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.