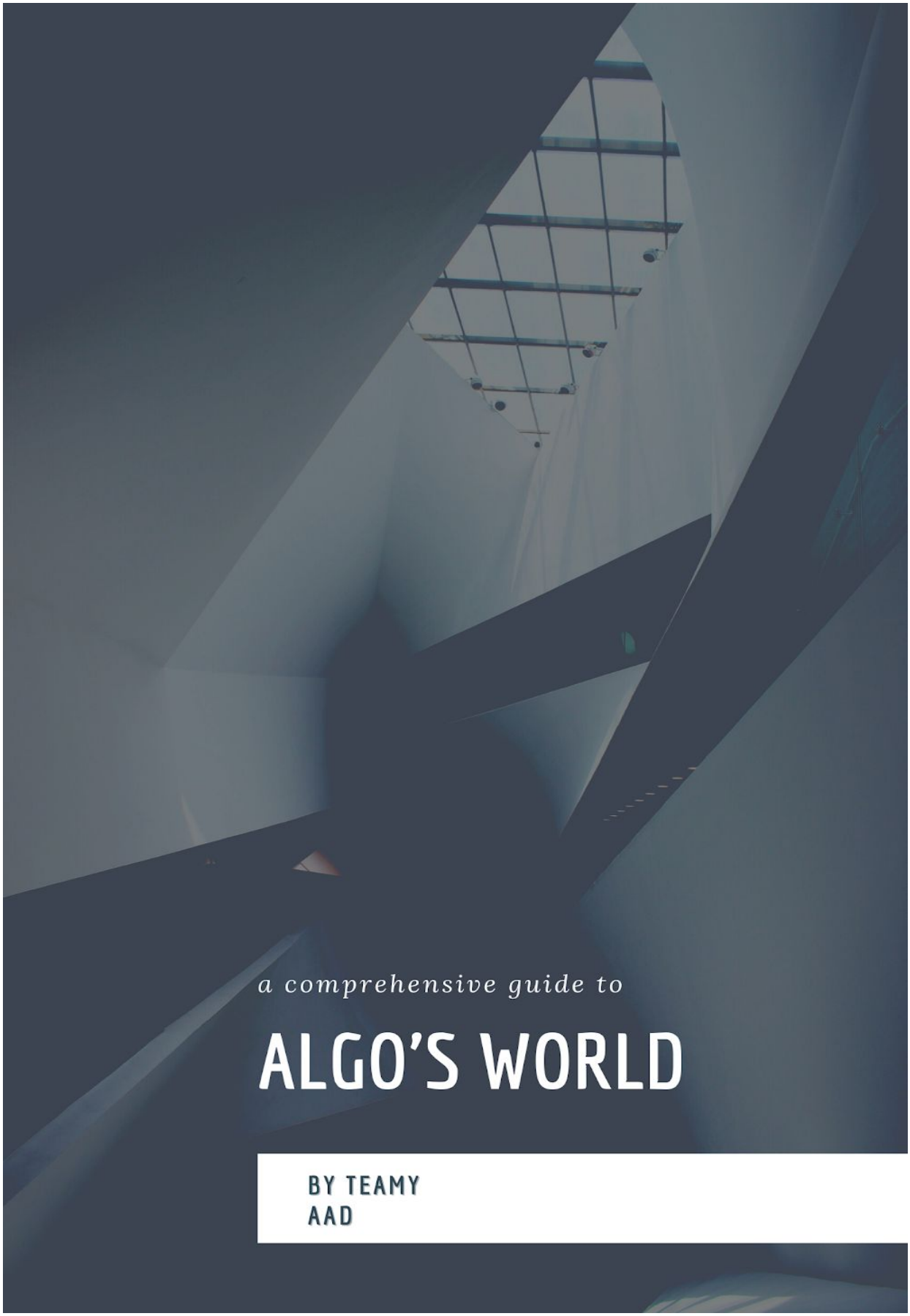


a comprehensive guide to

ALGO'S WORLD

BY TEAMY
AAD

INDEX

S.NO	Topics	Page No
1	Introduction	1
2	Educational institutions	3
3	Plagiarism	11
4	Google search	29
5	Digital Maps	48
6	Cryptography	60
7	Epicycles	75
8	Fourier transformation	81
9	Signal processing	99
10	Monte carlo	121
11	Pwd and Qr generation	131
12	Shopping Malls	148
13	Banking and Transactions	155
14	Stock Market	169
15	Econometrics	177
16	Smart city	191
17	Games	198
18	Metaphor based	238
19	Appendix	257

