Course Authored By:

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

##
##

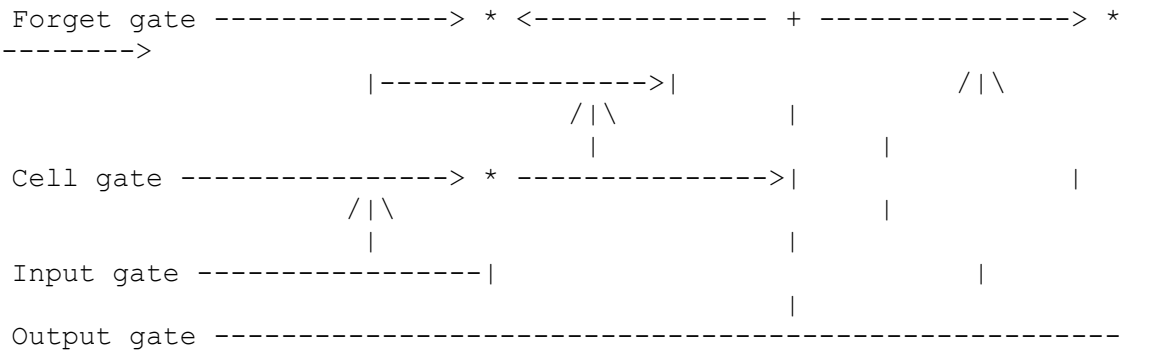
This is a non-linearly organized, continually updated set of course notes on miscellaneous topics in Graduate/Doctoral level
Computer Science and Machine Learning and supplements NeuronRain AsFer Design Notes in:

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:

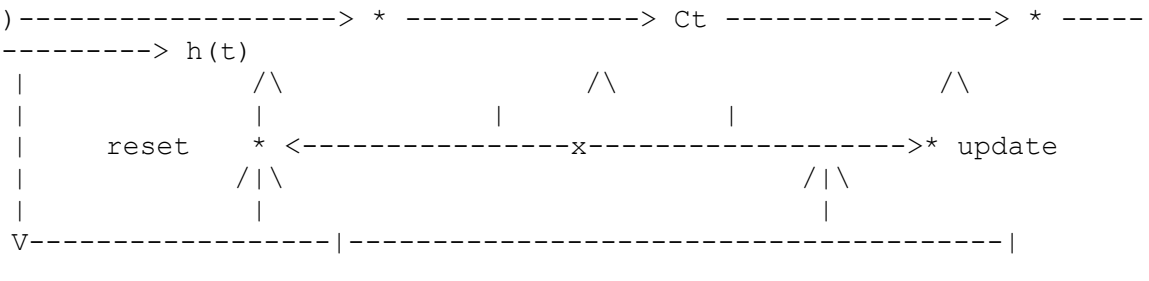
Traditional neural networks have a threshold function which outputs 1 or 0 based on a threshold. But they don't preserve state information over points in time. For example, if there is a requirement that next state depends on present state and an input, usual neural network cannot satisfy it. Recurrent 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:



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:



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 - 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 histogram can be defined by Earth Mover Distance or Wasserstein Distance in addition to RandIndex.

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_KHx8LfeAhVDL48KHQhTBOkQ6AEwAHoECAkQAQ#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), 151â€“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 2â€“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, Computational Geometry, Intrinsic Merit and Originality/Creative Genius - 4 June 2018,3 July 2018,7 March 2019,7 May 2020, 2,3,4 June 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

---

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.

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/

---

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,â€¦,n}; assume without loss of generality that no two points share the same x- and y-coordinates, and that each value in {1,â€¦,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] Ã— [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 xâ€"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]Ã—[j1,j2]), is answered in time O((k+1)lgÏƒ/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