from bottom left to top right of the grid is the Catalan Number = 1/(n+1) * 2nCn which is
same as number of Dyck words of the form XXYXX,... number of possible rooted binary trees of node size n and number of possible balanced
parenthesizations of an infix arithmetic expression. Catalan numbers are ubiquitous in combinatorial algorithms involving recursions and
self-similarity. Catalan number is also the number of random walks in the grid graph.

Most celebrated result involving Catalan numbers is the Bertrand Ballot Theorem: In an election of two candidates A and B, if A receives
p votes and B receives q votes, p > q, what is the probability A is strictly ahead of B throughout counting? This problem reduces to counting
dyck paths in the grid (time versus votes). Ballot Theorem applies to Streaming binary datasets and gives the probability of 1s dominating the stream if 1s outnumber 0s and vice versa.

Reference:

Puzzles To Puzzle You - Shakuntala Devi

Binary Search of a sorted array containing gaps - 6 February 2018

Q:Usual binary searches are made on arrays of contiguous sorted elements. How can binary search be made to work if the array has gaps/holes and yet the contents are in sorted order? E.g Array 12,33,44,-,-,56,-,66,-,-,-,88,99,-,-,123 is sorted ascending but has gaps.

A1: One possible solution is to fill the gaps with placeholder numbers or replicate the integers in hole boundaries to fill the gap. Previous example array is filled as 12,33,44,44,44,56,56,66,66,66,66,88,99,99,99,123.
A2: Other possibility is to fill the gaps with an arithmetic progression on difference of the integers on the boundaries. Previous example array is filled as 12,33,44,48,52,56,61,66,...
A3: Filling is necessary because to choose the subtree of search, an integer is necessary. Non-filling solution has to branch off to a subtree based on some other meta data on the gaps. Alternative: when a "-" is found, scan the array in one direction till an integer appears and branch off. This is similar to open addressing in hash tables. But this linear scan increases the amortized binary search cost from O(logN) to something higher. But filling the gaps by placeholders or arithmetic progressions is also linear and makes binary search superlogarithmic.

This problem has applications in splitting a single huge sorted array into multiple smaller arrays, distributed geographically but logically mapped to virtual memory pages in single address space, and searching them.

----
842. (THEORY and FEATURE) Computational Geometric Factorization and Planar Point Location, Wavelet Trees, Sublinear Multiple String Concatenation - related to all sections on String analytics and Planar Factor Point Location by Wavelet Trees - 8 February 2018, 18 July 2020

Q: Concatenation of multiple strings is trivially doable in O(N). Can N strings be concatenated in sublinear time?

A: Subject to certain assumptions following algorithm does sublinear multiple string concatenation:

Let the number of strings be N each of length l. Each string is fingerprinted/compressed to length logN by a standard algorithm e.g Rabin string fingerprint which computes a polynomial of degree l over Galois Field GF(2) and divides this by an irreducible polynomial of degree logN over GF(2) to create a fingerprint of logN-bit length.

Create a matrix of logN * N (transpose) which has logN rows and as many columns as number of strings. Entries of this matrix are the bits of string fingerprint hashes. This transformation converts N strings of length l to logN strings of length N. Hashes are stored as Rope strings to facilitate logN time pairwise computation. These logN strings are concatenated as a binary tree bottom-up and each pairwise concatenation is O(logN). Following series sums up the runtime:

```
      logN*(logN/2 + logN/4 + logN/8 + logN/16 + ...)
      = logN*logN*2
      = 2(logN)^2
```

This indirectly concatenates N strings in O(logN*logN) time. But it messes up with original string. This requires slight modification to pairwise Rope string concatenation routine. Before concatenation hash has to be reverse engineered (Rabin fingerprint polynomials have to be stored) to unicode string and location in the resultant single concatenation has to be ingredient of this routine.

Fingerprinting is not a necessity. Without fingerprint, previous matrix is l * N (l strings of length N) and the concatenation tree has following runtime geometric recurrence:

```
      logN*(l/2 + l/4 + l/8 + l/16 + ...)
```

```
      [because each internal node of concatenation tree needs O(logN) time for 2
Rope string concatenation]
      = logN*2l
      = 2*l*logN

This runtime is sublinear if:
      2*l*logN < N
      length of each string = l < N/(2*logN)

Example:
-------
Set of 5 strings of length 4:
      aaaa
      bbbb
      cccc
      dddd
      eeee
is transformed to transpose matrix of 4 strings of length 5:
      abcde
      abcde
      abcde
      abcde
Rope representation of these 4 strings are 4 binary trees. Rope concatenation
routine has to be changed to write the literals of new string in correct
locations in the final concatenation e.g abcde + abcde = abcdeabcde has to be
surgically mapped to aa--bb--cc--dd--ee--. Rope insertion is also O(logN). This
might require storing index information for each literal in original set of
strings.

Following is an example for the changed Rope concatenation by storing indices of
matrix entries for abcde and abcde:
      a(1,1)b(2,1)c(3,1)d(4,1)e(5,1)
      a(1,2)b(2,2)c(3,2)d(4,2)e(5,2)
In final concatenation, new indices for previous literals are
(length_of_string*(i-1) + j). Rope concatenation is just O(1) for merging two
trees as subtrees of a new root. Only updating sum of left subtree leaf weights
is O(logN). Storing matrix index information multiplies the string length by 5
(length of "(i,j)") which is a constant multiple and string lengths remain O(N).

Final concatenated string is stored as matrix in sublinear time 2*l*logN:
      a(1,1)b(2,1)c(3,1)d(4,1)e(5,1)
      a(1,2)b(2,2)c(3,2)d(4,2)e(5,2)
      a(1,3)b(2,3)c(3,3)d(4,3)e(5,3)
      a(1,4)b(2,4)c(3,4)d(4,4)e(5,4)
For example, accessing 10th element in this concatenation is O(length_of_string)
because (i,j) have to be found iteratively for all values of l
(length_of_string):
      l*(i-1) + j = 10
      4*(i-1) + j = 10
      i = (10-j)/4 + 1 and j=1,2,3,4

Thus total time to access an element in concatenation = 2*l*logN + l which is
sublinear if:
      l*(2logN + 1) < N
      => l < N/(2logN + 1) which is a tighter upperbound assumption for length
of strings, than previous N/2logN.

If each string is compressed as burrows-wheeler transform, columns in
concatenation matrix are compressed strings and l is reduced by compression
ratio. If N strings in the concatenation can be represented as N Wavelet trees
in compressed format (runlength encoding etc.,), each column in previous
concatenation matrix is a wavelet tree and access()/select()/rank() of a column
are logarithmic time. Rectified hyperbolic arc in computational geometric
factorization could be a huge string stored in wavelet trees and factor points
```

have to be located by rank() and select() queries. Fragments of Rectified hyperbolic arc segments in the vicinity of approximate factors found by number theoretic ray queries could be concatenated efficiently in sublogarithmic time to lessen the length of the relevant rectified hyperbolic arc string stored in wavelet tree which has to be searched for factor points because individually searching each fragment would require as many wavelet trees as number of factors. Sublinear string concatenation is considerably useful for text analytics problems as well involving huge set of strings (e.g. Bioinformatics)

References:

842.1. Rope Strings - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9450&rep=rep1&type=pdf - [hans-j. boehm, russ atkinson and michael plass, Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304, U.S.A.]
842.2. Rabin-Karp String Fingerprinting by Random polynomials - [Michael O.Rabin] - http://www.xmailserver.org/rabin.pdf
842.3. Myriad Virtues of Wavelet Trees - Pruned Wavelet Tree of Compressed Strings - [Paolo Ferragina, Raffaele Giancarlo, Giovanni Manzini] - http://www.ittc.ku.edu/~jsv/Papers/FGM09.wavelettrees.pdf

----
-
How would you move Mount Fuji? - 11 February 2018

-
This problem has parallels in moving a huge block of solid which can only be accessed in LIFO. Comparing
 with moving block of memory which can be randomly accessed, this problem is non-trivial. Moving mount
which is a 3D solid trivially involves cutting it into equal sized cubes and reconstructing the mount in
another location by moving the cubes. This is LIFO operation requiring an intermediate stack. Following
mountain is moved by an intermediate stack:

```
      1 2        6          2
      3 4  ---   4  ---   4
      5 6        2        6

               5  ---   1 2
               3      3 4
               1      5 6
```

Previous move is O(Volume_of_mount). Towers of Hanoi (Towers of Brahma in Kashi Vishwanath temple) problem is akin to this and requires exponential number of moves. For 64 disks of Towers of Brahma, this requires 2^64 - 1 moves which is legendary lifetime of universe (1 second per move translates to 585 billion years). Non-trivial requirement in this problem is no disk should be on smaller disk. Moving mount Fuji of height h sliced as horizontal disks instead of cubes is exactly Tower of Hanoi problem of time O(2^h-1).

Reference:

Towers of Brahma - https://en.wikipedia.org/wiki/Tower_of_Hanoi

----
----
843. (THEORY) Social networks, Bipartite and General Graph Maximum Matching, Permanent, Boolean majority,
Ramsey coloring - Number of Perfect (Mis)Matchings - Hat Puzzle - 17 February 2018, 18 July 2020 - related to 14, 801

There are N people in a congregation and they have to choose matching hat for each. But they endup choosing a non-matching hat at random. What is the probability of everyone choosing a non-matching hat?

This problem can be formulated as Bipartite Matching in Bipartite Graph - Set of vertices of people and Set of Hats forming the bipartisan. Each choice corresponds to an edge in this graph. Usual problem of perfect matching tries to find edges between these sets which create a bijection. Hat problem goes further beyond this and tries to match the index of the vertices too. For example:

```
p1 p2 p3
1  2  3
1  3  2
2  1  3
2  3  1
3  1  2
3  2  1
```

is the set of permutations of persons p1,p2,p3 choosing the numbered hats 1,2,3. Non-matching choices are:

```
p1 p2 p3
2  3  1
3  1  2
```

in which everyone has a mismatch.Counting the number of mismatches has the following algorithm:

```
for each person
{
  remove permutations which match the person's index from set of all
permutations
}
```

In previous example following are the iteratively curtailed set of string permutations:

```
person3:
1 3 2
2 3 1
3 1 2
3 2 1
person2:
1 3 2
2 3 1
3 1 2
person1:
2 3 1
3 1 2
```

An approximate recurrence for perfect mismatching (this is an alternative to Solution in reference):

[nPn - nPn/n] - Sigma_m=2_to_n[nPn/n - (n-m)P(n-m)]

for n=number_of_hats/persons, m=number of hats/persons not yet chosen. Intuition for this recurrence is obvious:

- Remove all strings ending with person index for pn.
- for all person indices m less than n, remove strings having m in index m minus set of all permuted strings ending with suffix (m, (m+1),...,(n)) already removed

Contrasting this with Mulmuley-Vazirani-Vazirani Theorem for number of perfect matchings by Isolation Lemma in randomized parallel polylog time, hat puzzle estimates Perfect Mismatches in Bipartite Graphs. Perfect matching in Bipartite graph is equal to Permanent of its incidence matrix. In Group Theoretic terms, previous number of perfect matchings is the number of permutations of cycle 6 in Symmetric Group S6 i.e each element in a permutation is mapped to a different element and all elements are moved.

Finding perfect mismatches has applications to majority voting in Social Networks - by replacing indices of hats by candidate indices. Each voter has to find a mismatching voter peer (who has voted differently). In previous hat puzzle, p1,p2 and p3 are voters who have voted to candidates 1,2,3 respectively and each of them need to find a mismatching candidate index (hat):

```
p1 p2 p3
2  3  1
3  1  2
```

An example of maximum (mis)match in a complete social network graph of 5 vertices (adjacency list) which are two colored (bipartisan) while earlier example is tripartisan. It is both a maximum match (number of edges having disjoint vertices is maximized) and a mismatch (because each pair of voter vertices in matching are complementarily 2-colored by candididate index 0-1):

```
v1 - v2,v3,v4,v5
v2 - v1,v3,v4,v5
v3 - v2,v1,v4,v5
v4 - v2,v3,v1,v5
v5 - v2,v3,v4,v1
```

One of the Maximum (mis)matchings is:

```
v1 - v3, v2 - v4
```

leaving v5 unmatched. v1,v2,v5 are colored red (voters of red) while v3,v4 are colored blue (voters of blue). Thus party red is the winner.

References:

843.1 Puzzle 113 and its Solution Recurrence (tends to 1/e for large n) - Mathematical Puzzles of SAM LOYD - Selected and Edited by MARTIN GARDNER
843.2 The Art Of Computer Programming - Combinatorial Algorithms - Volume 4a - [Don Knuth] - Section 7.2.1.2 - Generating All Permutations - Reverse Colex Order, Sims table for succinct representation of Symmetric Group elements.
843.3 Mulmuley-Vazirani-Vazirani Theorem and Perfect matchings - Theorem 5.5 and Theorem 5.6 (Isolation Lemma) - https://courses.cs.washington.edu/courses/cse521/16sp/521-lecture-5.pdf
843.4 Number of Perfect matchings in Bipartite graph = Permanent of Incidence matrix - Section 2 - https://lbgi.fr/~sereni/Lectures/GC_Spring09/gc09_4.pdf

----

Creating Biased Coin from Fair Coin - 27 February 2018

Q: Fair coin of Head and Tail has probability of 1/2 for either turning up. How can an unfair coin be created from fair coin?

A: 1) One possible solution is to have set of fair coins and tossing them all simultaneously. Return 1 if a regular expression occurs in the streak else 0. This would be unfair because percentage of regex matches outnumber percentage of regex mismatches and probability of unfairness follows. For example, from a set of 3 fair coins tossed simultaneously (0 for Head and 1 for Tail):

```
000
001
010
011
100
101
110
111
```

number of streaks matching regex 11 are 011,110,111 which is probability 3/8. This creates an unfair randomness bias and set of streaks matching regex correspond to 1 and rest are 0 in the unfair coin. Pr(streaks having 11=1) = 3/8 and Pr(streaks not having 11=0) = 5/8. This is a very primitive epsilon bias generator.

2) Another solution which expands a uniformly chosen permutation array by replicating an extra skew variable and all but skew variable have a biased probability of choice has been implemented in NeuronRain AsFer (https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/

EpsilonBiasNonUniformChoice.py) and is used in generating random 3SAT instances for SAT Solver - https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/CNFSATSolver.py. This is based on creating a Random Matrix per random 3SAT , computing Expected average per literal probability and is different from the standard Survey Propagation Message Passing Algorithm which represents SAT as a factor graph - https://arxiv.org/pdf/cs/0212002.pdf - graph having 2 types of vertices for variables and clauses and edges are between variables and clauses having variables - message passing belief propagation of potentials of a variable taking value 1 or 0.

Print the Nth element of a Fibonacci Sequence - 12 March 2018, 21 March 2018

Trivial solution uses the recurrence f(n) = f(n-1) + f(n-2) and f(0)=f(1)=1 and is exponential. Assuming Memoization/Cacheing of results,f(n-2) and f(n-1) can be memoized to compute f(n). Mathematically, Nth Fibonacci number is expressed in terms of Golden Ratio Phi = (1 + sqrt(5))/2 as:

```
      f(n) = (Phi^n-(1-Phi)^n/sqrt(5)
```

which is based on definition of Golden Ratio = f(n+1)/f(n) for large n

Related fibonacci recurrence is the problem of finding number of 1s in set of all n-bit strings. Number of 1s or 0s in set of all n-bit strings is denoted by the recurrence:

```
      f(n)=2*f(n-1) + 2^(n-1)
```

Expanding the recurrence recursively creates a geometric series summation which gives the Nth element in sequence:

```
      f(n) = [2^n-1 + 2 + 2^2 + 2^3 + ... + 2^(n-1)]
```

Probability of finding 1s or 0s in set of all n-bit strings = [2*f(n-1) + 2^(n-1)] / n*2^n = 0.5

This recurrence is quite ubiquitous in problems involving uniform distribution e.g number of positive/negative votes in voting patterns, number of Heads/Tails in Bernoulli Coin Toss Streaks etc.,.It has been mentioned in the context of 2-coloring/Complementation in https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt .

----
760. (THEORY and FEATURE) Newton-Raphson approximate factoring - 6 April 2018 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
https://kuja27.blogspot.in/2018/04/grafit-course-material-newton-raphson.html

References:
----
1. http://www.math.lsa.umich.edu/~lagarias/TALK-SLIDES/dioph-cplx-icerm2011aug.pdf - Binary Quadratic Diophantine Equations (BQDE) and Factorization are equivalent. BQDE is known to be in NP(Succinct Certificates for Solutions to BQDE). If BQDE is in P, Factorization is in P. Computational Geometric NC algorithm for Factorization probably implies BQDE is in P (probably because implication is in opposite direction).

----
761. (THEORY and FEATURE) Chomsky Sentences - 6 April 2018 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

----
https://kuja27.blogspot.in/2018/04/grafit-course-material-chomsky.html

----
750. (THEORY and FEATURE) Money-Changing Problem and minimum partition - 6 May 2018, 9 May 2018, 18 February 2020 - this section is an extended theory draft on set partitions, optimal denomination and coin problems among other topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
Q: How can all integers be generated as sum of elements of a minimum sized set (or) What are the least number
of denominations for a currency to sum up to all possible values of money?

A: Money Changing Problem or Coin Problem is an NP-hard problem (strong-NP or weak-NP depending on encoding) and is a variant of Integer Partition Problem. Most currencies use 1-2-5 series (and 10^x multiples of 1,2,5). Linear combinations of multiples of 1,2,5 can create all possible values minimizing the number of coins/bills e.g 2950 neuros (fictitious currency in NeuronRain) can be written as 2950 = 2*1000 + 1*500 + 2*200 + 1*50 and only 6 notes/coins are sufficient. There is a polynomial time Greedy algorithm for this as below(but exponential in number of bits):

```
      Sort the denominations descending (1-2-5 series and multiples)
      while (value != sum)
      {
            Choose the largest possible currency denomination remaining (ci) 
from sorted denominations
            subtract largest multiple m of it from value (value = value - ci*m)
            coins[ci] = coins[ci] + m
      }
```

This algorithm also maps the partition to a hash table of optimum size - if each currency is assigned a serial number and denominations are keys, each per-key bucket is a chain of serial numbers for that denomination.In previous example, 2950 is mapped to hash table as below for serial numbers s1,s2,...,s6 and denominations 1000,500,200,50:

```
      1000 - s1, s2
      500 - s3
      200 - s4, s5
      50 - s6
```

This algorithm solves the linear diophantine equation 1*exp(10)*x1 + 2*exp(10)*x2 + 5*exp(10)*x3 = N and also creates an exact cover/partition of the currencies represented by some diophantine.

Products of Other Integers - 22 May 2018

Q: Find an efficient algorithm to find the product of all other integers excluding an element or elements in an array. Naive algorithm for excluding one element is O(N). For example, array [2,3,7,5,6] is multiplied to find product of all : 2*3*7*5*6 = 1260 and array of products of all other integers is found by iterated division per element : [1260/2,1260/3,1260/7,1260/5,1260/6]. Generalizing this to all possible subset exclusions is non-trivial.

A: Following optimization is a way out. Downward closure subset product computations are cached in hash table at the outset as below by traversing array :

```
            (2,3) - 6
            (3,7) - 21
```

(7,5) - 35
(5,6) - 30
(2,3,7) - 42
(3,7,5) - 105
(7,5,6) - 210
(2,3,7,5) - 210
(3,7,5,6) - 630
(2,3,7,5,6) - 1260

This precomputation is O(N*N) and done only once as prerequisite. This does not compute 2^N=2^5=32 subsets but only (N-1 + N-2 + N-3=) 9 subsets.

For excluding subset {3,5} naive algorithm has to find product of set {3,5} or set difference {2,7,6} and compute the product 84 by division of 1260 which is O(N).

Algorithm based on previous lookup table:
----
1. For a subset X to exclude, find the maximum overlapping subset of minimum size S in the lookup table i.e X intersection S is maximum but S has smallest possible size.

2. Divide the product for S by the product for set difference of X with S to get product Z. (This could be recursive lookup for set difference in the previous cache)

3. Divide the product lookedup for all elements by Z for product of other integers

E.g 1. for excluding X={3,5}, lookup of the table results in S={3,7,5} which has maximum overlap with {3,5} but of smallest size ({2,3,7,5} also overlaps but is of larger size). Dividing the product lookedup for {3,7,5}=105 by set difference 7 with {3,5} yields Z=15. Finally, dividing product of all elements 1260 by 15 = 84 = 2*7*6.

E.g 2. for excluding X={3,6}, lookup of the table results in S={3,7,5,6} which has maximum overlap for {3,6}. Dividing the product lookup for {3,7,5,6}=630 by recursive lookup for set difference {7,5}=35 yields 630/35=18. Final exclusion is 1260/18 = 70 = 2*5*7

This algorithm is subset oblivious and is a slight improvement over brute-force multiplication of set or set difference because of cached subset products and is sublinear mostly. Previous examples required only 2 divisions whereas brute-force would need 3 multiplications assuming lookups are O(1). Cacheing has benefits in large arrays (when number of elements to be excluded are almost O(N)) for reducing number of multiplications.

----
751. (THEORY and FEATURE) Hashing Dynamic Sets - 24 May 2018, 31 October 2018, 11 March 2019, 8 October 2019, 6 November 2019, 18 February 2020, 1 May 2020, 5 May 2020, 2,3,4 June 2020, 30 July 2020,1 January 2021 - related to all sections on Set partitions, Computational geometry, Program analysis/Software analytics/Scheduler Analytics among other topics in NeuronRain Theory Drafts
----
Q: How can sets of elements which are dynamically modified at runtime be hashed by tabulation? E.g set of
clockticks remaining per process in an OS scheduler for 15 processes is [23,45,12,44,55,14] at time t=1. This
set is mapped to processes by hash table:

23 - p1,p2,p3
45 - p4,p5,p6,p7
12 - p8,p9

44 - p10
55 - p11,p12,p13
14 - p14,p15

which is a snapshot at time t=0. As timer ticks to t=1, previous hash table keys for remaining clockticks have to be decremented as:

22 - p1,p2,p3
44 - p4,p5,p6,p7
11 - p8,p9
43 - p10
54 - p11,p12,p13
13 - p14,p15

and new processes p16,p17,p18 are added at time t=2 with remaining timeslice clockticks 35,21,53 expanding the table to:

21 - p1,p2,p3,p17
43 - p4,p5,p6,p7
10 - p8,p9
42 - p10
53 - p11,p12,p13,p18
12 - p14,p15
35 - p16

Other clockticks are decremented based on timer. When a clocktick reaches 0, the queue of processes for it is removed.

A: Usually hash table keys are static not allowing dynamism. This requires an overloading/overriding of hash_code() and equals() functions programmatically in the respective implementation language which simulate equality of two keys so that value is appended to the correct queue bucket. Problem is how to lookup changing keys decremented by timer thread. An example equals() function is : key_clockticks1 - timerticks1 == key_clockticks2 - timerticks2. Rewriting the table:

23-2 - p1,p2,p3,p17
45-2 - p4,p5,p6,p7
12-2 - p8,p9
44-2 - p10
55-2 - p11,p12,p13,p18
14-2 - p14,p15
35 - p16

and hash_code() for a key is key_clockticks - timerticks. For example to lookup process p6, hash_code() returns 45-2=43 at time t=2 re-routing to bucket for 43 instead of 45 at time t=0. Similarly two processes pi and pj of time slices 14 and 12 but having elapsed timerticks 4 and 2 have equal hashcodes - 14-4 = 12-2 = 10 - pi is older than pj and pj is enqueued in scheduler 2 ticks after pi when 14-2 = 12-0 = 12. Therefore both pi and pj are in same clocktick queue for 10.

When implemented as LSH partition, clockticks-to-processes map is isomorphic to some random integer partition of n(number of processes) and both n and partition of n oscillate dynamically based on clockticks. Mining patterns in streaming dataset of clockticks-to-processes maps is an indicator of performance of the system. Each clockticks-to-processes dictionary can be represented as a matrix:

c1 p11 p12 ... p1m
c2 p21 p22 ... p2m
...
cn pn1 pn2 ... pnm

ci are clockticks and p(i,k) are processes having ci clockticks remaining before being swapped out of scheduler. Because of matrix representation each LSH partition is a graph too (previous matrix is its adjacency). Frequent subgraph mining algorithms can mine patterns in the clockticks-to-processes dictionaries. If the dictionary is string encoded, string search algorithms - multiple alignment, longest common subsequence etc., - can be applied for pattern mining. These elicited patterns are samples of how system behaves - number of processes consuming most clockticks, average load etc., Adjacency matrix for previous Survival Index Timeout Separate Chaining hashtable graph has edges of the form: <time_to_live_clockticks> -> <process_id> and this graph is dynamically refreshed after each timer tick. This graph can also be augmented by parent-child relation edges between processes (process tree and process groups) in different clocktick buckets and locks held/waited by them. Cycle detection algorithms applied on this graph for lock (hold/wait) cycles after each clocktick prevents deadlocks/races. This augmented hashtable chain directed graph has 4 types of edges:

<time_to_live_clockticks> -> <process_id>, <parent_process_id> -> <child_process_id>, <process_id> -> <mutex_id>, <mutex_id> -> <process_id>

Dynamism of this timeout hashtable/dictionary graph warrants mention of Dynamic Graph algorithm results - changes in the timeout hashtable after every clocktick is reducible to updates/insertion/deletion in a dynamic graph by previous definition of adjacency matrix from hashtable - insertions/deletions/updates in hashtable buckets are reflected in adjacency matrix for its graph. Reckoning only the <time_to_live_clockticks> to <process_id> edge, previous clockticks-to-processes dictionary is a dynamic stream of noncrossing (NC) set partitions - each block(bucket) in the partition for every clocktick lapse is an element in the Lattice of partitions defined by Hasse Diagram. Number of such partitions is given by Narayana Number. This timeout dictionary pattern occurs cutting across many arenas of theory and systems. Previous example of OS Scheduler is just mentioned for the sake of commentary and some official copyrighted implementations of this universal theoretical timeout pattern mentioned in references are in different software contexts. Precise example for exact time_to_live is the network routing in ISPs which have to timeout ageing packets. TCP/UDP and other protocol families support time_to_live in packet headers preset by user code.

An example pattern: Sort the previous pending clockticks(Survival Index) to processes map by descending values of clockticks. Percentage of processes flocking in top ranking clocktick bucket chains is a measure of system load - runqlat utility in linux kernel 4.x (BPF/bcc-tools) has close resemblance to clockticks-to-processes dictionary but in the histogram format (https://github.com/iovisor/bcc/blob/master/tools/runqlat_example.txt). But this histogram is a map of waiting clockticks to number of processes and not runqueue - consumed timeslice clockticks to processes.

Traditional timeout implementations are timer wheel based which are circular arrays of linked lists sweeped periodically like clockwork and are not hashable. Previous hashing of dynamic sets is also a timer wheel but takes a detour and converts hash table separate chaining itself into a dynamic clock in which, for example, hour keys are decremented periodically.

Mining Patterns in Survival Index Timeout:

Since every hashmap induces a set partition, previous timeout hashtable separate chaining partitions set of processes into buckets or baskets. Traditional Frequent Itemset Mining techniques - FPGrowth etc., - are applicable only for intra-hashtable patterns when process id(s) are multilocated across timeout buckets, which elicit frequently co-occurring set of process id(s) within the hashtable. Measuring inter-hashtable distance or distance between two survival index hashtable set partitions is a non-trivial problem. This partition distance problem is formulated by mapping two partitions to a distance graph and vertex cover on this partition distance graph (different from LSH graph of a hashtable previously described). This distance dynamically fluctuates based on processes forked and timedout and is a measure of system load.

Considering the stream of processes set partition induced by survival index timeout buckets as timeseries of set partitions, provides an alternative spectacle to view and mine patterns in OS Scheduler as ARMA or ARIMA polynomials. This requires mapping each set partition in stream to a scalar point in timeseries. Distance between two consecutive observations in timeseries is called Differencing and distance between any two consecutive processes set partition timeout histograms can be defined by Earth Mover Distance or Wasserstein Distance in addition to RandIndex. Linear complexity approximations of Earth Mover Distance (e.g. LC-RWMD) could be faster differencing measures for stream of timeout-to-processes set partitions.

Caveat:

Previous adaptation of Survival Index based Transaction Timeout Management (mentioned in the references) to OS Scheduler assumes prior knowledge of execution times of processes which is undecidable in exact sense by Halting Problem. Only an approximate estimate of execution time of a process can be derived by Analysis of control statements in the program (e.g Sum of execution times of control statements in longest path in the control flow graph of the program is the upperbound). Most of the static code analysis tools for worst case execution time (WCET) do not depend on input size and concentrate only on realtime operating systems and WCET for non-realtime OS implementations therefore can be dynamically derived from theoretical worst case execution time of algorithm underlying executable by reckoning input size e.g Master Theorem T(n) = aT(n/b) + f(n) for divide-and-conquer estimates worst case upperbound running time of algorithm underlying a process executable based on toplevel recursion function f(n) and constants a and b for every inner level of recursion - sorting executable is O(NlogN) and by choice of constant a, approximate worst case execution time of process executable is aNlogN. Constants have to be found by trial-and-error and mostly are architecture dependent. There are few trivial exceptions to what Master Theorem can estimate and there is no necessity of CFG longest path estimation. Busy Beaver Function is an alternative formulation which quantifies the maximum number of steps of a halting Turing Machine of N states defined as:

BB(N) = maximum number of 1s written by a Halting Turing Machine of N states on tape

But BB(N) requires prior knowledge of number of states of Turing Machine for a process executable. Brainfuck is a Turing-complete programming language for designing Turing Machines and host of tools are available to translate a high level programming language (C,C++) source code of a process executable to Brainfuck (.bf) format. If the translated Brainfuck code for a high level language source of process executable has N states its worst case runtime is upperbounded by BB(N) which is a relaxation of master theorem upperbound.

Survival Index Timeout as Earliest Deadline First (EDF) OS Scheduler:

Linux Kernel has Earliest Deadline First Scheduler which requires user to specify Worst Case Execution Time (WCET) of a process and deadline (runtime << deadline) explicitly by chrt in commandline or programmatically by sched_setattr(). EDF scheduler prioritizes low deadline processes/threads first, causing long deadline processes to wait longer. Commentaries in References below mention an example constraint to be satisfied:

WCET_deadline_timeout_value_of_bucket * number_of_processes_in_the_bucket = constant.

This constraint makes the OS Scheduler histogram lopsided - shortest deadline buckets are longest and longest deadline buckets are shortest. Enforcement of this kind of EDF constraint is tantamount to mapping deadline(timeout) values of buckets to process priorities (nice values) - lowest deadlines/timeouts have highest priorities and viceversa.

Usual scheduler race deadlock anomalies of priority inversion arise in previous Survival Index EDF scheduler too e.g low deadline thread/process id spawns another thread/process id of long deadline and waits for high deadline thread/process to end or blocks for a resource locked by high deadline thread/process. This causes starvation of both low deadline process/thread and high deadline process/thread - low deadline process waits for high deadline process to release the lock; High deadline process holding lock, which may not be scheduled in near future, blocks low deadline process stagnating further scheduling of both high and low deadline processes resulting in system freeze. This requires high deadline process to be reprioritized and deadlock avoidance.

Timeout as Graph Partition (Dynamic Process id(s) tree partition):

Previous Survival Index based OS process dynamic set partition can be theorized by a Dynamic Graph Partition too - each process has a dependency to some other process by parent/child fork() relationship as graph edge and processes as vertices. At any instant, survival index graph partition captures both the buckets of the processes set partition for timeout values and dependencies among buckets.

Process id dynamic set/graph partition and rectilinear partition - a computational geometric perspective:

Aforementioned Survival Index Worst case execution time partition of process id(s) in an OS kernel can be viewed as computational geometric problem of partitioning a rectilinear orthogonal polygon into rectangles - Every timeout value bucket in set-partition histogram is visually a rectangle of dimensions 1 * number_of_processes_per_bucket. If size of set of process id(s) is factorizable as 2-dimensional orthogonal polygon, per-bucket rectangles tile this rectilinear process space. NeuronRain theory drafts describe and implement a set-partition to Lagranges four square theorem tile cover reduction which finds a rectangle by factorization of size of set-partition and tiles it by squares - an example of rectilinear partition. As a matter of fact, every histogram set partition is theoretically a rectangle partition (of dimensions 1 * size_of_set_of_processes).

References:

1.Previous algorithm is a generalization of Survival Index Based Transaction Timeout Management mentioned in patent disclosure https://sites.google.com/site/kuja27/SurvivalIndexBasedTxnTimeoutManager.pdf (Patent Pending - Copyright - Sun Microsystems/Oracle) - Specific to Java Hashmap (CustomizedHashMap) and Open Addressing - related to erstwhile iPlanet Application Server (iAS - now GlassFish appserver - https://github.com/javaee/glassfish/tree/master/appserver) patents - https://patents.google.com/?inventor=Srinivasan+Kannan&assignee=Sun+Microsystems%2c+Inc.
2.Continuous Parameter Markov Chains - Probability and Statistics,Reliability,Queueing and Computer Science - [Kishore Shridharbhai Trivedi, Duke University] - Birth and Death Processes - Response Time of RoundRobin Scheduling - Little's Formula - [Chapter 8] and Network of Queues - Open Queueing Networks - [Chapter 9] - Program state transition markov chain
3.Program Analysis - Analysis of Control Statements - expected and variance of execution times and laplace transforms - Appendix E - Probability and Statistics, Queueing, Reliability and Computer Science - [Kishore Shridharbhai Trivedi, Duke University]
4.Introduction to Algorithms - [Cormen,Leiserson,Rivest,Stein] - Chapter 12 - Hashing by Chaining - http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap12.htm - Static Hashing - all runtime analyses for searching in static hashtables still apply to hashing dynamic sets - Note: Hashing Dynamic Sets is different from Dynamic Hashing which facilitates fast insertion/deletion of elements and not dynamism of element itself. Also previous adaptation of hash chaining for timeout and schedulers assumes the hash function for any process is defined as:
h(p) = new_execution_time_of(p) = old_execution_time_of(p) - clockticks_elapsed
5.Hashing by Chaining - http://cglab.ca/~morin/teaching/5408/notes/hashing.pdf - Section 1.3.1 - Gonnet's result on worst case search time in a Chain.
6.Structured Programming - Proper Programs - [Linger] and [Beizer] - https://books.google.co.in/books?id=GZ6WiU-GVgIC&pg=PA264&lpg=PA264&dq=Linger+beizer+proper+program&source=bl&ots=e169qHibIm&sig=Tp_GOlaenwMOLpkc9ip7scs4xus&hl=en&sa=X&ved=2ahUKEwjv_KHx8LfeAhVDL48KHQhTBO

kQ6AEwAHoECAkQAQ#v=onepage&q=Linger%20beizer%20proper%20program&f=false - Page 264 - Probability and Statistics,Reliability,Queueing and Computer Science - [Kishor Shridharbhai Trivedi] - Expected Execution Time of a Program
7.ISO C++ Bucket Interface - Page 920 - Chapter 31 - C++ Programming Language - [Bjarne Stroustrup] - https://books.google.co.in/books?id=PSUNAAAAQBAJ&pg=PA919&lpg=PA919&dq=hash+table+bucket+size+bucket_count+C%2B%2B&source=bl&ots=DrvmDeg57M&sig=Kj4qXTnTfI-x5qh50bPoiBYoaFw&hl=en&sa=X&ved=2ahUKEwiW6f35187eAhUHVH0KHf9bClc4ChDoATADegQIBhAB#v=onepage&q=hash%20table%20bucket%20size%20bucket_count%20C%2B%2B&f=false - Bucket Interface provides const and mutable iterables for each bucket in the unordered_map chained hash table. In the context of previous Survival Index timeout table, mutability is defined by the dynamic hash_code(). Aside: ThoughtNet - an Evocatives based Hypergraph - in NeuronRain AsFer has been implemented as Reinforcement Learning Contextual Multi Armed Bandit dictionary which maps an evocative class to set of sentences of that class (from some classifier). ThoughtNet can also be viewed as a Hash table Chaining in which Bucket linked lists of sentences for each evocative are interconnected (Hashmap-cum-LinkedList diagram in https://sites.google.com/site/kuja27/SurvivalIndexBasedTxnTimeoutManager.pdf). ThoughtNet Hypergraph and previous Survival Index Timeout Separate Chaining can be represented by an adjacency matrix (Hypermatrix Tensors - https://www.sciencedirect.com/science/article/abs/pii/S0167506008700578, http://courses.cs.vt.edu/cs6824/2014-spring/lectures/student-presentations/2014-01-27-student-presentations.pdf). In the case of ThoughtNet Hypergraph Chaining same value(thought or sentence id) could exist in multiple buckets making a hyperedge connection sprawling over 2 or more buckets which is ruled out in Survival Index Timeout but for the processes having multiple threads each having different timers which demands multilocating a process id and therefore a hyperedge. Survival Index hypergraph is dynamic (edges and vertices are inserted and deleted over time).
8.Mining LSH Partitions/Dictionaries - DictDiffer in Python for difference between two dictionaries - https://github.com/inveniosoftware/dictdiffer
9.Noncrossing Partitions and TV Narayana Number - https://en.wikipedia.org/wiki/Noncrossing_partition
10.Dynamic Graph Algorithms - http://cs.ioc.ee/ewscs/2012/italiano/dynamic1.pdf
11.Different version of Timeout in Global Decision Platform 3.0 - C++ - https://sites.google.com/site/kuja27/PhDThesisProposal.pdf - (Copyright: Global Analytics)
12.Intel Threading Building Blocks (TBB) Concurrent Hash Table Bucket Interface - https://www.threadingbuildingblocks.org/docs/help/index.htm#reference/containers_overview/concurrent_unordered_map_cls.html
13.Introduction to Algorithms - [Cormen-Leiserson-Rivest-Stein] - Coupon Collector Problem/Balls and Bins Problem - Page 134 (5.4.2), Page 1201 (C.4)
14.Balls and Bins Problem, Set Partitions, Bell Numbers and Multinomial Theorem - https://math.dartmouth.edu/~m68f15/lectec.pdf - Multinomial Theorem is applicable to previous Survival Index Separate Chain Set Partition if size of each bucket is some constant - e.g Processes are numbered balls and Timeout values are numbered bins and Multinomial Theorem gives all possible configurations of Timeout datastructure subject to rider: bin for timeout value t(i) must contain m(i) processes. Stirling Numbers of Second Kind (Bell Numbers) is the number of all possible Timeout datastructure configurations of p processes and n timeout values (unrestricted bin size).
15. Kruskal-Katona Theorem and Erdos-Ko-Rado Theorem for families of intersecting sets - https://en.wikipedia.org/wiki/Kruskal%E2%80%93Katona_theorem - Uniform Hypergraphs (size of each hyperedge set is equal) are families of intersecting sets if the hyperedges have non-empty intersection. These two theorems upperbound number of hyperedges in a hypergraph by a binomial coefficient. ThoughtNet which is Hypergraph of sentence hyperedges and each hyperedge is set of evocative vertices usually has high intersection (each element in intersection is represented by a stack hypervertex).
16. Linux Kernel Timer Wheel implementation - https://lwn.net/Articles/646950/ - as tree hierarchy of arrays of linked lists
17. Earliest Deadline First Scheduler (EDF) in Linux Kernel - Part 1 - Dhall Effect in multicores - https://lwn.net/Articles/743740/ - "...The run time is

the amount of CPU time that the application needs to produce the output. In the most conservative case, the runtime must be the worst-case execution time (WCET), which is the maximum amount of time the task needs to process one period's worth of work. For example, a video processing tool may take, in the worst case, five milliseconds to process the image. Hence its run time is five milliseconds...."
18. Earliest Deadline First Scheduler (EDF) in Linux Kernel - Part 2 - sched_setattr() for setting worst case execution time and deadline explicitly - https://lwn.net/Articles/743946/
19. Partition distance - [D. Gusfield.] - Partition-distance: A problem and a class of perfect graphs
arising in clustering - https://csiflabs.cs.ucdavis.edu/~gusfield/cpartition.pdf
20. Various distance measures between 2 set partitions - http://igm.univ-mlv.fr/~gambette/Re20121025.pdf - Overlap Distance - Minimum Number of elements to be removed to remove all overlaps between subsets of 2 partitions
21. Rand Index - Distance between two sets of classified subsets (set partitions) - https://en.wikipedia.org/wiki/Rand_index
22. Graph Partition - https://en.wikipedia.org/wiki/Graph_partition
23. Dynamic Graph Partition - [Chen Avin et al] - https://arxiv.org/pdf/1511.02074.pdf - Previous Survival Index OS Scheduler is a Dynamic Graph Partition and requires online algorithms (all inputs are not readily available and processes are streamed data) because processes are forked and timedout frequently and parent-child fork relation edges between process vertices (which could be in different timeout buckets) are deleted and created dynamically.
24. Dynamic Tree Partition - Linear weighted tree partition - [Sukhamay Kundu, Jayadev Misra] - https://epubs.siam.org/doi/abs/10.1137/0206012?journalCode=smjcat - SIAM J. Comput., 6(1), 151–154. (4 pages) 1977 - Previous Survival Index (Worst Case Execution Time) OS Scheduler is a Dynamic Tree Partition if processes are related only by parent-child fork relation - at any instant set of process id(s) waiting to be scheduled on CPU form an n-ary tree and partitions of process id(s) which are timeout value buckets can be arbitrary and need not be rooted subtrees (could be non-rooted subforests). Good scheduling therefore reduces to balanced process tree partitions: number of processes per timeout(deadline) value bucket must be inversely proportional to timeout(deadline) value of the bucket (or number_of_processes_per_timeout_bucket * timeout = constant) which is exactly Earliest Deadline First (EDF) scheduling - small duration processes are preferred while longhaul processes are delayed.
25. Histogram Distance Measures - Because survival index scheduler is a dynamic histogram of timeout versus processes, all histogram distance metrics are relevant for analyzing stream of OS scheduler histograms (Wasserstein-Earth Mover Distance, Correlation,ChiSquare,Intersection,Bhattacharya-Hellinger - https://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html). Earth Mover Distance is a variant of Transportation Problem and Simplex. Histogram Stream of Scheduler Queue Checkpoints can be plotted as timeseries of distances between pair of consecutive scheduler runqueue histograms.
26. Program Analysis - Worst Case Execution Time - Estimation - Survey - https://arcb.csc.ncsu.edu/~mueller/ftp/pub/mueller/papers/1257.pdf - Control Flow Graph, IPET, Longest Path, Syntax tree structure based
27. Rectangle Partitions of Orthogonal Polygon - [David Eppstein] - Graph-Theoretic Solutions to Computational Geometry Problems - Section 3 and Figure 2 - Matching and Maximum Independent Sets in Bipartite Graph of Intersections of Good Diagonals - https://arxiv.org/pdf/0908.3916v1.pdf
28. The Heptane Static Worst-Case Execution Time Estimation Tool - ARM and MIPS instruction sets only - [Damien Hardy,Benjamin Rouxel,and Isabelle Puaut] - https://drops.dagstuhl.de/opus/volltexte/2017/7303/pdf/OASIcs-WCET-2017-8.pdf
29. Worst Case Execution Time and Earliest Deadline First Scheduler in Linux Kernel - SCHED_DEADLINE - https://www.kernel.org/doc/html/latest/scheduler/sched-deadline.html
30. Master Theorem - [Bentley-Haken-Saxe] - https://apps.dtic.mil/dtic/tr/fulltext/u2/a064294.pdf
31. Master Theorem - Case 2 - [Goodrich-Tamassia] - Algorithm Design: Foundation, Analysis and Internet Examples - Pages 268-270
32. Discrete Parameter Markov Chains - Probability and