Introduction

We created this as part of our AAD course project.Our motive is to make a book that consists of a description of algorithms used in various fields. We intend to give the reader an overall idea about the algorithm to the extent possible and give necessary technical details about it. We will be giving appropriate references wherever necessary for the reader to explore more about their desired algorithms. The book will be filled with many intriguing puzzles and astonishing facts nearing the end of the chapter that will lighten the mood of the book.

The chapters in the book will come from various diverse fields such as security, customer focussed sectors, some google functionalities, and also many topics which involve a great deal of core subjects like electronics, biology, maths etcetera. Since all these topics are diverse, we have included some small detours at the end of every major topic. We intend to keep the reader engaged and spice things up by our motto of beauty in diversity

Towards the end of the book, we intend to provide the reader with basic algorithms in the field of data structures and programming languages that will help beginners as well as any reader for immediate reference. These will be included in the glossary with some minimal explanation. Last but not least , we have added some references for the reader to explore more about any topic they are interested in. Enjoy the experience of a beautiful world, The **Algo's World**!!!

Contributors

1. **MC Bhavana 2019101100**
2. **M. Pravalika 2019101098**
3. **Ganne Jyotheeshwar 209101099**
4. **Shri Vidhatri M M 2019113006**
5. **Abhinav Chowdary M 2019101109**

1. Educational institution

Every child needs education. Education helps people to understand the world better and change the world to its better. Nowadays education has become a business. Education is one of the largest and steadily growing sectors worldwide.

In this digital age, algorithms are ubiquitous!

From your phone to your computer to even your microwave, practically every device you use has something to do with algorithms. They are the literal foundation of every technology known to man. Given their importance in our daily lives, it's worth taking a brief look at them to better understand what they are and how they became so relevant in our lives.

Here's our attempt to explain the algorithms used in the educational field.

Evolution

An algorithm is one that aims to solve a particular problem. The first evidence of the use of algorithms is the artificial fire in the caves in Africa and Babylonians had developed a numerical system using *cuneiform numerals* to count and they preserved those calculations on tablets.

Also, we can find instances of algorithms in Euclidean mathematics, Archimedes' approximation of Pi, or **Eratosthenes'** *to calculate the prime number*s and Boolean

system which takes the most of the algorithmic world. Next, Alan Turing came up with a mathematical model of a hypothetical computing machine. The states required for counting with the
Turning machines are some of the easiest to understand. In the educational system from priests teaching to AI teaching. The major problem in Covid was the grading system(grading algorithm) by an algorithm that has wrong predictions.

Prediction of graduation rate

Researchers analyze student data to predict the graduation rate by looking at the characteristics of students enrolled and to take corrective actions at an early stage or improve the admission process

This graduation prediction rate helps institutions to analyze their students' performance and also helps students to know at what stage they are in.

Before this discovery of the graduation prediction algorithm, both institutions are having doubts about whether their students will pass or not and students also don't have a clear image of their performance. Using these algorithms helps both of them to function in a better way.

Econometric Modeling

Colleges and universities use econometric modeling to predict how many students will enroll, how much revenue they will raise, and how much of a discount they need to give certain students to ensure they enroll. There are plenty

of other uses for machine intelligence, some positive and some highly questionable:

- **Prescriptive learning**

  Migrant students who miss part of the school year due to the family's need to travel to work benefit from perspective learning. So does the learner who needs a quick refresher or just wants to repeat a lesson. Individualized and differentiated instruction gives all students what they need to become successful learners. Algorithms make it possible to scaffold support, assess learning, and speed up or slow down the curriculum.

- **Risk identification**

  Police departments have been using predictive algorithms to determine the relationship between potential suspects, gang members, social media posts, and the likelihood a person would commit a crime. Schools have adopted the algorithmic formula to identify potential dropouts. While we want to keep our students in school, there's a real risk of profiling through systemic bias.

- **Unified enrollment**

  What if a computer decided where your children went to school or which teacher they will have?UE is already happening in cities around the country. Complex algorithms recommend the best matches between students and the schools on their "preferred" lists. The algorithmic initiative is meant to remove bias and subjectivity from the enrollment process. It's also supposed

to prevent any opportunity to game the admissions process.

E-LEARNING

Having virtual classrooms has changed communication and interaction between teachers and students, education resources, and others. Although e-learning has developed new educational models, researchers have become aware of the need to sustain the development of abilities and competencies to promote intellectual capital.

Clustering algorithm

Clustering is an analysis of unsupervised learning problems. It is often **used as** a data analysis technique for discovering interesting patterns in data, such as groups of students based on their performance. There are many algorithms. Let's see the k-means algorithm which clusters the data.

The k-MeansAlgorithm

k-Means Is one of the simplest unsupervised learning algorithms used for clustering. Given D, a data set of n objects, and k, the number of clusters to form, a partitioning algorithm organizes the objects into k partitions ($k \leq n$), where each partition represents a cluster. The clusters are formed to optimize an objective partitioning criterion, such as a dissimilarity function based on distance. The algorithm is composed of the following steps:

1. Place k points into the space represented by the objects that are being clustered. These points are present initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the k centroids.
4. Repeat steps 2 and 3until the centroids no longer move.This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

The k-means simple clustering algorithm that has been improved to several problem domains.
This algorithm uses an iterative refinement technique.
Given an initial set of k means $m_1^{(1)},...,m_k^{(1)}$ (see below), the algorithm proceeds by alternating between two steps:

Assignment step: Assign each observation to the cluster with the nearest mean: that with the least squared Euclidean distance. (Mathematically, this means partitioning the observations according to the Voronoi diagram generated by the means.)

Update step: Recalculate means (centroids) for observations assigned to each cluster.

AI in the educational field.

- **Personalised learning**

Offering personalised, adaptive learning platforms recognises the diversity that is part of any learning ecosystem. This will be a significant change for universities, as it moves away from the traditional model of "one module guide for all". It will see educators equipped with data sets to analyse and understand the needs of individuals. And work can be automatically adapted to the style and pace of learning for each particular student.

- **Smart campus**

Smart classrooms will also enhance the learning experience of the students. A classroom connected to the Internet of Things equipped can adapt to the personalised settings to prepare the classroom for different faculty members. Monitoring attendance and invigilating exams will also be automated and made much more robust.

- **Monitoring performance**

Another dimension of using AI innovations in universities will be the use of block chains. This will revolutionise how universities operate, as higher

education institutions use this technology to automate recognition and the transfer of credits, potentially opening up learning opportunities across universities.

These facilities make students use those institution facilities in a more efficient way and helps both students and management in their functionality.

Small detour!! :)

Have you ever wondered how a selection process takes place when people on both sides rank the ones on the other side, and then pairings are made to satisfy everyone in the best way possible? To make this problem more clear, let's assume there are n men and n women, all women have their own ranking of men in their mind and vice versa. Now, we should pair them off in such a way that they all get the best prospective spouses according to their individual rankings. This problem is formally called the gale shapley algorithm. Here stability is defined as both of the participants not being happier with an alternative match. And the explanation of the algorithm is given below.

The Gale–Shapley algorithm (also known as the deferred acceptance algorithm) involves a number of "rounds" (or "iterations"):

1. In the first round, first a) each unengaged man proposes to the woman he prefers most, and then b) each woman replies "maybe" to her suitor she most prefers and "no" to all other suitors. She is then provisionally "engaged" to the suitor she most prefers so far, and that suitor is likewise

provisionally engaged to her.