Statistics,Reliability,Queueing and Computer Science - [Kishore Shridharbhai Trivedi, Duke University] - Chapter 7 - 7.8 Analysis of Program Execution Time - Program Flow Graph - Page 358 - [Knuth 1973] - Problem 1 - Stochastic Program Flow Graph
33. The Art of Computer Programming - Volume 1 - [Don Knuth] - Analysis of Program Execution Time - pages 190-192 - 1.3.3 - Basic concepts - Timing - Kirchoff's First Law for Program Control Flow Graph: sum of incoming edges = sum of outgoing edges - pages 383-389 - 2.3.4 - Free tree and Flow chart of a Program - Theorem K - [Thomas Ball, James R.Laurus] - ACM Transactions on Programs and Systems 16 (1994), 1319-1360.
34. Efficient Path Profiling - [Ball-Laurus] - Published in the Proceedings of MICRO-29, December 2–4, 1996, in Paris, France. - ftp://ftp.cs.wisc.edu/wwt/micro96.pdf
35. Brainfuck Language for Turing Machines - https://esolangs.org/wiki/Brainfuck - Hello world example
36. C to Brainfuck compiler - C2BF - https://github.com/arthaud/c2bf
37. FBP - High level language to Brainfuck compiler - https://esolangs.org/wiki/FBP
38. (Vide reference 7) Bucketization - Incorrectly assigning class labels to objects - [Ethem Alpaydin] - Introduction to Machine Learning - https://www.cmpe.boun.edu.tr/~ethem/i2ml/ - 3.3 Losses and Risks - Page 51 - "... Let us define action ai as the decision to assign the input to class Ci and Lik as the loss incurred for taking action ai when the input actually belongs to Ck. Then the expected risk for taking action ai is R(ai) = Sigma(Lik * P(Ck)) ..."

----

762. (THEORY and FEATURE) Markov Chains Random Walks on a Random Graph and Television Viewership, Merit of Large Scale Visuals, Media Analytics, Business Intelligence, Timeseries, Computational Geometry, Intrinsic Merit and Originality/Creative Genius, Graphical Event Models and EventNet - 4 June 2018,3 July 2018,7 March 2019,7 May 2020, 2,3,4 June 2020, 25 January 2021, 8 April 2021,14 April 2021 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

----

Television viewership is the most researched subject for Media and Advertisement Analytics (Business Intelligence). A viewer randomly shuffling channels creates a Channel Random Graph dynamically where :

*) Channels are the vertices
*) Switching Channels creates a random hyperlink edge between two Channel vertices c1 and c2 with some probability

This random graph is similar to World Wide Web and a converging random walk on this Channel graph implies viewer is finally satisfied at some point (Random Walk Mixing Time) after traversing the hyperlinks. Each such random edge is a Markov transition depending on previous state. This is similar to PageRank iteration applied on the Channel Random Graph which is converging Markov Random Walk and most ranked vertices/channels can be approximate preferences of the viewer. Infact there is more to it than PageRank - amount of time spent per channel between switches is crucial. There is a subtlety: Predicting Viewers Makes them More Unpredictable - This is because ranking channel vertices from previous random graph and directing more ads to the topmost channel, repels the viewer and causes a random channel switch. Again the PageRank has to be recomputed for finding new topmost channel and this process repeats endlessly - kind of Uncertainty Principle in macrocosm - measuring momentum and location of a subatomic particle changes its momentum and location.

Previous example differs from reputation rankings on the net because TV ads cannot be personalized similar to web adverts and each viewer creates a channel switch random graph independent of others (assuming each individual viewership statistics is recorded in a device or access meter in a set-top box and transmitted). This creates set of ranking preferences per viewer all of which have to be rank correlated and mapped to TRP for ad(s) which satisfies majority. Rank correlations are measures which measure similarity between two rank labeled sets. In Psychology, Spearman Rank Correlation (RC) of two ranked datasets of size n (e.g manual rankings of same dataset by two individuals) is computed by RC = 1 - [6*Sum(euclidean_distance^2) / n*(n^2 - 1)]. High distance minimizes rank correlation.

Previous Converging markov random walk algorithm applies to Business Analytics particularly in FMCG where customers have lot of options to try out. An alternative histogram analytics perspective for mining stream of business intelligence dictionary data (distance metric based on rand index) is described in
https://gitlab.com/shrinivaasanka/asfer-github-code/blob/03333f9fbf0dd087a2fa90bd1856887c8e2eca0f/asfer-docs/AstroInferDesign.txt. Regression analysis is the primary tool for business and economics research which connects a dependent variable and set of independent variables by a linear or logistic regression equation. For example, Y = Sum(ai*Xi) + b defines a linear regression model of independent variables Xi for dependent variable Y. Weights ai are found by least squares method on equations for N observed values of Y and Xi:

```
Sum(Y) = Sum(ai * Sum(Xi)) + Nb
Sum(XY) = Sum(ai * Sum(Xi*Xi)) + b*Sum(Xi)
```

Previous PageRank computation on Channel Switch Random Graph per viewer can be mapped to a histogram of PageRank score range buckets clustering almost similarly scored channels (intuitively channels of similar genre have almost similar scores thereby partitioning the set of channels by genre buckets) which is a probability distribution per viewer. Majority viewership trend has to be inferred from this stream of per viewer histograms e.g clustering the viewer histograms by adjusted rand index distance measure and largest cluster histogram trend wins by majority vote. It is worth noting that Television Rating Points are intrinsic merit measures for media analytics.

An intrinsic alternative to previous PageRank based voting by viewers is to rank content of channels by EventNet Tensor Products Audio-Visual Merit algorithm implemented in NeuronRain AsFer which considers every audio-visual as stream of causally related frame graphs inferred from ImageNet. This algorithm is not restricted to videos alone but models physical reality of event cause-effect (kind of simplified PetriNets where places are events and transitions are causations between events). Based on genre (Sports, Info, Entertainment etc.,) Empath or LIWC sentiment analysis might be necessary for emotional content. EventNet is a Graphical Event Model(GEM) and there are already GEM algorithms - OGEM, PGEM - which learn graphical dependency from timeseries stream of events of mostly business intelligence, economic or political genre. Formulating Video (timeseries stream of frames) as Graphical Event Model generalizes OGEM and PGEM algorithms to learn EventNet Graph of dependent frames. EventNet Tensor Products algorithm is thus a Graphical Event Model (GEM) algorithm. EventNet and Graphical or Causal Event Models are best suited for analyzing astronomical event timeseries e.g event series denoted by ordered pairs of the form (n-body celestial configuration, terrestrial event) and predicting bayesian likelihood of weather events.

H-index measure of merit in academic research is defined as h number of articles by an academic each of which have citations by atleast h other academics. Similar notion can be generalized to text,audio,video and people too. In the context of video merit, for example, h number of atleast h-times retweeted videos is a measure of quality. An alternative definition of merit in the context of music has been presented in NeuronRain AstroInfer Audio and Music Analytics which is based on how original an Audio waveform is, measured by distance dissimilarity (between other composers) and similarity (theme amongst works of oneself). Similar definition of originality can be arrived at for following categories of merit:

Text - Semantic (Conceptual) Dissimilarity/Similarity between TextGraphs of academic publications by different authors and Self

People - Semantic Dissimilarity/Similarity between Career transition (modelled by some state machine automaton) of different people and self - Choices made across tenures define people

Video - Narrative Dissimilarity/Similarity between FaceGraph (Voronoi tesselated frames by centroid tracking) and EventNet Tensor Product representation of movies, youtube videos by different creators and Self

References:

1.Ranks as Symmetric Permutation Group Sn and Rank correlations - Spearman's Footrule as measure of Disarray - [Persi Diaconis and R.L.Graham] - https://statweb.stanford.edu/~cgates/PERSI/papers/77_04_spearmans.pdf
2.H-index - Measure of Academic Research Quality - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1283832/
3.Ordinal Graphical Event Models - OGEM - https://www.ibm.com/blogs/research/2021/01/ijcai-graph-flow/,https://www.ijcai.org/Proceedings/2020/274 - Quite similar to EventNet but without intraevent actors and applicable to many BigData sets having temporal causality - "...OGEMs go a step further. They aim to capture the effect of the order in which preceding events have occurred and detail how each one has affected the event of interest. They do so using a new algorithm we’ve developed, which learns an event’s causes and the quantification of the effect of the order of the causes using event streams as input....The graph itself may include cycles and even self-loops for event labels, capturing the dynamics of the process. For instance, event type C in fig. 1b depends on historical occurrences of event types A and B — meaning there is a parameter for every potential order of every subset of {A, B}...."
4.Proximal Graphical Event Models - PGEM - https://papers.nips.cc/paper/2018/file/f1ababf130ee6a25f12da7478af8f1ac-Paper.pdf - "...ICEWS. We consider the Integrated Crisis Early Warning System (ICEWS) political relational event dataset [O’Brien, 2010], where events take the form ‘who does what to whom’, i.e. an event z involves a source actor az performing an action/verb vz on a target actor a0z , denoted z = (az, vz, a0z). In ICEWS, actors and actions come from the Conflict and Mediation Event Observations (CAMEO) ..."
5.Graphical Event Models and Causal Event Models - http://www.contrib.andrew.cmu.edu/org/cfe/simplicity-workshop-2014/workshop%20talks/Meek2014.pdf - "...Treat data as a realization of a marked point process: x = t1, l1 , … , tn, ln Forward in time likelihood: p x = i=1 n p(ti , li |hi) where the history hi = hi(x) = t1, l1 , … , ti−1, li−1...Correlation does not imply Causation"

----
763. (THEORY and FEATURE) Non-graph theoretic Intrinsic merit measures of texts, Reputation Rankings, Sybils and Collusions - 16 July 2018, 23 April 2020, 29 June 2020, 30 June 2020 - this section is an extended draft of sections 783,815 and respective topics in NeuronRain AstroInfer design - Intrinsic Merit of texts, Vowelless text compression and Hyphenated Syllable vectorspace of words - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
Most search engines rank websites based on their fame/reputation which is a function of incoming links to a website and reputations of vertices from which the links are incoming e.g PageRank. These Reputations can be manipulated by creating fake incoming links (Sybils) and collusions between websites to inflate PageRank artificially. Identifying Sybils and Collusions is an open problem. Intrinsic fitness/merit of a website is a valuable measure to filter Sybils. It is known from most research papers on Fame Versus Merit that Fame is either linearly or almost-exponentially proportional to the merit of a social/academic

profile vertex in the context of Social networking and Citations in Science publications. If the function relating merit to fame is known approximately by some least squares fit on a training dataset, Fame of a new website can be related to Intrinsic fitness of the website. Huge Distance between observed Fame and Fame predicted by least squares regression from training dataset could be a prima facie indicator of a Sybil. There are quite a few standard non-graph theoretic tools to quantify connectedness of words in text of a website and its narrative style which could confront Fame measures - Coh Metrix, L2 Syntactic Complexity, TAACO, WAT among others. Most of these metrics measure local cohesion(intra-sentence connectivity), global cohesion (inter-sentence connectivity), coherence (mental representation of meaning) quantitatively by correlation between a text and human pre-judged essays. Natural Language Texts of good local coherence and less global cohesion could be simulated by Markov chain models of Information theory which are kind of Turing tests.

References:

763.1 TAACO - https://alsl.gsu.edu/files/2014/03/The-tool-for-the-automatic-analysis-of-text-cohesion-TAACO_Automatic-assessment-of-local-global-and-text-cohesion.pdf - [Scott A. Crossley, Kristopher Kyle, Danielle S. McNamara] - Cohesion features - Connectives,Givenness,Type-Token Ratio(TTR),Lexical overlap,Synonymy overlap,Semantic overlap - Table 1 - Of these Lexical,Synonymy and Semantic overlap are already subsumed by Recursive Gloss Overlap and Recursive Lambda Function Growth (which approximates a natural language text by a Lambda function tree - Turing Machine - thus attaining maximum theoretical limit) TextGraph algorithms for graph complexity merit of text implemented in NeuronRain.
763.2 Manipulability of PageRank under Sybil Strategies - 4.1 - Theorem 2 - http://www.eecs.harvard.edu/cs286r/courses/fall09/papers/friedman2.pdf
763.3 Markov Models of Text Analysis - https://www.stat.purdue.edu/~mdw/CSOI/MarkovLab.html - natural language text could be artificially created by modelling text as probabilities of state transitions between alphabets and words - markov order 1 and order 3 word sequences - manufactured sentence simulates coherence of a human writing but is meaningless - Phonetic Syllable Text (De)Compression models English texts as markov sequences of vowels and consonants - probabilities of vowel succeeding n-grams of consonants are the priors - on the average every second or third letter is a vowel in an English text creating 2-grams and 3-grams of consonants.
763.4 Markov Models of Text Analysis - https://www.cs.princeton.edu/courses/archive/spring05/cos126/assignments/markov.html - an example news article and its Markov text of order 7 (each state is a string of 7 alphabets which depend on previous states)
763.5 Markov Models of Text - [Shannon] - http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

----
Difference between two trees (delta) - 7 August 2018

Two graphs are similar if they are isomorphic (i.e there is a bijection between 2 graphs by vertex renumbering).
Finding difference between two graphs or trees is therefore Graph Non-Isomorphism problem (GNI). Tree difference
is a frequent requirement in source code version control systems and file sync-ing software which transmit delta
(what changed) between source and destination. For example, SVN delta editor in https://subversion.apache.org/docs/api/1.9/svn__delta_8h_source.html overlays the new revision delta on existing tree by replicating only the
changed subtrees while unchanged tree is shared between versions.

Stable Matchings for Dynamic Population - 7 September 2018

[This is mentioned more like an open puzzle/question than answering it]
Stable Marriage Theorem implies there exists an algorithm for finding bipartite matchings between two
sets (bipartite graph) when vertices in both sets have ranking preferences of choosing a match in other set.
Gale-Shapley algorithm finds such an optimal matching between two sets based on preferences in quadratic time.
This algorithm is for static bipartite sets/graphs. Would the same hold for dynamic bipartite graphs in which
either set grows/diminishes over time? A real world example: Population is a bipartite graph of either genders
and stable marriage theorem implies there is always an optimal match between vertices of two genders. This graph grows in time and size of both sets (gender populations) remain equal approximately despite births/deaths (which is a natural mystery implying order emerging from an apparent random process).

References:
----
1. Gale-Shapley Algorithm - Stable Marriage Problem - https://en.wikipedia.org/wiki/Stable_marriage_problem

----
----
764. (THEORY and FEATURE) Gordian Knot and One Way Functions - 15 October 2018, 19 October 2018 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----

Gordian Knot is an impossible-to-unravel knot which was allegedly solved by Alexander the Great by cutting it.
There exists a striking parallel between one way functions for Psuedorandom Generators and difficulty in
untying knots. Gordian knot is open problem in knot theory. One way functions are defined as:

      f(x) = y
      Pr(finverse(y) = x) = 2^(-n) for bit length n of x.

Hardness of inversion and difficulty in untying a knot can be correlated by following contrivance:
Assuming a knot is a map of sequence of points in straightline to sequence of non-linear points in 3 dimensions,
every knot is a polynomial of degree 3 drawing a locus in 3-D plane. A function for this polynomial is the mapping :

      f(x1,x2,x3):<set of straightline points in 3-D plane>-----<knot polynomial configuration in 3-D plane>.

Inverting the previous function implies untying a knot to straightline points:

      finverse(x1,x2,x3):<knot polynomial configuration in 3-D plane>-----<set of straightline points in 3-D plane>.

On the contrary a proof of existence of one way functions implies there exists an impossible-to-unravel knot by previous reduction:

      Pr(finverse(knot polynomial configuration in 3-D plane>) == <set of straightline points in 3-D plane>) is exponentially small.

Infact this definition of One Way Functions is a stronger version of Gordian Knot because inversion should restore the same status-quo-ante straightline configuration of a sequence of points earlier and not some other alignment.

Defining Boolean Gordian Knot One Way Function is not straightforward: For example every point on a string to knot has to be defined as binary inputs to some boolean function which outputs 0 or 1 corresponding to some bit position of a point on the resultant knot polynomial. If there are n points on string and m bit positions each per point, this requires m*n boolean functions of the form f: {0,1}^m-{0,1} all of which have to be inverted to unravel the knot - this is a family of one way boolean functions harder than plain one way boolean function.

References:
----
1. Gordian Knot Simulation - [Keith Devlin] - https://www.theguardian.com/science/2001/sep/13/physicalsciences.highereducation
2. Knot Polynomials - Jones and Alexander - https://en.wikipedia.org/wiki/Knot_theory#Knot_polynomials - Topologically, Knot is an embedding of a circle in R^3 (and also to all the homeomorphisms of the circle obtained by deformations - Knot equivalence - ambient isotopy) - Previous definition of one way function maps a circular or straightline string to one of the homeomorphic knot denoted by a knot polynomial and there are as many one way functions as there are knot polynomials. Inversion of one way function reduces to inverse homeomorphism. Example: Handwritings of different persons (of same language and text) are homeomorphic deformations in R^2 preserving genus (holes or maximum number of cuts required without disconnecting the manifold).
3. Homeomorphic inverse - http://at.yorku.ca/cgi-bin/bbqa?forum=ask_a_topologist_2010&task=show_msg&msg=1138.0001
 - "Thirdly, the inverse of f^{-1} is just f itself - in other words, the inverse of the inverse of f is f itself, so that (f^{-1})^{-1} = f. By assumption, f is continuous, and as f is the inverse of f^{-1}, the function f^{-1} has a continuous inverse."

----
765. (THEORY) Circle Packing and Planarity - 21 March 2019 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
Drawing non-crossing paths between points is a non-trivial planar graph embedding problem. There are planarity criteria defined by various theorems as below:

(*) Wagner's Theorem - Graph is planar if and only if it is free from K5 or K3,3 minors. Kn is a complete graph of n vertices and K3,3 is a complete bipartite graph on 2 sets of size 3. Graph minor is obtained by contracting edges to vertices.

(*) Circle Packing Theorem - Circles of varied sizes are drawn tangentially (osculation) on plane and a graph comprising edges among osculating circles is the coin graph. Graph is planar if and only if it is a circle intersection graph or coin graph.

References:
----
1. Mathematical Puzzles of Sam Lloyd - Selected and Edited by Martin Gardner - Chicken Puzzle - Puzzle 82
2. Circle Packing Theorem - [Koebe-Andreev-Thurston] - https://en.wikipedia.org/wiki/Circle_packing_theorem

----
766. (THEORY) Computational Geometric Factorization, 2-D Cellular Automaton and Multidimensional array slicing - 12 November 2019, 13 November 2019, 21 April 2020 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----

Naive 2-dimensional (multidimensional) array slicing loops through rows and finds per row slice which are united to a square slice. In Python SciPy and NumPy have a multidimensional subscript slicing facility e.g x[2:10,2:10] for an ndarray object x extracts a square slice of 9 * 9. Naive loop slicing is O(n^d) for d dimensional arrays. Low level languages C,C++ etc., store arrays in contiguous memory locations in flat 1-dimension and 2-d slice of (a,b) is expressed by an equation r*(x+a) + y (r <= b, y <= a) which uniquely identifies an element in square slice of origin (x,y) obviating loops. But extracting the square slice still needs looping. Doing better than naive bound necessitates storing the two dimensional array in a wavelet tree and computational geometric range search on it:

(*) Wavelet Tree for Computational Geometric Planar Range Search in 2 dimensions - https://www.researchgate.net/profile/Christos_Makris2/publication/266871959_Wavelet_trees_A_survey/links/5729c5f708ae057b0a05a885/Wavelet-trees-A-survey.pdf?origin=publication_detail - "... Therefore, consider a set of points in the xy-plane with the x- and y-coordinates taking values in {1,…,n}; assume without loss of generality that no two points share the same x- and y-coordinates, and that each value in {1,…,n} appears as a x- and y- coordinate. We need a data structure in order to count the points that are in a range [lx, rx] × [by, uy] in time O(logn), and permits the retrieval of each of these points in O(logn) time. ... this structure is essentially equivalent to the wavelet tree. The structure is a perfect binary tree, according to the x–coordinates of the search points, with each node of the tree storing its corresponding set of points ordered according to the y-coordinate. In this way the tree mimics the distinct phases of a mergesort procedure that sorts the points according to the y-coordinate, assuming that the initial order was given by the x-coordinate. ..."

(*) Entropy bound for Wavelet Tree point grids - Lemma 2 - https://www.sciencedirect.com/science/article/pii/S0925772113000953 - [Arash Farzan, Travis Gagie, Gonzalo Navarro] - set of all possible point grids (slices) of size m carved from n * n square are populated in a wavelet tree and size of this set is {n^2Cm} and of entropy log{n^2Cm} - "...Furthermore, query rel_acc(i1,i2,j1,j2) (giving all the k points in [i1,i2]×[j1,j2]), is answered in time O((k+1)lgσ/lglgn)...". Query rel_acc() range reports all k points in the rectangular slice [i1,i2] * [j1,j2] of n*n square for an alphabet size (which could be 2 or 10 depending on binary or decimal radix of the 2-dimensional array) in O((n^2+1)lg2/lglgn) and O((n^2+1)lg10/lglgn) for k=O(n^2). This is slightly better than O(n^2) naive bound because O((n^2+1)lg2/lglgn) and O((n^2+1)lg10/lglgn) = O(n^2/lglgn) < O(n^2). 2-dimensional arrays are labelled points on 2-d plane and computational geometric wavelet tree planar range search selects an array slice in its entirety in time O(n^2/lglgN).

Previous improvement in 2-dimensional array slicing is quite useful in speedup of delta vicinity search for exact factors around approximate factors in both Randomized NC (Section 752) and Exact NC-PRAM-BSP Computational Geometric Factorization algorithms. Once the ray shooting queries find the approximate factors on the hyperbolic arc bow, square vicinity (in contrast to circular radius) of approximate factor can be retrieved in subquadratic time. Every square vicinity of approximate factor found by ray query induces a 2-dimensional cellular automaton centered at approximate factor which sweeps the plane in 8 directions and locates the exact factor in consecutive generations by growth rules.

----

767. (THEORY and FEATURE) Leaky Bucket Algorithm and Time Series Analysis - 30 November 2019 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

----

NeuronRain AstroInfer Research Version in SourceForge implements lot of algorithms to mine patterns in strings especially encoded astronomical datasets of celestial configurations which might be helpful to correlate gravitational influence of an n-body planetary system on sky and terrestrial weather-seismic events (example Sequence Mined astronomical pattern based on swiss ephemeris and maitreya8t - commit - https://sourceforge.net/p/asfer/code/2606/) . NOAA JAWF Climate Prediction Centre precipitation analytics based on time series and leaky bucket are alternative examples of machine learning driven weather forecasts (Leaky Bucket Model - https://www.cpc.ncep.noaa.gov/products/JAWF_Monitoring/India/monthly_maps.shtml, Rainfall Time Series - https://www.cpc.ncep.noaa.gov/products/JAWF_Monitoring/India/30d_time_series.shtml).

Leaky Bucket Algorithm is primarily used in traffic policing and scheduling of networks (in NOAA example previously, daily rainfall statistics replace network traffic and are plotted as timeseries) and operates as follows:

(*) Bucket datastructure (e.g store-forward buffers in routers) is predefined for certain average expected traffic

(*) Network Traffic trickles in a leaky bucket datastructure at random rate

(*) Some amount of traffic leaks out of bucket at random rate

(*) Bucket might either overflow or be emptied depending on rate of incoming network packets (similar to detour of vehicles in a jammed junction)

(*) Bandwidth rate limiting and outlier detection can be enforced based on leaky bucket model (e.g any overflow is caused by outlier packets and indicates abnormal exorbitant incoming traffic)

(*) Bursts of packets in network traffic can also be plotted as time series of periodic intervals (number of packets versus time)

(*) Time-Series of network traffic and Leaky-Bucket model often have an one-to-one correspondence - Any peak (outlier) in timeseries might trigger a bucket overflow and vice-versa.

References:

767.1 Visual and Audio Data Mining - Section 11.3.3 - Page 670 - Data Mining- [Jiawei Han-Micheline Kamber] - Visual Data mining of Rainfalls in SAS Enterprise Miner and Mining data as music or audio signals

----

768. (THEORY) Finding penultimate element in a linked list, sublinear Depth First Search and Breadth First Search, Survival Index Timeout WCET EDF OS Scheduler - 3 January 2020 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

----

Finding the last but one element in a singly linked list of N elements requires linear traversal of the list, pushing the elements to stack and popping top two elements from it by a naive O(N) algorithm. Breadth First and Depth First Searches are cornerstones of Artificial Intelligence having vast literature in motion planning and robotics. Because singly linked list is a directed acyclic line graph parallel versions of traditional O(V+E) breadth first search and depth first search algorithms which are sub-linear and logarithmic can locate the penultimate element in a singly linked list in parallel faster. There are many parallel BFS and DFS algorithms e.g iterative deepening A* (IDA*), parallel shortest path, Depth First Branch and Bound which are based on PRAMs and thus are polylogdepth NC circuits. Parallel DFS and BFS are quite useful in Survival Index Timeout OS EDF Scheduler algorithm described earlier where every per WCET timeout bucket is a linked list of process id(s) and a process id needs to be searched.

References:

1.Parallel DFS - [Nageshwara Rao - Vipin Kumar] - https://www.lrde.epita.fr/~bleton/doc/parallel-depth-first-search.pdf
2.Parallel DFS - Chapter 11 - Introduction to Parallel Computing - http://parallelcomp.uw.hu/ch11lev1sec4.html
3.Parallel DFS - http://www2.inf.uos.de/papers_html/zeus_95/node5.html
4.Parallel RAM Breadth First Search - https://en.wikipedia.org/wiki/Parallel_breadth-first_search
5.Parallel Breadth First Search and Depth First Search - [Taenam Kim & Kyungyong Chwa] - https://www.tandfonline.com/doi/abs/10.1080/00207168608803503?journalCode=gcom20 - is of parallel time O(log d - log n) for diameter d (longest of shortest paths between all pairs of vertices) of the graph and n is the number of vertices - "...we develop a parallel breadth first search algorithm for general graphs and a parallel depth first search algorithm for acyclic digraphs which run in time O(logd-logn) using 0(n^2[n/log n]) processors....". For singly linked lists diameter d = number of vertices n

----
838. (THEORY) 12 May 2020 - Finding number of elements in a linked list - Sequential and Parallel - List Ranking - Pointer Jumping - related to 751,768

Counting number of elements in a linked list of N elements has a naive sequential bound of O(N). List
Ranking is the problem of computing distance of every element in a linked list from the end of the list.
Thus counting number of elements in a linked list is a List Ranking problem for finding distance of
the first element from end of the list. List Ranking has a Parallel RAM pointer jumping algorithm which
finds the rank in O(logN) parallel time. Following is a pointer jumping pseudocode for parallel
list ranking:

```
            allocate one element per processor
            for every processor and element i
                  distance[i] += distance[next[i]]
                  next[i] = next[next[i]]
```

Sequential sublinear algorithm for counting number of elements in a linked list and list ranking is an
open problem. Every linked list is an inorder traversal of a balanced AVL tree equipped with successor
(and predecessor in doubly linked list) pointers - this structure can be exploited to approximately
estimate number of elements in a linked list as 2^(average length of root to leaves). Parallel List
Ranking by Pointer Jumping is central to many parallel algorithms for linked lists. Primitive next[i]
is architecture dependent and if its recursive version next(next(next....))) could be upperbounded by
(log M)^k for M >> N, N can be written as N = a*N/(log M)^k + b, a <= (log M)^k. Huge linked lists could
be approximately estimated by this heuristic for sequential pointer jumping a times plus additional b
sequential traversals.

References:
----
1. Algorithms - [Cormen-Leiserson-Rivest-Stein] - Page 692 - Algorithms for Parallel Computers - 30.1.1
- List ranking
###

##
Course Authored By:

Srinivasan Kannan
(also known as: Shrinivaasan Kannan, Shrinivas Kannan)
Ph: 9791499106, 9003082186
Krishna iResearch Open Source Products Profiles:
http://sourceforge.net/users/ka_shrinivaasan,
https://github.com/shrinivaasanka,
https://www.openhub.net/accounts/ka_shrinivaasan
Personal website(research): https://sites.google.com/site/kuja27/
emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com, kashrinivaasan@live.com
----
##

This is a non-linearly organized, code puzzles oriented, continually updated set of course notes on AngularJS. This
complements NeuronRain course materials on Linux Kernel, Cloud, BigData Analytics and Machine Learning.
----

2 June 2017

AngularJS is the client side Advanced Java Scripting (AJAX) engine runtime supported by most browsers. AngularJS follows Model-View-Controller pattern:
- View is the HTML visible to user
- Model is the form layout and frames of the Dynamic HTML page
- Controller acts on the model data and eschews HTML output from Server Side JavaScript (Node.js) or other frameworks(Django/Flask/Tornado)

Model-View-Controller decouples the HTML rendering from the hidden dynamic logic. NeuronRain GUI frontend has two client/server interfaces: Plain HTML-to-Tornado and AngularJS-to-Tornado webservers (https://github.com/shrinivaasanka/asfer-github-code/tree/master/python-src/webserver_rest_ui/). AngularJS HTML template for a simple form submit has following structure:
- form tag with ng-app directive defining the application name of the controller
- Model name for input elements in the form
- script tag which defines a controller object from ng-app and defines a function for form processing invoked from submit widget of the form
- AngularJS script in controller has access to builtin(s) like $scope, $http etc., for accessing model data in the scope. Form data can be submitted to a REST URL with GET or POST with stringified JSON data.

##

##
##
Course Authored By:
##
##
K.Srinivasan
Personal website(research): https://sites.google.com/site/kuja27/
NeuronRain Documentation,FAQ,Licensing: http://neuronrain-documentation.readthedocs.io/
##
#########################
This is a non-linearly organized, code puzzles oriented, continually updated set of course notes on Cloud computing frameworks and BigData analysis.

----

7 Februrary 2017

Apache Spark is a Cloud computing software for processing bigdata. It is based on concept of Resilient Distributed Datasets (RDD) which are partitions of a dataset executed in parallel. Spark is divided into 2 components: 1) Driver and 2) Executor. Driver splits the dataset into RDDs and allocates each RDD to an executor in parallel. Parallelization can be in two ways: 1) For objects like lists,arrays etc., 2) For data in HDFS,cassandra, S3 etc.,

While executing in parallel, there is a necessity to share mutable state across executors. This is done in two ways: Broadcast variables and Accumulators (only increment is allowed). Spark streaming is a feature that allows realtime processing of streaming data by an abstraction of Discretized Streams or DStreams. Following code in neuronrain asfer receives generic data from any URL, does an ETL on it and stores in RDDs:
https://github.com/shrinivaasanka/asfer-github-code/blob/master/java-src/bigdata_analytics/spark_streaming/SparkGenericStreaming.java

There are 2 operations performed on RDDs: 1) Transformations - create a new set or subset of RDDs 2) Actions - do some iteration on transformed RDDs. Spark streaming allows custom streaming by implementing/overriding receive() method in Receiver interface. Receiver can be started and stopped by onStart() and onStop() methods. Receive method is overridden with customized code to access a remote URL, fetch HTML, parse it and do any ETL operation as deemed fit. From Spark 2.0.0 , support for lambda functions (new feature in Java 8) instead of *.function.* (Function objects) for RDD transformations has been included. Previous Spark Streaming code demonstrates these features and uses Jsoup GET RESTful API for ETL/scraping of remote URL data.

20 February 2017

Spark SQL + Hive (Shark) provides synergy of bigdata processing with an SQL storage backend. Hive is implemented on top of Thrift RPC protocol which is modern version of Interface Definition Language based Web Service Architectures like CORBA, Google Protocol buffers , SOAP etc., Streamed data received is an iterable (e.g lines in SparkGenericStreaming.java implementation in https://github.com/shrinivaasanka/asfer-github-code/blob/master/java-src/bigdata_analytics/spark_streaming/SparkGenericStreaming.java) which is further transformed with map/filter operations quite similar to Java Streams. Java Streams work on similar concept of creating a stream from iterable (arrays, lists etc.,) and applying map/filter transformations. Spark's saveAsTable() saves the streaming data into a hive table in Spark Metastore or Hive metastore (this requires hive-site.xml in Spark conf directory). (MAC currency in AsFer+KingCobra electronic money cloud perfect forwarding move is implemented on Protocol Buffers.)