2. In each subsequent round, first a) each unengaged man proposes to the most-preferred woman to whom he has not yet proposed (regardless of whether the woman is already engaged), and then b) each woman replies "maybe" if she is currently not engaged or if she prefers this man over her current provisional partner (in this case, she rejects her current provisional partner who becomes unengaged). The provisional nature of engagements preserves the right of an already-engaged woman to "trade up" (and, in the process, to "jilt" her until-then partner).
3. This process is repeated until everyone is engaged.

This algorithm is guaranteed to produce a stable marriage for all participants in time $O(n^2)$ where n is the number of men or women.

The downside to this algorithm is that it always yields the one that is best for all men among all stable matchings, and worst for all women.

2. Plagiarism Detection

"Plagiarism is commonly defined as the theft of intellectual property". And this plagiarism has been a major drawback when it comes to reviewing academic works. Thus the detection of this plagiarism is a very important issue in today's internet era. There have been various approaches and methods to solve this plagiarism issue. We have listed some of those, tried to give pseudocodes for those algorithms and made our own feeble and baby steps towards that algorithm detection.

Plagiarism detectors will break down your entered document into phrases and then use search engines to find web pages that have matching phrases. Obviously if too many matching phrases, especially from the same source then your document is most likely plagiarized. The phrases are usually the sentences or chunks of sentences from your document.

We will go from the most basic idea of comparing lines/ a set of words from two documents to trying to understand some pretty complex plagiarism detection codes(like moss :))

Given below is a very naive approach to comparing two documents that have similar text in them and we observe approximately 42 percent plagiarism even with a lot of similarities.

This could be stated as a first step taken towards plagiarism detection and as we move on in this chapter we will see a few more attempts to make this code efficient.

The next approach is the string comparison type of algorithm. We will use algorithms like the Knuth-Morris-Pratt algorithm for pattern searching or the

Rabin-Karp hashing algorithm that would be used for searching a string within another string. But before we use those algorithms, we are going to apply a few tricks that will make our pattern search more reliable for plagiarism detection.

- The first thing to do is to convert the given document into a group of words that may be in an array and remove punctuation marks, unimportant numbers, and convert all words to the same case. This will make sure that simple case conversion or extra punctuation marks would still be considered plagiarised.

  This process can also be called tokenization. Tokenization is usually used in e-commerce businesses to minimize cost and complexity and for security reasons. But here, we can use that same tokenization idea to achieve our first step.

  //plain tokenization

```
import nltk
word_data = """input doc as text"""
nltk_tokens = nltk.word_tokenize(word_data)
print (nltk_tokens)
```

- The second step is the stopword removal step. This step removes basic words to make sure we compare only the important useful parts of the document.

Below is a list of some common stopwords that offer little addition to the real meaning of a sentence.

```
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
"you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in',
'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there',
'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other',
'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't',
'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
"doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't",
'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't",
'wouldn', "wouldn't"]
```

Here are a few key benefits of removing stopwords:

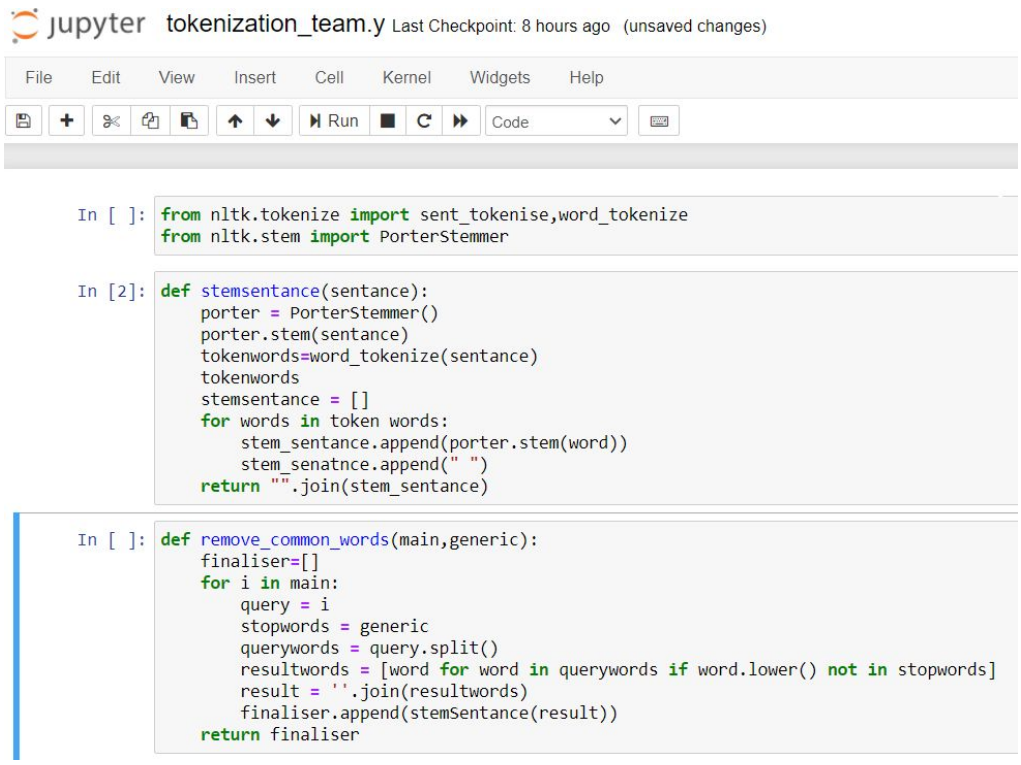
- On removing stopwords, dataset size decreases and the time to train the do the string comparison also increases
- Removing stop words can very well help improve the performance as there are fewer and only meaningful word tokens left. Thus, it could increase the accuracy of the document information.

- Even search engines like Google remove stopwords for fast and relevant retrieval of data from the database.
- The third process is termed as stemming. This is a type of string functionality that removes the prefixes and suffixes and leaves the basic words for comparison. This is very useful to compare when documents have been changed according to their tense (eg: walked to walking).
- Stemming process can be done using two famous libraries in the nltk package in python: Porter stemmer and Lancaster stemmer. The below code gives a comparative idea between the two.

```
word_list = ["friend", "friendship", "friends",
"friendships","stabil","destabilize","misunderstanding","railroad","moonlight","f
ootball"]
print("{0:20}{1:20}{2:20}".format("Word","Porter Stemmer","lancaster
Stemmer"))
for word in word_list:

print("{0:20}{1:20}{2:20}".format(word,porter.stem(word),lancaster.stem(word
)))
```

```
Word                 Porter Stemmer      Lancaster Stemmer
friend               friend              friend
friendship           friendship          friend
friends              friend              friend
friendships          friendship          friend
```

stabil	stabil	stabl
destabilize	destabil	dest
misunderstanding	misunderstand	misunderstand
railroad	railroad	railroad
moonlight	moonlight	moonlight
football	footbal	footbal



- The fourth and final step can be done in various ways. One of these is the KMP algorithm mentioned earlier. This algorithm is used to check if a string is the substring of another and thus can be used to check plagiarism in the strings obtained from the previous steps.

prefix[]

```
prefix[0] = 0, border = 0
for i := 1 to |S| - 1:
   while (border > 0) and (S[i] != S[border]) :
       border = prefix[border - 1]
   if S[i] == S[border] :
      border = border + 1
   else: border = 0
   prefix[i] = border
```

Time Complexity: O(m + n)

Space Complexity: O(m + n)

Another algorithm that can do the similar task is theRobin-karp algorithm that uses hashing.The steps are given below

- Rolling-Hash data structure maintains the hash value of a string as seen from a sliding window.
- Generate an empty rolling hash where strings are represented as multi-digit numbers using the given base. The hash-function used is h(x) = x mod prime.
- Generate the hash value obtained after appending the new character to the current internal string.
- Generate the hash value obtained after removing the first character from the current internal string.
- Generate the hash value obtained after sliding the window one step to the left, removing the previous first character and appending a new last character to the current internal string.