18 January 2018

Spark cloud processing framework has support for global variables in two flavours: 1) Accumulators and 2) Broadcast variables. Both of these are mechanisms to reflect global state across Resilient Distributed Data Set nodes in Spark clusters. Accumulators have a single operation add() which incrementally adds a value on a local RDD to the global accumulator variable and is reflected across all nodes in Spark cluster. Both Accumulators and Broadcast variables are instantiated from Spark Context. Broadcast variables are plain read-only global variables which are broadcast as the name suggests to all RDDs in Spark cluster. An example code and logs for how accumulators and broadcast work has been demonstrated in code/Spark_Broadcast_Accumulator.py and code/testlogs/Spark_Broadcast_Accumulator.log.18January2018. Accumulator constructor can be optionally passed on an object of type AccumulatorParam (or its subclassed types which override add()). Presently accumulators and broadcasts are only way to provide global state across nodes in Spark cluster.

----
4 October 2018 - Representational State Transfer - CRUD - RESTful and WebServices in Cloud
----
Traditional Client-Server Architecture in Distributed Computing involves client making a socket connection to
a listener server, completing a handshake and establishing a two-way message transport.Over the years, with the
advent of cloud, every application on web is deemed to be a finite state automaton of 4 states - Create-Read-Update-Delete (CRUD) respective HTTP primitives being PUT,GET,POST,DELETE which create a resource in server, update it, read it and delete. Every resource is identified by a URL or WebService. Though this indirectly wraps the underlying socket communication, benefit is in statelessness of each request - every request is independent of previous request and state is remembered only in client side and server is state oblivious. Nomenclature RESTful stems from the state getting transferred from client to server for every HTTP request and responded in JSON objects. An example of RESTful API is Facebook Graph API SDK for retrieving user profile information, connections, comments, likes etc., A REST Python Client which GETs/PUTs objects to Facebook wall has been described in code/GRAFIT_automatic_wallposter.py. It internally issues HTTP requests to Graph API REST endpoints. Invocation to put_object() has been commented. This is an updated version of https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/Streaming_FacebookData.py specific to GRAFIT (for Grafit Open Learning facebook profile https://www.facebook.com/shrinivaasan.ka which imports @NeuronRain_Comm - https://twitter.com/neuronrain_comm - automatic commit tweets). RESTful implies every cloud distributed computation is a string from alphabets {GET, POST, PUT, DELETE} amongst the nodes in cloud URL graph.

----
5 December 2018 - Apache 2 Web Server, Apache HTTPD modules, Configs and Hooks, Application Servers - example
----
Apache webserver provides facilities to plugin user developed modules. One specific standard example is mod_ssl which is an SSL plugin module for Apache webserver. An example Apache module implementation and related forum Q&A have been cited in the references. This implementation was a part of Sun Microsystems/Oracle iPlanet Application Server which had a 3-tier J2EE compliant middleware architecture (Clients <-> WebServer <-> ApplicationServer). Application Servers are ancestors of present day cloud implementations which are Service Oriented Architectures. HTTP requests are served by Apache Webserver and responded. Necessity for an Apache module arises when Apache webserver is used as a loadbalancer for a cluster of application servers and requests have to be routed to it. Example module snippet in the reference defines config parameters

for Apache webserver - the loadbalancer XML file containing details about cluster of iPlanet Application Servers and Locale. The register_hooks() function has callback functions for init, name translation (e.g. URL rewriting for session ids and cookies), and handle requests (e.g routing requests to another application server). These config parameters are set by invoking ap_set_string_slot() functions denoted by assignment to take1 structure member (function takes 1 argument)

References:

1.Apache Modules Mailing list - Apache 1.3.27 and iPlanet Application Server - https://marc.info/?l=apache-modules&m=105610024116012 (Copyright: Sun Microsystems/Oracle)
2.Apache Modules Mailing list - Apache 2.0.47 and iPlanet Application Server - https://marc.info/?l=apache-modules&m=106267009905554 (Copyright: Sun Microsystems/Oracle)
3.Apache Config Directives - ApacheCon - http://events17.linuxfoundation.org/sites/events/files/slides/ConfigurationDirectiveAPI.pdf

----
752. (THEORY and FEATURE) 2 January 2019, 9 January 2019, 8 January 2020, 22 January 2020, 23 January 2020, 29 January 2020, 31 January 2020, 2 February 2020,4 May 2020,12 May 2020,15 May 2020, 2,3,4,7 July 2020,28 October 2020,3 November 2020,2 December 2020,5 December 2020,1 January 2021,29 January 2021,1 February 2021,5 March 2021,1 May 2021,8 June 2021 - Searching, Indexing, Computational Geometric point location queries, Factorization in Randomized NC - Binary Search Nuances, B-Trees, Sharding - related to 227,666,730,762,828,831,864,870,1123 and all sections of People Analytics,Histogram Analytics,Social Network Analysis,Majority Voting,Syllable vectorspace text analytics,Space filling,Cellular automata,Graphical Event Models and Causal Event Models (GEM and CEM) and Computational Geometric Planar Point Location Factorization in NeuronRain Theory Drafts
----
Traditional Binary Search still remains the standard for sifting huge datasets and has undergone significant
refinements. Usual Binary Search is centred around midpoint computation and branching:

```
      midpoint = (left + right)/2
      if query > midpoint:
            search interval [midpoint, right]
      else:
            search interval [left, midpoint]
```

Bug in Midpoint computation by previous averaging in some earlier versions of languages like Java caused an overflow error in 32-bit architectures which necessitated replacing it by >>> operator (unsigned right shift):

```
      midpoint = (right-left) >>> 1;
```

B-Trees are generalized Binary Search Trees in which each internal and root nodes contain more than one element and every element of a node acts as a separator for values of its subtrees as in example:

```
                        2,10
                        |
                  1 ----- 3,9 ---- 11,12
                        |
                        4,5
```

B+-Trees extend B-Trees by additional linked-list of leaf nodes.

Indexing Bigdata e.g web pages involves storing them as huge set of key-value pairs of the form:

```
            word  --- (document1, location1), (document2, location2) ...
```

which is termed as posting in inverted indices. Size of an Index in search engine could be unimaginably large requiring partitions into set of rows located across geographically distant servers on cloud. This is horizontal partition based on rows as against vertical partitions of columns in DBMS Normalizations. Each partition is named a Shard.

Fundamental question is: can items in a list be found efficiently by a search graph instead of binary search trees. Lower bound for search is Omega(logN). Binary Search Trees can be alternatively defined as recursive bipartite graph which is an indefinite recursive fractal bipartition of sets - Example:- Set of integers [2,4,5,1,3,7,9,10,8,6] are represented recursively as bipartite graph by assuming an initial midpoint pivot separator of each subset similar to quicksort which partitions a set into two halves of elements > pivot and elements < pivot in each depth of recursion:

```
      pivot 6
      [2,3,5,1,4] ----- [7,9,10,8,6]
      pivot 3                pivot 8
      [[4,5,3]--[2,1]] ---- [[6,8,7]--[10,9]]
      pivot 4                pivot 7
      [[[4,3]-[5]]--[2,1]] ---- [[[7,8]-[6]]---[[10]-[9]]]
      pivot 4          pivot 2                 pivot 7
      [[[[4]-[3]]-[5]]--[[2]-[1]]] ---- [[[[7]-[8]]-[6]]---[[10]-[9]]]
```

Edges are denoted by ----- which connect each parenthesized subset vertex for a subtree obtained by traversing the binary search tree. Last line encodes a recursive bipartite search graph which is the top view of the binary search tree and is a multidimensional tensor - a kind of binary space partition. Searching this tensor locates a point in the nested parentheses by comparing against pivots. Advantage of recursive bipartite graph representation of binary search tree is each subtree can be retrieved by array indexing.

Planar Point Location in Computational Geometry is the most fundamental generalization of conventional one dimensional list search to arbitrary dimensions. Computational Geometric Factorization algorithm implemented in NeuronRain AstroInfer is a two dimensional parallel geometric search which locates , in polylogarithmic time, factor points with in rectangular faces of planar straight line graph (PSLG) formed by rasterizing (pixel polygon approximation of an algebraic curve on a grid) hyperbolic arc bow on Parallel RAMs - this search involves two phases:

(*) Locating a rectangle in PSLG polygon subdivision containing factor point in polylogarithmic time by some Parallel RAM Planar Point Location algorithm - PSLG has ~(3N+1) vertices and ~4N edges and each polygonal face is a pixel array rectangle of dimensions 1 * N/[x(x-1)]

(*) Binary search the rectangle containing factor point - every rectangle in PSLG of rasterized hyperbolic arc is an arithmetic progression of pixels (ordinate products)

Hardness of Factorization has bearing on susceptibility of cryptographically secure Majority Voting to manipulation and its credibility.

```
--------------------------------------------------------------------------------
-
An example hyperbolic arc rasterization - schematic - pixel array polygonal
faces
(arithmetic progression rectangles marked f denote faces containing factor
points):
--------------------------------------------------------------------------------
-

      ########
            ###f####
                  ###############
                               #########
                                    ###########f###
```

Such a Planar Point Location can be extended to arbitrary dimensions > 2 and any other algebraic curve - E.g. algebraic curve uvwxyz=N is a 6-dimensional hyperbola and requires rasterization in 6-dimensional hyperplane to locate factor point (u,v,w,x,y,z). Multidimensional Point Location factorization is the problem of Factorisatio Numerorum (multiplicative partition).

Randomized NC Geometric Search of a 2-dimensional grid and Linear Program Formulation
----
A Grid filling algorithm which maximizes linear program of sum of binary values of ordinate points in Randomized NC has been described in https://sites.google.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf and its Cellular Automaton, Apollonian Gasket-Circle Packing and Factorization-based Set-Partition-to-Lagrangian-Tile-Cover versions have been drafted in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt . Grid Filling as constraint satisfaction problem which maximizes the number of points hit on a 2-dimensional space is a planar point location geometric search if BigData is visualized as values of ordinates within a 2-dimensional rectangle and a value for a query point has to be found by a parallel pseudorandom generator which simultaneously churns out multiple ordinate points (xi,yi) or circles of small radii centred around random ordinate points within the rectangle (Monte Carlo Simulation). This Randomized NC pseudorandom generator plane sweep covers most of the 2-dimensional rectangle with high probability. Advantage of RNC Grid filling: if query points constitute a meagre percentage of total 2-dimensional rectangular space e.g polynomial curve points, Randomized NC grid filling is able to find query point with high probability in polylogdepth NC without sorting and binary search. It is noteworthy that planar point location for Integer Factorization in exact NC can also be formulated as less stringent Randomized NC Grid filling which locates a factor on hyperbolic curve with high probability. Chaos theoretic RNC parallel pseudorandom generator defined in https://sites.google.com/site/kuja27/ChaoticPRG.pdf could output pseudorandom ordinate bits in parallel.

Probability of hitting a query point within time t is a Bernoulli trial: Probability of hitting a query point within a rectangle = number of query points or length of query polynomial curve embedded in 2-dimensional plane (p) / Area of rectangle of sides a and b (ab) = p/ab

If in time t there are m trials, probability of hitting only query points by pseudorandom sampling = (p/ab)^m, m trials are parallelized by Parallel Chaotic Pseudorandom Bits Generator which generates m 2-dimensional ordinate points simultaneously in Randomized NC.

Previous linear program for maximizing sum of (values are 0 or 1, pseudorandomly flipped) q ordinates x1 + x2 + x3 + ... + xq is exactly the sum of binary random variables Xi: X1 + X2 + ... + Xq. Berry-Esseen Central Limit Theorem - https://en.wikipedia.org/wiki/Berry%E2%80%93Esseen_theorem - implies normalized sum (mean) of random variables tends to a Normal distribution bell curve. Excluding tails of Normal distribution area of bell curve integral(f(x)*dx) from -left_tail to +right_tail is the approximate expected sum of ordinates randomly set to 0 or 1: x1 + x2 + x3 + ... + xq

Probability of hitting a query point after m-th trial (success after m-1 trials):

    (1-[p/ab])^(m-1)[p/ab]

In the context of factorization, geometric search reduces to finding factor points on a hyperbolic arc bow embedded in 2-dimensional rectangular space.

Though less accurate compared to Exact NC-PRAM-BSP computational geometric planar point location, success amplification of RNC pseudorandom factor point location can be achieved by narrowing down the domain of pseudorandom search to a delta-rectangular strip vicinity in lower halfspace created by leading diagonal of the rectangle:

Length of the leading diagonal = sqrt(a^2 + b^2)

Area of delta-vicinity rectangular strip beneath the leading diagonal which contains the hyperbolic arc = delta * sqrt(a^2 + b^2)

Side of the rectangular strip - delta - is the distance between tangent of hyperbolic arc at (sqrt(N), sqrt(N)) and line connecting (1,N) and (N,1) which is a very small fraction of area of rectangle a*b.

Probability of hitting a query point within the rectangular strip =
number of query points or length of query polynomial curve embedded in 2-dimensional plane / Area of rectangular strip = p/[delta * sqrt(a^2 + b^2)] >> p/[ab]

Probability of hitting a query point after m-th trial (success after m-1 trials):

(1-[p/(delta * sqrt(a^2 + b^2))])^(m-1)[p/(delta * sqrt(a^2 + b^2))]

which amplifies success probability of finding a factor point.

For finding prime factors, number of query points p = kloglogN (Hardy-Ramanujan Theorem) and success probability of finding a factor point is substituted for p:

(1-[kloglogN/(delta * sqrt(a^2 + b^2))])^(m-1)[kloglogN/(delta * sqrt(a^2 + b^2))]

For a square embedding of hyperbolic arc a=b and success probability amplifies to (delta is configurable):

(1-[kloglogN/(delta * N * sqrt(2))])^(m-1)[kloglogN/(delta * N * sqrt(2))]

Neglecting delta vicinity, obvious randomization is to pseudorandomly choose an x-axis ordinate x between 1 and N and compute N/x which is a random ordered pair (x,N/x). For Randomized NC, x-axis is divided to N/(logN)^k intervals of width (logN)^k each. Each interval is assigned to a PRAM. Thus N/(logN)^k pseudorandom ordinate ordered pairs are simultaneously generated by a Chaotic Parallel PRG on N/(logN)^k PRAMs which is looped for (logN)^k iterations making it an RNC factorization.

NeuronRain AstroInfer implements a sequential Chaotic PRG based on 1-dimensional binary Cellular Automata which grow chaotically by growth rule fraction lambda, Undecidable Mandelbrot sets, Logistic and Lehmer-Palmore chaotic PRGs. Spread of memes in Social Networks, pandemics, cybercrimes et al all have common Chaos,Game theory and 2-Dimensional Cellular Automata underpinnings (FTrace analyzer in USBmd is a game theoretic botnet defense model and a design of mechanism to counteract - antigame - as well) - For example Verhulst-Pearl-Reed Logistic Law and its Biological variant (lambda*x(1-x) and N0*exp(lambda(1-Nt))) analysis of CoronaVirus2019 pandemic reveals Bifurcation parameter lambda hovering approximately at 4.829487535509905... > 3.57 when chaos sets for a very small initial condition which implies emergence of order in Chaotic spread explained by Period Doubling ratio limit of Bifurcation parameter [lambda(n-1) - lambda(n-2)] / [lambda(n) - lambda(n-1)] - Feigenbaum universality constant 1 = 4.669201609... (https://en.wikipedia.org/wiki/Feigenbaum_constants).

By pandemic cellular automaton model, spread of memes in Social networks can be formulated as RNC space filling problem - social network vertices spreading memes are parallelly chosen random points (people) on a rectangular space effecting a 2-dimensional cellular automaton of 8-neighbours (influenced by meme) per vertex. There are linear stretch PRGs (https://dl.acm.org/doi/10.1007/s00037-007-0237-6 - [Applebaum-Ishai-Kushilevitz]) subject to assumptions in NC0 from which ordinate points can be computed in NC by splitting bit sequence (klogN bit pseudorandom string in NC is split into k (x,y) ordinates of length (logN/2, logN/2) each). As memes automata evenly cover the surface there is an overlap of neighbours adjoining all or some random points. By increment growth rule, overlapped neighbour vertices are incremented implying a high value vertex (point on grid) has high value neighbours and vice versa simulating any parallel random process (this is in discrete contrast to the continuous plane sweep by circles of small radii mentioned earlier). Thus a cellular automaton leads to a random cellular automaton planar graph (CAGraph):

(*) Vertices are randomly chosen and labelled by instantaneous values of points on cellular automaton grid.

(*) New edges are created when a point and its neighbours have non-zero value after some generations of increment growth rule

(*) Decrement growth rule would conversely obviate edges amongst zero points and neighbours thus simulating Susceptible-Infected-Recovered Erdos-Renyi SIR random graph model.

(*) Between every non-zero valued point vertex and its non-zero neighbours (maximum degree = 8)
on grid there is an edge

(*) Clustering traits of previous CAGraph determine diffusion of a concept in community - strongly connected components of high value vertices are the most influenced giants.

References:

1.The Art of Computer Programming - Volume 3 - [Donald Knuth] - Sorting and Searching - 6.2.2
2.Beautiful Code - Finding Things - [Tim Bray - Sun Microsystems]
3.Google AI Blog - [Joshua Bloch] - https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html
4.Topological Sorting - Fast Parallel Algorithms - [SA Cook] - https://www.sciencedirect.com/science/article/pii/S0019995885800413 - If set of integers are randomly assigned as labels of a random graph, topological sorting of vertices by some ordering can be done efficiently in NC^2 and a following binary search can find the element in additional logarithmic time. But the topologically sorted vertex list may not be sorted by total ordering always.Dekel-Nassimi-Sahni algorithm works by Repeated Matrix Multiplication logarithmically many times and sorting by longest paths.
5.Planar Separator Theorem for Graphs - [Richard J. Lipton and Robert Endre Tarjan] - https://epubs.siam.org/doi/abs/10.1137/0209046 - PSLG from rasterized hyperbolic arc xy=N can be partitioned to almost equal subgraphs by removing O(sqrt(3N+1)) vertices - quite useful in parallelizing and fair load balancing the rasterized polygon faces amongst PRAMs for point location queries

----

1144. (THEORY and FEATURE) Computational Geometric point location queries, Factorization in Randomized NC, SETH, String complexity and distance measures, Causal and Graphical Event Models, Space filling and Cellular automata - related to 752 and all sections of People Analytics,Histogram Analytics,Social Network Analysis,Majority Voting,Syllable vectorspace text analytics,Graphical Event Models and Causal Event Models (GEM and CEM) and Computational Geometric Planar Point Location Factorization in NeuronRain Theory Drafts - 8 June 2021

Causal Event Models and Graphical Event Models (CEM and GEM) could be useful for Novel SARS CoronaVirus 2 2019 timeseries analysis. NeuronRain COVID19 analyzer is Python and R based. NeuronRain implements an EventNet GEM based on Recursive Lambda Function Growth Textgraph algorithm. CEM and GEM Integrated Crisis Early Warning System for COVID19 could hypothetically mine news and research articles on COVID19 by Recursive Lambda Function Growth EventNet GEM and grow textgraph from articles which unravels hidden cause-effects of actors and events(on how the keywords in the article are connected) - a random walk on the textgraph implies causality. NeuronRain Cellular Automaton Space Filling ER-SIR graph (CAGraph) has average degree (which depends also on population density) of

multiples of 8:
Per day Spread = 8 * w1 * I(nfected) - 1 * w2 * R(ecovered) - 1 * w3 * S(usceptible) + 1 * w4 * recovery_time ... + constant
COVID19 ER-SIR graph empirical data show R-number of 406 for 30 days (average degree > 400 - https://www.hindustantimes.com/india-news/1-covid-patient-can-infect-406-if-physical-distancing-measures-are-not-followed-101619441843605.html) which makes it a potential expander graph. Random walk on COVID19 CAGraph (ER-SIR random graph) simulates transmission. Infection CAGraph is incidentally a Dynamic Graphical Event Model (GEM) because actors (S,I and R) and causations (infections) are dynamically created and removed - a timeseries of ordered pairs (actor_infected,time) if viewed as one dimensional stream of data wherein some ordered pair in stream is caused (infected) by some ordered pair in the past.

Chaotic universality and Ramsey theoretic emergence of monochromatic arithmetic progressions in pseudorandomly colored sequences independently certify "Orderly Disorder" paradox. Recent Chaos Machine PRG by [Maciej A. Czyzewski] has a minimal example implementation of a Push-Hybrid-Pull Chaos Machine PRG computed by non-linear Chaos maps (Butterfly effect, Logistic map, Gingerbreadman map - Chaos Machine source code - https://eprint.iacr.org/2016/468.pdf):

Number of prime factors in [1,N] interval = aloglogN for constant a
Number of prime factors in unit interval = aloglogN/N

Number of prime factors in each of N/(logN)^k intervals of width (logN)^k = [aloglogN/N]*[(logN)^k]

Probability of chaotic pseudorandom choice of a prime factor within each (logN)^k width interval = [aloglogN/N]*(logN)^k/(logN)^k = aloglogN/N = Probability of parallelly locating a prime factor per interval in each Bernoulli trial.

Success amplification - probability of locating a prime factor in each of the (logN)^k Bernoulli trial iterations (Randomized NC) in parallel = (aloglogN/N)^(logN)^k which is exponentially small. Suitable choice of constant a and depth k of RNC circuit (number of trials) bounds the error and thus in BPNC.

Geometric search can be generic to any dataset beyond numeric including strings and texts. Words in text documents can be embedded in an m-dimensional vector space A^m for alphabet A (or language and script independent syllable space A^m if texts are syllable hyphenated) and length of longest word m which is a word embedding kernel for texts. E.g strings "word1" and "word2" in text are embedded as vectors [w,o,r,d,1,...] and [w,o,r,d,2,...] of m dimensions:

Size of the alphabet A (or syllables) = |A|

Length of the longest word in text (or longest syllable hyphenation) = m

Previous kind of alphabet-syllable word embedding of text in a space of A^m clusters similar words by usual numeric euclidean distance measures as opposed to levenshtein edit distance. Traditional Latent Semantic Indexing Singular Value Decomposes a term-document matrix and computes low rank approximation for document similarity. Similarly LSI and SVD could be computed for previous word embedding of text which is an N*m matrix of word_id-alphabets for a text of N words of maximum length m.

NeuronRain AstroInfer has a polynomial encoding implementation of texts which curve fits a text to a polynomial in R^2 [points on polynomial are ordered pairs (alphabet_location,alphabet_unicode)] and PRP error correcting polynomial implementations. Multiple encoded polynomials of texts can be embedded on a 2-D plane and planar point location could be underneath rank() and select() queries on strings. Similarly, People analytics involve intrinsic merit vectors of people e.g [academic credentials, work experience,...] for people clustering.

Finding closest pair of points in a set of points on a vectorspace has quite a few practical applications in sea and air traffic control and vehicle collision avoidance. Replacing the vehicle points by textual strings on a vectorspace (mentioned earlier) is helpful in finding closest pair of strings within a set of strings. Subquadratic closest pair of points detection is a geometric search problem and distance similarities amongst string datapoints could be ranked (by iteratively removing every closest point to a pivot point and recursively recomputing closest pair algorithm). Subquadratic string distance algorithms are important from complexity theoretic standpoint - Edit distance is a standard measure for string similarity (number of additions and deletions of alphabets necessary to morph one string to another) and subquadratic algorithm for edit distance might nullify SETH. If edit distance is expressible in terms of computational geometric algorithm for closest pair of string points, edit distance could be computed in subquadratic time.

Prerequisite for finding Closest pair of string vehicle datapoints by computational geometric divide and conquer earlier is: Number of strings (N) must equal Length of each string(n) (to attain O(nlogn) edit distance subquadratic bound).

Example:
----
string1 - abcde - embedded as point [a,b,c,d,e]
string2 - fghij - embedded as point [f,g,h,i,j]
string3 - klmno - embedded as point [k,l,m,n,o]
string4 - pqrst - embedded as point [p,q,r,s,t]
string5 - uvwxy - embedded as point [u,v,w,x,y]

Each alphabet is encoded as scalar from 1 to 26 and string vectors are embedded on 26^n vectorspace (similarly for ascii or unicode alphanumeric strings, vectorspace is 256^n or 512^n):
string1 - [a,b,c,d,e] - [1,2,3,4,5]
string2 - [f,g,h,i,j] - [6,7,8,9,10]
string3 - [k,l,m,n,o] - [11,12,13,14,15]

Edit distance between string1 and string2 = 5. Euclidean distance between string1 and string2 = sqrt(5^2+5^2+5^2+5^2+5^2) = 5*sqrt(5). In general, euclidean_distance >= function_of(edit_distance)

Earth mover distance and Word mover distance are current state-of-the-art similarity measures for BigData sets. Recently Earth Mover Distance has been approximated in linear time (Relaxed Word Mover Distance - [Kusner] - http://proceedings.mlr.press/v37/kusnerb15.pdf) though exact computation is quadratic or cubic depending on algorithm. If Edit distance is expressible in terms of Earth mover distance, edit distance can be approximated in linear time which is subquadratic (closer to proving SETH is false) and if exact edit distance could be derived from this approximation in subquadratic time SETH is false. Following is an example reduction from Earth mover distance to Edit distance by tokenizing each string to 1-alphabet arrays or buckets creating a 2D array or histogram per string:

abcde - [[a],[b],[c],[d],[e]]
fghijk - [[f],[g],[h],[i],[j],[k]]

Earth mover distance or Relaxed Word Mover distance between 2 string histograms earlier is exactly equal to edit distance (work necessary in adding, updating or deleting 1-alphabet buckets for transforming one string to another). NeuronRain AsFer implements Earth Mover Distance similarity measure between two 2D syllabified string tensors.

Edit distance between 2 strings a and b is defined as recurrence solved by dynamic programming in quadratic time by Wagner-Fischer memoization algorithm:
minimum(d(i-1,j-1),d(i-1,j) + delete(ai),d(i,j-1) + insert(bj),d(i-1,j-1)

+ substitute(ai,bj))

Traditional edit distance dynamic programming recurrence memoization table could be split into quadrants - one of constant size (bottom left) and the other 3 quadrants of variable size. Constant size quadrant of the table could be computed by Earth Mover Distance reduction (LC-RWMD or LC-ACT) in linear time exactly by suitably choosing the number of iterations for 100% accuracy. Rest of the 3 quadrants could be computed by conventional quadratic time edit distance algorithms. Cumulatively total time to compute edit distance is subquadratic (exponent k in O(n^k) is 2-epsilon < 2) because a constant fraction of tableau is computed in linear time. For example exponent could be 1.9999999.... which adheres to definition of subquadratic though superlinear thus proving SETH is 100% false.

Linear Complexity Approximate Constraints Transfer (LC-ACT) is the generalization of all WMD measures and its accuracy can be controlled by number of iterations(k). Linear Complexity of LC-ACT is achieved by setting k to be a constant. LC-ACT has maximum accuracy of 97.83% for 15 iterations for MNIST dataset (http://yann.lecun.com/exdb/mnist/). Increasing the constant for number of iterations would asymptotically boost accuracy close to 100%. 99% accuracy of LC-RWMD or LC-ACT for increased iterations implies edit distance is approximable by previous Edit-Distance to EMD reduction in linear time and SETH is false with 99% probability (or) there is an error of +1% or -1%. Deriving exact edit distance from approximate LC-ACT distance in linear time would prove SETH is 100% false. Edit distance from previous dynamic programming recurrence is an integer if weights for delete,insert and substitute primitives are integers. E.g For exact edit distance of 200 between 2 strings, 99% accurate LC-ACT-k computes an approximate distance of 198 or 202 in linear time. Each of the integral values in +1% or -1% error range - 198,199,200,201,202 - have to be sifted for exact edit distance. This requires a verification algorithm in linear time which accepts a numeric argument in the error range and 2 strings and prints if it is exact edit distance between 2 strings. Pseudorandomly choosing one of the integers in error range and repeating the trials might amplify success. Each recursive step in dynamic programming recurrence earlier for edit distance could be computed by LC-ACT-k at 99% accuracy. If definition of subquadratic time allows fractional exponent in runtime O(n^k) for 1 <= k < 2, then SETH is still false though ACT is super-linear but still subquadratic and ACT iterations could be increased accordingly for close to 100% accuracy.

Edit distance dynamic programming recurrence could be summarized as:
Edit distance = n(s) + n(i) + n(d)
n(s) = number of substitutions
n(i) = number of insertions
n(d) = number of deletions

Each of the approximate edit distances in LC-ACT error range could be equated to previous summation:
Approximate LC-ACT Edit Distance = n(s) + n(i) + n(d)

For 2 strings of equal lengths n(i) and n(d) are zero => n(s) = approximate edit distance in LC-ACT error range(-1% to +1% error band for 99% accuracy)

Machine Learning solution for deriving exact edit distance from LC-ACT approximate edit distance:

Create a training dataset of strings and compute exact edit distances and LC-ACT approximate edit distances for every pair of strings and find the average deviation between exact and approximate edit distances.This learnt deviation could be applied as heuristic correction for exact edit distance to LC-ACT approximate edit distances between 2 arbitrary strings.

References:

1.The Art of Computer Programming - Volume 3 - [Donald Knuth] - Sorting and Searching - 6.2.2
2.Beautiful Code - Finding Things - [Tim Bray - Sun Microsystems]
3.Google AI Blog - [Joshua Bloch] - https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html
4.Topological Sorting - Fast Parallel Algorithms - [SA Cook] - https://www.sciencedirect.com/science/article/pii/S0019995885800413 - If set of integers are randomly assigned as labels of a random graph, topological sorting of vertices by some ordering can be done efficiently in NC^2 and a following binary search can find the element in additional logarithmic time. But the topologically sorted vertex list may not be sorted by total ordering always.Dekel-Nassimi-Sahni algorithm works by Repeated Matrix Multiplication logarithmically many times and sorting by longest paths.
5.Planar Separator Theorem for Graphs - [Richard J. Lipton and Robert Endre Tarjan] - https://epubs.siam.org/doi/abs/10.1137/0209046 - PSLG from rasterized hyperbolic arc xy=N can be partitioned to almost equal subgraphs by removing O(sqrt(3N+1)) vertices - quite useful in parallelizing and fair load balancing the rasterized polygon faces amongst PRAMs for point location queries
6.Planar Point Location in sublogarithmic time - [(Late) Mihai Patrascu] and [Timothy Chan] - https://ieeexplore.ieee.org/document/4031368
7.Algorithms - [Cormen-Leiserson-Rivest-Stein] - Page 845 - Number theoretic algorithms - Integer Factorization - 33.9 - Pollard's Rho Heurisitc - "...It is infeasible with today's supercomputers and the best algorithms to date to factor an arbitrary 200-decimal-digit number..."
8.On Implementing an o(log n) Planar Point Location Algorithm - [Yifei Zhang, Wei Yu, Decheng Dai] - https://dsa.cs.tsinghua.edu.cn/~deng/cg/project/2006f/2006f-b.pdf
9.Clarkson-Shor Random Sampling, Partitioning Theorem - http://courses.cs.tau.ac.il/0368-3249-01/michasodaslid.pdf - "...Q a set of n points in the plane R a random sample of r points of Q Then, with high probability, any triangle that does not contain a point of R contains at most cn/r * log r points of Q (c an absolute constant)...For any r < n, P can be partitioned into O (r) subsets P1, . . . , Pt, each of ≤ n/r points, so that any hyperplane h separates the points of only O (r^(1 − 1/d)) subsets..." - Query could be a hyperplane or triangle and not necessarily a point. Random sampling is quite necessary in pre-poll and post-poll forecast analytics - if voters are points on a 2-d plane, Clarkson-Shor estimate upperbounds the extent of voters excluded by a random sampling triangle. For plural multipartisan voting, Partitioning theorem has a direct bearing - Set of Voters P is partitioned to r subsets of candidates voted for and query hyperplane (forecast sample) separates only a fraction of subsets.
10.SIR ODE model of How Gossips and Rumours spread in Social network - https://scholarship.claremont.edu/cgi/viewcontent.cgi?article=1036&context=codee
11.Random Walks and Diffusion in Networks - Brownian Motion - https://www.sciencedirect.com/science/article/pii/S0370157317302946
12.Finding Closest Pair of Points - Subquadratic O(NlogN) divide and conquer algorithm better than naive O(N*N) bruteforce for obstacle avoidance - Section 35.4 - Chapter 35: Computational Geometry - Algorithms - [Cormen-Leiserson-Rivest-Stein] - Page 908 - "... System for controlling air or sea traffic might need to know which are the two closest vehicles in order to detect potential collisions ..."
13.Linear Complexity Relaxed Word Mover Distance - LC-RWMD - [Kubilay Atasu-Thomas Mittelholzer] - IBM Research, Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. - http://proceedings.mlr.press/v97/atasu19a/atasu19a.pdf - Figure 2 on computing distance between 2 histograms as transportation cost minimization problem - Algorithm 3 for ACT - Section 5 - Table 2 - LC-RWMD has average time complexity O(vhm + nh) and average space complexity O(nh + vm + vh) for n=Number of database histograms, v=Size of the vocabulary, m=Dimensionality of the vectors, h=Average histogram size - Table 5 on comparison of precisions of various distance measures and maximum precision of approximation is > 97% implying EMD-to-EditDistance reduction could be approximated in linear time with > 97% accuracy (or SETH is false with > 97% probability).

14. Implications of ETH being false - https://en.wikipedia.org/wiki/Exponential_time_hypothesis - "... However, if the exponential time hypothesis fails, it would have no implication for the P versus NP problem. There exist NP-complete problems for which the best known running times have the form O(2^n^c) for c < 1, and if the best possible running time for 3-SAT were of this form, then P would be unequal to NP (because 3-SAT is NP-complete and this time bound is not polynomial) but the exponential time hypothesis would be false...."
15.CGAL 5.2 Planar Point Location Implementation Strategies and Illustrations - C++ - Section 3 and Figure 34.7 - https://doc.cgal.org/latest/Arrangement_on_surface_2/index.html - CGAL parallelizes by calls to Intel TBB multicore library - CGAL dependencies - https://doc.cgal.org/latest/Manual/thirdparty.html
16. Minimum Edit Distance Dynamic Programming Table - https://web.stanford.edu/class/cs124/lec/med.pdf

----
758. (THEORY and FEATURE) 21 January 2019 - FP Growth Algorithm for mining frequent patterns and mining timeout dictionaries example - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
Timeout design pattern implemented as a separate chaining/dictionary of timeouts-to-processes has been described in ../AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt. Discussion there is restricted to a process id present in only one bucket per timeout value. But there are possibilities when same process_id has to be multilocated in many timeout buckets e.g multiple threads spawned by a process can have different timer values for varied functionalities and all those threads per process have to be timedout.

Spark MLlib implements parallel version of FP-Growth algorithm which mines frequent itemsets in multiple baskets with no candidate generation (downward closure) by growing suffix trees. Survival Index Timeout Map is also a set of baskets (buckets) from which frequently occurring process subsets can be mined by Spark FPGrowth. Example Spark code for this has been committed to code/Spark_FPGrowth_SurvivalIndexTimeout.py and logs in code/testlogs/Spark_FPGrowth_SurvivalIndexTimeout.log.21January2019 which demonstrate minimum support for frequent occurrences and minimum confidence for association rules. These frequently occurring process subsets point to some lurking relationship among the colocated processes in some way and thus a measure of system load and behaviour.

Multilocation of process id(s) in multiple time out buckets creates hyperedges amongst the timer bucket vertices and thereby a hypergraph.

References:

1.Spark MLLib Documentation - https://spark.apache.org/docs/2.3.0/ml-frequent-pattern-mining.html

----
19 March 2019 - String Search in Large Files

BigData or Large Text Files on clouds often require a functionality to quickly search for a string of patterns or text. There are standard string matching algorithms like Knuth-Morris-Pratt, Boyer-Moore etc., which are standard algorithms implemented in text editors for "find". For large filesystems, wavelet trees are fast alternatives which support rank(c,p) [number of

occurrences of character c before position p], select(c,q) [q-th occurrence of character c], access(k)[access k-th position] operations in O(1) time for binary vectors. For substring pattern p of size n, repetitive n invocations of select(p[i],q) returns all substrings of pattern p in the large text file - Example Pseudocode below for first match:

```
            for k in xrange(n):
                  pos=wavelettree.select(p[k],1)
                  if prevpos + 1 != pos
                        return matchfound==false
                  prevpos=pos
            return matchfound==true
```

----
26 March 2019 - Example MapReduce in Spark 2.4 Cloud - Bitonic Sequences of Integers
----
Spark framework parallelizes work by partitioning a bigdata set into Resilient Distributed Datasets
which are map()-ped to Spark cloud nodes by Spark Executor Driver and local computations in cloud node are unified by reduce(). Spark 2.4 + Python Spark code example in code/Spark_MapReduce.py defines two functions map() and reduce(). Spark context is instantiated and an array of integers is parallelized to
Spark resilient distributed dataset partitions and map() is invoked per RDD on Map function - Map() which returns a single element array as a tuple. This is followed by reduce() on Reduce function - Reduce() which takes as args two mapped tuples and merges the integer arrays after > or <= checks to create a subarray which is either strictly ascending or descending. Resultant array is checked for sortedness by a function and mapreduces further if not. Logs for this in Spark_MapReduce.log.26March2019 demonstrate the bitonic sequence n1 + n2 which fluctuate (ascending-descending-ascending):

```
Reduce: (1, [15], 1, [16], 1, [12], 1, [7], 1, [6], 1, [3], 1, [2], 1, [4], 1, [5])
n1: (1, [15], 1, [16], 1, [12], 1, [7], 1, [6], 1, [3], 1, [2], 1, [4], 1, [5])
n2: (1, [23], 1, [32])
Reduce: (1, [15], 1, [16], 1, [12], 1, [7], 1, [6], 1, [3], 1, [2], 1, [4], 1, [5], 1, [23], 1, [32])
Bitonic Sequence of Integers:
[15, 16, 12, 7, 6, 3, 2, 4, 5, 23, 32]
```

As can be seen, the reduce produces a huge tuple from which an integer array is extracted after typecheck by Python keyword "type".

----
757. (THEORY and FEATURE) 16 March 2019 - Set Partitions in SymPy, Survival Index Timeout OS Scheduler
- this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
Spark_FPGrowth_SurvivalIndexTimeout.py example previously described for FPGrowth mining of dictionaries
has been changed to print all possible partitions of set of processes in OS. Partition API in SymPy have been invoked for this. OS Scheduler/Timer is in itself a set partition histogram which maps timeout values to subsets of processes. This demonstrates the integer partition and LSH/set partition-separate chaining isomorphism. Logs in Spark_FPGrowth_SurvivalIndexTimeout.log.16April2019 show the all possible permutations of process set partitions.Rank of a partition is printed which is

the difference between size of largest part and number of parts (or) equivalently side of largest square in Durfee Diagram.

References:

1.Durfee Square - https://en.wikipedia.org/wiki/Durfee_square
2.Rank of a partition and Partitions fitting within a rectangle - Gauss Binomial Coefficient - https://en.wikipedia.org/wiki/Rank_of_a_partition
3.Ferrers Diagram or Young Tableau of a Partition - https://en.wikipedia.org/wiki/Partition_(number_theory)#Ferrers_diagram

----
837. (THEORY and FEATURE) People Analytics and Economic merit (Spending Analyzer) - Credit Card Datasets - 9 July 2019 - Fraud Analytics in Spark 2.4.3

NeuronRain AstroInfer implements a primitive fraud analytics on a credit card transactions dataset - https://gitlab.com/shrinivaasanka/asfer-github-code/blob/master/python-src/FraudAnalytics.py which analyzes https://www.kaggle.com/mlg-ulb/creditcardfraud csv dataset. Alternatively Spark provides some basic statistics functions for analyzing DataFrames of BigData. An example Spark Python code on Spark 2.4.3 at code/Spark_FraudAnalytics.py does the following :

(*) Group the transactions by first few columns of cardholder identity and computes average amount withdrawn by aggregating column "Amount"

(*) Find Frequent Items in the DataFrame

(*) Describe the columns and compute basic statistics - mean, standard deviation etc.,

```
...
+-------+--------------------+--------------------+--------------------
+--------------------+--------------------+--------------------
+--------------------+--------------------+--------------------
+------------------+
|summary|                  V1|                  V2|                  V3|
V4|                  V5|                  V6|                  V7|
V8|                  V9|            Amount|
+-------+--------------------+--------------------+--------------------
+--------------------+--------------------+--------------------
+--------------------+--------------------+--------------------
+------------------+
|  count|              284807|              284807|              284807|
284807|              284807|              284807|              284807|
284807|              284807|            284807|
|   mean|3.742631994571313...|4.949726613980831...|-7.91956258236933...|
2.685625859585728...|-1.52643182031150...|1.813835301123298...|-
1.74517780292937...|-2.00384093565998...|-3.14088095662162...|
88.34961925089794|
| stddev|  1.9586958038574895|  1.6513085794769824|   1.516255005177769|
1.4158685749409246|  1.3802467340314435|  1.3322710897575698|
1.237093598182658|  1.1943529026692032|  1.0986320892243098|250.12010924018742|
|    min|-0.00012931370800...|-0.00010296722561...|-0.00010859127517...|-
0.00011921826106...|-0.00010366562678...|-0.00010234903761...|-
0.00010533581684...|-0.00010065655617...|-0.00010181309940...|
0|
|    max|7.55406974741191e-05|    9.99769856171626|9.67444968403876e-05|
9.92501936512661|9.99846034625664e-05|    9.91116576052911|    9.97044721041161|
9.90825458583455|9.96754672601408e-05|             999.9|
+-------+--------------------+--------------------+--------------------
+--------------------+--------------------+--------------------
+--------------------+--------------------+--------------------
+------------------+
```

----

756. (THEORY and FEATURE) 21 October 2019 - Advertisement Analytics - Recommender Systems - ALS Collaborative Filtering - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

----

Collaborative Filtering based Recommender Systems works by plotting a matrix of users versus their item choices and predicting preferences of new users or missing preferences based on inferences from the user-items matrix.
Advertisement Analytics by computing PageRank of viewer channel switch graph (converging markov random walk) is explained in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt. Spark MLLib predefines a function for alternating least squares (ALS) collaborative filtering on cloud. Code example in Spark_RecommenderSystems.py which is a modified version of documentation example in https://spark.apache.org/docs/latest/ml-collaborative-filtering.html adapts to channel recommendation systems based on user surveyed matrix of viewer-channel matrix.This is an alternative advertisment analytics.Channel ratings of viewers are read from text file AdvertisementAnalytics_RecommenderSystemsCF.txt having fields - viewer, channelid, rating and timestamp.
ALS factorizes the user-item matrix into User and Item matrix factors and minimizes a quadratic optimization function:

    UserItem = User * Item (Factorization)
    (UserItem - User * Item)^2 (Minimization)

ALS makes either User or Item factor constant alternatingly and converts to a quadratic optimization problem.

----

755. (THEORY and FEATURE) 23 October 2019, 1 November 2019 - Sequence Mining of Astronomical Datasets by Spark PrefixSpan - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

----

NeuronRain Research version of AstroInfer in SourceForge mines astronomical data e.g encoded ephemeris degree locations of celestial bodies, planetary positions on zodiac etc., by native sequential implementation of GSP Sequence Mining algorithm. Spark MLlib provides cloud implementation of an advanced sequence mining algorithm - PrefixSpan which recursively projects a sequence dataset to smaller fine grained datasets and locally mines frequent patterns in smaller databases which are grown (https://ieeexplore.ieee.org/document/1339268 - Mining sequential patterns by pattern-growth: the PrefixSpan approach - Jian Pei,Sch. of Comput. Sci., Simon Fraser Univ., Burnaby, BC, Canada,Jiawei Han,B. Mortazavi-Asl,Jianyong Wang,H. Pinto,Qiming Chen,U. Dayal,Mei-Chun Hsu). An example code for Spark PrefixSpan pattern mining for encoded astronomical datasets (encoding is numeric 1-to-9 for each of the 9 planets spread across 12 zodiac houses delimited by #) is at code/Spark_PrefixSpan.py. It prints frequently occurring celestial juxtapositions of planets in the dataset. Large scale mining of celestial data can be used for variety of scientific applications - solving n-body gravitational differential equations, correlating frequent planetary patterns to terrestrial seismic events and weather forecast. Curiously enough, each encoded string of celestial configuration is a set partition and is an instance of balls-bins problem - set of celestial bodies are bucketed by some permutation amongst 12 houses - number of such celestial configurations are governed by Bell and Stirling numbers which is derived as below (background n-body problem choreography analysis is in NeuronRain FAQ - https://neuronrain-documentation.readthedocs.io/en/latest/):

    Number of ordered partitions of length p of 9 celestial bodies = N(p)
    Summation_p=1-to-9(N(p)) = B9 - 9th Bell Number (= binomial series

summation of Stirling Numbers of second kind) = 7087261
Number of all possible choreographic arrangements of 9 celestial bodies amongst 12 zodiac degree divisions = Summation_p=1-to-m(12CN(p)) which is lowerbounded by B9 >= 7087261 = number of celestial patterns to be correlated to terrestial events. This is a finite number implying repetitive patterns in gravity induced events.

References:

1.Sage OrderedSetPartition - http://doc.sagemath.org/html/en/reference/combinat/sage/combinat/set_partition_ordered.html

----
754. (THEORY and FEATURE) 18 December 2019, 19 December 2019, 22 July 2020, 23 July 2020 - Intrinsic Performance Ratings and Sports Analytics, Streaming Histogram analytics, Partition distance measures, Random graph streams, Logic and Game theory - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
----
Non-perceptive intrinsic performance rankings (IPR) have been mentioned as one of the multiple classes of measuring intrinsic fitness/merit as part of People Analytics in https://neuronrain-documentation.readthedocs.io/en/latest/ without voting. IPRs are widely used to rank Chess players (e.g Elo ratings). As an alternative bigdata usecase, stream of cricket match statistics is analyzed for intrinsic merit (Example hawkeye statistics - wagon wheel, trajectories, pitch maps - in https://www.bcci.tv/events/15305/india-v-west-indies-2019/match/15309/1st-odi?tab=overview#hawkeye). NeuronRain Set Partition and Histogram analytics implementations elicit patterns from stream of histogram set partitions by multitude of distance measures - earth mover distance, partition rank, correlation among others:
(*) Batting, Fielding and Bowling figures per match for 11+11=22 players can be plotted as a histogram of player versus runs/wickets.
(*) Streaming Set partition analytics is a special case of Histogram analytics - stream of histograms can have varied total sum of parts while stream of set partitions have oscillating parts but constant total sum of parts.
(*) Stream of match statistics histograms (which always have constant 11 parts per team) as time series has the imprint of team performance over the period of time.
(*) Partition Rank which is defined as maximum part - number of parts is an indicator of skewness in the team performance (maximum runs/wickets scored by a player - 11 per team)
(*) Durfee square which is the side of largest square inscribed in Ferrer diagram of the match statistics histogram measures if the performance is equitable - larger the square, more players performed nearly equally.
(*) Time series of Earth mover distance between consecutive match statistics histograms in the stream sheds light on seasonal trend in team performance.
(*) Patterns in Wagon wheel graphics of shots played per player measures strengths of each player - stream of wagon wheels can be mapped to a histogram of shots and aforementioned metrics apply.
(*) Clustering of Bowling trajectories/pitch maps extract stereotypes.

Most games have been analyzed for their computational complexity - E.g Go, Chess are either in PSPACE or EXPTIME depending on rules. Cricket on the other hand could be theoretically formulated as two True Quantified Boolean Formulae Satisfiability problems in PSPACE - TQBF1 and TQBF2 - forall and existential quantifiers in each formula symbolize team1 and team2 per innings (adversary-defender game). Batting and Bowling Roles (respectively governed by existential and forall quantifiers) for 22 player statistics variables are interchanged in each innings. Maximum satisfied formula decides the winner and game is undecidable if drawn/tied.

Mining soccer patterns is complex because of random trajectories of the ball which draws a random directed graph on field - trajectory is a brownian motion random walk - whose vertices are 2-colored (bichromatic for vertices belonging to two opponent teams). Consecutive match statistics create a stream of 2-colored random graphs and frequent subgraph patterns in this stream of random trajectory graphs show patterns across matches.

Reference:

1.Real Plus Minus of NBA - http://www.espn.com/nba/statistics/rpm - Player Efficiency Ratings in Basketball
2.Go, PSPACE, Combinatorial Game Theory - https://en.wikipedia.org/wiki/Go_and_mathematics
3.Computational Complexity of Games and Puzzles - https://www.ics.uci.edu/~eppstein/cgt/hard.html

----

753. (THEORY and FEATURE) Social Network Analytics, People Analytics, Preferential Attachment, Urban Sprawls, Graph Edit Distance and Intrinsic Merit Rankings - 20 February 2020,26 February 2020,29 April 2020,17 May 2020,18 May 2020,28 May 2020,15 August 2020,24 December 2020 - related to 762, 821, 1128 and all sections on urban planning analytics, GIS analytics among other topics in NeuronRain Theory Drafts

----

NeuronRain AstroInfer codebase has implementations for urban planning analytics including analytics of Remote sensing imagery , GIS 2-D patches extraction, Watershed image segmentation specific to urban sprawls et al. Urban planning analytics could be termed a special category lying in intersection of People analytics and Social Network analytics - difference being people profiles and social profile vertices in world wide web are replaced by real humans and urban clusters. There exists a striking parallel between preferential attachment in social networks and formation and growth of urban agglomerations - Processes underneath why certain social profiles attract more incoming links and why people flock to megapolises (and megalopolises formed by countermagnets) have clustering similarities. Preferential attachment in social networks causes profiles of large followers to be larger or famous become more famous while big cities become megacities. It is self-explanatory that perception underlies preferential attachment (because population is swayed by and gravitates to majority). Average degree of a vertex in Social networks is replaced by population density in Urban sprawls.

On the contrary, rationale for ranking social profiles by their intrinsic least energy merit characterized by inverse lognormal sum of education(E), wealth(W) and valour(V) (energy of social profile vertex is directly proportional to 1/(logE + logW + logV)) defined and implemented in NeuronRain AstroInfer applies as well to the ability of an urban sprawl to attract population and ranking urban areas absolutely. Some of the following example standard of living factors could be classified into three divisions of lognormal sum (inverse lognormal sum could be broken-up into following finer contributing factors):

- Harbours, Coastal areas (Wealth)
- Infrastructure (Wealth)
- Manufacturing,IT and ITES jobs (Wealth)
- Education and R&D institutions (Education)
- Financial institutions (Wealth)
- Antiquity, Tourism and Culture (Valour)
- Salubrious and Cool climes (less likely)
- ...

Previous division of Social profile vertices into 3 categories may infact be 4 if Wealth category is segregated into Working class and Business profiles:

(*) Profiles of Academics and Researchers - Education (E)
(*) Profiles of Sports personalities - Valour (V)
(*) Profiles of Business - Wealth1 (W1)
(*) Profiles of Working class - Wealth2 (W2)

which would consummate to lognormal least energy sum logE + logV + logW1 + logW2. On the contrary such a division of social profiles may not be Four color theorem compliant - there is no or least avoidance of colors (categories) of neighbours (though W1 and W2 are repulsive). NeuronRain People Analytics implements Chaotic HMM state machine model for tenure transitions of a profile and an alternative weighted automaton model is put forth (state transitions between organizations are incentivized by weights). Attributing merit to people tenure transitions reduces to rating incentives for choices made e.g:

(*) transition from industry to academics is considered nobler than industry-to-industry.
(*) transition from academics to industry is majority choice
(*) transition from low remuneration to high is a majority choice
(*) transition from high remuneration to low is rare
(*) transition from employment to entrepreneur is rare and considered less risk-averse
(*) transition from low designation to high is majority choice

Weights of the transitions in Weighted Automata Tenure model are decided by ranking of rarity of choices earlier. Comparing state machines (ChaoticHMM or Weighted Automata) of tenures of two social profiles is the problem of Equivalence of Regular languages (verify if two automata are similar) solved by Table filling algorithm. NeuronRain AstroInfer People Analytics implements career transition analytics based on a weighted automaton derived from graded incentives for previous state transition choices - rare transitions are rewarded. Fluctuating weights between consecutive transitions as points on polynomials could be interpolated to a polynomial career graph and similar profiles could be clustered by distance of those career polynomials e.g Distance between polynomials is defined by inner product spaces - http://web.mit.edu/18.06/www/Spring17/Orthogonal-Polynomials.pdf. In the context of "original music score" merit measurement, equivalence of weighted automata of two music compositions is verifiable by Table filling.

GIS imagery of any urban sprawl can be segmented by watershed algorithm or contour detection to following categories (or by approximate color of pixels):

(*) Manufacturing,IT and ITES
(*) Residential
(*) Commercial
(*) Agricultural lands, Water bodies (Groundwater and Surfacewater) & Greenery

which correspond to four color cartography of the urban sprawl. By Four color theorem, segments of urban sprawl master plan could be colored by any of the four categories in a way which prevents any neighbor of a segment to be of same category thereby ensuring equitable and sustainable development. Fair Division of resources within an urban sprawl (water,amenities,...) is a Pareto optimal Multiple Agent Resource Allocation (MARA) problem. An allocation is Pareto optimal if no agent is preferentially treated at the expense of its peers (Envy-free) while a Nash equilibrium occurs when every agent is at its best and can't do better. Four categories or colors of an urban sprawl GIS segmentation earlier correspond to Four agents among whom a resource has to be fairly divided - an analytics alternative to Four color theorem. Segments of an urban sprawl GIS imagery form a Dual graph induced by segment vertices (faces) and their adjacency. Four coloring theorem stems from 4-colorability of unavoidable sets and 1482 reducible configurations. On the lines of TCP congestion control in electronic networks, Bottleneck measures (e.g Expanders, Cheeger constant) of 4-colored FaceGraph of a segmented urban sprawl GIS imagery help in theoretically solving the traffic congestion problem in urban sprawls - Most transport in urban sprawls happens amongst 3 face categories of FaceGraph - Manufacturing-IT-ITES,Residential and Commercial. Four colored Facegraph of segmented urban sprawl GIS imagery has an inherent feature which decongests traffic snarls - every s-t path between vertices of Facegraph is 4-colored (two consecutive vertices in a path avoid same color) implying every route in transport network traverses perfect mix of 4 face categories earlier. Graph Edit Distance between FaceGraphs of 2 urban sprawls is a theoretical measure of similarity which quantifies Urbanization and thus could be used in Ranking Urban agglomerations. FaceGraph as a Flow network (Maxflow-Mincut problem) of face vertices maximizes traffic between source and sink faces through minimum cut edges which if removed would disconnect source from sink. Apart from these population dynamics of urban sprawls are of statistical importance and are necessary for redrawing and expanding urban boundaries. Ricker non-linear logistic N(t+1) = N(t)e^(r(1-N(t)/K) formalizes growth of population from generation t to (t+1) from which future population projections for an urban sprawl could be inferred (implemented as R script in GRAFIT course material). Increasing population implies increasing population density (average degree) for constant sprawl.

Extent of correlation between Rankings of Urban Sprawls by Population (mostly caused by preferential attachment - Urban Area Rankings in Section 675 of https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) and Intrinsic Lognormal Sum Least Energy Merit is a measure of Fame-Merit coincidence.

References:

1.Preferential attachment,Bose-Einstein Condensation in Networks,Double Jeopardy in business intelligence (low market share brands have low buyers and low brand loyalty - low market share brand is not necessarily of low merit) - https://en.wikipedia.org/wiki/Preferential_attachment
2.Automata - [Hopcroft-Motwani-Ullman] - Page 157 - 4.4.2 - Table Filling algorithm for Equivalence of two automata (NFA and DFA) - If two states are not distinguished by Table filling, they are equivalent
3.NASA SEDAC GIS Population Estimator - https://sedac.ciesin.columbia.edu/mapping/popest/gpw-v4/ - PDF of 6-colored Terrain Population Density of Settlement Points in Southern India (2015) and Convex Hull Population estimator - Gridded Population of the World (GPW) Version 4 - has been committed to code/testlogs/NASA_SEDAC_Population_Estimator.6coloredGIS.16August2020.pdf
4.Inner Product Spaces of Polynomials - Examples - http://www.math.ucdenver.edu/~esulliva/LinearAlgebra/InnerProducts.pdf - "... <p, q> = p(t0)q(t0) + p(t1)q(t1) + · · · + p(tn)q(tn) ..." - For career polynomials of two professional profiles p and q, p(ti) and q(ti) are the fluctuating weights (value of polynomial) at time ti where weight could be any metric including remuneration,designation et al of a profile.

----
821. (THEORY and FEATURE) Word Embedding on a Vectorspace - Spark SkipGram Log Likelihood - Text Similarity, Dissimilarity and Merit Measure of Originality - 10 May 2020,11 May 2020 - related to 762,815 and all sections on Kernel Lifting, Intrinsic Merit of Academic publications/Large Scale Visuals/Music/People, Psychoanalysis and Shell Turing Machines
----
Word2Vec Embedding in Spark MLLib maps words in a text to vectors in a Vectorspace which facilitates clustering of synonymous words - Words of similar meaning and context are in proximity on the vector space. This kind of kernel lifting from one dimensional text to multidimensional vectorspace of words in Spark MLLib is implemented as a SkipGram. SkipGram maximizes average loglikelihood for a word wt and a context training window for wt of width 2k - w(t-k),...,w(t+k):

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{j=-k}^{k} \log p(w_{t+j}|w_t)$$

Code example Spark_Word2Vec.py defines a function bibliometrics_word2vec() which analyzes text file bibliographies from Semantic Scholar(https://www.semanticscholar.org/author/Ka.-Shrinivaasan/1861803), DBLP(https://dblp.dagstuhl.de/pers/hd/s/Shrinivaasan:Ka=) and GoogleScholar (https://scholar.google.co.in/citations?user=eLZY7CIAAAAJ&hl=en) profiles of author on quadcore and Python 3.7.5 and tries to find synonyms and clustering similarities (by cosine distance). Sections 762 and 815 propound an alternative merit measure of "Originality" (how dissimilar a work is from other authorships and common theme similarity amongst author's works) which can be best captured by Word2Vec vectorspace embeddings of texts wherein computation of euclidean distance clustering similarity between word vectors is easier. Analyzing academic publications is a field in itself - Bibliometrics and Scientometrics - once touted to be an alternative to peer review. It is worth noting that TextGraph representation of texts offers Graph Isomorphism and Graph Edit Distance as graph theoretic concept similarity alternatives. Originality thus distinguishes seminal papers (differentiated by huge conceptual distance from earlier works), independent of H-index (Fame measure based on voting - citations), which ushered a revolution when they were published. H-index is the Durfee Square of the Integer Partition of Citations amongst papers and approximately equals 0.54*sqrt(N) for total citations N. Concept similarity is defined by archetypes (Thought motif from which copies are made) in Carl Jungian Psychoanalysis (Pattern of Thought in collective unconscious).

References:

821.1 H-index - https://en.wikipedia.org/wiki/H-index
821.2 Bibliometrics for Internet Media: Applying H-index to YouTube - https://arxiv.org/abs/1303.0766
821.3 Spark MLLib Word2Vec - https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=word2vec

----
822. (FEATURE) Word Embedding of Text on Vectorspace - Merit of Academic Publications - an alternative to H-index - implementation changes - 11 May 2020 - related to 821

1.Spark_Word2Vec.py has been refactored to include separate mapreduce functions for tokenizing the text
to words after utf-8 encoding.
2.To circumvent Spark Java String to Iterable cast error in Word2Vec fit(), list comprehension in mapreduce creates one element arrays per word (instead of array per line earlier).
3.Parsed data is printed by DataFrame show() and printSchema() - StringType Spark type has been imported.
4.GoogleScholar bibliography has been included.
5.Spark logs are at testlogs/Spark_Word2Vec.log.11May2020

----
826. (FEATURE) Spark SkipGram Word2Vec Bibliometrics - increased vocabulary and accuracy - 9,11 June 2020
- related to 821,822

1.Spark_Word2Vec.py has been changed to parse an expanded Bibliography text file (Spark_Word2Vec_Bibliography.txt) which is a concatenation of BibTeX from various Bibliography sources (Google Scholar,CiteSeerX,Microsoft Academic,Semantic Scholar,Harvard NASA ADS,DBLP) instead of parsing a single source.
2.Because of enlarged BibTeX training data, vocabulary of Word2Vec bibliometrics has improved and more meaningful words e.g
author,year,Science,merit,title,document have been vectorized
3.All BibTeX are from author search of self - "Ka Shrinivaasan" - from URLs:
      Google Scholar - https://scholar.google.co.in/citations?user=eLZY7CIAAAAJ&hl=en
      DBLP - http://dblp.dagstuhl.de/pers/hd/s/Shrinivaasan:Ka=

Microsoft Academic - https://academic.microsoft.com/search?q=ka%20shrinivaasan&qe=%40%40%40Composite(AA.AuN%3D%3D%27ka%20shrinivaasan%27)&f=&orderBy=4&skip=0&take=10

Semantic Scholar - https://www.semanticscholar.org/author/Ka.-Shrinivaasan/1861803

CiteSeerX - https://citeseerx.ist.psu.edu/search?q=Ka.+Shrinivaasan

NASA/ADS - https://ui.adsabs.harvard.edu/search/q=author%3A%22Shrinivaasan%2C%20Ka.%22&sort=date%20desc%2C%20bibcode%20desc&p_=0

4.Vector size and Seed size have been set to 5 and 42 respectively

5.Vocabulary could be further improved by word2vec embedding of entire article instead of BibTex abstracts which is a concept cloud of vectors.

----

827. (THEORY and FEATURE) Intrinsic Merit of Academic Publications - Bibliometrics - First Order Logic, Monoidal Categories, Girard Geometry of Interactions (GoI), Sequent Calculus, ProofNets - related to 624,821,822,826 - 10,11,13,14,15,16,17 June 2020

There exists a fine division in quantifying merit of a natural language text and academic publications cutting across disciplines - theoretical and experimental. Natural language texts are measurable by mental representation of meaning and its lambda function approximation (Recursive Lambda Function Growth) whereas every academic publication could be construed as sequence of first order logic formulae connected by logical implications - from conjecture to QED. In Sequent Calculus proof has a tree structure and every line of a proof is a conditional tautology on previous lines. Girard Geometry of Interactions maps every theorem-proof to geometry of logical statements (ProofNets) by defining trip operator (Long Trip Criterion) on proof network. For instance, Coq Theorem Prover has been used to verify proof of Four Color Theorem by [Appel-Haken]. Thus Concept distance for Originality merit measure between any two academic publications could be defined as the distance between their ProofNet GoIs (e.g graph edit distance between ProofNets). On a related note, Distance between two First order logic (FOL) statements is the problem of equivalence of 2 FOL formulae F1 and F2: If F1 == F2, distance is 0 else distance is the number of mismatches in truth table - To prove F1 == F2, both F1 => F2 and F2 => F1 have to be established. There are two other prominent proof calculi - Hilbert and Natural Deduction. Analyzing academic publications and theorem-lemma-proofs therein thus has two facets: 1) Natural Language Processing - Lambda Function approximation, Word2Vec embedding of concepts and their clustering 2) Formal Logic - Proof theory - Sequents, Gentzen Cut-Elimination (A |- B,C and D,B |- E then A,D |- C,E), ProofNet-GoIs . There is an alternative Category theoretic formalism of Proofs wherein every proof entailment A |- B is a morphism A -> B between objects A,B in a monoidal category M having a binary operator * - entailment morphisms thus constitute a ProofNet directed graph (a quiver in Category jargon - https://ncatlab.org/nlab/show/quiver). Geometry of Interaction in Linear Logic and Type theory considers A |- B as an endomorphism.

References:

827.1 Coq Proof Assistant and Four Color Theorem - https://www.ams.org/notices/200811/tx081101382p.pdf

827.2 Proof Nets - Sequent Calculus and Girard Geometry of Interactions - https://en.wikipedia.org/wiki/Proof_net

827.3 E Theorem prover - https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html

827.4 Concept Cloud Wordle example for publications - https://ui.adsabs.harvard.edu/search/q=author%3A%22Shrinivaasan%2C%20Ka.%22&sort=date%20desc%2C%20bibcode%20desc/concept-cloud - prominent concepts in a set of publications are highlighted

827.5 Equivalence of FOL statements - Biconditional - http://iiitdm.ac.in/old/Faculty_Teaching/Sadagopan/pdf/Discrete/Logic.pdf - Indian Institute of Information Technology Design and Manufacturing, Kancheepuram, Chennai 600 127, India

827.6 n-Category Lab - Geometry of Interaction - https://ncatlab.org/nlab/show/Geometry+of+Interaction

----
829. (THEORY and FEATURE) Intrinsic Merit of Academic Publications - Bibliometrics - SkipGram Word2Vec of publication full text - Concept Cloud Wordle - TOP Categories, High Dimensional Inference, Shell Turing Machines and Kernel Lifting,Lambda functions and Beta Reduction, Meaning representation, Graph Edit Distance - related to 827 - 13,14,15,16,17 June 2020,13 July 2020, 6 December 2020, 25 January 2021, 5 March 2021

1.This commit revamps Spark_Word2Vec.py implementation to parse an entire article text to extract a concept cloud wordle from it by Spark SkipGram Word2Vec. Word2Vec is computationally intensive and corpora-hungry for producing accurate embedding.
2.New text file Spark_Word2Vec_PublicationFullText.txt is read which contains a draft publication of author on "Majority Voting" - https://5d99cf42-a-62cb3a1a-s-sites.googlegroups.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC_2014.pdf
3.Concepts and respective vectors for them are pretty-printed. Concept cloud of full text is quite meaningful and captures the essential words in the text. Clustering of related concepts is implied by Synonyms printed.
4.Recursive Gloss Overlap and Recursive Lambda Function Growth algorithm implementations in NeuronRain could benefit from Word2Vec in filtering less important words while computing textgraphs, euclidean distance among them and their dense subgraph k-cores which serve as unsupervised text classifiers.
5.Extracting a composition of Lambda Functions from Natural Language Text of a Publication and First order logic (FOL) ProofNet representation of Theorems therein are empirically equivalent - former is a Turing machine while the latter is its FOL version. Example Beta Reduction ((lambda.x t)s) in reference 829.1 translates a natural language sentence to lambda function:

    Whiskers disdained catnip => disdained(Whiskers,catnip)

which could be written as FOL statement:

    There exist x There exist y (x disdained y)

and model {Whiskers,catnip} satisfies it. Recursive Lambda Function Growth algorithm works by Beta Reduction and is motivated by Grounded Cognition in Cognitive Psychology. Thus ProofNet or Category morphisms representation of texts is not just limited to academic publications but pervades any natural language text.
Meanings of Natural Language Texts captured by Recursive Gloss Overlap and Recursive Lambda Function Growth TextGraph representation algorithms could be compared by Graph Edit Distance algorithms which is NP-Hard and APX-Hard (hard to approximate) though there are polynomial approximations for Graph Edit Distance. Graph Edit Distance is the problem of Inexact Graph Matching for dissimilar graphs (while Graph Isomorphism is Exact Graph Matching for similar graphs) solved by computing optimal edit paths through A*-algorithm. Graph Edit Distance generalizes String edit distance - every string is a connected directed acyclic graph of maximum indegree 1 and outdegree 1.
6.Vectorspace embedding of concepts in an article basically does a kernel lifting of the model to arbitrary dimensional Topological space and from equivalence of Lambda functions-First order logic mentioned earlier, every first order logic statement in the ProofNet is indirectly kernel lifted and implication entailments between them are replaced by monoidal category morphisms between one logical statement embedded in space S1 to another embedded in S2 (A(S1) |- B(S2) is a morphism A(S1) -> B(S2) in a theorem monoidal category). Word2Vec typically embeds all concepts in single space and thus S1=S2.
7.Thus every theorem-proof could be stated as TOP category of first order logic objects in some topological space and morphisms amongst them.
8.Spark_Word2Vec.py has been made Python 3.7 compatible by autopep8 and 2to3.

References:

829.1 Meaning Representation by First order logic and Lambda calculus - https://www.cs.mcgill.ca/~jcheung/teaching/fall-2015/comp599/lectures/lecture14.pdf
829.2 Logical Representations of Sentence Meaning - Chapter 16 - Speech and Language Processing - [Jurafsky-Martin] - https://web.stanford.edu/~jurafsky/slp3/edbook_oct162019.pdf - Model theory, Currying and Beta-reduction in Lambda calculus,Tense and Temporal Logics - Modus Ponens on FOL sentences and Lambda function meaning and event representation - Examples: " ... VegetarianRestaurant(AyCaramba) ⇒ Serves(AyCaramba,VegetarianFood) ... ∃ e Eating(e) ∧ Eater(e,Speaker)∧ Eaten(e,TurkeySandwich) ∧ Meal(e,Lunch) ∧ Location(e,Desk)..."
829.3 Graph Edit Distance - https://en.wikipedia.org/wiki/Graph_edit_distance
829.4 Graph Edit Distance - Basics and Trends - https://www.slideshare.net/LucBrun2/graph-edit-distance-basics-trends
829.5 Comparing Stars: On Approximating Graph Edit Distance - Polynomial time approximation of Graph Edit Distance - https://web.archive.org/web/20170810170852/http://www.vldb.org/pvldb/2/vldb09-568.pdf
829.6 High Dimensional Inference - Robust Testing of Low Dimensional Functions - k-juntas - https://eccc.weizmann.ac.il/report/2021/005/ - [Anindya De,Elchanan Mossel,Joe Neeman] - Problem is to distinguish two functions one of which depends on subspace of k dimensions and the other transcends k dimensions - Two functions on different dimensions are two Shell Turing Machines - "...A natural problem in high-dimensional inference is to decide if a classifier f:R^n -> −1 depends on a small number of linear directions of its input data. Call a function g:R^n -> −1 , a linear k-junta if it is completely determined by some k-dimensional subspace of the input space.A recent work of the authors showed that linear k-juntas are testable. Thus there exists an algorithm to distinguish between:

1. f:R^n−>-1 which is a linear k-junta with surface area s,
2. f is epsilon-far from any linear k-junta with surface area (1+epsilon)s, where the query complexity of the algorithm is independent of the ambient dimension n...."

829.7 Formal Models of Language Learning - [Steven Pinker] - Tree-fitting heuristic - Pages 248-251 - https://stevenpinker.com/files/pinker/files/10.1.1.124.5000.pdf - Has parallels to Recursive Lambda Function Growth Meaning Representation Algorithm

----
836. (THEORY and FEATURE) People Analytics - Theoretical (Drone) Electronic Voting Machines - Population Count - Number of 1s (Decimal parity) in a bitstream - Various algorithms - Parallel computation in Spark - 3 July 2020

Finding parity of a boolean string is a major open problem in Computational Complexity which is not known to be in AC0 (constant depth) but in NC1 (log depth parallel computing).
BigData version of parity is the problem of Population Count which pertains to finding number of 1s in a stream of bits. Lot of algorithms for population count have been invented in Hardware and Artificial Intelligence literature - Divide and Conquer, HAKMEM, CSA(Carry Save Adder) etc., . Code example Spark_PopulationCount.py reads a CSV file containing a huge binary string and computes the number of 1s in it by sequential and parallel mapreduce. String is split into words of length 32 and parallelized to RDDs. Each word is then scanned by a mapbitstream() and population_count() functions which sequentially compute the function:

    pop(x) = -1 * Summation((x << n)) for each bit position n.

Bit left shift has been replaced by access to literal in bitstream string. Reduce function reducebitstream() gathers and sums each 32-bit word. Population count gains importance in the context of efficiently counting number of votes in a theoretical Electronic Voting Machine.

References:

1.The Quest for an Accelerated Population Count - Beautiful Code - [Oram-Wilson] - Chapter 10 - Population Count by Divide and Conquer, HAKMEM, CSA
2.HAKMEM - [Beeler-Gosper-Schroeppel] - MIT Artificial Intelligence Laboratory AIM 239, February 1972 - https://w3.pppl.gov/~hammett/work/2009/AIM-239-ocr.pdf
3.Computer Architecture: A Quantitative approach - [Hennessy - Patterson] - Full Adder of 3 1-bit inputs and 2 1-bit outputs of sum and carry - CSA is sequence of full adders

----
839. (THEORY and FEATURE) Bibliometrics - Patents as ProofNets and Search - Novelty detection and Originality merit measure, Latent Semantic Analysis, Low Rank Approximation - 6,7,8 July 2020 - related to 829

Patents are uniquely granted to individuals and corporations who have invented an original concept which could be underneath any STEM artifact. Patent Search is a time consuming process and could take years to validate a patent application and involves human effort to find Concept similarity (Novelty detection,Prior art). Novelty detection and Originality are thus synonymous - Latent Semantic Analysis by SVD (X = U(Eigen)Z^T) and low rank approximation (choose creamy layer of UEZ^T) of term-document matrix of patents mines concept words related to each other. Alternatively, First order logic ProofNet format of patent applications written in natural languages could be an apt model to quantify conceptual distance and patent search. Patent database is a massive open online dataset (MOOD) and Concept similarity search could be automated by a ProofNet distance computations amongst pairs of patent GoIs on a cloud thus replacing humans. This commit computes a Concept Cloud wordle of set of example team software patents author was involved as former staff (Sun Microsystems-Oracle)

References:

839.1 Sun Microsystems-Oracle iPlanet Application Server (iAS) 6.5 patents - http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-adv.htm&r=0&f=S&l=50&d=PTXT&Query=%22kannan+srinivasan%22+AND+%22sun+microsystems%22
839.2 Patent Novelty Detection - https://www.sciencedirect.com/science/article/abs/pii/S147403461830421X - Latent Semantic Analysis by Low Rank Approximation of Term-Document matrix

----
840. (THEORY and FEATURE) Fraud Analytics - Cybercrime analyzers - related to 752, 770 (USBWWAN wireless traffic and Ftrace analyzers) - 6,7,8 July 2020

Cybercrimes are a scourge on society causing immense economic damage. Distinguishing a spoofed email from genuine origin is an artificial intelligence problem and is still less solved. Spam filters and Firewalls in Mail service providers have machine learning algorithm implementations which segregate botnet senders from humans to some extent (Spam folders in mailboxes use Bayesian filters to differentiate frequency of words in human and robot senders). A commonsense workaround to spoofs: As against keyboard typed emails, text could be a handwritten image (scanned or touchscreen stylus written) backedup by a private hardcopy which is digitally watermarked and sent through usual HTTPS/IMAP/POP/SMTP infrastructure. Written emails are less spoof-prone and less forgeable (because of private hardcopy duplication) and introduce an additional layer of reCAPTCHA Turing tests to discern a robot from human. Old school Post office mails are written and stamped hardcopies. Most cybercrimes are difficult to probe not just because of their international impact but as well due to complex traffic routing (e.g Onion routers) which obfuscate IP, munge headers and fudge dates. Parallels between pandemics and cybercrime transmission are fascinating:

(*) Pandemics are air,vector,contact borne while Cybercrimes are ISP-borne, proliferate by address book lookups (e.g Worm spreads by infecting an id and recursively traversing contacts - contacts per id could be a huge number and thus infection graph could be an expander of high degree)
(*) Both pandemics and cybercrimes follow Game theoretic (Botnet adversary-defender game), SIR and SIS models of infections

Predominant attacks which render RSA dysfunctional are Radio-Frequency Van Eck Phreaking (capturing RF signals from monitors),Keystroke loggers (Trojan/Malware capturing keystroke signals), SQL and JavaScript XSS Injection attacks which execute malicious code in client side et al. Some example past cybercrime incidents of author's credentials (compromise of linkedin id by password reset, spoofs of yahoo webmail and past institutional affiliation ids) and their traffic pattern (headers) have been committed to code/testlogs/Cybercrime_examples/

Recursive Lambda Function Growth textgraph algorithm implemented in NeuronRain could be a spam filter - growing a textgraph of an email text and classifying the text by genre of top percentile core number word vertices is a mail segregation mechanism.