- Apply Karp-Rabin to check whether string s is contained into text t.

Pseudocode

```
class Rolling_Hash:
    def __init__(self, base, prime):
        self.base = base
        self.prime = prime
        self.base_inverse = pow(base, prime-2, prime)
        self.magic = 1
        self.hash = 0
    def append(self, new):
        self.hash = (self.hash*self.base + ord(new)) % self.prime
        self.magic = (self.magic*self.base) % self.prime
    def remove(self, old):
        self.magic = (self.magic*self.base_inverse) % self.prime
        self.hash = (self.hash - ord(old)*self.magic) % self.prime
    def slide(self, old, new):
        self.hash = (self.hash*self.base - ord(old)*self.magic + ord(new))%self.prime
def karp_rabin(s, t, p=257):
    rhs = Rolling_Hash(256, p)
    rht = Rolling_Hash(256, p)
    for i in range(len(s)):
        rhs.append(s[i])
        rht.append(t[i])
    if rhs.hash == rht.hash and s == t[:len(s)]:
        return True
    for i in range(len(s), len(t)):
        rht.slide(t[i-len(s)], t[i])
        if rhs.hash == rht.hash and s == t[i-len(s)+1:i+1]:
            return True
    return False
```

- Another algorithm called boyer moore algorithm is a similarly used one for string comparison.

The four steps thus listed above together form the string comparison type plagiarism detection.

Plagiarism detection using vector space models

- A vector space model represents the documents by term sets.This approach has three types: unigram, bigram and trigram. For unigram every word in the document is considered one term, in the bigram every two words and every three words in the trigram.
- In this the preprocessing is done by eliminating prefixes and suffixes, removing case differences and punctuation marks,diagrams and pictures. After this, the data is separated into uni, bi and tri gram models. These vectors are then compared using cosine similarity.
- The unique point to be noted in this approach is that the weight was also considered according to a formula “td-idf” to find out the final plagiarism amount. Usually it is to be noted that the unigram model of approach is less reliable compared to the bigram and trigram model. This approach was more efficient when the cosine measure was used rather than the Jaccard measure. These measures are more clearly explained in the next subtopic.

On a general note , there are some algorithms and measures that help a great deal in finding similarity in text documents. Some of them are

- Minkowski distance

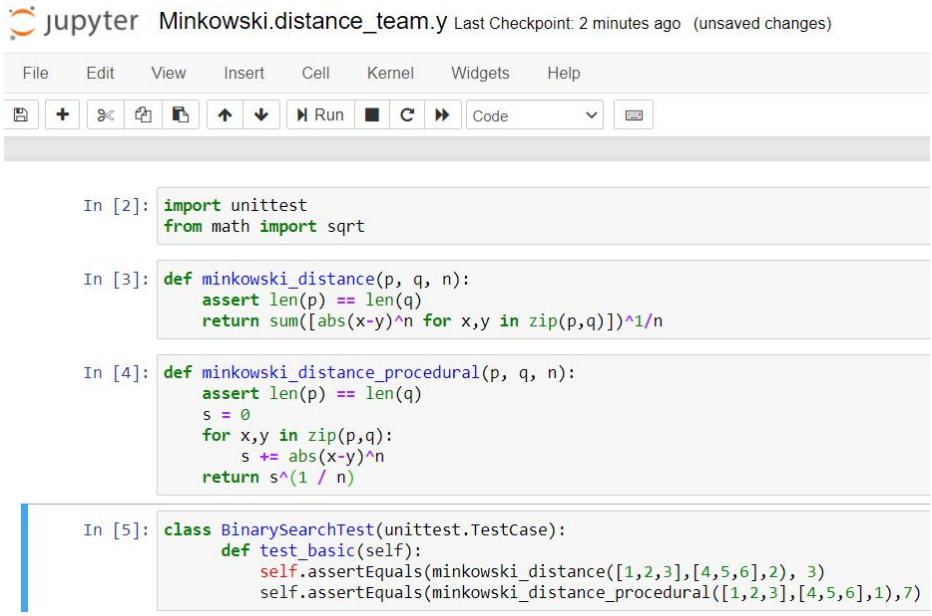
- Manhattan distance
- Euclidean distance
- Hamming distance
- Cosine distance

We will try to explain some of these to the extent possible.