References:

840.1 Van Eck Phreaking of Electronic Voting Machines - https://yro.slashdot.org/story/09/11/22/027229/brazilian-breaks-secrecy-of-brazils-e-voting-machines-with-van-eck-phreaking

----
841. (THEORY and FEATURE) Digital Watermarking of Large Scale Visuals in OpenCV - related to 581,840 and all sections on Large Scale Visual Recognition Analytics, Handwriting Recognition and Fraud analytics - 10,11,12 July 2020

This commit implements an OpenCV python function which reads a large scale visual (Video) and a watermark
image and writes out a new MP4 video overlaying the watermark image on each frame of the visual. New
MP4 video captures the OpenHub,SourceForge,GitHub and GitLab NeuronRain repository profiles of Krishna iResearch Free Open Source Software initiative of the author having both images and text (testlogs/Krishna_iResearch_NeuronRain_Repositories-2020-07-10_13.17.20.mp4). Function watermark() in DigitalWatermarking.py reads each frame of the video and watermark image in BGRA 4-channel mode (Blue-Green-Red-Alpha). Alpha channel controls the opacity of each pixel.An overlay ndarray is created of the same dimension as each frame of the video. Pixels of overlay array are filled by watermark image. OpenCV addWeighted() function computes a weighted sum of each pixel of watermark and frame images in BGRA 4-color channel - opacity of watermark is controlled by weights. This example watermarks first 2 frames of the video by a Sanskrit name text image (in testlogs/). Alternatively, Image text could be a handwritten signature. Previous watermark is quite primitive and overlay array could be filled by an arbitrary complex pixel cryptographic hash checksum function prior to addWeighted() invocation. Such a steganographic information hiding finds applications in Copyright Protection, Licensing violations, Software Piracy, Currency watermarking, Fraud and Tamper protection, Antitheft Cybercrime software inter alia.

References:

841.1 Digital Watermarking - overview - https://www.sciencedirect.com/topics/computer-science/digital-watermark - Movie Piracy example - every frame of DVD is copyright watermarked and can be tracked. On the similar lines watermarked open source software repositories should be able to track each fork-off and flag OSS violations if necessary.

----
848. (THEORY and FEATURE) 1 August 2020 - Number crunching - Sublogarithmic Prime power encoding of huge integers by Computational Geometric Factorization and 2060 bits integer factorization quadcore benchmark - related to 2 and all sections on Text Compression and Compressed Sensing, Factorization and its applications, Multiplicative Integer Partitions of NeuronRain Theory Drafts
----

It is often necessary to store numeric bigdata succinctly - Numeric compression vis-a-vis Text Compression. Usual binary storage requires logN bits. From Unique Factorization Theorem every integer can be expressed uniquely as product of powers of consecutive primes. Number of prime factors of an integer N is kloglogN by Hardy-Ramanujan theorem. Thus huge integers could be stored as concatenated strings of exponents of consecutive primes (primes are implicit and prior factorization is assumed). Identifying the powers of prime factors is the multiplicative partition problem and sum of prime powers in unique factorization is defined by Big Omega. Thus storing the powers of prime factors (partition of Big Omega - all possible huge integers could be generated by partition function of Big Omega) and prime factors could be efficient (sublogarithmic space) compared to storing a huge integer N by logN bits if following holds:

```
      Average length (bits) of each prime exponent in Big Omega of
multiplicative partition = l1
      Average length (bits) of each prime in multiplicative partition = l2
      Total number of bits required to store huge integer = (l1 + l2) * kloglogN
      if (l1 +l2) * kloglogN < logN, prime exponent encoding of integers is
sublogarithmic:
            => (l1 + l2) < logN / kloglogN
            => For integers of low prime factors (Little Omega) prime exponent
encoding is quite
            feasible because of high upperbound for l1 + l2
```

Code example code/Spark_PrimePowersEncoding.py invokes the Computational geometric PRAM-NC-BSP factorization for 2060 bits integer
(9933353950998727431474807777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777761444238267790503463029217) implemented in NeuronRain by subprocess Python feature on quadcore + Python 3.7.5 and factors are persisted to a JSON list as under (only few small and large factors have been found including non-prime factors). This is the largest integer factored by far in NeuronRain - factors are printed from 22 seconds onwards and depth of the non-uniform logdepth circuit is 186:

[103,
96440329621346868266745706580366774541531823085221143473570658036677454153182308
52211434735706580366774541531823085221143473570658036677454153182308522114347357
06580366774541531823085221143473570658036677454153182308522114347357065803667745
41531823085221143473570658036677454153182308522114347357065803667745415318230852
21143473570658036677454153182308522114347357065803667745415318230852211434735706
58036677454153182308522114347357065803667745415318230852211434735706580366774541
53182308522114347357065803667745415318230852211434735706580366774541531823085221
1434735706580366774541531823085220984895517163014596728439, 673,
14759812705793057104717396400858510813934290903087336965494469209179461779758956
57916460293874855539045732210665345880799075449892686148258213637114082879313191
34885256727753013042760442463265643057619283473666831764900115568763414231467723
29536073964008585108139342909030873369654944692091794617797589565791646029387485
55390457322106653458807990754498926861482582136371140828793131913488525672775301
30427604424632656430576192834736668317649001155687634142314677232953607396400858
51081393429090308733696549446920917946177975895657916460293874855539045732210665

3458807990754498926861482582136371116559046460312783748929]

Theoretically, by equivalence of Parallel RAM and Nick's class (depth of NC circuit = polylog PRAM time, size of NC circuit = number of PRAMs), previous integer requires O(N/(logN)^186) = constant*2^(2060)/ (2060)^186 = constant*5527 PRAMS

References:

848.1 Prime Omega Functions - Little Omega and Big Omega - https://en.wikipedia.org/wiki/Prime_omega_function - Little Omega in terms of partition function
848.2 Prime Big Omega - http://oeis.org/wiki/Omega(n),_number_of_prime_factors_of_n_(with_multiplicity)

----

859. (THEORY and FEATURE) Numeric Compression by Unique Integer Factorization - updated - Spark
NC-PRAM-BSP 2063 bits integer factorization - Euclidean Semiprimes - 11 August 2020, 16 August 2020

Spark_PrimePowersEncoding.py has been updated to parse the JSON factors file written by DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py after renaming ./DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.factors to Spark_PrimePowersEncoding.factors in subprocess.call("mv"). Factors are parsed from JSON and their bits length are printed. Small semiprime integer 6541 of 13 bits has been factorized and the length of factors 211 and 31 are printed (code/testlogs/Spark_PrimePowersEncoding.log2.11August2020). Euclid's algorithm has been applied to create semiprime integer 6541 - by multiplying consecutive primes and incrementing by 1:

      (2*3*5 + 1) * (2*3*5*7 + 1) = 31 * 211 = 6541

In previous example sum of lengths of prime factors (12.675295499) is slightly exceeded by length of integer (12.67529549909406):

```
Length of integer -  6541  - factorized (bits): 12.67529549909406
=======================================
Length of factor -  1  (bits): 0.0
Length of factor -  211  (bits): 7.721099188707186
Length of factor -  6541  (bits): 12.67529549909406
Length of factor -  31  (bits): 4.954196310386876
```

Similarly huge semiprimes could be found and the encoding is independent of radix (Numeric compression is valid for any log N to base M M-ary representation for M >= 2 because prime factors and their exponents can be encoded in base M - compression ratio remains same).

Huge 2063 bits integer (arbitrary) -
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999392838238928
93829892838298398298392898392893892893892898829389283989223982983989999999999999
9999999999999999999999999298828388823822323232323299999459999 - has been factorized (largest integer factorized thus far in NeuronRain) finding compression scheme for which is time consuming because of huge multiplicity, complexity of isolating prime factors and finding their exponents - excerpts from code/testlogs/Spark_PrimePowersEncoding.log.2063bits.11August2020 print two factors in 14 seconds:
('Factors are: (', 29, ',',

344827586206896551724137931034482758620689655172413793103448275862068965517241379310344827586206896551724137931034482758620689655172413793103448275862068965517241379310344827586206896551724137931034482758620689655172413793103448275862068965517241379310344827586206896551724137931034482758620689655172413793103448275862068965517220442697894101320654925458565458565272358375651479099755132047906344455999413241034482758620689655172413793103448275837890634097373183559735286206877931, ') (at ', 'Thu, 02 Jul 2020 08:19:01 GMT', ')')

Following are few examples of Sublogarithmic space Numeric Compression by Unique Factorization:
----
*) 1024 = 2^10 in binary has 10 bits. But by encoding 2^10 in unique prime factor basis - set of ordered pairs of the form (prime_factor,exponent) - as (2,10) which requires 2 + 4 = 6 bits only.
*) 60466196 = 2^10*3^10 = 6^10 is 26 bits integer while unique prime factor encoding (2,10)(3,10) requires only 2 + 4 + 2 + 4 = 12 bits (space for ordered pair parenthesization is an additional kloglogN which has been neglected). If each parenthesis is of 3 bits for "(",","," and ")" delimiters, total space increases to 12 + 3 + 3 = 18 bits in unique factor scheme but still less than 26.
*) CPU instruction set microcodes might have to compute exponentiation and products of prime exponents which increases per instruction time in spite of compressed space.

1129. (THEORY and FEATURE) People Analytics - Feasibility of Automated Talent Recruitment - Fame Versus
Merit - Some Examples - related to 365,420,443,572,613,632,753,1128 and all sections on People Analytics, Fame, Intrinsic Merit, Majority Voting, Sybils, Social Network Analytics, Interview LTF/PTF, BKS Conjecture, Learning LTFs - 28 December 2020

From literature cited in the references (1129.1) on Fame versus Merit, there is an empirical and statistical evidence of Fame being exponentially related to Merit:

" ... Fame, F, grows exponentially with achievement, F(A) = exp(β × A),..."

and achievement is equivalent to intrinsic merit (e.g acdemic publications). Fame is modelled as memepool which evolves over time (Diffusion of Opinion in Social Networks) according to differential equation d(Fame)/dt = v(Achievement). Resemblance of this DE to Recursive Mistake Correction Tree model of Experiential Learning is uncanny.

Section 1128 defines and implements a weighted state machine model of career transition - every human is non-deterministic indefinite state machine defined by choices made (Legendary "Road not taken"). Following example usecases motivate the necessity for automated objective recruitment of talent vis-a-vis subjective majority voting based choice of cadidates in People Analytics (Assumption: Each voter/interviewer uses an Interview LTF which is both subjective and objective multiple choice questionnaire of n questions denoted by q(i) and weighted by w(i) - q1*w1 + q2*w2 + ... + qn*wn):

Example 1:

If intrinsic merit of a candidate is m and there are k voters deciding the candidature (perceived merit
or Fame), for each vote being denoted by v(i):

m < v(1) < v(2) < ... < v(k)

from assumption - "Intrinsic Merit Lowerbounds Fame in the absence of Sybils" -

derived from statistical evidence of exponential separation between the two. $v(1)$ is the least upperbound Fame for Merit m implying every voter overrates candidate (Interview LTF succeeds for candidate).

Example 2:

If intrinsic merit upperbounds Fame:

$$v(1) < v(2) < \dots < v(k) < m$$

implying every voter underrates candidate (Interview LTF fails for candidate)

Example 3:

If intrinsic merit is an average (weighted mean) of Fame:

$$v1 < v2 < \dots < vx < m < vy < \dots < vk$$

implying some voters underrate while others overrate candidate. This is the most prevalent scenario in
real world setting (Interview LTF neither fails nor succeeds for candidate)

Of these, Example usecase1 is supported by statistical evidence. Example1 and Example2 are consensual but yet inaccurate. Previous scenarios demonstrate error probability of real world manual Interview processes which are a mixture of Majority Voting and LTF, thus semi-intrinsic. Replacing manual interviews by objective LTF which is dynamically machine learnt from People profile datasets (e.g Career Transition Automaton, Source Lines of Code for IT domain) minimizes errors and is more absolute. From Computational Learning Theory, LTFs can be learnt in polynomial time in the presence of classification noise.

Career Polynomials - Inner Product Spaces - Example 4:

Career Transition of a profile could be plotted as points on polynomials and similarity of two profiles p and q could be ascertained by inner product spaces of interpolated career polynomials defined as:

$$\langle p, q\rangle = p(t0)q(t0) + p(t1)q(t1) + \cdot \cdot \cdot \cdot + p(tn)q(tn)$$

where p(ti) and q(ti) are the values (e.g remuneration) of polynomials at time ti. Usual algebraic notations of orthonormal basis apply and p,q are orthogonal if $\langle p,q\rangle$ is 0 (Gram-Schmidt Orthogonalization). For two profiles making extremely different career choices $\langle p,q\rangle$ reaches its infimum - Example:

$$p(0) = 0, p(1) = 100000, p(2) = 0, p(3) = 100000$$
$$q(0) = 100000, q(1) = 0, q(2) = 100000, q(3) = 0$$

$\langle p,q\rangle = 0$ (Orthogonal) implying profiles p and q made diametrically opposite career choices during similar time points.

References:

1129.1 Theory of Aces: Fame by chance or merit? - [Simkin-Roychowdhury] - https://arxiv.org/pdf/cond-mat/0310049.pdf - Equation 2
1129.2 Measuring Celebrity - [Lamport] - Microsoft Research - https://lamport.azurewebsites.net/pubs/celebrity.pdf - "... A preliminary conclusion would be that, in science, fame is based only on publications, but celebrity also depends on TV appearances ..."
1129.3 MEASURING FAME QUANTITATIVELY. V. WHO'S THE MOST FAMOUS OF THEM ALL? (PART 2) - [Schulman] - dBHa unit of Fame - https://www.improbable.com/wp-content/uploads/2016/08/MEASURING-FAME-part2-2016-09.pdf
1129.4 Learning Linear Threshold Functions in the presence of Classification Noise - [Bylander] - https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.481&rep=rep1&type=pdf
1129.5 Polynomial time algorithm for learning noisy linear threshold functions - [Blum-Frieze-Kannan-Vempala] - https://www.math.cmu.edu/~af1p/Texfiles/learn.pdf
1129.6 Measuring Fame - [Schulman] - https://www.improbable.com/2009/02/28/measuring-fame-quantitatively-iv-whos-the-most-famous-of-them-all/

1131.(THEORY and FEATURE) Bibliometrics - Merit of Academic Publications - Transformers Attention Model implementation - related to 702,707 ,839 and all sections on Named Entity Recognition, Bibliometrics, Novelty Detection, People Analytics and Embedding in Vector Spaces - 3,4,5,21 January 2021

1.Transformer attention models are recent deep learning advances in NLP replacing Recurrent NNs and LSTMs and are widely used in machine translation and text analytics. BERT and GPT based text writing systems have massive pretrained transformers underneath.
2.Transformer attention model learns Query weights (Q), Key weights (K), Value weights (V) for embedding xi for word i defined by:

```
      Qi = xi * qi
      Ki = xi * ki
      Vi = xi * vi
      Attention(Q,K,V) = softmax(Q * K^T/sqrt(d^k))*V, d^k = dimension of Keys vector
      a(i,j) = attention of word i to word j = qi * kj
      softmax(xi) = e^xi/Sigma(e^xj) which normalizes sum to 1
```

3.Weights are learnt by Gradient Descent Perceptron code for which has been implemented in Transformers_PerceptronAndGradient.py (invoked by Transformers.py) which is based on PerceptronAndGradient.py in NeuronRain AstroInfer but replicated for Transformer specific enhancements.
4.This commit implements transformer model in Transformers.py and Transformers_PerceptronAndGradient.py which are invoked from Spark_Word2vec.py for learning previous attention weights for word2vec embeddings of academic publications. More attention to a word indicates the importance and Novelty associated with it which is particularly necessary in analyzing patent prior art.
5.Transformers object is initialized as:

```
          def
__init__(self,wordembedding,queries,keys,values,expected,dimension)
```

which are input training gradient descent args for bibliometrics word2vec embedding, prior weights for queries-keys-values, expected attention and dimension of keys vector.
6.Logs testlogs/Spark_Word2Vec.log.5January2021 for Spark 3.0.1 + Python 3.7.5 execution depict the attention weights learnt by Gradient Descent.
7.Named Entity Recognition PoS Conditional Random Field implementation in NeuronRain has non-statistical alternative which resembles a transformer sequence model pipeline.

References:

1131.1 Transformers - https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)

1140. (THEORY and FEATURE) Teapot Shards and Space filling - related to 135,649,819,752 and all sections on Space filling, Equal and Unequal Circle packing, Grid filling, Integer Linear Programming and Cellular automata, Quantum Versus Classical Computation, Parallel Pseudorandom Generators, Berry-Esseen Central Limit Theorem - 12 May 2021

Bocherds Teapot Shards experiment for predicting number of shards teapot would break into has uncanny similarities to Randomized NC Parallel Space Filling of many natural random parallel processes (Rain drops falling at random in parallel, Distribution of randomly strewn sand particles,...). Teapot shards which are randomly created in parallel are of unequal sizes and fill a rectangular space. NeuronRain Randomized NC Cellular automaton parallel PRG space filling algorithm assumes equal sized circles of infinitesimal radii sweeping the 2D plane which is an alternative formulation of Circle Packing

Problem. Filling a 2D space at randomly chosen ordinates (which are centroids of circles of small radii) requires classical parallel PRGs (Tygar-Reif,Applebaum Linear strech PRG in NC). Teapot shards are fragmented at unequal random shapes and predicting number of shards by Quantum computation is equivalent to predicting total number of randomly chosen ordinate points in Classical Parallel PRG Cellular Automaton Grid filling version e.g finding total number of rain drops - rainfall is a massive natural parallel random computation deserving a quantum supremacy analogue and a quantum PRG might be suitable for simulating rain - if randomly chosen ordinate centroids are random variables set to 1 from 0 implying a filling, sampled average of sum of binary random variables converges to expected value (mean) by Central Limit Theorem.

References:

1140.1 Teapot Shards - https://www.youtube.com/watch?v=sFhhQRxWTIM
1140.2 Teapot Shards and Quantum supremacy - https://www.scottaaronson.com/blog/?p=5460 - "...I do, however, have one substantive disagreement. At one point, Borcherds argues that sampling-based quantum supremacy itself fails his teapot test. For consider the computational problem of predicting how many pieces a teapot will break into if it’s dropped on the ground. Clearly, he says, the teapot itself will outperform any simulation running on any existing classical computer at that task, and will therefore achieve “teapot supremacy.”..."

1141. (THEORY and FEATURE) Digital Watermarking - refactored - Image and Video functions - related to 841 and all sections on Merit of Large Scale Visuals, Antitamper/Antispoof/Cybercrime analytics, Image Segmentation and Classification, ThoughtNet - 18 May 2021

DigitalWatermarking.py implementation has been refactored to define 2 separate watermarking functions for Videos and Images which thus far had definition only for Video files (watermark_video and watermark_image). Few example group photo image files (Copyright: 95CSE PSG Tech, Coimbatore) of author (.jpg and .png) have been watermarked by Python 3.7.5 + OpenCV2 overlay(). Digital watermarking overlay is a primitive image classifier - overlay of similar segmented images causes high number of superimposed image segments (or) facegraphs of similar segmented images are more isomorphic and high number face vertices of overlayed images get superimposed exactly. Vertices of Facegraphs of the overlayed segmented images are stacked creating a multiplanar graph (in which every vertex is a stack) which is a visual version of ThoughtNet if thoughts are visualized.

1145. (THEORY and FEATURE) Causal and Graphical Event Models, Collatz Conjecture, Economic Datasets, Complementary Functions and Sequences, Ramsey 2-Coloring, Pseudorandomness, Quantitative Finance, Chaos, Multifractals, Discrete Fourier Transform - related to 1144 and all sections on Economic Merit, Complement functions-sequences, 2-coloring and Event Timeseries analysis - 8 June 2021,19 June 2021

Collatz conjecture states:
$x(n+1) = x(n)/2$ if $x(n)$ is even and $3x(n) + 1$ if $x(n)$ is odd and $\{x(n)\}$ sequence always ends in 1.

Collatz map is a chaotic fractal which exhibits pseudorandomness and generalized Collatz conjecture ($g(n) = a(i)*n+b(i)$, $n=i \bmod P$ for rational $a(i)$ and $b(i)$ and $g(n)$ is integer) is undecidable. NeuronRain implements a Chaotic PRG based on conventional actuarial and fractal logistic maps (Verhulste-Pearl-Reed and Mandelbrot). Collatz conjecture could be useful for modelling Economic Event

Timeseries as Graphical Event Model E.g stock price quote tickers alternately demonstrate ebb and tide which are explained by Chaotic Mandelbrot sets (Multifractals - https://www.scientificamerican.com/article/multifractals-explain-wall-street/). Collatz sequence is complementarily 2-colored by odd and even integers determined by rule earlier. Collatz conjecture for economic datasets (quarterly GDP data of Boom-Recession, Demand driven-Supply driven) is a plausible usage for Graphical Event Models by following hypothetical rephrasal:

x(n+1) = Boom/2 (Recession), if x(n) = Boom (simulates penalty for supply glut after boom)

and

3*Recession + 1 (Boom) if x(n) = Recession (simulates incentive for demand growth after recession)

Stock prices are influenced by global cues and sectorwise stocks fall or rise according to ripple effect of global events making stream of stock quotes an event timeseries for Graphical Event Model. Boom is tentatively defined as > 7% GDP growth QoQ while Recession is defined as < 2% GDP growth over two consecutive quarters. Thus GEM and CEM are heavily related to Chaos and Non-linear dynamics. Basic dictum of market event causality cycle goes by: Boom causes supply glut, supply glut causes low demand and recession, and low supply caused by recession spurs demand which is thus a cyclical CEM and GEM. Multifractal model of stock markets defines stream of multiple local exponents for logistic instead of one exponent. Discrete Fourier Transform of stream of stock quotes could reveal hidden periodicities - For stream q0,q1,q2,...qN:

X(k) = Sum_n=0_to_N-1(x(n) * e^(2*pi*n*k/N)), k=0,1,2,...,N-1

X(k) are the Fourier spectrum frequencies.

1147. (THEORY and FEATURE) ThoughtNet Hypergraph - Difficulty in unique determination of a hyperedge - 24 June 2021 - related to all sections on ThoughtNet Hypergraph Evocation, Evocation WordNet, Contextual Multiarmed Bandit model of ThoughtNet, Reinforcement Learning and Word Sense Disambiguation, Cognitive Psychology

----

NeuronRain postulates and implements a Contextual Multiarmed Bandit model of thought evocation named "ThoughtNet" which is a hypergraph of thoughts and every vertex of multiplanar ThoughtNet is a stack. Reinforcement Learning based Contextual Bandit retrieves thought of maximum reward for each evocative. For example, three stack hypervertices c1,c2,c3 of ThoughtNet could be evocatives or modal classes and each hypervertex stack is populated by thoughts t1,t2,t3,...,tn. Hypergraph nature of ThoughtNet allows a thought hyperedge to connect multiple hypervertices of ThoughtNet. Evocation problem is to uniquely determine a hyperedge on utterances of evocatives c1,c2 and c3. As opposed to contextual multi armed bandit solution which ranks thoughts evoked by numeric rewards and chooses the maximum, unique thought could be retrieved non-numerically if intersection of sets of hypervertices has size 1. Any intersection of size > 1 would neccessitate disambiguation - Example:

c1 - {t1,t3,t7}
c2 - {t1,t2,t5,t6}
c3 - {t1,t3,t4,t5}

For previous ThoughtNet hypergraph of three hypervertices c1,c2,c3 and hyperedges t1,t2,t3,t4,t5,t6,t7, evocatives c1,c2,c3 uniquely determine {t1} by intersection while evocatives c1,c3 retrieve thoughts {t1,t3} which have to be ranked. Datastructure used for intersection (e.g type of hashing) is crucial.

```
##########################################################################################
################################################################
#Course Authored By:
#-----------------------------------------------------------------------------------------
----------------------------
#K.Srinivasan
#NeuronRain Documentation and Licensing: http://neuronrain-
documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#-----------------------------------------------------------------------------------------
----------------------------
##########################################################################################
###############################################################
*/
```

This is a non-linearly organized, code puzzles oriented, continually updated set of course notes on C language. Though C is at present
rapidly diminishing in usage because of many high level languages and big data tools for cloud processing, a primer in C is essential for
grasping nuances of operating system memory management, networking and threading
----

References:
----

1. C Puzzle Book - Alan R.Feuer
2. C Programming Language - ANSI C - [Kernighan-Ritchie]

2 February 2017

Arithmetic Operators:

```
#include <stdio.h>

main()
{
        int x;
        x = -3 + 4 * 5 - 6; printf("%d\n",x);
        x = 3 + 4 % 5 - 6; printf("%d\n",x);
        x = -3 * 4 % -6 / 5; printf("%d\n",x);
        x = (7 + 6) % 5 / 2; printf("%d\n",x);
}

/*
        Above prints:
        11 (-3 unary operator has precedence over binary, * has precedence over
+ and -. (-3)+(20) -6 = 11 )
        1 ( remainder operator (modulus) % has precedence. 3 + (4) - 6 = 1)
        0 ( (((-3 * 4) % -6)/5) = (-12 % -6) % (-1) = 0. Operator precedence is
same and evaluation is from left to right )
        1 ( ((13) % 5) / 2 = 3 % 2 = 1 )
*/
```

----
10 March 2017

Pointers in C are basic low level primitives of allocating memory and indexing them by contiguous bytes (malloc/free). Most of the vulnerabilities found in software are traced back to memory buffer overruns. Buffer overruns are conditions in which a pointer to a contiguously allocated block of memory goes past the bounds. Two major bugs in modern internet history are OpenSSL's Heartbleed and Cloudflare's Cloudbleed both being bugs in array bounds checking crept into software.

1. HeartBleed fix for OpenSSL - https://git.openssl.org/gitweb/?

p=openssl.git;a=commitdiff;h=96db902 - This bug was introduced into OpenSSL in HeartBeat extensions for SSL connections keep-alive feature. Pointer was getting past without record length checks enforced in the fix commit diffs
2. CloudBleed fix for Cloudflare proxies HTML parsing - https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug/. This is similar to HeartBleed buffer overrun, which happened during equality check for end of buffer pointer (which bypasses an increment check condition within Ragel HTML parser).

Duff's Device - 7 August 2018

Duff's device is C loop unrolling design pattern which mixes switch and looping for fall throughs. Example code is in code/duffs_device.c which decrements a counter in a loop - switch statement falls through when counter is not divisible by 5 to case statement selected by switch and number for chosen case statement is decremented by fall through to other cases till end. This is an optimized loop unrolling inspired by jmp statements with in loops in assembly language routines. Previous example prints (in "case: counter" format):

```
0: 29
1: 28
2: 27
3: 26
4: 25
0: 24
1: 23
2: 22
3: 21
4: 20
0: 19
1: 18
2: 17
3: 16
4: 15
0: 14
1: 13
2: 12
3: 11
4: 10
0: 9
1: 8
2: 7
3: 6
4: 5
0: 4
1: 3
2: 2
3: 1
4: 0
```

----
10 February 2020 - C++ Class simulation in C, Function pointers, Private and Public
----

Example code class_in_c.c simulates object oriented programming of C++ in C. C only has the facility of
struct {} declaration which is exploited to declare few private and public variables (variables are structs containing primitive type unions of type "struct variable". A flag to distinguish private variables is part of the struct variable) and member function pointers for class "struct classinc". Member private and public variables are assigned to by set<type>() functions which check private flag of a variable and deny access for private variables. Member

function function1() clears the private flag and invokes set<type>() reinstating the private flag after for member variable thus simulating privileged member function access of private. set<type>() functions simulate type overloading. Logs in code/logs/class_in_c.log.10February2020 trace the public and private access by functions pointed to.

----

7 May 2020 - Simulation of loop by recursion, void typecast, Binary search tree example

----

Functional programming defines loop imperatives by recursion. Code example recursionasloop.c demonstrates
incrementing a counter by recursion without recourse to for loop. Function loop() defines two clauses
for simulating increment by recursion and to check if an array representation of a binary tree is indeed
a binary search tree by verifying if inorder traversal of the tree is sorted ascending and adheres to subtree comparison ordering. Both int object and int[] array are passed to loop as void* where arg is reinterpret_cast()-ed. Two binary trees one of which is not a BST are tested. Pairwise subtree comparison and loopless increment are printed to logs/recursionasloop.log.7May2020.

6,7,8 June 2020 - Dynamic Nested For Loop (variant of question from UGC NET/SLET/JRF)

Writing hardcoded nested for loops (of constant depth) in C is trivial. But for generalizing loops to
arbitrary nested depths is less straightforward because loop body has to be dynamically indented which
requires on-the-fly code generation. Code example dynamicnestedloop.c implements a contrived recursive
version of nested looping of arbitrary depth. Function nestedloop() accepts from,to and function pointer
to body of the loop. nestedloop() is recursively invoked along with loopbody() till depth is decremented
to 0. Subtle difference to usual n depth for loop of range 10 - counter is incremented to 10^n + 10^(n-1) + ... instead of 10^n (and thus powerful compared to hardcoded nested for loop) which is filtered by depth==1 check within loopbody(). Removing depth==1 clause increments counter to 1110 and not 1000 (depth 3 nesting). Iterative version of dynamic nesting is again non-trivial.

2,3,4,7 July 2020 - Arrays as Pointers, Recursive exponentiation in binary (variant of questions
22,48 from GATE 2020)

Questions 22 and 48 of GATE 2020 have been merged to code example multidimensionalarrays.c which
demonstrate:

- 2-d arrays as objects and their multidimensional pointer dereferencing
- exponentiation function pp(a,b) which raises a to the exponent b after converting b
to binary string by modulo 2 division recursively (to-binary function - tob())

pp() function obtains the binary string of b from tob() which is looped through and tot is updated

only for bits "1" in binary string of b. Auto variable array arr[] is allocated on stack and dereferenced
by tob().
##
##

##
##
Course Authored By:

K.Srinivasan
Personal website(research): https://sites.google.com/site/kuja27/
NeuronRain GitHub and SourceForge Documentation: http://neuronrain-documentation.readthedocs.io/

##
##

This is a non-linearly organized, continually updated set of course notes on miscellaneous topics in Computer Science (algorithms, datastructures, graph theory etc.,). Puzzles/Problems/Questions are sourced from many text books.

----
9 February 2017

A graph G with 100 vertices numbered from 1 to 100 has edges between vertices p and q iff p-q=8 or p-q=12. Number of connected components of G is:

Number of components of an undirected graph is number of subgraphs with no path between them. Above set of adjacent vertices can be written as merger of 2 arithmetic progressions of common different 8 and 12 as below:

1,9,13,17,25,33,37,41,49,...
2,10,14,18,26,34,38,42,50,...
3,11,15,19,27,35,39,43,51,...
4,12,16,20,28,36,40,44,52,...
5,13,17,21,29,37,41,45,53,...
6,14,18,22,30,38,42,46,54,...
7,15,19,23,31,39,43,47,55,...
8,16,20,24,32,40,44,48,56,...

Further sets are repetitions of the above. Of the previous 8 sets 4 have common elements between them and form a connected component. Thus there are 4 connected components in the graph G. This is equal to GCD of two common differences 4 and 8.

----
15 February 2017

Linear Programming and Simplex

Linear programming is defined as: Given an objective function to minimize or maximize with set of constraints expressed as inequalities, finding set of solutions which maximize or minimize the objective function. For example following is a linear program:

maximize x1 + 6x2 = P (written as -x1-6x2+P = 0)
subject to constraints:
2x1 + x2 <= 10
x1 + 5x2 <= 9

There are 3 types of solutions possible for a Linear Program:

*) Basic feasible solution - which is the set of vectors which satisfy the constraints. There are two types of feasible solutions:
 *) Bounded feasible - the satisfying vectors maximize the objective function to a bounded value
 *) Unbounded feasible - the satisfying vectors maximize the objective function to an unbounded value

*) Infeasible - there are no vectors which satisfy the constraints.

Simplex algorithm solves an LP in following tableau steps:

while (there are no negative indicators in ObjF row)
{
*) Choose least positive pivot row and most negative pivot column as pivots:
 - most negative indicator column is chosen as pivot column
 - divide the last column entries by pivot column entries and choose least positive row as pivot row

	x1	x2	s1	s2	P		
c1	2	1	1	0	0	10	10/1 = 10
c2	1	5	0	1	0	9	9/5 = 1.8 (least positive pivot row - c2)
ObjF	-1	-6	0	0	1	0	

most negative pivot column (x2)

*) the pivot entry in the intersection of c2 and x2 is 5. Interchanging the row and column variables row2 is x2. Divide by pivot to obtain 1:

	x1	c2	s1	s2	P	
c1	2	1	1	0	0	10
x2	1/5	5/5	0	1/5	0	9/5
ObjF	-1	-6	0	0	1	0

and do row operations till all other row entries in pivot column are zero.

*) perform a row mapping operation - Rc1 = Rc1 - Rx2:

	x1	c2	s1	s2	P	
c1	9/5	0	1	-1/5	0	41/5
x2	1/5	1	0	1/5	0	9/5
ObjF	-1	-6	0	0	1	0

*) perform a row mapping operation - RObjF = 6Rx2 + RObjF:

	x1	c2	s1	s2	P		
c1	9/5	0	1	-1/5	0	41/5	
x2	1/5	1	0	1/5	0	9/5	
ObjF	1/5	0	0	6/5	1	54/5	(no negative indicators, iteration stops)

}

From previous last iteration feasible solution satisfying the constraints is :
x2=9/5 , x1=0 and maximized objective function value is 54/5 for x1 + 6x2

= P

LL,LR and Shift-Reduce Parsers

Parsing with reference to modern programming languages is defined as:
Given a set of Context Free Grammar Production Rules produce a parse tree in either top-down or bottom-up fashion resulting in a single non-terminal symbol.

LL parsing is Left-Right, Top-Down scanning of the symbols with Leftmost Derivation while LR parsing is Left-Right, Bottom-Up scanning of the symbols with Rightmost Derivation. Shift-reduce parsing which is an LR parsing is implemented with a stack,lookahead symbol(usually 1) and yet to be scanned set of terminals.

Leftmost Derivation applies production rule to leftmost non-terminal in a sentential form(top-down parsing). Rightmost Derivation applies a production rule to rightmost non-terminal in a sentential form(bottom-up parsing)

Example CFG for simple arithmetic operation (+ and *):

```
      E = E * T
      T = E + E
      E = id
```

Previous CFG is applied to parse a sentence:

```
      x * x + x
```

Leftmost Bottom-Up Derivation Parser:

```
      x * x + x
      E = E * T
      E = id * T (E = id)
      E = id * E + E (T = E + E)
      E = id * id + E (E = id)
      E = id * id + id (E = id)
```

Rightmost Top-Down Derivation Parser (Complete Reversal of Previous Parsing Steps):

```
      E = E * T
      E = E * E + E (T = E + E)
      E = E * E + id (E = id)
      E = E * id + id (E = id)
      E = id * id + id (E = id)
```

Bottom-Up Shift-Reduce Parsing with a stack for previous Rightmost Derivation:

Stack	Lookahead	Yet-to-be-scanned	ProductionRule
-	id	* id + id	-
(Shift)			
E	*	* id + id	E = id
(Reduce)			
E	*	id + id	-
(Shift)			
E*	id	+ id	E = id
(Reduce)			
E*E	+	id	-
(Reduce)			
E*E	+	id	-
(Shift)			
E*E+	id	-	E = id
(Reduce)			
E*E+E	-	-	T = E + E
(Reduce)			
E*T	-	-	E = E * T

```
      (Reduce)
      E                 -                 -                 -                 -
```

Shift-Reduce parser actions at each step are determined by valid combinations of top of the stack and the lookahead - a pushdown automaton state diagram. Shift step advances to next unscanned symbol and Reduce step applies a production to top of the stack based on lookahead.

----

23 February 2017

Evaluate prefix expression:

      *+3+3^3+333

This can be evaluated with two stacks as below by popping topmost and storing it in another stack, evaluating next two operands and operator on first stack and pushing back the result and topmost popped from second stack to first stack:

```
      3           3           3           3           3           =  2205
      3           6           729         732         735
      3           3           3           3           *
      +           ^           +           +
      3           3           3           *
      ^           +           +
      3           3           *
      +           +
      3           *
      +
      *
```

An egg dropped from a floor between 1 to 100 breaks if floor value is x or above. With 2 eggs find minimum number of drops required to find the highest floor from which egg does not break.

Binary search on the floors 1 to 100 would require log(100) eggs. With 2 eggs binary search won't work. For this number 100 is split into intervals of powers of 2 (this is the idea behind logarithmic counters) - 2,4,16,32,64. First egg is dropped sequentially from 2,4,16,32,64th floors. Drop when first egg breaks is denoted as m. Maximum number of drops is 6 for first egg. Then the second egg is sequentially dropped from 2^(m-1) floor to 2^m floor which is equal to 2^(m-1). Thus total drops required is m + 2^(m-1). For maximum value of m=6, this is 6 + 36=42 . Thus with 42 drops, highest floor from where egg does not break can be found. Trivial one level binary search would require 50 (100/2) drops for second egg.

----

28 February 2017

There is a list with integers in range 1..n with length n+1 elements. Find the repeated element (in least space without hash tables)

Brute force requires O(n) space and hash tables require O(n) space too. This list has a pattern that all elements are in range 1..n with one repeat. Sum of this list can be computed in O(logN) space - single counter incremented - which is Total = n(n+1)/2 + repeating_element. Total - n(n+1)/2 is the repeating element. Any other value of |Total - n(n+1)/2| which is not in 1..n indicates more than one repeating element and more than 1 repetition per repeating element. For arbitrary length of list, and multiple repetitive elements, this problem reduces to Element Distinctness Problem which has non-trivial algorithms (NlogN lowerbound with sorting, decision tree model, property testing, quantum etc.,).

----

6 March 2017

Types of Latches and FlipFlops:

SR latch (Set-Reset):

S	R	Q
1	0	1
0	1	0

JK Latch(SR Latch with Toggle Q and !Q states for 00 and 11):

J	K	Q
1	0	1
0	1	0

D FlipFlop:

D	Q
0	0
1	1

----
20 September 2017

Bernoulli Trials and Geometric Distribution:

Question: While tossing a coin what is the expected number of coin tosses before Head appears? (i.e expected number of failures before success)

Answer: Probability of head = Pr(H) = 1/2.
Let N be the number of coin tosses before Head appears. N is a Bernoulli trial random variable.
Pr(N) is the probability of Head appearing after N coin tosses.
Pr(N) = (1-Pr(H))^(N-1)*Pr(H) (first N-1 tosses is streak of Tails and Nth toss is Head i.e TTTTT...TH)
Expected value of N = E(N) = sum_1_to_infinity(N*(1-Pr(H))^(N-1)*Pr(H)) = (1/2) + (1/2)^2 + (1/2)^3 + ... = 1/1-0.5 = 2 = (1-Pr(H))/Pr(H)
Expected value of N = (1-Pr(H))/Pr(H) = 0.5/0.5 = 1

This problem has applications in predicting binary streams of 0s and 1s in network traffic. But probability of 1 and 0 need not be a fair coin toss i.e Pr(1) != Pr(0). If a network packet traffic stream has Pr(1) = 0.75 and Pr(0) = 0.25, Expected number of bits after which bit 1 appears = (1-0.75)/0.75 = 1/3 and Expected number of bits before 0 appears = (1-0.25)/0.25 = 3

It has to be noted that Expectation of Appearance of Head and Expectation of Failures before Head Appears are two different random variables.

Reference: https://en.wikipedia.org/wiki/Geometric_distribution

----
13 October 2017

How can all nodes of same level in a binary tree be retrieved (of least time complexity)?

Assuming array representation of a binary tree, each node of the array of size 2^(logN) for N tree nodes is filled from the tree top-down and left-right and missing nodes are blank in array. In this array representation, nodes of level l are the consecutive nodes from indices 2^(l-1) to (2^l) - 1 and can be printed by iterating through array.

----
27 November 2017

----
847. (THEORY) One way functions - How can a hashmap be inverted i.e keys and values are interchanged? - related to all sections on Hardness Amplification, Locality Sensitive Hashing, Separate Chaining, One-way functions and Parallel algorithms
----

If the hashmap is perfect hash function, there are programming language supports e.g bidirectional dictionaries - bidict in Python, Ruby Hash#invert - http://ruby-doc.org/core-1.9.3/Hash.html#method-i-invert. If the hashmap is not perfect and has collisions each key is mapped to multiple values. Inverting this amounts to unwinding each of the collision buckets as keys and map them to the erstwhile key as value. Bruteforce algorithm is O(N) time where N is total number of values in the map. An alternative version of this problem is : Given a value v in the map find the key k which maps to v. Theoretically, this problem is parallel to Proving existence of One Way Function which is defined as Probability[f(finverse(y))==y] < epsilon for a function f and its inverse. For a hashmap H, this is equivalently stated as Probability[H(Hinverse(v))=v] where Hinverse is hashmap inverted. Difference is Function has unique range value per domain key while a hashmap has multiple values. Sublinear parallel algorithm to invert a hashmap which is not perfect could be as below:

```
      for each key k in hashmap H1 accessed in parallel
      {
            each element v=H1[k] in collision bucket is accessed in parallel and
added as key in a new hashmap H2 as H2[v] = k
      }
```

Both keys and buckets are accessed in parallel. Theoretically if hashmap is on a PRAM, previous algorithm is O(1) parallel time and requires
O(N) PRAMs.

----
2 January 2018 - Find Sum of Bit differences (Distance) between all possible pairs of integers in an array
----

Example: For array [1,2,3] sum of bit differences for pairs:

```
       distance(1,2) = 01,10 = 2
       distance(1,3) = 01,11 = 1
       distance(2,3) = 10,11 = 1
                       --------
                           4
                       --------
```

Trivial bruteforce algorithm has to compute all possible pairs and find the distance of each pair which is O(NC2*logM) where N is the size of array and M is the largest integer in array.

Following algorithm finds the bit distance sum in O(logM*N) time for all pairs of subsets better than above bruteforce:


```
       (#) Represent the array of size N and largest number M as N * logM 2-
dimensional array.
       (#) For each column (O(logM)):
       {
               Find the number of rows R having same bits - 0 or 1 (O(N)).
               /*
                       Number of pairs differing in this bit position are :
```

```
                        Set of all possible pairs - (Set of all possible pairs
having both 1s + Set of all possible pairs having both 0s)
                        This contributes to the sum of distances for all bits
               */
               BitDifferenceSum += NC2-((N-R)C2 + RC2)
       }
```

Following example executes above algorithm for array: [5,6,7,8,10] represented as 5*ceil(log10) 2-dimensional array - there are 5C2=10 possible sets of pairs:

```
       0101
       0110
       0111
       1000
       1010
       ---------------------
       2+1+3+4+1+3+2+4+3+1=24
       ---------------------
       There 3C2+2C2 pairs of (1,1) and (0,0) which are subtracted:
       5C2-(3C2+2C2) + 5C2-(3C2+2C2) + 5C2-(3C2+2C2) + 5C2-(3C2+2C2) = 10-4 +
10-4 + 10-4 + 10-4 = 6+6+6+6 = 24
```

An implementation of this algorithm has been included in NeuronRain AsFer Pairwise Encoded String Pattern Mining function in:
https://github.com/shrinivaasanka/asfer-github-code/blob/master/cpp-src/asferencodestr.cpp

----
9 January 2018 - Find the heavier ball

Question: There are 8 balls of same size. One of them is heavier and all other 7 are of equal lesser weight. Find the heavier ball
in just 2 weighings.

Answer: Assuming a balance, split the 8 balls into 2 sets of 4 each placed on either trays of the balance. One of the trays goes down
Heavier ball is one of the 4 in going down tray. Split the set having heavier element into 2 sets of 2 each placed on either trays. This is
second weighing. One of the trays goes down. Heavier ball is one of the 2 in down tray. Third weighing is not necessary because one ball each
can be removed from each tray and there are two possibilities: 1) The trays level - ball taken from going up tray is heavier, 2) the trays don't level - both balls taken are of equal weight and down tray has heavier ball.

Previous problem has deep rooted connection to finding the larger value in a balanced search tree. Two trays of the balance are two subtrees of a BST in each weighing and query time is O(logN).

----
834. (THEORY and FEATURE) Social Network Analysis - People Analytics - Unique Identification - 12 January 2018 - How would you uniquely identify an object from a plethora of objects? - Birthday Paradox, Contextual Name Parsing, PIPL Name syllable based unique person search - related to all sections of People Analytics
----

That is, how can a universally unique identifier be created? This is an open-ended question. In the context of Public Key Infrastructure,
creating unique non-repetitive keys has been a challenge - Sections 4.1 and 4.2 in https://factorable.net/weakkeys12.conference.pdf on repetitive and factorable keys. Boost C++ has UUID generation algorithm based on
https://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx. UUID is a misnomer

because there is a negligible probability of collisions known as birthday paradox. Birthday paradox is problem of any two persons in a congregation having same birthday. If a unique id is of maximum value N and size of population is M (M < N):

probability of person2 having different id than person1 = (N-1)/N

probability of person3 having different id than person1 and person2 = (N-2)/N

probability of person4 having different id than person1, person2 and person3 = (N-3)/N

... and so on.

Probability of all persons in population having different UUIDs = (N-1)(N-2)....(N-M) / N*N*N...N
Probability of some of the population having same UUIDs = 1 - [(N-1)(N-2)....(N-M)/ N^M]

This problem has direct bearing on Tabulation Hashing where two objects have same hash digest and are hashed to same bucket. Minimizing hash collisions therefore requires increase in denominator and lowering numerator i.e N >> M.

This problem is all the more serious in People Analytics where a person has to be uniquely identified in ,say, social networks undisputably (e.g there are abundant duplicate social profiles of same name, photos etc., and handles are sometimes prefixed as "real<name>"). Usually it suffices that N is logM bit integer. Digital Unique IDs stored on cloud often are vulnerable to attacks. An example Unique Person Search therefore might involve a decision tree of depth O(logM) and UniqueID is not just an integer but is a concatenation of various not-so-forgeable human non-digital features (e.g age, voice metric, biometric, blood group, height, birthmarks, educational qualifications, work experience, circle of human acquaintances, non-invasive unique private event in a person's timeline history etc.,)

Name Syllable based Unique Person Search (by scraping PIPL.com data) is described and PIPL.com python API search of similar persons is implemented in NeuronRain AstroInfer commit https://gitlab.com/shrinivaasanka/asfer-github-code/commit/d855882c3e8c2f97f6709c6f4a03728f95377ca1 (and in GitHub and SourceForge AstroInfer repositories). Contextual Name Parsing to parse First and Last Names from Full Name based on ID context has been implemented in NeuronRain AstroInfer commit https://gitlab.com/shrinivaasanka/asfer-github-code/commit/3ae9054a1311176c87304ed85e3835510657fc8b (and in GitHub and SourceForge AstroInfer repositories)

----

835. (THEORY and FEATURE) Unsorted Search - Kernel lifting and Computational Geometric search - 19 January 2018 - How would you find needle in haystack? - related to all sections on Computational Geometric Hyperplanar Point Location, Locality Sensitive Hashing and Shell Turing Machines

----

Question can be translated to finding a number in a set of unsorted numbers. Trivial brute-force search is of O(N). Can this be improved? There are some ways to go about:

1. Searching set of numbers is one dimensional. If each integer is mapped to a tuple in n-dimensional space, then searching reduces to shooting a ray from origin and doing a sweepline which is computational geometric reduction of search problem. For example, 2387283 is mapped to [2,3,8,7,2,8,3] which is 7 dimensional tuple. Each dimension has maximum of 10 values - 0 to 9. Each number in set is thus a tuple in this space. This has some parallels to kernel trick and Support Vector Machines in Machine Learning which lift data in low dimension to higher dimension (Mercer polynomial Kernels). Previous

reduction depends on the radix of the numbers. Representing in hexadecimals, widens the ease of search. This reduction creates a new d-dimensional dataset from one dimensional and distance of any point from origin is the L2 norm upperbounded by sqrt(d*100).This shrinks the distance of any point in the haystack from other point from O(N) to O(sqrt(d)).

2. Represent the numbers in dataset as M*N matrix where N is the size of set and M is the number of digits. Each contiguous subset of columns are transformed into a hashtable of size N corresponding to O(M^2) substrings. Searching the number is then looking up each digit/substring of the query in respective digit's/substring's hashtable. For example, 123,235,534,323,333,343 form the contiguous column subsets:

```
1 2 3 12 23
2 3 5 23 35
5 3 4 53 34
3 2 3 32 23
3 3 3 33 33
3 4 3 34 43
```

Next create 5 hashtables for 3*2-1 digit substrings: [1,2,5,3,3,3],[2,3,3,2,3,4],[3,5,4,3,3,3],[12,23,53,32,33,34],[23,35,34,23,33,43]
Searching 323 then requires 5 lookups for 3,2,3,32,23 in each of previous tables successively.
Searching 545 returns true (false positive) if 4th and 5th contiguous substring columns are not considered which fail the lookup of 54 and 45.
Number of substrings of an integer string of length M is M + M-1 + M-2 + ... + 1 = M(M-1)/2 and each hashtable is for a substring.
Trivial M-length string hashtable is not created (which obviates this algorithm). This is O(M^2) total lookup assuming O(1) per table hash lookup and no sorting is needed. Overhead is the preprocessing step of creating hashtables.
If M*M << N, this O(M^2) lookups is too less than bruteforce search of O(N).
Number of possible M-digit strings are 10^M which is maximum possible value of N, and M^2 << 10^M.

Algorithm 1 can search a query point by Algorithm 2.

This algorithm has been implemented in NeuronRain AsFer - https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/UnsortedSearch.py

Example unsorted search 1 - https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/testlogs/UnsortedSearch.log.21January2018

```
---------------------------------------------------
...
Is Queried integer 99455 in unsorted array: False
Is Queried integer 43 in unsorted array: True
Is Queried integer 31 in unsorted array: True
Is Queried integer 17 in unsorted array: True
Is Queried integer 3278 in unsorted array: False
Is Queried integer 333 in unsorted array: False
Is Queried integer 29 in unsorted array: True
---------------------------------------------------
```

Example unsorted search 2 - https://raw.githubusercontent.com/shrinivaasanka/asfer-github-code/1b543857e41824a1ed0e10211ceb1e0906bd670c/python-src/testlogs/UnsortedSearch.log.23March2018

```
----------------------------------------------------
Is Queried integer 99455 in unsorted array: False
Is Queried integer 43 in unsorted array: True
Is Queried integer 31 in unsorted array: True
Is Queried integer 17 in unsorted array: True
Is Queried integer 3278 in unsorted array: False
Is Queried integer 333 in unsorted array: False
Is Queried integer 29 in unsorted array: True
```

Is Queried integer 327 in unsorted array: False
Is Queried integer 115 in unsorted array: False

Previous matrix representation of the unsorted numerical dataset replaces the traditional array storage and requires O(M^2*N) space. Sequential search time of O(M^2) derived previously can be shrunk to O(1) by searching each substring parallelly in M^2 processors. If number of digits is upperbounded by a constant, storage is O(N) which is of same space complexity as array storage. With respect to overhead of creating hashtables, the benefit of Unsorted Search is the ability to exit the search if prefix or suffix is not found and entire number string need not be matched and is significant optimization for large M. Insertion of an M-digit integer into this Hash storage is O(M^2) for all substrings whereas in traditional lists appending is O(1). But lookup in traditional unsorted list is O(N) and O(M^2) in Unsorted Search of Substring hashtables. Thus the latency of Unsorted Search is only in adding an integer to list and is O(N*M^2) for all integers.

Present implementation in NeuronRain AsFer creates hashtables only for all prefixes and suffixes of strings. Similar to Bloom Filters, Unsorted Search of single digit hashtable lookup can have only False Positives and not False Negatives - if a number does not exist it can be wrongly flagged as match, but if it exists match is always correct. False Positives are removed by matching all prefixes and suffixes. Hashtables for just prefixes and suffixes are sufficient because Set of Unsorted Integers can be stored in a TRIE M-ary Tree datastructure in which numbers in list are marked in internal nodes by a delimiter and all root to nodes paths are the prefixes. Mismatch is flagged by an absence of root-to-node path prefix in TRIE. Maximum size of the TRIE is 10^M.

How can a Binary Search Tree be verified? - 12 March 2018

An obvious solution is to do an inorder traversal of BST and verify the list of traversed nodes is sorted ascending or descending. This is O(n) for number of nodes in tree n. Array representation of BST can be directly scanned for sortedness.

----
850. (THEORY and FEATURE) How would you avert collision of two trains speeding along on same track in opposing directions? - Critical Sections and Synchronization - 3 November 2018, 9 August 2020 - related to all sections on Program Analysis, Software Analytics, Software Transactional Memory, Lockfree datastructures, VIRGO Bakery Algorithm implementation, VIRGO Read-Copy-Update of NeuronRain Theory Drafts
----
(*) This puzzle is based on P and V Binary Semaphores - Mutexes (Dijkstra):

```
      Process1(Train1)             Process2(Train2)
      ----------------             -----------------
      P()                          P()
      track(critical_section)            track(critical_section)
      V()                          V()
```

(*) P():

```
      while mutex==0
      {
            wait()
      }
```

(*) V():

```
      mutex=1
```

(*) Difference between Binary Semaphores and this puzzle is how to avert

collision after both processes have entered critical section by mistake (track) which is more susceptible to happen (e.g signals P and V fail) in real life than operating systems (even in Operating Systems this can occur when atomic instructions for P and V fail theoretically).
(*) In modern architectures, synchronization is supported in hardware by a fine-grained primitive - compare and swap instruction - atomically which has following algorithm:

```
      compare_and_swap(p,old,new)
      {
            if(p != old)
                  return false
            p = new
            return true
      }
```

(*) Multiprocessor Hardwares support memory fencing instructions, bus locking etc., for serializing instructions issued before fence and to allow maximum of one thread of execution to load and store.
(*) VIRGO32 and VIRGO64 linux kernels of NeuronRain implement various userspace and kernelspace synchronization primitives - Software Transactional Memory Lockfree Userspace C++ usecase, Read-Copy-Update userspace C++ usecase and Bakery algorithm locking synchronization kernel driver which can be module imported (in two variants - 1 or 2 for loops - parametrized)

Some assumptions for solving this puzzle:

(*) Assuming electric traction or maglev, shutting down power supply is an obvious option. But this requires atleast one of the trains to be aware that the other train is on same track or a third party's intervention. Assuming zero-knowledge by everyone (e.g no GPS location info) and both trains are unstoppable, there exists a non-zero probability of collision.

Variant of the previous critical section when two tracks cross
----
Problem is to cross two tracks by overlap. Railway tracks are undirected cyclic graphs having 2 parallel edges. Overlapping two track edges makes it non-planar and following example gadget schematic creates a planar crossing of two parallel track edges (disconnects the graph at the crossing and adds two point vertices (and)). This is a dimensional critical section which allows a planar intersection (reduces 3D to 2D):

```
\ \  / /
 \ \/ /
  )  (
 / /\ \
  / /  \ \
```

References:

850.1. Compare and Swap can simulate Lock-free synchronization and mutexes - [Maurice Herlihy] and P and V Semaphores - https://en.wikipedia.org/wiki/Compare-and-swap and https://en.wikipedia.org/wiki/Semaphore_(programming)
850.2. UNIX Internals: New Frontiers - [Uresh Vahalia - EMC] - Multiprocessor Synchronization Issues - Pages 193,200,201 - Semaphores, Convoys, Necessity of Atomic test_and_set instruction in multiprocessors. Convoys are formed in Semaphores when a queued process in Scheduler waiting on a lock is prevented from being runnable because another process holds the lock.
850.3. UNIX - [Maurice J.Bach - AT&T] - Page 396 - Monitors as Synchronization Primitives in Multiprocessor UNIX - Monitors differ from Semaphores by modularizing scope of critical section to subroutine and serializing access by processes.

850.4. Operating Systems - [Milan Milenkovic - IBM Corporation] - pages 110-112 - Section 3.4.4 - Compare and Swap instruction in IBM 370 series - Global is copied to Oldreg local registers of interleaving concurrent processes P1,P2,...Pn. Each process computes the local copy of new value of Global in Newreg registers. Condition Oldreg == Global is checked in each process which guarantees Global has not been tampered with by other processes and condition codes are set. Only one process goes beyond this sanity check if condition is satisfied and updates the Global with Newreg local copy. All other processes fail the Global == Oldreg test because Global is updated by only one of the process by its Newreg, and all other local copies of processes are out of sync which branch to loop again to read new value of Global. Similarities to Train collision example earlier have to be noted - Both the trains have to simulate a local copy of the critical section information of the track which, for example, could be Global Positioning System (GPS) location of begin-end of critical section.
850.5. Operating Systems - [Silberchatz-Galvin-Gagne] - pages 197-200 - Section 7.3 - TestAndSet and Swap instructions (no comparison) - Synchronization Hardware

----
1 May 2020 - Insertion in numbered lists

Q: It is trivial to insert an element in unnumbered lists and linked lists. How can an element be inserted
in numbered lists without re-ordering - e.g element 6 is inserted in 1,2,3,4,5,6,7,8,9,10 between 6 and 7 making the list 1,2,3,4,5,6,6+1,7+1,8+1,9+1,10+1 (all elements after 6 are re-numbered):

A1: (*) TRIE solution is to append a Dewey decimal suffix to 6 as 6.1 - In previous example 1,2,3,4,5,6,7,8,9,10 becomes 1,2,3,4,5,6,6.1,7,8,9,10 preserving sorted order without renumbering.
A2: (*) List is represented as variable expression array which is lazy evaluated before and after insertion - 1+x,2+x,3+x,4+x,5+x,6+x,7+x,8+x,9+x,10+x for a global shared pointer variable x initialized to 0. All element expressions upto insertion point are evaluated for x=0. After insertion of 6+x next to 6+0, x is incremented by 1 and all element expressions after insertion point are evaluated:

    1+0,2+0,3+0,4+0,5+0,6+0,6+1,7+1,8+1,9+1,10+1

This is not a sequential renumbering because global increment of x is reflected at once across all elements of variable array.

----
849. (THEORY and FEATURE) 29 May 2020 - Probability of Odd number of Heads in Coin Toss - related to
all sections on Majority Voting, Efficient Population count, Voting Analytics of NeuronRain Theory
Drafts

Q: A coin is tossed n times. What is the chance that the Head will present itself odd number of times (IIT-JEE 1970)

A: Total number of possible toss strings from alphabet {H,T} are 2^n. Number of possibilities of odd number of Heads

    = number of ways of arranging odd number of Hs in toss string
    = number of ways of choosing strings of 1H + number of ways of choosing strings of 3H + number of ways of choosing strings of 5H + ...
    = (n,1) + (n,3) + (n,5) + ... = 2^(n-1)

=> Probability of odd number of heads = 2^(n-1) / 2^n = 0.5

For binary strings H and T are replaced by 1 and 0 and previous probability corresponds to a randomly chosen binary string to be of odd parity. Bipartisan (2-colored) voting patterns are random binary strings.

----

846. (THEORY) 2,3,4 June 2020 - Union of Probabilities, Bayes Rule, Venn Diagrams, Pairwise and Mutual independence - related to all sections on Majority Voting and Correlated Majority, Statistical dependence of voters

----

Q: Three outcomes of an experiment are w1,w2 and w3 such that w1 is twice as likely as w2 which is twice likely as w3. What are the probabilities of w1,w2 and w3 (UPSC Civil Services - IAS(Main) - 2003)

A: P(w1) = 2P(w2), P(w2) = 2P(w3)

Straightforward solution neglecting dependence of outcomes:

4P(w3) + 2P(w3) + P(w3) = 1
P(w3) = 1/7, P(w2) = 2/7 and P(w1) = 4/7

Dependent events (if "as likely as" implies dependence):

=> By union bound for dependent events w1,w2 and w3, P(w1 U w2 U w3) = P(w1) + P(w2) + P(w3) - P(w1 /\ w2 /\ w3) = 1

7P(w3) - P(w1 /\ w2 /\ w3) = 1

By Bayes Rule for Total Probability if W is the total outcome event space, probability of total outcome:
      P(W) = Sum(P(W/Wi)*P(Wi))
from which per event conditional probability is derived as:
      P(Wi/W) = P(W /\ Wi)/P(W)

By General Multiplication Chain Rule for dependent events:
      P(w1 /\ w2 /\ w3) = P(w1 / (w2 /\ w3)) * P(w2 / w3) * P(w3)

=> 7P(w3) - P(w1 / (w2 /\ w3)) * P(w2 / w3) * P(w3) = 1 solving which requires knowledge of conditional probabilities

P(w1 /\ w2 /\ w3) is the intersection of three circles for P(w1),P(w2) and P(w3) in Venn Diagram - overlap of circles is the dependence of events.

Independent events - Pairwise and Mutual:
----
if the outcomes of experiment are mutually independent, P(w1 /\ w2 /\ w3) = P(w1)P(w2)P(w3) = 8P(w3) = 8/7 > 1 for P(w3) = 1/7 implying outcomes are not mutually independent (which is stricter than pairwise independence for w1-w2, w2-w3 and w1-w3)

References:

846.1.Probability and Statistics with Reliability,Queueing and Computer Science Applications - [Kishor Shridharbhai Trivedi] - Chapter 1 - Page 33 - Problem 4 - General Multiplication Rule (GMR)
846.2.Correlated Majority Voting - https://en.wikipedia.org/wiki/Condorcet%27s_jury_theorem#Correlated_votes - "...Condorcet's theorem assumes that the votes are statistically independent. But real votes are not independent: voters are often influenced by other voters, causing a peer pressure effect...In a jury comprising an odd number of jurors {\displaystyle n} n, let {\displaystyle p} p be the probability of a juror voting for the correct alternative and {\

displaystyle c} c be the (second-order) correlation coefficient between any two correct votes. If all higher-order correlation coefficients in the Bahadur representation[6] of the joint probability distribution of votes equal to zero, and {\displaystyle (p,c)\in {\mathcal {B}}_{n}} {\displaystyle (p,c)\in {\mathcal {B}}_{n}} is an admissible pair, then: The probability of the jury collectively reaching the correct decision (Condorcet probability) under simple majority is given by: {\displaystyle P(n,p,c)=I_{p}\left({\frac {n+1}{2}},{\frac {n+1}{2}}\right)+0.5c(n-1)(0.5-p){\frac {\partial I_{p}({\frac {n+1}{2}},{\frac {n+1}{2}})}{\partial p}},} {\displaystyle P(n,p,c)=I_{p}\left({\frac {n+1}{2}},{\frac {n+1}{2}}\right)+0.5c(n-1)(0.5-p){\frac {\partial I_{p}({\frac {n+1}{2}},{\frac {n+1}{2}})}{\partial p}},} where {\displaystyle I_{p}} I_p is the regularized incomplete beta function...."

----

830. (THEORY and FEATURE) Set Partition Analytics, Voting Analytics, Ramsey 2-coloring - m men and n women are to be seated in a row so that no two women sit together. If m > n, show that the number of ways in which they can be seated is m! (m+1)! / (m-n+1)! - (Question from IIT-JEE 1983) - related to all sections of NeuronRain Theory Drafts on Set Partitions, Theoretical Electronic Voting Machines, Ramsey coloring of sequences, Complementary Sets, Avoidance Patterns in Primes - 18 June 2020, 19 June 2020

A1) m men can be seated in m! ways and for each such permutation, n women can be seated in (m+1)
vacancies - including an extra seat beyond a row - between each male in (m+1)Pn = (m+1)!/(m+1-n)! ways. Multiplying, total number of such arrangements are m! * (m+1)!/(m-n+1)!

A2) But previous answer assumes set of m men is partitioned into m parts of size 1 each. Generalizing
it to a set partition of m males of arbitrary sized parts (unrestricted set partition) no two women might still sit together - a woman sandwiched between any 2 male parts - if number of male parts exceed number of women:
- m men could be partitioned in B(m) ways where B(m) is the Bell number = number of partitions of a set of size m = Sum of Stirling numbers of second kind
- Number of ways m men could be partitioned into k parts = Stirling number of second kind {m,k} = 1/k!*Sum_over_i=0,...,k((-1)^i (k,i) (k-i)^n)
- n women could be seated in each of the k vacancies for all k > n and each of k vacancies (parts) are created in {m,k} ways
- Total arrangements possible = Sum_over_k[{m,k}*kPn] for all k > n

A3) Problem is symmetric:
- instead of partitioning m males first, n females could be partitioned in {n,1} ways where {n,1} is the Stirling number of second kind for number of partitions of size 1 parts (or n!).
- m males could be partitioned into k parts in {m,k} ways (from A2)
- k parts of set of males could be seated in (n+1) vacancies between each female - including extra seat beyond end of row - in Sum_over_k[(n+1)Pk] ways for all k = n,n+1 and each of the k parts could be found in {m,k} ways
- k should not be less than n because it might juxtapose two women
- Total arrangements possible = {n,1}*Sum_over_k[{m,k}*(n+1)Pk] for all k = n,n+1

A2 and A3 are equivalent:
Sum_over_k[{m,k}*kPn] for all k > n = {n,1}*Sum_over_k[{m,k}*(n+1)Pk] for all k = n,n+1

Computer Science Theory Applications of such an avoidance are numerous - biased binary strings, patterns in primes, 2-colored sequences, bipartisan voting patterns are some of them (by replacing men-women by 0-1 or Red-Blue colors).

----
845. (THEORY) Finding average number of comparisons per element in a Binary Search Tree - based on question 2 of ETS GRE Major Field Test model - Computer Science Subject GRE - https://www.ets.org/Media/Tests/MFT/pdf/mft_samp_questions_compsci.pdf - DFS traversal of a Binary Search Tree - 21,22 July 2020 - related to 751,768

Following is an example of a complete binary search tree which stores a sorted array of integers in adjacency list:

```
            8 - 4,12
            4 - 2,6
            12 - 10,14
            2 - 1,3
            6 - 5,7
            10 - 9,11
            14 - 13,15
```

Inorder traversal of the BST produces the sorted list 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15. Problem is to find the average number of comparisons required to find an element. Generic expression for average number of comparisons could be derived as:

```
      Number of comparisons per root-to-leaf traversal * Number of root-to-leaf
traversals
      --------------------------------------------------------------------------
-----------
                  Total number of elements in Binary search tree
```

Average root-to-leaf depth of the search tree = d
Number of comparisons per root-to-leaf traversal = 1 + 2 + 3 + ... + d = d(d+1)/2 by Gauss Formula
Number of root-to-leaf traversals = 2^(d-1)
Total number of elements in Binary search tree = 2^d - 1
=> Average number of comparisons per element of the Binary Search Tree = d(d+1) * 2^(d-1) / [2*(2^d -1)]
For previous example d=4 (including root) => Average comparisons = 4*5 * 8 / [2*15] = 80/15 = 5.33 while Worst case number of comparisons is O(logN) ~ 4k which is counterintuitive because worst case complexity is exceeded by average case complexity (unless k > 1).

Previous is an approximate estimate which does not subtract overlapping traversals - Every root-to-leaf DFS traversal left-to-right marks the nodes "visited" for which comparisons have been already computed. Comparisons for visited nodes must be excluded from total comparisons. In previous example following is the DFS traversal which marks the visited nodes and number of comparisons per root-to-leaf traversal in parentheses:

```
      8-4-2-1 (1+2+3+4=10) - 4 nodes
      8-4-2 (visited), 3 (4) - 1 node
      8-4 (visited), 6-5 (3+4=7) - 2 nodes
      8-4-6 (visited), 7 (4) - 1 node
      8 (visited), 12-10-9 (2+3+4=9) - 3 nodes
      8-12-10 (visited), 11 (4) - 1 node
      8-12 (visited), 14-13 (3+4=7) - 2 nodes
      8-12-14 (visited), 15 (4) - 1 node
```

Thus there are 49 total unique comparisons for 15 nodes ignoring the visited nodes and thus average comparisons = 49/15 ~ 3.2666...
=> worst case complexity 4k is more than average case complexity 3.2666... for k >= 1.
/*
##
##

###
#Course Authored By:
#---
#K.Srinivasan
#NeuronRain Documentation and Licensing: http://neuronrain-documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#---
##
*/

This is a non-linearly organized, code puzzles oriented, continually updated set of course notes on C++ language. This
complements NeuronRain course materials on Linux Kernel, Cloud, BigData Analytics and Machine Learning and covers
fundamentals of C++.
----

22 February 2017

An example on C++ templates and Runtime type identification:

Example code snippet in code/templates.cpp implements a simple templatized book class with book type string as template parameter. Template
book is instantiated with template and typename keywords and type T can be any subject type passed in as template parameter. Template class
EBook derives from base class Book<T>. A subtlety in this example is absence of default constructor for Book<T> causes following compiler
error:

```
g++ -g -o templates -I/usr/local/include -L/usr/local/lib -std=c++14 *.cpp
templates.cpp: In instantiation of ‘EBook<T>::EBook(T) [with T = std::__cxx11::basic_string<char>]’:
templates.cpp:47:28:   required from here
templates.cpp:36:2: error: no matching function for call to ‘Book<std::__cxx11::basic_string<char> >::Book()’
  {
  ^
templates.cpp:18:2: note: candidate: Book<T>::Book(T) [with T = std::__cxx11::basic_string<char>]
  Book(T type)
  ^
templates.cpp:18:2: note:   candidate expects 1 argument, 0 provided
templates.cpp:8:7: note: candidate: Book<std::__cxx11::basic_string<char> >::Book(const Book<std::__cxx11::basic_string<char> >&)
 class Book
       ^
templates.cpp:8:7: note:   candidate expects 1 argument, 0 provided
templates.cpp:8:7: note: candidate: Book<std::__cxx11::basic_string<char> >::Book(Book<std::__cxx11::basic_string<char> >&&)
templates.cpp:8:7: note:   candidate expects 1 argument, 0 provided
```

```
---------------------
Adding default constructor:
      Book()
        {
        }
removes compilation error and following is printed:
---------------------------------------------------
Instantiating Book of type Maths
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating Book of type ComputerScience
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating Book of type Physics
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating Book of type History
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating EBook of type English
Read book
Read book
Read book
Read book
Read book
-----------------------------------------------------------------
```

In above example, read_book() is a virtual function in superclass Book which can be overridden in derived classes. Previous output indicates how dynamic polymorphism works and read_book() of Book<T> is invoked from derived class EBook<T>. Type information is printed by typeid keyword of C++. This example is built using G++ with C++14 standard compiler option.

----

858. (THEORY and FEATURE) 17 July 2017,9 August 2020 - Self-Aware Software, Quines - related to all sections on Formal Languages, Program Analysis, Software Analytics

----

Question: How can a program print its source itself as output ? [Quine - self-aware code]

Answer: Theoretically, there exists a lambda function with a fixed point i.e f(x)=x. Unix/Linux binaries are stored in ELF format which have debugging information embedded in DWARF entries (as set of DIEs - Debugging Information Entries). There are utilities like objdump and dwarfdump which display the DIEs. For example, following is the DWARF dump of asfer executable pointing to the compilation source directory in DW_AT_comp_dir:

```
root@Inspiron-1545:/media/shrinivaasanka/6944b01d-ff0d-43eb-8699-cca469511742/
home/shrinivaasanka/Krishna_iResearch_OpenSource/GitHub/asfer-github-code/cpp-
src# objdump --dwarf=info asfer |more

asfer:     file format elf32-i386

Contents of the .debug_info section:

  Compilation Unit @ offset 0x0:
   Length:        0x16f7e (32-bit)
   Version:       4
   Abbrev Offset: 0x0
   Pointer Size:  4
 <0><b>: Abbrev Number: 1 (DW_TAG_compile_unit)
    <c>   DW_AT_producer    : (indirect string, offset: 0x35560): GNU C++14
5.2.1 20151010 -mtune=generic -march=i686 -g -std=c++14 -fstack-pro
tector-strong
    <10>   DW_AT_language    : 4   (C++)
    <11>   DW_AT_name        : (indirect string, offset: 0xe304):
DecisionTreeClassifier.cpp
    <15>   DW_AT_comp_dir    : (indirect string, offset: 0x7933):
```

/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea24/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/cpp-src

```
    <19>   DW_AT_ranges      : 0x210
    <1d>   DW_AT_low_pc      : 0x0
    <21>   DW_AT_stmt_list   : 0x0
```

There are libdwarf libraries for programmatically querying ELF DWARF DIEs. Thus a wrapper reflection code invoking the dwarfdump on executable can be written to print the source(assumes fresh compilation everytime).

----

12 September 2017

Question: Remove duplicates in a string in-place (no extra space) e.g shrink aabbcba to abc

Answer: One possible solution is to sort the string (by an in place sorting algorithm like quicksort) in ascending order of unicode value e.g aabbcba is sorted to aaabbbc. Implement the string as linked-list of literals, Scan the string linked list and remove repetitive alphabets and repeat till duplicates are removed. Code example for this is at code/remove_duplicates.cpp uses STL string sort() to sort the string, STL erase() to simulate a linked list node erasure, and shows how STL iterators for strings begin() and end() are applied.

```
---------------------------------------------------
str:9aaabbbcccddddddddeeeffggjjksss
str:9aaabbbcccddddddddeeeffggjjksss
str:9aaabbbcccddddddddeeeffggjjksss
str:9aabbbcccddddddddeeeffggjjksss
str:9aabbbcccddddddddeeeffggjjksss
str:9aabbcccddddddddeeeffggjjksss
str:9aabbcccddddddddeeeffggjjksss
str:9aabbccddddddddeeeffggjjksss
str:9aabbccddddddddeeeffggjjksss
str:9aabbccdddddddeeeffggjjksss
str:9aabbccddddddeeeffggjjksss
str:9aabbccdddddeeeffggjjksss
str:9aabbccddddeeeffggjjksss
str:9aabbccddddeeffggjjksss
str:9aabbccddddeeffggjjksss
str:9aabbccddddeefggjjksss
str:9aabbccddddeefgjjksss
str:9aabbccddddeefgjksss
str:9aabbccddddeefgjksss
duplicateexists(): true
str:9aabbccddddeefgjkss
str:9aabbccddddeefgjkss
str:9aabbccddddeefgjkss
str:9aabbccddddeefgjkss
str:9abbccddddeefgjkss
str:9abccddddeefgjkss
str:9abcddddeefgjkss
str:9abcdddeefgjkss
str:9abcddeefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
```

str:9abcdefgjks
str:9abcdefgjks
str:9abcdefgjks
str:9abcdefgjks
str:9abcdefgjks

----
2 October 2017 - Placement New and Operator Overloading

C++ provides mechanisms to override default storage allocation by overloading operator new. There are two types of operator new(): Plain overload and Placement Overload. Placement new supplies storage as argument to operator new. Code example in code/placement_new.cpp illustrates this
as below (this has been compiled to C++2017 standard). Older ways of overriding pointer this have been described in comments. Recent compilers
do not allow direct *this overrides and prefer operator new. Operator new facility is useful for writing new storage allocators and memory debuggers which can instrument and bypass default memory allocation for profiling. This example also explains the rvalue reference operator && for *this. Rvalue references alias the right side of an assignment while Lvalue references(&) alias the left side of assignment.

```
=====================================
auto allocation
=====================================
this...0xffa08a90
overwriting this...
rvaluethis :0xffa08a90
=====================================
operator new:
=====================================
operator new overloaded and this is from a heap allocator
this...0x96a3e18
overwriting this...
rvaluethis :0x96a3e18
operator delete overloaded and this is freed to a heap allocator
=====================================
placement operator new:
=====================================
this...0xffa08aa8
overwriting this...
rvaluethis :0xffa08aa8
```

----
21 December 2017 - Rvalue References in C++ and Move semantics
----

Rvalue references were introduced in C++11 standard specification. Cloud move implementation of NeuronRain Neuro Currency applies the
move semantics and rvalue references (client,server and header in
https://github.com/shrinivaasanka/asfer-github-code/blob/master/cpp-src/cloud_move/). Traditionally lvalue refers to LHS of an assignment operator and rvalue to RHS of it. For example:

```
      int x=5
```

assigns rvalue 5 to lvalue x. Lvalue references are declared by alias operator & as:

```
      int& y=x
```

and Rvalue references are declared by && operator:

```
      int&& y=10
```

Move semantics in C++ specify moving an object by move constructor (std::move() and operator= overload) vis-a-vis copying an object by
copy constructor. Move constructor is defined in Neuro currency as:

```
      T& operator=(cloudmove<T>&& rvalue) {
      ...
      }
```
and this move constructor is invoked by:
```
      cloudmove<currency::Currency> currency_src(&c1,"localhost");
      cloudmove<currency::Currency> currency_dest(&c2,"localhost");
      ...
      currency_dest = std::move(currency_src);
```
Move differs from Copy by returning rvalue of the argument to std::move() and renders the operand currency_src nullified by moving the resources to lvalue currency_dest.

----

7 August 2018 - Substring/Regular Expression Matcher

Matching a substring within a larger string is regular expression matching problem of writing a DFA. Deterministic Finite State Automatons are usually state transition tables on a graph. String is looped through and state transition table is looked up for next state till accept is reached. Designing this as a recursion saves lot of lines of code. An example recursive regexp substring matcher is in code/regexp.cpp which prints all matching positions of a substring as below:

```
:regexp matchks does not match at 0
 :regexp matchks does not match at 1
 :regexp matchks does not match at 2
 :regexp matchks does not match at 3
 :regexp matchks does not match at 4
 :regexp matchks does not match at 5
 :regexp matchks does not match at 6
 :regexp matchks does not match at 7
 :regexp matchks does not match at 8
 :regexp matchks does not match at 9
 :regexp matchks does not match at 10
 :regexp matchks matches at 11
 :regexp atchks matches at 12
 :regexp tchks matches at 13
 :regexp chks matches at 14
 :regexp hks does not match at 15
 :regexp matchks does not match at 16
 :regexp matchks does not match at 17
 :regexp matchks does not match at 18
 :regexp matchks does not match at 19
 :regexp matchks does not match at 20
 :regexp matchks matches at 21
 :regexp atchks matches at 22
 :regexp tchks matches at 23
 :regexp chks matches at 24
 :regexp hks matches at 25
 :regexp ks matches at 26
 :regexp s matches at 27
```

----

7 September 2018 - Unordered Map, Hash table buckets and Auto Iterator

----

C++ supports hashtables via unordered_map which is initialized either by emplace() or by {{...}} notation.
C++ from 2011 has new kind of iterators similar to Java 8 which automatically identify the type by auto keyword:
```
      auto& it: <container>
```

Bucket containing an entry in the map is accessed by bucket() member function. An example code which populates
an unordered_map by process-clockticks pairs, auto iterates them and prints the buckets is committed in:
code/unordered_map_auto_iter.cpp and logs are committed to code/logs/unordered_map_auto_iter.log.7September2018.

----
10 September 2018 - Unordered Map and for_each()

Previous example for auto iterator has been changed to iterate the unordered map by for_each() primitive
from <algorithm>. This is C++ equivalent of map() in python which invokes a function on each element of the
container. Unordered map has std::pair<> elements accessed by .first and .second members.

----
855. (THEORY and FEATURE) 24 September 2018,9 August 2020 - Fowler-Noll-Vo Hashing, Custom Hash Functions in unordered_map, Nested Template Classes - related to all sections on Locality Sensitive Hashing, Separate Chaining Bucketization
----
FNV or Fowler-Noll-Vo Hashing is a non-cryptographic hash algorithm which has high dispersion and minimizes collisions in same bucket. It iterates through literals in text and multiplies their unicode values by a prime and XORs with an offset. This has avalanche effect - hash is very sensitive to small change in input. An example FNV implementation based on Boost C++ example has been added to course material at code/fnv.cpp. This defines a namespace class and nested fnv templatized struct through which prime number and offsets can be passed as arguments. FNV hashing is widely used in search engines, text processing, MS Visual Studio, memcache etc.,

References:

855.1.Boost FNV example - https://www.boost.org/doc/libs/1_68_0/libs/unordered/examples/fnv1.hpp
855.2.Fowler-Noll-Vo - FNV - Hashing: http://www.isthe.com/chongo/tech/comp/fnv/
855.3.Go Lang FNV package - https://golang.org/pkg/hash/fnv/

----
5 October 2018 - C++ Move-Assign Threads, Unordered Map Rehash and Concurrent Access
----
In C++ threads can be created in C++ specific move-assign paradigm which moves RHS thread object to LHS and
destroys LHS. Move-assign is done by std:thread() operator= overloaded function which takes thread function
and arguments to it as parameters. An example C++ source file threads.cpp has been committed in code/ which
creates 50 thread objects, move-assigns thread objects to them by invoking a function to populate an unordered
map. populate_hashmap() waits for few nanoseconds, makes a key-value pair and places them in unordered map.
Load factor (number of items/number of buckets ratio) is recomputed to by invoking max_load_factor() and
rehash() functions alternately for odd and even values. This is a contrived example to demonstrate concurrent

accesses to a container in C++. Logs for this example are in code/logs/threads.log.5October2018.

----

854. (ThEORY and FEATURE) 28 October 2018,9 August 2020 - Three Distances Theorem and Fibonacci Hashing - related to all sections on Locality Sensitive Hashing, Separate Chaining Bucketization

----

Three Distances Theorem - Proof of Steinhaus Conjecture:
If Phi=(sqrt(5)-1)/2, and sequences of points {Phi}, {2*Phi}, {3*Phi}, ... are plotted in [0...1] y-interval,and successive line segments are inserted in [0...1] y-interval from (k, {k*Phi}) to (n, {k*Phi}) the line segments are of sets of 3 lengths.[{k*Phi} is the fraction obtained subtracting the integer floor(k*Phi) from k*Phi]

An example C++ code which implements this as a hash function to an unordered_map has been described in code/threedistances.cpp.

Following are the size of each line segment sets grepped from log:

```
-----------------------------------------------------------------
# grep "big " logs/threedistances.log.28October2018 |wc -l
68
# grep "bigger " logs/threedistances.log.28October2018 |wc -l
66
# grep "biggest " logs/threedistances.log.28October2018 |wc -l
66
```

References:

854.1.The Art of Computer Programming - Volume 3 - Sorting and Searching - Page 518 - [Don Knuth] - Proof of Steinhaus Conjecture - Theorem S - [Vera Turan Sos]

----

1 November 2018, 2 November 2018, 3 November 2018 - Polymorphism, RTTI, Pure Virtual Functions, Friend classes, Scope Resolution operator, protected and private members, Initializers in Constructors, const correctness

----

C++ specifies polymorphic classes by deriving a base super class by syntax:

```
      class <derived> : <qualifier> <super>
```

code example in code/polymorphism.cpp defines a base class Animal and 2 derived classes: Tiger and Lion.
Keyword protected in derived classes imply the derived class access to base class's protected members. Base
class Animal has a private member which is accessible by the derived classes through friend class declarations
in base class. There are two virtual functions in base class one of which is declared pure and makes Animal
an Abstract Data Type. Derived classes Tiger and Lion implement the pure virtual function in abstract base
class and override the other virtual function. Runtime Type Identification (RTTI) is from typeinfo infrastructureprovided by C++ standard for inferring the type of the object at runtime - typeid() keyword prints the typename
of the object. Constructor Initializers are mentioned by a list of variables suffixed by () operators and values assigned to private member variables. Const qualifier informs the compiler that the function should not alter
the variables (immutables).

Scope resolution operator :: resolves the private (by friendship) and protected members of the super class(by
protected derivative classing). Header cxxabi.h has been included for C++ ABI

name demangling of RTTI typenames.
const disambiguation has been demonstrated by two functions legs() with const and without const qualifier. Both
legs() are invoked by base class pointer Animal* (->legs()) and as member invocation (.legs()) and difference
in behaviour is obvious from logs/polymorphism.log.3November2018.

References:
----
1. The C++ Programming Language - [Bjarne Stroustrup]
2. Essential C++ - C++ in depth series : Bjarne Stroustrup - [Stanley Lippman, Dreamworks] - const example - Section 5.9 - Page 161 - Previous example differs because of g++-6 idiosyncracy: const Animal* is required to invoke legs() having const qualifier.

----
29 November 2018 - Pointers and References (Lvalue and Rvalue)

An example C++ code for miscellaneous permutations of pointers and aliases usage has been committed at code/pointerstew.cpp. C++ pointers which are supersets of C pointers have additional facilities for aliasing to an object location in the form of right value references(&& operator) and left value references(& operator). References or aliases do not consume extra memory storage as opposed to pointers which are object memory locations themselves. Points-to and Reference-to graph of the variables declared in pointerstew.cpp is below
(legend: pointer ######>, rvaluereferences: ==========>, lvaluereferences: ------------->):

```
            psptr ############> ps <========= psrvalueref
            pint1 ############> ps.rvaluerefx1 ====> ps.x
            pint2 ############> ps.lvaluerefx2 -----> ps.x
            func1() parameter y =========> forwarded rvalue of arg to func1()
```

Assigning an lvalue to rvalue of same datatype throws following GCC error:

```
pointerstew.cpp: In function ‘int main()’:
pointerstew.cpp:33:28: error: cannot bind ‘pointerstew’ lvalue to
‘pointerstew&&’
  pointerstew&& psrvalueref=ps;
```

Logs for this example code have some surprising values for rvalue references. Assigning values directly to rvalue references corrupts the rvalue in GCC (shown in logs):

```
      int&& rvaluerefx1=1;
```

whereas std::Forward<int>(1) is required to forward the rvalue to lvalue for any assignment and across function invocations as parameters:

```
      int&& rvaluerefx1=std::forward<int>(x);
```

func1() has been overloaded with parameters and without them. Difference between effect of post-increment of rvalue (ps.rvaluerefx1++;) with and without std::forward() (previous two ways of initializing rvaluerefx1 within pointerstew object) is evident. One time std::move of rvalue and std::forward() of rvalue is demonstrated by static value of xx across multiple invocations in std::move() while rvalues always reflect x dynamically.

References:

1.C++ Programming Language - [Bjarne Stroustrup]
2.C Puzzles - Pointer Stew - [Alan Feuer]

----
856. (THEORY and FEATURE) 26 December 2018 - Bridge and Iterator Design Patterns

- related to all sections on Software Analytics, Program Analysis, Survival Index Timeout and Scheduler of NeuronRain Theory Drafts
----
Bridge is a design pattern mentioned in Gang-of-Four catalog of C++ Design Patterns. Bridge separates implementations and the interfaces by defining implementation itself as an abstract data type. Iterators are the patterns to enumerate iterable containers - arrays, hashmaps, linkedlists etc.,C++ code example bridgeiteratordesignpatterns.cpp which demonstrates how timeout pattern described in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt fits as an amalgamation of Bridge and Iterator Design Patterns, has been committed in C++/code. Timeout is a dictionary of timeout values to lists of objects to timeout. Timeout as a pattern has universal occurrence across whole gamut of software engineering. Code example defines following classes - Timeout and TCPTimeout are interfaces and TimeoutImp and TCPTimeoutImp are implementations which are bridged by a pointer reference Timeout holds to TimeoutImp. This Decoupling is by passing any derivative TimeoutImp object in TCPTimeout constructor which in turns assigns to timeoutimp reference in Timeout. timeout() overridden virtual member function in TCPTimeout invokes imptimeout() unaware of implementation TimeoutImp :

```
      Timeout -------------------- implemented-by ----------------- TimeoutImp
(derived by TCPTimeoutImp)
(derived by TCPTimeout)
```

References:

856.1.Design Patterns - Elements of Reusable Object Oriented Software - [1995] - [Erich Gamma - Richard Helm - Ralph Johnson - John Vlissides] - Bridge and Iterator Patterns - Page 160 - Shared Strings Class - [Coplien] and [Stroustrup]

----
851. (THEORY and FEATURE) 12 February 2019, 9 August 2020 - Software Transactional Memory - related to Program Analysis, Software Analytics, Software Transactional Memory, Lockfree datastructures, Bakery Algorithm, Read-copy-update sections of NeuronRain Theory Drafts
----
Software Transactional Memory is supported by C++ by synchronized blocks of compound statements and transaction_safe directive in function declaration. Software Transaction Memory is the intrinsic facility for transactional rollback or commit of set of statements similar to RDBMS - either all are executed or none. An example transactional memory code has been committed to code/softwaretransactionalmemory.cpp. It declares two functions - function1() executing a synchronized block and function2() declared transaction_safe which is a tighter restriction preventing unsafe code. Compiler error flagged for unsafe function calls have been added in code comments. Curious statement in the code is:

```
            t=std::thread([]{for(int n=0; n < 10;n++) function1(n);});
```

which is a lambda expression doing null capture by [] operator and just invoking the function function1() within lambda expression loop block. Auto iterator variable t is assigned to the thread object. Logs in logs/softwaretransactionalmemory.log.12February2019 show serialized execution of 10 threads instantiated. VIRGO32 and VIRGO64 kernels implement a Bakery algorithm locking primitive as kernel driver while in userspace VIRGO system calls could be wrapped by C++ transaction memory primitives.

References:

851.1.C++ Lambda Expressions - https://en.cppreference.com/w/cpp/language/lambda
851.2.C++ Software Transactional Memory -

https://en.cppreference.com/w/cpp/language/transactional_memory

----
15 February 2019 - Lambda Functions and Capture, Functional Programming - std::function
----
C++ supports lambda functional programming constructs similar to other languages like Python and Java.
An example C++ code which dynamically creates lambda functions and returns them is shown in code/lambdafunctions.cpp. It defines a struct and member function dynamicfunctions() which populates an unordered_map of string-to-int by parameters defined by () operator. It also captures this object from its present scope by [] operator which is internally accessed by on-the-fly lambda function block for the hashmap member. Capturing is intended for data communication between lambda function block and external scope. Member function dynamicfunctions() is returned as std::function object function1 returning int and taking (string,int) as arguments. Returned dynamic function object function1 is invoked like any other function by passing (string,int) arguments twice. Resultant hashmap is printed by auto iterator. Logs for this are committed to logs/lambdafunctions.log.15February2019.

----
852. (THEORY and FEATURE) 23 February 2019,9 August 2020 - Concurrency, Promise and Future Asynchronous I/O - related to Program Analysis, Software Analytics, Software Transactional Memory, Lockfree datastructures, Bakery Algorithm, Read-copy-update, Drone Autonomous Delivery sections of NeuronRain Theory Drafts
----
C++ facilitates asynchronous communication between concurrent threads by Promise and Future. Promise is instantiated by std::promise template and passed on as arguments to threads similar to shared_ptr which are shared mutables within thread functions. Future associated to Promise is acquired at a later time point asynchronously and value set by threads is readable. Code example at code/promisefuture.cpp describes two fictitious train threads having access to Promise and sets the nanoseconds time duration between present and an epoch as its value. std::chrono high resolution clock is invoked for time duration in nanoseconds. Future value for this Promise is later read by get() on Future object. Logs are shown in logs/promisefuture.log.23February2019. C++ SDK asynchronous I/O code for Drone telemetry could be augmented by Promise and Future code blocks.

----
853. (THEORY and FEATURE) 6 February 2020, 9 August 2020 - Read-Copy-Update mentioned in VIRGO Design Document of NeuronRain Theory Drafts has been implemented in userspace - related to Program Analysis, Software Analytics, Software Transactional Memory, Lockfree datastructures, Bakery Algorithm, Read-copy-update sections of NeuronRain Theory Drafts
----
Read-Copy-Update (RCU) has been mentioned as a feature in VIRGO32 and VIRGO64 design documents. Read-Copy-Update which is an efficient synchronization primitive implemented in most OS kernels works quite similar to local CPU caches of global RAM memory:

(*) READ - Read the variable
(*) COPY - Copy it to a temporary variable
(*) UPDATE - Update the temporary variable
(*) WRITEBACK - Write back temporary variable to the actual source

Advantage of Read-Copy-Update is the lack of necessity of mutexes:

(*) multiple concurrent readers have access to an older version of

variable while a writer updates the copy of it
(*) older version of the variable is updated by new after all existing reads of older versions are done and no new read is allowed.
(*) older version is updated by the new version of the writer's working copy.

All the previous steps require no synchronization though it has to be ensured no new reads are performed while older version is updated. This kind of lockfree synchronization is quite useful for multiple readers of a linked list while some writer removes an element of the linked list. [Example of such a necessity is the WCET EDF Survival Index Scheduler design in GRAFIT course material - https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt - which is a set partition of linked list of process id(s) where frequently process id(s) are read by almost every component of OS kernel and deleted by scheduler]. Writer marks the node to delete and updates the links bypassing the deleted node. Thus both node pointed to by new link and deletion-marked node of the linked list are available to existing readers. After all existing queued reads are over, node marked for deletion is really deleted. Efficiency stems from the fact that no locks are necessay for concurrent RCU.

Code example in code/readupdatecopy.cpp implements a C++ class and wraps the RCU assign functionality as its operator= overloaded member function. As evident from example synchronized blocks for software transactional memory and mutexes have been commented. It can be compiled by commandline - g++ -g readcopyupdate.cpp -fgnu-tm -lpthread -o readcopyupdate. Three kinds of copy have been shown - invoking operator=, copy assign of RCU object and copy assign of members. Third clause for copy assign of member has the following schematic:

```
                        valuecopy=value; //READ and COPY
                                cout<<"COPY: valuecopy="<<valuecopy<<endl;
                                valuecopy=rvalue.value; //UPDATE
                                cout<<"UPDATE: lvaluecopy
updated="<<valuecopy<<endl;
                                value=valuecopy; //WRITEBACK

                        ===================

                        value
                        | (READ)
                        V
                  (COPY) valuecopy <- rvalue.value (UPDATE)
                        |
                        V
                        value (WRITEBACK)
```

Logs in code/logs/readcopyupdate.log.6February2020 demonstrate a concurrent read-write by RCU of 200 threads. This code example is in effect a userspace implementation of RCU in VIRGO linux kernel (and is a spillover code of VIRGO repositories in GRAFIT) because reads and writes in thread functions can be replaced by syscall invocations of virgo_get() and virgo_set() preceded and succeeded by a global virgo_malloc() and virgo_free() respectively and no kernel codechange is necessary.

References:

853.1.Read-Copy-Update - https://en.wikipedia.org/wiki/Read-copy-update

----

857. (THEORY and FEATURE) 28 April 2020 - Name filter - C++ STL containers and algorithms - copy,copy_if,shared_ptr,tokenizer - related to all sections on People Analytics, Named Entity Recognition, Name filters (learning proper nouns

in a text)

C++ Standard Templates Library implements algorithms for manipulating containers - for copying,filtering
and erasure. A contrived example C++ name filter class has been defined in namefilter.cpp which accepts a textfile and parses it to filter the words having a substring name pattern. An example list of names from linkedin profile of author is namefiltered for a certain prefix. Lines are tokenized by stringstream iterator and copied to a vector by copy(). Name filter is done twice - for non-zero length of strings in lambda function capture of copy_if() and in auto iterator loop by find() of the pattern. Shared pointers are C++ STL facility for refcounted pointers. Wordcount of the strings containing pattern is incremented via a shared_ptr. Arbitrary filtering implementation can be plugged-in to lambda function capture block of copy_if() - Most names are of persons,organizations,locations and namefiltering or proper noun extraction has multiple solutions:

(*) NER PoS tagging by Conditional Random Fields(Supervised-costly-requires culture neutral training corpora e.g https://www.aclweb.org/anthology/P95-1032.pdf)

(*) Natural Language Dictionary or Ontology lookup(Unsupervised-preferred-no training data-if word is not in dictionary or semantic network - WordNet,ConceptNet,NameNet - https://pdfs.semanticscholar.org/56f9/cf53333a46c9ea355578f6b7b9424a4737e2.pdf - it is most likely a proper noun) are some of them. NeuronRain AstroInfer People Analytics implements dictionary filter.

----

1 November 2020, 2 November 2020 - Mediator-Colleague Design Pattern - C++ example

----

Mediator Design Pattern encapsulates the set of colleague objects and a director object which mediates the interactions between colleague objects. Colleagues do not interact among themselves directly but are moderated by the mediator object. C++ code example mediatordesignpattern.cpp implements two classes for director-mediator and colleagues - colleague objects invoke the singleton mediator for interaction amongst them and do not communicate with each other. Such a pattern is necessary in GUI event oriented programming - set of widgets which do not know each other notify a mediator about an event and mediator acts accordingly issuing further directives.

References:

1..Design Patterns - Elements of Reusable Object Oriented Software - [1995] - [Erich Gamma - Richard Helm - Ralph Johnson - John Vlissides] - Mediator Behavioural Pattern - Page 273

----

25 November 2020 - Reference wrappers, Arrays of references, Array move, C++ Array Objects, Reference
to C++ Array object, Array Rotation

Code example arraymove.cpp demonstrates the following on C++ bounded array objects:

(*) Define a primitive integer array
(*) Perform memmove() on elements of integer array
(*) Instantiate a vector from integer array (copies array to a vector)
(*) Rotate the vector data and print them (would not affect source array)
(*) Define & alias operator to std::string type as reference_wrapper<string> object - from <functional> library

(*) Define array object of reference_wrapper<string> objects
(*) Invoke a function and pass bounded array object of reference_wrapper<string> objects by reference to function - (&array)[length]
(*) Perform memmove() on elements of array object of reference_wrapper<string> objects
(*) auto iterate the memmove()-ed integer array object and reference_wrapper<string> array objects

References:

1.Clockwise-Spiral rule for C type inference - http://c-faq.com/decl/spiral.anderson.html
2.C++ Reference wrapper - https://en.cppreference.com/w/cpp/utility/functional/reference_wrapper
3.C++ Rotate - https://en.cppreference.com/w/cpp/algorithm/rotate

----

22 December 2020 - Russian Peasant Multiplication in Linear Time by BitShift operator

----

Code example russianpeasant.cpp implements the Russian Peasant Multiplication Algorithm for 2 integers which is in Linear time (Theta(n)). It takes two integers as commandline arguments and casts them to type "unsigned long long" by strtoull(). Linear complexity product is obtained by Carry-Save adder, Parity and Majority Functions. Russian Peasant Algorithm works by:

(*) Bit shifting right operand a and Bit shifting left operand b by 1 till one of the operands reaches single bit
(*) Add all rows for Left shifted values of b for which corresponding right shifted rows of a are odd to get the product a*b.

Logs logs/russianpeasant.log.22December2020 show few example products by Russian Peasant algorithm.
Conventional multiplication is quadratic time - O(n^2)

References:
----

1. Algorithms - [Cormen-Leiserson-Rivest-Stein] - Chapter 29 - Arithmetic Circuits - Section 29.4.2 Clocked Circuits - Faster Russian Peasant Multiplication Circuit in Linear Time - http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap29.htm

Course Authored By:

Srinivasan Kannan
(also known as: Shrinivaasan Kannan, Shrinivas Kannan)
Ph: 9791499106, 9003082186
Krishna iResearch Open Source Products Profiles:
http://sourceforge.net/users/ka_shrinivaasan,

https://github.com/shrinivaasanka,
https://www.openhub.net/accounts/ka_shrinivaasan
Personal website(research): https://sites.google.com/site/kuja27/
emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
kashrinivaasan@live.com

###

This is a non-linearly organized, code puzzles oriented, continually updated set of course notes on Java language. This complements NeuronRain course materials on Linux Kernel, Cloud, BigData Analytics and Machine Learning and covers fundamentals of Java.
----

7 February 2017

Generics in Java are equivalents of Standard Templates Library/Boost in C++ with facility to parametrize Classes and Function signatures. Java Generics define a variable, class, interface and function with syntax:

```
variablename<T1,T2,...> v;
classname<T1,T2,...> { };
interfacename<T1,T2,...> { };
T1 functionname(T2 t2,...);
```

Spark Streaming code in https://github.com/shrinivaasanka/asfer-github-code/blob/master/java-src/bigdata_analytics/spark_streaming/SparkGenericStreaming.java uses Generic JavaDStream with type parameter <String> and JavaPairDStream with parameter <String, Integer>. Diamond notation ignores typenames and runtime inference is done for following declaration:

```
T1<T2,T3> t1 = new T1<>();
```

which automatically infers type as T1<T2,T3>.

21 May 2019

Lambda functions in Java are illustrated by LambdaExpressions.java which implements a functional interface and a lambda expression to print N consecutive numbers without looping. Function recursion() implements the recursive part while lambda expression just consists of printing an integer which is captured from argument to lambda function. Logs for this code example are in testlogs/LambdaExpressions.log.21May2019.
###

###
Course Authored By:

Srinivasan Kannan
(also known as: Shrinivaasan Kannan, Shrinivas Kannan)
Ph: 9791499106, 9003082186
Krishna iResearch Open Source Products Profiles:
http://sourceforge.net/users/ka_shrinivaasan,
https://github.com/shrinivaasanka,

https://www.openhub.net/accounts/ka_shrinivaasan
Personal website(research): https://sites.google.com/site/kuja27/
emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
kashrinivaasan@live.com

##
##

This is a non-linearly organized, continually updated set of course notes on
Linux Kernel and Cloud
and supplements NeuronRain USBmd, VIRGO Linux and KingCobra Design Notes in:

NeuronRain Enterprise Version Design Documents:

USBmd USB and WiFi network analytics - https://github.com/shrinivaasanka/usb-md-github-code/blob/master/USBmd_notes.txt

VIRGO Linux -
https://github.com/shrinivaasanka/virgo-linux-github-code/blob/master/virgo-docs/VirgoDesign.txt

KingCobra Kernelspace Messaging - https://github.com/shrinivaasanka/kingcobra-github-code/blob/master/KingCobraDesignNotes.txt
----
NeuronRain Research Version Design Documents:

USBmd USB and WiFi network analytics -
https://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt

VIRGO Linux -
https://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VirgoDesign.txt

KingCobra Kernelspace Messaging -
https://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt
----
----
17 March 2017

VIRGO Linux Kernel Build Steps

VIRGO Linux kernel is an overlay of VIRGO codebase on kernel mainline (presently 4.1.5) source tree. Building a custom kernel for VIRGO
is required for building new system calls and kernel modules in it. Shell script for builing kernel mainline is in:
      https://github.com/shrinivaasanka/virgo-linux-github-code/blob/master/buildscript_4.1.5.sh

https://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/buildscript_4.1.5.sh

VIRGO Linux kernel has following new system calls:

Cloud RPC system call:
      virgo_clone()
Cloud Kernel Memory Cache (kernelspace equivalent of memcache):
      virgo_malloc()
      virgo_get()
      virgo_set()
      virgo_free()
Cloud File System:
      virgo_open()

virgo_close()
virgo_read()
virgo_write()

VIRGO Linux kernel has following new kernel modules (kernelsocket listeners) corresponding to previous system call clients:
----
1. cpupooling virtualization - VIRGO_clone() system call and VIRGO cpupooling driver by which a remote procedure can be invoked in kernelspace.(port: 10000)
2. memorypooling virtualization - VIRGO_malloc(), VIRGO_get(), VIRGO_set(), VIRGO_free() system calls and VIRGO memorypooling driver by which kernel memory can be allocated in remote node, written to, read and freed - A kernelspace memcache-ing.(port: 30000)
3. filesystem virtualization - VIRGO_open(), VIRGO_read(), VIRGO_write(), VIRGO_close() system calls and VIRGO cloud filesystem driver by which file IO in remote node can be done in kernelspace.(port: 50000)
4. config - VIRGO config driver for configuration symbols export.
5. queueing - VIRGO Queuing driver kernel service for queuing incoming requests, handle them with workqueue and invoke KingCobra service routines in kernelspace. (port: 60000)
6. cloudsync - kernel module for synchronization primitives (Bakery algorithm etc.,) with exported symbols that can be used in other VIRGO cloud modules for critical section lock() and unlock()
7. utils - utility driver that exports miscellaneous kernel functions that can be used across VIRGO Linux kernel
8. EventNet - eventnet kernel driver to vfs_read()/vfs_write() text files for EventNet vertex and edge messages (port: 20000)
9. Kernel_Analytics - kernel module that reads machine-learnt config key-value pairs set in /etc/virgo_kernel_analytics.conf. Any machine learning software can be used to get the key-value pairs for the config. This merges three facets - Machine Learning, Cloud Modules in VIRGO Linux-KingCobra-USBmd , Mainline Linux Kernel
10. SATURN program analysis wrapper driver.
and userspace test cases for the above.

Prerequisites for building VIRGO Linux kernel are similar to mainline kernel:
apt install libncurses5-dev gcc make git exuberant-ctags bc libssl-dev

Presently VIRGO kernel is 32-bit. Building 64 bit linux kernel requires long mode CPU flag (lm) in /proc/cpuinfo. Also booting a 64 bit kernel built on 32 bit kernel could cause init to fail as "init not found". This is because init is a symbolic link to systemd binary when kernel boots up which is 32-bit and not 64-bit. For this menuconfig in Kbuild provides IA32 Emulation config parameter.

Booting a custom kernel could also cause partition issues while grub is updated sometimes with error "no such partition" and could drop to grub rescue> prompt. This can be remedied by:
    1. grub rescue> ls
which lists the partitions in format (hd0, msdos<#number>)
    2. grub rescue> set root=(hd0, msdos<#number>) {for each such partition}
    3. grub rescue> set prefix=(hd0, msdos<#number>)/boot/grub
    4. grub rescue> insmod normal
    5. grub rescue> normal
which boots grub normally. This has to be persisted with:
    6. update-grub2
    7. grub-install /dev/sda
on reboot

Some grub errors can be debugged by adding kernel boot parameters in /boot/grub/grub.cfg or at boot time edit of the entry with "... $vt_handoff". Appending options "debug init=<path-to-systemd>" to this line might help.

Sometimes repetitive builds and grub updates also cause a rare file system

corruption as below which causes root shell drop:
      ALERT! <diskid> does not exist
      Boot args (cat /proc/cmdline)
      - Check rootdelay
      - Check root
      Missing modules (cat /proc/modules, ls /dev)
      Dropping to root shell
      (initramfs)
This is remedied by:
      (initramfs) modprobe dm-mod
      (initramfs) lvm
      lvm> vgchange -ay
      lvm> exit
      (initramfs) exit
followed by:
      mount -o remount,rw /dev/sda<number> <mount_point_directory>

Complete documentation of VIRGO Linux with Design Documents is in:
https://github.com/shrinivaasanka/virgo-linux-github-code/tree/master/virgo-docs/
https://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/

----
20 March 2017

VIRGO Linux 64 bit build - continued:

*) After configuring IA32 simulation and a successful build, there could still be initramfs errors
because of i915 drivers mismatch as below:
      "Possible missing firmware --------- for i915"
*) Previous error requires installation of latest linux i915 graphics drivers (though there may not be a matching hardware for it) listed in https://01.org/linuxgraphics/downloads/firmware: for Kabylake, Skylake, Broxton graphics processors
*) update-initramfs has to be done again for correct /boot initrd image to be created for 4.10.3 after updating i915 drivers previously listed
*) In some cases image may be hidden in grub boot menu. For this an already available boot-repair tool scans the /boot images and reinstalls grub menu.
*) On successful 64 bit build, uname -a should have x86_64 as below:
Linux kashrinivaasan-Inspiron-1545 4.10.3 #1 SMP Fri Mar 17 18:30:40 IST 2017 x86_64 x86_64 x86_64 GNU/Linux

----
24 September 2017

NeuronRain VIRGO64 presently has been ported to 4.13.3 Linux Kernel. Also the accompanying USBmd and KingCobra kernel modules have been
split into 32-bit and 64-bit versions and separate repositories have been created for them in SourceForge and GitHub. 4.13.3 kernel has
transport layer security enshrined in kernel which is essential for securing kernelspace cloud traffic. A detailed FAQ on technical aspects
of VIRGO 64-bit version and KTLS is in http://neuronrain-documentation.readthedocs.io/en/latest/ .

Logging Servers and Clients - Some Implementation Examples - 28 September 2018

In large cloud installations, there often arises a need to log frequently occuring network event traces to log files realtime minimizing network latency. Example 1 in references is the Trace utility implemented in an application server (Java) for logging Global and Local JTS Transaction (XA) events with facility for Trace Levels. Example 2 describes sequential and parallel implementations of a logging server in a network and logging clients connecting to this server. Reactor pattern is a Listener on pending socket descriptors (select/poll) and events to be serviced. Example 3 in the references: NeuronRain VIRGO32 and VIRGO64 Linux Kernels implement a kernel service module and logging client utility kernel function for writing EventNet log messages to EventNet Edges and Vertices files sent from remote cloud nodes by eventnet_log() - implements Acceptor-Worker Kernel Threads (Router-Dealer) pattern. Example 4 is the ZeroMQ Publish-Subscribe protocol for designing a log collector subscriber and log client publishers.

References:

1.Oracle Glassfish Java EE - (Earlier Sun Microsystems iPlanet Application Server - IAS) - JTS XA Transaction Logging - https://github.com/javaee/glassfish/blob/master/appserver/transaction/jts/src/main/java/com/sun/jts/trace and https://github.com/javaee/glassfish/tree/master/appserver/transaction/jts/src/main/java/com/sun/jts/utils - IAS_JTS_TRACE - authors: "mailto:k.venugopal@sun.comi,kannan.srinivasan@sun.com" - "...public static void setTraceWriter(PrintWriter traceWriter) { m_traceWriter = traceWriter; }..." - Wrapper Facade pattern - wraps a writer object
2.Beautiful Code - Chapter 26 - Labor-Saving Architecture - An Object Oriented Framework for Networked Architecture - Concurrent Logging Servers in C++ - Reactor Pattern for Logging Concurrent Events
3.EventNet Kernel Service Module and EventNet Logging Client function in NeuronRain VIRGO Linux Kernel - https://gitlab.com/shrinivaasanka/virgo64-linux-github-code/tree/master/linux-kernel-extensions/drivers/virgo/eventnet and https://gitlab.com/shrinivaasanka/virgo64-linux-github-code/blob/master/linux-kernel-extensions/drivers/virgo/utils/virgo_generic_kernelsock_client.c - eventnet_log()
4.ZeroMQ Pub-Sub for distributed logging - http://zguide.zeromq.org/php:chapter8#toc31

Linux Kernel Development (System calls and Drivers) - some low level architecture specific issues - 14 May 2019

Following are few architecture specific idiosyncracies and heisen bugs that could cause harrowing experience while writing new kernel system calls and drivers (32 bit versus 64 bit, dual core versus quad core versus octa core):

1.strcpy() buffer overflow (kernel has its own implementation of glic string library in include/linux) - there is a documented intel x86_64 buffer overrun error in some chips
2.char * to const char* cast requirement
3. (u8*) cast for sin_addr in kernel sockets
4. memcpy() versus copy_to_user() or copy_from_user() - in some architectures memcpy() works while in others copy_xxx() pairs work without crashes
5. Requirement for __user qualifier macro for some system call parameters passed to copy_from_user() and copy_to_user()
6. Correct location of kernel glibc string library headers in Makefile paths
7. strscpy() instead of strcpy() - works some times
8. Usage of BUF_SIZE (buffer size as macro) instead of strlen()
9. kstrdup() might crash sometimes in strlen() which can be replaced by strcpy() and strcat()
10. User memory access warning/error in KASAN because of copy_from_user() and copy_to_user() (access_ok() assertions crash)
11. memcpy() is less secure than copy_xxx() pairs

12. __put_user() and strncpy_from_user() might sometimes circumvent crashes
13. Setting the segment correctly - get_fs()/set_ds(KERNEL_DS) must correctly encapsulate kernel data access.
14. copy_user_generic() avoids crashes in some cases if previous do not work
15. cast char to unsigned long long which bypasses lot of buffer issues and __put_user() supports only unsigned long long
16. unsigned long long has the advantage of abstracting any datatype.
17. Being compatible for both 32 and 64 bits simultaneously could be a problem - u8 and const char* cast
18. memcpy() could fail in certain multicores
19. strncpy_from_user() in place of copy_from_user() creates stabler kernel syscalls and driver builds
20. __put_user() in place of copy_to_user() is stabler in some multicores
21. unsigned long long is sometimes stabler than char __user*
22. sizeof() might have to be replaced by BUF_SIZE macro for all copy functions
23. in4_pton() might fail in some cases while in_aton() might work

----

Linux Kernel Development (System calls and Drivers) - some low level architecture specific issues 2 - 10 June 2019

Following are some architecture specific problems while invoking kernel sockets (e.g simulating a telnet client and server within
kernel):
1. System calls (especially written new) often involve userspace strings passed on to kernel (and sometimes transported to a remote
kernel by kernel sockets) which are mandated to be marked as "const __user char*"
2. __user pointer marked as previous informs the kernel that a userspace data has to be handled by kernel by copying it to kernelspace
3. Accessing __user pointers directly within system calls and kernel modules (e.g in printk) creates a fault because a userspace
pointer is accessed in kernel space illegitimately.
4. Special functions like copy_from_user()/strncpy_from_user() are meant for copying userspace __user pointed data to kernelspace
pointers.
5. This userspace to kernelspace copy requires setting kernel data segments appropriately by set_ds(KERNEL_DS) followed by the invocation and
resetting the datasegment to status quo ante.
6. Buffers used within the kernel modules and system calls for copying user buffers must have sufficient size and strlen() may not
work for strings which are not null terminated causing overflow.
7. copy_to_user() causes weird faults and there is no equivalent strncpy_to_user() for out parameters in system calls which might necessitate __put_user() to a generic data type e.g unsigned long long and reinterpret-cast to a char literal. Sequential invocation
of these creates an array of chars or strings.
8. -32, -107, -101 errors are often witnessed in kernel_connect() which is probably caused by in4_pton()

Course Authored By:
##
##

K.Srinivasan
NeuronRain Documentation and Licensing: http://neuronrain-documentation.readthedocs.io/en/latest/
Personal website(research): https://sites.google.com/site/kuja27/

This is a non-linearly organized, code puzzles oriented, continually updated set of course notes on Python language. This
complements NeuronRain course materials on Linux Kernel, Cloud, BigData Analytics and Machine Learning and covers
fundamentals of Python.
----
6 February 2017

Code Reference:
----
https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/CNFSATSolver.py

Python's object oriented paradigm is quite similar to most of the languages like Java and C++ but the difference in number of lines code
for same algorithm between Python and C++/Java is what makes it favourable for Natural Language Processing and writing microservices (services
with small functionality which are interconnected). Python is more Haskell or LISP functional language-like and achieves both worlds of functional programming and imperative procedural programming by Lambda on-th-fly functions. Previous code demonstrates some of the minimum basic features of Python:

- Python Classes
- Tuples
- Lists
- List Slicing
- Set operations on python objects
- Control structures (for, if)
- Python object member functions and self keyword

Python classes are defined with class <class>(<baseclass>) and member functions are defined with def <function>(). Base class by default is object unless explicitly stated. Tuples are ordered pairs of arbitrary dimensions equivalent to vectorspaces in mathematics defined with (). Lists are equivalent to arrays in C defined with [] subscripts. Accessing an element in both tuples and lists are by [] subscript. Concepts of slicing is central to list comprehension in Python. Slicing can return a contiguous subset of a list by [<start>:<end>] notation. When either start or end is ignored implicit list start and end are assumed. For loops in python can iterate over any "iterable" object. Iterables include lists, dictionaries and user defined containers. if..else..elif is the python equivalent of conditional clauses with truth values being boolean keywords "True" or "False" which are builtins. Python classes denote self keyword to be the present object instantiated (equivalent to "this" in C++)

----
27 February 2017 - Python Generators and Yield

Python has a notion of iterables where any sequential data structure can be made to return an element and resume from where it left to return the next element in the sequence. This is quite useful for problems which need to remember the last element accessed and resume from next element (can be constrasted with static keyword in C/C++ which live across function invocations with global scope) - typical streaming scenario.

Streaming Abstract Generator implemented in :
https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/Streaming_AbstractGenerator.py

is based on this idiom. It basically is facade frontend abstraction for multiple backend streaming datasources. It overrides the __iter__() method to access an element and yield it instead of returning it. User of this class, instantiates with desired data storage constructor arguments and iterates through it without any knowledge of how the data is accessed. In python terms, data is "generated" and "yielded" iteratively in a loop accessing consecutive elements. No storage is allocated by generator explicitly and backend client objects for HBase/Cassandra/Spark/Hive/File datasources handle them internally.

This is a typical example of Iterator/Facade design pattern listed in Gof4.

----
844. (THEORY and FEATURE) 6 January 2018,18 July 2020 - Currying and Partial Function Application in Python - related to all sections on Recursive Lambda Function Growth algorithm implementation for learning lambda functions from Natural Language Texts and Text analytics
----
Python has functional programming (Haskell) equivalents of Currying and Partial Application support. Currying converts a function of n parameters to n functions of one parameter each invoked in nesting, and returns a function in each function. Partial Application Function is similar to currying but takes 2 arguments instead of 1 in currying, and returns a function. Code examples for these are below - committed to code/Currying.py and code/PartialFunctionApplication.py

```
------------
Currying.py:
------------
1
2
3
4
5
6
===========
Curried :
===========
6 5 4 3 2 1

------------------------------
PartialFunctionApplication.py:
------------------------------
=========================================
Equivalent Partially applied functions
=========================================
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
```

Currying by Partial Functions (as Beta reduction) are used in Recursive Lambda Function Growth - Graph Tensor Neuron Network algorithm implementation of NeuronRain which converts a natural language text to tree composition of functions of 2 arguments (a function operator acting on two operands) by traversing a random walk or cycle of the textgraph. Example:
maximum_per_random_walk_graph_tensor_neuron_network_intrinsic_merit=
('one((integer(digit,(definite_quantity(abstraction,(measure(number,
(command(act,(speech_act(psychological_feature,(order(abstraction,(event(order,
(speech_act(event,(act(abstraction,(digit(three,(psychological_feature(number,
(abstraction(measure,

(definite_quantity(integer,command)))))))))))))))))))))))))',
10.804699467199468)

----
7 August 2018 - Python Dictionaries

Python dictionaries are the hashtable implementations using linear probing(open addressing) to find next
available slot (preferred to Chaining/Buckets). Open addressing in python applies the following function
recurrently to find next available slot:
      x = 5*x + perturb + 1
      perturb >> PERTURB_SHIFT (some constant)
This function has been found to be optimal in python benchmarks. Simulating buckets/chaining in builtin
dictionaries is therefore done by an alternative dictionary implementation defaultdict() which initializes
the dictionary to a default key whose type is defined by argument to constructor. An example, chaining/buckets
has been described in code/Dictionaries.py which prints :

['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'iterkeys', 'itervalues', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
defaultdict(<type 'list'>, {0: [0], 1: [1, 81], 4: [4, 64], 5: [25], 6: [16, 36], 9: [9, 49]})

References:
----
1. CPython PyDictObject source - code comments - https://github.com/python/cpython/blob/master/Objects/dictobject.c
2. Beautiful Code - [Greg wilson]

----
7 October 2018 - Python Reflection, Derived Classes and Class Methods
----

Python has support for reflection in the form of function and code objects. Python function objects are accessed
by __func__ member of any function considered as an object. Python classmethods are decorators declared before
defining a function by @classmethod annotation and defined as class method and defined by def <func>(cls, ...)
declarative containing cls keyword. Class methods are useful for defining a classwide method common to all instances . This is different from static keyword in other languages and there is a @staticmethod decorator in python, the static equivalent. Class methods called on derived class take derived class objects for cls. An example for reflection and class methods is in code/Reflection.py and logs in code/logs/Reflection.log.7October2018

----
13 March 2019 - Async/Await - Asynchronous IO in Python 3.7

Python 3.7 supports asynchronous invocations of coroutines by async and await pairs of keywords similar to promise-future in C++. An example Chatbot Client and Server in AsyncIO_Client.py and AsyncIO_Server.py
define two classes for Chat Server and Client which implement connection_made() and data_received() interface functions. Notable feature in 3.7 is the "async def" for function declarations in lieu of @asyncio coroutine decorators in previous versions of Python 3. Both client and server instantiate a loop object and invoke create_connection() and create_server() respectively for client and server by await semantics. Notion is to make client-server transport completely event driven by await and non-blocking by async.This example applies as a pattern to any algorithm doing asyncio. Logs in logs/AsyncIO_ClientServer.log.13March2019 show a sample chat.

References:

1.Python 3.7 documentation - Async/Await - https://docs.python.org/3/library/asyncio-protocol.html

----
14 November 2019, 23 April 2020 - Multidimensional Array Slicing - List comprehension, slice() function and NumPy - and StringIO

Python has in-built slice() function which is an iterator for array indices to slice. Numeric Python
NumPy has library support for multidimensional array slicing by subscripts. Code example code/Slicing.py
demonstrates 5 variants of slicing of a 2 dimensional ndarray parsed from a string text. Text of multiple
lines separated by newline "\n" is read by StringIO object by genfromtxt() and parsed to a multidimensional array by delimiter ",". Five different slicings in the example are:

1. slicedarray1 - equal slices for 2 dimensions by slice()
2. slicedarray2 - unequal slices for 2 dimensions by slice() - only larger slice is effected
3. slicedarray3 - slice() for one dimension and tuple of indices for the other - only indices from tuple are effected
4. slicedarray4 - equal slices of 2 dimensions with an added step parameter - only one dimension is sliced
5. tuple of indices for both the dimensions raises Python error:

```
      Traceback (most recent call last):
        File "Slicing.py", line 23, in <module>
            slicedarray3=parsedarray[(0,3),(0,3)]
      IndexError: index 3 is out of bounds for axis 0 with size 2
```

Code example chooses either Slicing or List comprehension to extract a slice by a flag - List comprehension being the most fundamental Python primitive is the obvious choice for slicing. Following list comprehension:

```
        slicedarray5=[row[2:5] for row in parsedarray]
```

extracts a rectangular slice:

```
      [['1', '2', '3', '4', '5', '6', '7', '8'], ['9', '10', '11', '12', '13', '14', '15', '16']]
      slicedarray5 - list comprehension:
      [['3', '4', '5'], ['11', '12', '13']]
```

----
14 December 2019 - Rounding off, Floating point division, Python 2.7 and Python 3.7.5, PDB

Following code snippet defined within a class of code/RoundOff.py demonstrates difference in division behaviour between Python 2.7 and Python 3.7.5 by

disassembly of Python bytecode in PDB debugger:
      x1 = (1-2)/2
        x2 = (2-1)/2
==
Python 2.7
==
python RoundOff.py
('x1:', -1)
('x2:', 0)
==
Python 3.7.5
==
python3.7m RoundOff.py
x1: -0.5
x2: 0.5

Python 3.7.5 does auto type promotion to float while Python 2.7 rounds off to floor. PDB is imported by -m pdb in commandline and disassembler is imported as dis in pdb. Roundoff class is loaded in PDB. PDB Python VM bytecode disassembly shows absence of binary divide opcode in Python 3.7.5

----
26 June 2020 - Object Marshalling and Unmarshalling - Python Serialization and Persistence
----
Python has variety of infrastructure to support serialization and deserialization of objects - some of
them being Pickle,Marshal,DBM and Shelve modules. While pickling is widely used, marshal,dbm and shelve are lowlevel alternatives - marshal is most widely used for persisting compiled binaries (.pyc) and restricted to python types while shelve and DBM depend on file and linux database backends and can persist arbitrary user defined types as persistence dictionaries of name-values. Thus lookup is easier making them powerful than pickle and marshal. Code example Marshal.py demonstrates the serialization and deserialization of an example Python 3.7 class and primitive list datatypes. Class Marshal defines __init__(), marshal(),unmarshal() and sync() functions which respectively wrap following shelve innards:

- init - opening a file persistence by shelve.open() in writeback=True mode and retrieve serializer dict
- marshal - populating serializer dict by object name ("array1" and "exampleobject1") and object.
- unmarshal - retrieval of a persisted object by object name from serializer dict
- sync - which synchronizes the persistence if writeback=True

Marshal databases and logs are in:
code/MarshalDB.bak
code/MarshalDB.dat
code/MarshalDB.dir
code/logs/Marshal.log.26June2020

Fields of deserialized Example() object and array are printed by print():

Marshalling object: array1
Shelve serializer: [('array1', [1, 2, 3]), ('exampleobject1', <__main__.Example object at 0xb7396aec>)]
Unmarshalling object: array1

Shelve serializer: [('array1', [1, 2, 3]), ('exampleobject1', <__main__.Example object at 0xb7396aec>)]
[1, 2, 3]
Marshalling object: exampleobject1
Shelve serializer: [('array1', [1, 2, 3]), ('exampleobject1', <__main__.Example object at 0xb7396aec>)]
Unmarshalling object: exampleobject1
Shelve serializer: [('array1', [1, 2, 3]), ('exampleobject1', <__main__.Example object at 0xb746188c>)]
field1: field1
field2: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

----
30 July 2020 - Pointers, C-types, Copy-On-Write, Mutables, Immutables in Python 3.7.5

Python compiles to bytecodes which are executed on Python Virtual Machine. Basic datatype objects are
immutable while iterables are mutable in python. Code example Pointers.py defines a function to
simulate pointers on Python by importing ctypes C wrapper library and instantiating pointers
to some string and integer objects by c_wchar_p() and c_void_p(). Illegal access Exception while assigning to immutable string literal is handled and pointer to string is created by writing a copy (Copy-On-Write) of original string which is different from C pointers. Similarly pointer copy to an integer data is assigned to while leaving the original unchanged. An alternative to C-type string creation by create_string_buffer() is demonstrated.

Logs from code/logs/Pointers.log.30July2020 :
Immutable - sentence1: This is a sentence
Exception: 'str' object does not support item assignment
Pointer to sentence1 string: c_wchar_p(3075052112)
Changed pointer copy of sentence1: This is different sentence
sentence2 - RAM contents after create_string_buffer(): b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Pointer to integer1: c_void_p(200)
immutable - integer1: 100
Changed pointer copy of integer1: 200

References:
----
1. Python 3.8.5 documentation - https://docs.python.org/3/library/ctypes.html

----
862. (THEORY and FEATURE) NeuronRainApps - NeuronRain Usecases - MAVSDK Python Async I/O Drone Simulator - related to 637,713,722 and all sections on Drone Autonomous Delivery, Drone Electronic Voting Machines, Large Scale Visuals/Urban Sprawl/GIS Analytics, VIRGO PXRC Flight Controller Kernel Driver - 18 August 2020
----
1. This commit implements Drone_MAVSDK_Client.py and Drone_MAVSDK_Server.py asynchronous Drone I/O by importing MAVSDK (Micro Air Vehicle SDK) Python library which contains an inferface for drone telemetry - mavsdk_server. Earlier NeuronRainApps usecase implementations for Autonomous Online Shopping Delivery (and Conceptual Set Partition Drone Electronic Voting Machines) depend on Dronecode Python SDK which might alternatively import MAVSDK.
2. Asynchronous I/O Chatbot Python class example has been augmented to import mavsdk and defines a new asynchronous function drone_async_io() which takes a Drone System object, Drone port, Drone action as arguments and accordingly

async-invokes (await) MAVSDK internal functions for those actions.
3. Because of absence of a licensed Drone, this is a conceptual-only usecase implementation of a Drone simulator and "No System" exceptions are thrown which are handled and printed. Standalone product-specific Simulators (e.g jMAVSIM, Gazebo and AirSim for PX4 - https://dev.px4.io/v1.9.0/en/simulation/jmavsim.html, ROS-Simulator - https://dev.px4.io/v1.9.0/en/simulation/ros_interface.html) could be used for development testing.
4. Drone action clauses for "info","takeoff","arm","camera","goto_location" have been implemented in drone_async_io() which respectively print flight information, control takeoff, arm the AV, control camera video streaming, and pass on Longitude-Latitude-Altitude-Yaw GPS destination data for autopilot of the AV.
5. drone_async_io() is an async member of DroneMAVSDKClient class. Drone object is instantiated by System() and drone_async_io() initializes connect() to mavsdk_server Drone server port.

----

863. (THEORY and FEATURE) MAVSDK Python jMAVSIM Drone flight simulator - Proof-of-Concept implementation related to 862 and all sections on Drone Autonomous Delivery, Drone EVMs, Large Scale Visuals/Urban Sprawl/GIS Analytics, VIRGO PXRC Flight controller kernel driver - 26 August 2020

----

1. This commit revamps Drone_MAVSDK_Client.py earlier to define a new command "all" in drone_async_io() which connects to a drone, prints its flight information, arms, takes-off, navigates, streams visuals to ground station and lands a drone - everything in a simulator (jMAVSIM)
2. Before arming the drone its connection state, UUID and GPS health checks are preferable code for which has been introduced in 2 async iterators. "If" clause for Command "all" is a ubiquitous pattern which might be frequently necessary in Drone Autonomous Delivery and Drone electronic voting machines.
3. Drone_MAVSDK_Client.py connects to a PX4 jMAVSIM flight simulator which can be installed by instructions in https://dev.px4.io/v1.9.0/en/setup/dev_env_linux.html (Section on JMAVSIM and Gazebo simulation) and https://dev.px4.io/v1.9.0/en/simulation/jmavsim.html. PX4 Firmware, ROS and ECL dependencies are installed by sourcing shell script https://raw.githubusercontent.com/PX4/Devguide/v1.9.0/build_scripts/ubuntu_sim.sh which clones PX4 dependencies and creates build scripts. PX4 jMAVSIM SITL (Software in the loop) is built by target "make --debug px4_sitl jmavsim" in PX4/Firmware source which starts the Simulator GUI and SITL CLI pxh> prompt.
4. Drone_MAVSDK_Client.py is either executed from apython (asynchronous python CLI installed by aioconsole) or usual python commandline interface which connects to jMAVSIM GUI server.
5. jMAVSIM flight simulation logs for a multirotor aerial verhicle is at code/logs/PX4_Drone_JMAVSIM_Flight_Simulation.log.26August2020 which shows the jMAVSIM server side logs for GPS, arm, vertical takeoff, navigation and vertical landing. Camera streaming is failed with Error "DENIED".
6. drone_async_io() might be significantly augmented and imported across every Drone dependent analytics code in NeuronRain which can be developed and tested on a 3D Virtual Reality simulation without a real drone.
7. Efficient Drone Autonomous Delivery is an NP-complete Travelling Salesman-Hamiltonian Cycle problem which traverses lowest cost circuit connecting set of delivery points on a transporation network graph.
8. Drone FOSS Code Repositories and References:
    8.1 MAVSDK Python - https://github.com/mavlink/MAVSDK-Python
    8.2 PX4 Drone Autopilot Firmware - https://github.com/PX4/Firmware
    8.3 PX4 Estimation and Control - https://github.com/PX4/ecl
    8.4 PX4 JMAVSIM - https://github.com/PX4/jMAVSim
    8.5 Auterion MAVSDK Java - Android Client - QGroundControl - https://auterion.com/getting-started-with-mavsdk-java/

8.6 PX4-ROS-Gazebo simulator - Graphic Illustration - MAVROS - https://dev.px4.io/v1.9.0/en/simulation/ros_interface.html
8.7 PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms
- "...Our system architecture is centered around a publish-subscribe object request broker on top of a POSIX application programming interface. This allows to reuse common Unix knowledge and experience, including a bash-like shell. We demonstrate with a vertical takeoff and landing (VTOL) use case that the system modularity is well suited for novel and experimental vehicle platforms. We also show how the system architecture allows a direct interface to ROS and ..." - https://ieeexplore.ieee.org/document/7140074
9. jMAVSIM replay logs have been committed to code/logs/jMAVSIM_replay_logs/

----
5 September 2020,7 September 2020 - Python Decorators, Static Type Checking, Type Hints, Mypy and IDE typecheckers, Final constant type hint, final decorator in Python 3.8

Python traditionally implements Duck typing or Dynamic typing and types of objects could be checked
at runtime by type() and isinstance(). Decorators in Python are features which could be used to instrument another function and return a wrapped new function having additional code. Lack of support for static
type checking and constant definitions in Python have been answered in Python 3.8 which implements series
of PEPs for final decorator for final methods which cannot be overridden in derived classes and Final
type hint which aids Typecheckers (mypy, PyCharm IDE) to statically analyze code and flag type errors.
Code example Decorators.py defines two classes - Base and Derived - which import various typecheck artefacts from typing module. Class BaseDecorated defines a constant var1 by type hint Final and a method cannotoverride() by @final decorator which can be type-enforced by linters in mypy (http://mypy-lang.org/) and PyCharm. Both base and derived classes define member functions to which argument type hints and return type hints are annotated by ":" and "->" operators. Most importantly member function funcdecorator() defines an inner function wrapper() which takes a function argument and instruments additional code around it and returns a new decorated function by typing.cast() typecasting. Declaration of T defines a Callable type variable (TypeVar) of arbitrary number of arguments specified by Any. Logs code/logs/Decorators.log.5September2020 capture the decorator callstack.

----
866. (THEORY and FEATURE) A* (A-Star) Best First Search Algorithm Implementation - Python 3.8.5 - 8 September 2020, 27 October 2020 - related to all sections on Drone Autonomous Delivery Navigation, Drone Obstacle Avoidance, Drone Electronic Voting Machines and Graph Analytics
----
1.This commit implements the standard A-Star Path Finding algorithm widely used in Robotics as a general
purpose graph search implementation which could be invoked by multitude of NeuronRain code dependent on Graph Analytics as well as a routine prerequisite for motion planning in NeuronRain Drone Navigation.
2.A-Star algorithm of [Hart-Nilsson-Raphael] described in https://en.wikipedia.org/wiki/A*_search_algorithm is the reference for this implementation in Python 3.8.5.
3.This implementation uses a plain list in place of a Priority Queue for Open set (Discovered nodes)
4.A-Star algorithm improves upon Dijsktra's Shortest Path by finding the path which minimizes the cost function with the help of a heuristic:

argmin(f(n) = g(n) + heuristic(n))
where f(n) is the fscore map, g(n) is the cost of traversing to node n (gscore map) from start and heuristic(n) is the estimated cost of traversing to end vertex from n.
5.Logs in code/logs/AStar_BestFirstSearch.log.8September2020 compute the Best First Search Path [3,6,7] from node 3 to node 7 in an 8 vertex graph denoted by adjacency matrix which marks lack of edges by -1.
6.Navigation in Drone Autonomous Delivery and EVMs is a TSP-Hamiltonian Cycle NP-complete problem wherein Drone has to efficiently visit set of all delivery points-voter residences once while A-Star motion planning is necessary for finding the least cost trajectory between longitude-latitude-altitude of any two delivery points or voter residences (by looking up the addresses in a map service e.g Google Maps).

References:

866.1 Finding Closest Pair of Points - O(NlogN) divide and conquer algorithm better than naive O(N*N) bruteforce for obstacle avoidance - Section 35.4 - Chapter 35: Computational Geometry - Algorithms - [Cormen-Leiserson-Rivest-Stein] - Page 908 - "... System for controlling air or sea traffic might need to know which are the two closest vehicles in order to detect potential collisions ..." - quite relevant for Drone swarms

----
871. (THEORY and FEATURE) Generating all possible permuted strings of an alphabet - related to 843 and all sections on Social networks, Bipartite and General Graph Maximum Matching, Symmetric Group, Permanent, Boolean majority, Ramsey coloring - Number of Perfect (Mis)Matchings - 28 October 2020

1.Related to a question in IIT-JEE: Find sum of integers > 10000 having only 0,2,4,6,8 as digits without repetition (implies < 99999 because more digits would repeat).
2.This commit implements Permutations.py utility which generates all possible permutations of a list of integers and creates permuted integers from them. Generating permutations is required in section 843 for perfect (mis)matches.
3.In the function genpermutations() random permutation is obtained from numpy.random.permutation() and its numeric equivalent is hashed to a dictionary iteratively till all permutations are generated (which is N!).
4.Finding sum of the permuted integers is tricky and makes use of the fact that every digit in the list of permutation is independently and identically distributed - For example, there are 120 (5!) integer permutations of 0,2,4,6,8 and each of [0,2,4,6,8] occurs 120/5 = 24 times per digit in the permutations.
5.Thus per digit sum is 24*8 + 24*6 + 24*4 + 24*2 + 24*0 = 480 and following schematic depicts carry forward:

```
      53+ 53+ 52+ 48+
      |  |  |  |  |
      ....(480 for each digit)
      |  |  |  |  |
      -------------
       533 3  2  8  0
```

and sum of the permuted integers is 5333280.
6.Permutations.py demonstrates this fact and prints:
('summation:', 5333280)
('permutations:', defaultdict(<type 'int'>, {46082: 56, 24068: 97, 20486: 24, 86024: 36, 82604: 75, 60428: 15, 62480: 52, 4628: 81, 24086: 70, 86042: 65, 64028: 40, 24608: 85, 80426: 43, 8246: 107, 26804: 95, 84026: 35, 60482: 55, 8264: 41, 4682: 78, 80462: 28, 64082: 80, 42068: 9, 24860: 91, 2648: 110, 84062: 39, 62048: 99, 42086: 21, 24680: 100, 68204: 77, 42608: 71, 6248: 116, 84602: 86, 2684: 108, 82046: 82, 46208: 51, 48260: 44, 84620: 7, 68240: 57, 20648: 79, 82460: 58, 4268: 61, 40628: 8, 42680: 2, 6842: 62, 86204: 83, 4286: 68, 46280: 109, 20684: 102, 82640: 30, 46802: 69, 26840: 113, 4826: 120, 86240: 6, 64208: 31, 46820: 76, 24806: 14, 8426: 63, 84206: 13, 80624: 3, 28406: 18, 4862: 98,

80642: 72, 8462: 96, 64280: 32, 62084: 48, 6428: 49, 2846: 89, 64802: 92, 84260: 27, 26408: 22, 28460: 26, 2864: 12, 68402: 104, 64820: 114, 42806: 17, 68420: 54, 6284: 90, 40268: 53, 6482: 34, 62804: 115, 60248: 105, 40286: 84, 82064: 93, 42860: 64, 20846: 106, 26480: 4, 80246: 10, 62840: 47, 40826: 29, 60284: 20, 40682: 118, 20864: 67, 86402: 37, 80264: 60, 28046: 88, 86420: 101, 60824: 119, 48026: 94, 40862: 112, 28064: 74, 2468: 42, 60842: 19, 8624: 45, 2486: 103, 68024: 5, 28604: 66, 48062: 33, 26048: 25, 8642: 23, 62408: 59, 68042: 11, 46028: 16, 48206: 117, 48602: 50, 28640: 46, 26084: 38, 82406: 1, 48620: 87, 6824: 73, 20468: 111}))
('numbers:', 120)

1149. (THEORY and FEATURE) ThoughtNet Evocation of Thought Hyperedges by Set Intersection of Hypervertices - related to 620,1147 and all sections of ThoughtNet Hypergraph Evocation, Contextual Multiarmed Bandits, Reinforcement Learning, Word Sense Disambiguation, Balls-Bins Problem - 28 June 2021, 29 June 2021, 30 June 2021, 1 July 2021

1.This commit implements HypergraphIntersections.py which computes the ThoughtNet hyperedge thought evocations from evocative modal classes of hypervertices within the function hypervertices_intersections(). This code is a non-machine learning alternative to ThoughtNet Hypergraph Contextual Multiarmed Bandit Reinforcement Learning implementation in NeuronRain AstroInfer and demonstrates unique intersection example mentioned in 1147.
2.Function hypervertices_intersections() could be imported in NeuronRain AsFer ThoughtNet as graph theoretic evocation alternative to Contextual Multiarmed bandits.
3.hypervertices_intersections() takes the hypergraph encoded as dictionary and the evocative modal classes (which could be learnt from some machine learning classifier including unsupervised dense subgraph core number classifier implemented in NeuronRain) as arguments.
4.For example, some fictitious news headlines have been populated as thought hyperedges which connect some or all of hypervertices 'water','air' and 'agriculture' assigned by a classifier:

```
-----------------------------------------------------------------------------------------------------------------
'water':['flood caused by storms wrought havoc','water scarcity hits crops','cost effective sea water desalination would benefit agriculture'],
'air':['flood caused by storms wrought havoc','air traffic flourished in last financial year','air pollution hits new high'],
'agriculture':['water scarcity hits crops','cost effective sea water desalination would benefit agriculture']
-----------------------------------------------------------------------------------------------------------------
```

5.Thoughts evoked for some set of modal classes are:
hypervertices intersection of ['water', 'air'] : {'flood caused by storms wrought havoc'}
hypervertices intersection of ['water', 'agriculture'] : {'water scarcity hits crops', 'cost effective sea water desalination would benefit agriculture'}
hypervertices intersection of ['water', 'air', 'agriculture'] : set()
6.One of the evocations is not unique and multiple thoughts are evoked which might be difficult to assign numeric weights (as implemented in the Contextual Multiarmed Bandit Reinforcement Learning version) and thoughts could be a partial ordering instead of total ordering (all thoughts evoked might be equally important). Complexity of unique evocation increases proportional to number of hyperedges incident per hypervertex (or average degree of ThoughtNet hypergraph)
7.ThoughtNet Hypergraph is a variant of Balls-Bins (Coupon Collector) problem in which each bin is a hypervertex and balls connected by a hyperedge (each hyperedge connecting similarly labelled balls across hypervertices is uniquely colored) are thoughts pigeonholed to modal class bins.

References:

1149.1 Modality Effect in Psychology - https://en.wikipedia.org/wiki/Modality_effect - "...Modality can refer to a number of characteristics of the presented study material. However, this term is usually used to describe the improved recall of the final items of a list when that list is presented verbally in comparison with a visual representation. The effect is seen in free recall (recall of list items in any given order), serial recall (recall of list items in the order of study), short-term sentence recall (recall specific words from sentences with similar meanings) and paired associate recall (recall of a pair from presentation of one of its members)...." - ThoughtNet evocation by multiarmed bandit could be likened to a serial recall while evocation by intersection could be a free recall.

----
823. (THEORY and FEATURE) People Analytics - Social Network and Urban Sprawl Analytics - Ricker - Population Dynamics - Functions, Loops, Variables, Timeseries and Plots in R - 22,23,24,25,26 May 2020, 2,3,4 June 2020 - related to 487,572,770

R along with SAS is a standard BigData analytics programming language having MATLAB capabilities. Code example Ricker.R in code/ defines a function ricker() which computes the Ricker Chaotic Biological logistic function for population dynamics (Nt * exp(r(1 - Nt/K)) a variant of [Robert May] logistic for pandemic non-linear dynamics modelling (Initial condition is per generation and K=1). The code example demonstrates assignment of variables by "<-" operator, for loop block, list initialization by ":", initialization of timeseries by ts and plotting graphics by plot(). Function block by "<- function()" has been commented which can be run as main. R scripts could be executed either by RStudio or Rscript CLI. R Graphics plot from RStudio and logs from Rscript have been committed.

References:

1.R for Beginners - [Emmanuel Paradis] - https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
2.Biological Logistic - [Robert May] - http://abel.harvard.edu/archive/118r_spring_05/docs/may.pdf - equation 4

----
824. (FEATURE) People Analytics - Social Network Analysis - CoronaVirus2019 analyzer - Statistics in R - Data Frames,Factors,Histograms,Fitting a Probability Distribution - 27 May 2020, 2,3,4 June 2020 - related to 487,572,770

R is the prominent choice for data science because of immense builtins for statistical analysis. Contrived Code example FileIOVectStats.R demonstrates analysis of CoronaVirus2019 dataset from https://www.worldometers.info/coronavirus/ by computing following metrics from it:

- Histogram
- ECDF - Cumulative Density Function
- Stem
- Rug
- Find the correlation to Bell curve - Shapiro-Wilk test
- Find the correlation to Bell curve - Kolmogorov-Smirnov (KS) test
- T-two sample test
- Mean,Median,Quantiles summary

COVID2019 data requires preprocessing of the text data and R script executes the following:

- attaching a dataframe by attach()
- instantiate Data Frame after reading the data file by read.table()
- summary() from dataframe
- accessing each field of read.table() dataframe by [[]] operator
- converting string fields to numeric by as.numeric()
- splitting sample into two for T-test
- instantiate a factor object (which categorizes the data)

Statistical tests - Shapiro,KS,T-test - try to find the proximity of the dataset to normal distribution.
Histogram bucketization of COVID2019 shows an initial huge bucket followed by almost equal sized buckets.

----
825. (THEORY and FEATURE) People Analytics - Social Network Analysis - CoronaVirus2019 analyzer - Correlation coefficients in R - Concatenation of Vectors, c() function, Vector arithmetic, Correlation of two datasets - related to 487,572,770 - 2,3,4 June 2020

R provides various facilities for creating sequences - seq(), array(), c(), data.frame(), vector(). Code example VerhulstPearlReed.R analyzes the correlation between a Chaotic sequence from Verhulst-Pearl-Reed logistic law (and biological logistic) and CoronaVirus2019 dataset, both of same length. Spread of memes,fads,epidemics and diffusion of concepts in community is a Chaotic process which can be modelled by Erdos-Renyi random graphs and Cellular automata. Though Chaotic fractal datasets (Mandelbrot) are undecidable, from social network analysis literature it is evident that emergence of giant components in scalefree (Pareto 80-20 rule - number of vertices of degree r is propertional to 1/r^k - high degree vertices are least and low degree vertices are numerous) random graphs are inevitable. Approximate correlation to onset of Chaos could be deciphered by varying initial condition and bifurcation parameter(lambda) of logistic and finding maximum correlation to CoronaVirus2019 dataset. COVID2019 dataset is scaled to decimals in interval [0,1] by vector division feature of R which applies division by maximum element to each element of sequence. Vector division could also be performed on dataframe objects (commented). Scale-normalized dataset is then correlated to Chaotic sequence by Kendall,Spearman,Pearson coefficients (~45% maximum). Concatenation is done by c() function within for loop which reassigns the vector after concatenating next element. Double precision arithmetic has been coerced. Factor objects of both logistics are printed which show the levels of data. Graphics are plotted by timeseries plot() function (most recent to the left) - Chaotic sequence oscillates heavily by period doubling. Low p-value implies null hypothesis (no

correlation) can be rejected with high confidence. Maximum correlation occurs at lambda=4.0 and initial condition=0.00000000000001 and for increasing lambda both logistics coincide (resemble a heaviside step function). Return value is a concatenated vector by c().

References:

825.1 R and S documentation - https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/c

----
828. (THEORY and FEATURE) People Analytics - Social Network Analysis - CoronaVirus2019 analyzer - Linear Models - 11,12 June 2020 - related to 487,572,752,770,823,824,825

R has variety of builtins for Linear Regression and Logistic Regression - Linear Models - lm() and glm() functions. Code example LinearModels.R computes linear model lm() and generalized linear model glm() for CoronaVirus2019 dataset which correlate per-day and total fields by various family of fit measures - gaussian, gamma, poisson, binomial and link functions - logit, identity, log, inverse, 1/mu^2. It also demonstrates matrix creation by cbind() which combines vectors and $ notation for dataframe fields. Summary including linear.predictor for each model is printed to testlogs/LinearModels.log.11June2020. Pandemics as Social network fad diffusion are traditionally fit to Chaos, Cellular automata, ERSIR random graph model, SIS random graph model, exponential or poisson distributions. An example Theoretical regression model which explains the spread based on population density (average degree in urban sprawl social networks) could be:

Per day Spread = weight1 * total_infected - weight2 * total_recovered - weight3 * deaths + weight4 * recovery_time ... + constant

for variables total_infected, deaths, recovery time and total_recovered and weights weights3=1, weight4, weight1=average_degree and weight2=1 (spread is en masse and depends on degree of network and recovery time but recovery benefits only individual). By CAGraph social network concept diffusion model, weight1=average_degree=8 for 2 dimensional cellular automaton increment growth rule. Linear model is dependent mostly on recovery time which then becomes:

Per day Spread = 8 * total_infected - 1 * total_recovered - 1 * deaths + weight4 * recovery_time ... + constant

----
831. (THEORY and FEATURE) People Analytics - Social Network Analysis - CoronaVirus2019 analyzer - Cellular Automaton Graph (CAGraph) - Regression Models of Diffusion - Iteratively Re-Weighted Least Squares (IWLS) - 22,23,24,25 June 2020 - related to 678,740,762,828,830 and all sections on Business Intelligence, Voting Analytics, Urban Sprawl Analytics, Random Walks on Expanders of NeuronRain Theory Drafts
----
1.In continuation of R CoronaVirus2019 models earlier, Cellular Automaton Graph diffusion logit in 828 has been implemented by glm() and lm() utilities of R library.
2.Two R functions in CAGraphLogit.R - cagraphlogit() and cagraphprojections() - respectively learn a regression model from COVID19 data and project it to a later datapoint to obtain per day global spread.
3.glm() and lm() functions learn the model:

Perdiem ~ Infected + Deaths + Recovered + RecoveryTime

based on Susceptible-Infected-Recovered (SIR) data gathered from few giant geographic clusters.
4.Logs of code/testlogs/CAGraphLogit.log.24June2020 detail the predictors, coefficients and fitted model.
5.Regression model coefficients are learnt by Iteratively Re-Weighted Least

Squares implementations of glm() and lm(). CoronaVirus2019 dataset has been chosen as a representational dataset because of its unprecedented global scale and randomness which could simulate any other random process involving people including business, economy and majority voting (Preferential attachment of high market share brands versus Double Jeopardy of low market share brands - new products gain market share by word of mouth and promos, spread of opinions as opposed to pandemics decide elections than individual rational decision making - correlated majority or statistical dependence of voters - CJT for Correlated Majority -
https://www.sciencedirect.com/science/article/abs/pii/016726819400068P ,
Markets' spikes and corrections are explained by Bounded rationality and Irrational exuberance).
6.All learnt weights have been multiplied by 400 except total_infected which is multiplied by 8*400 = 3200 as 2-dimensional Cellular Automaton Graph heuristics for High degree Expander graphs (from Section 740, degree 400 graphs are Expanders which facilitate huge spread subject to |eigenvalue(k)|/d-regularity <= 1/10) - Rationale being Spread in an Urban Sprawl social network is a Random walk on Expander Cellular Automaton Graph. Function cagraphprojections() computes the following global spread per day from the learnt model applying signs (and multiplications by 8 and 400) for each dependent variable of recent COVID19 SIR data (by abs() of weights to ignore signs and without abs()). Model could be refined by incorporating additional dependent variables:

```
[1] "Per day spread(abs):"
[1] 121087.5
[1] "Per day spread:"
[1] 149975.7
```

7. Section 830 has a derivation for number of possible arrangements or 2-colorings of people subject to avoidance criteria. Number of possible arrangements of people (uniquely identified by integers) bijectively equal number of possible complementary equations or complement diophantines defined on the integer sequences (Complementary Equations, Functional equations, Beatty sequences, Taichi - partition of Integer sequences -
https://cs.uwaterloo.ca/journals/JIS/VOL10/Kimberling/kimberling26.pdf). In the context of diffusion in social network, 2-coloring reduces to Infected-Uninfected. An example combinatorial avoidance: Infected and Uninfected couldn't be juxtaposed.
8. Section 762 defines least square model for variables in business and econometric datasets - By replacing the variables of COVID19 analyzer, similar models could be learnt for those datasets as well.

----
870. (THEORY and FEATURE) People Analytics - Social Network Analysis - CoronaVirus2019 Analyzer - Exponential Fit in R - related to 744,831
and all sections on Chaos, Cybercrime analyzers, Game theoretic Cybersecurity, Pseudorandomness, Random walks on Cellular Automata and Expander Graphs - 11 October 2020, 12 October 2020
----
1. CoronaVirus2019 dataset has been thus far fit to non-exponential probability distributions and Chaotic non-linear models.
2. Chaotic models suffer from theoretical limitations of computational undecidability thereby prohibiting accurately learnt models.
3. This commit implements exponential fit of CoronaVirus2019 dataset till 11 October 2020, by lm() applying logarithmic version of:

```
Fatalities = A*exp(B*days)
```

which is log2(Fatalities) = logA + B*days (R linear model is used for learning non-linear exponential model)
4. Logs code/testlogs/Exponential.log.12October2020 print the exponential fit model.
5. Previous exponential model is the solution for differential equation d(Fatalities)/Fatalities = B*d(days) which is similar to DEs in Erdos-Renyi Susceptible-Infected-Recovered random graph epidemic model.
6. Differential equation in (5) implies rate of change of fatalities depends on

instantaneous fatalities (Geometric progression or evolving m-ary tree).
7. From coefficients in logs, COVID19 dataset is fit to A=11.27683950 and B=0.04250611