Minkowski distance

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Minkowski distance is a generalized distance metric. We can manipulate the above formula by substituting ‘p’ to calculate the distance between two data points in different ways. Some common values of ‘p’ are:- p = 1, Manhattan Distance,p = 2, Euclidean Distance,p = infinity, Chebyshev Distance. The p=1 and p=2 are very famous and important kinds of distances in data mining studies.



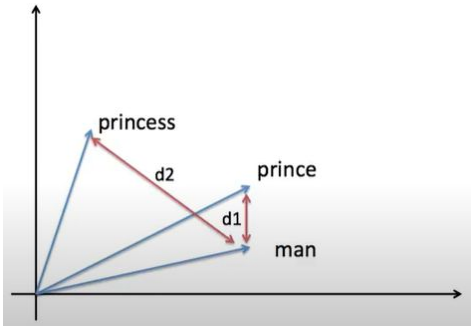
Euclidean distance

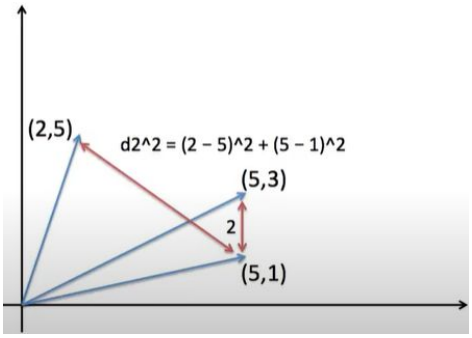
The general formula for euclidean distance is

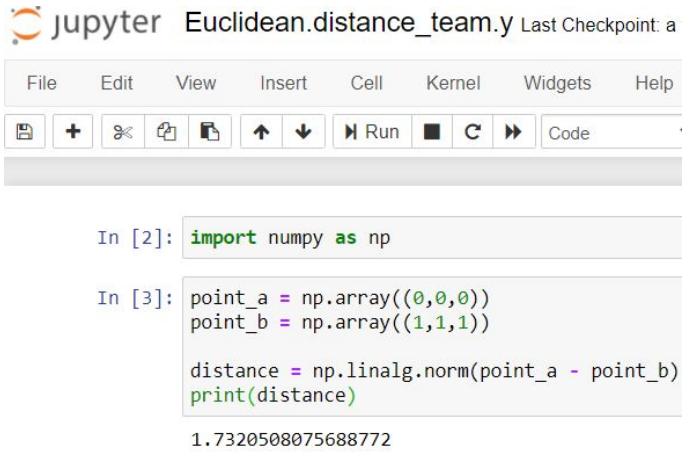
$$dist = \sqrt{\sum_{k=1}^{n}(p_k - q_k)^2}$$

here, n is number of dimensions(or attributes) , p_k and q_k are respectively the k^{th} elements of the objects p and q

This value is higher, when the objects are more dissimilar.







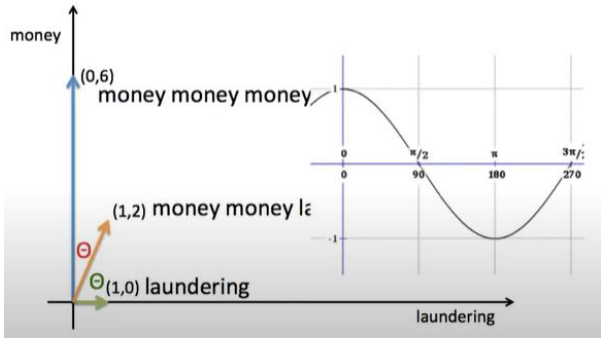
Manhattan distance

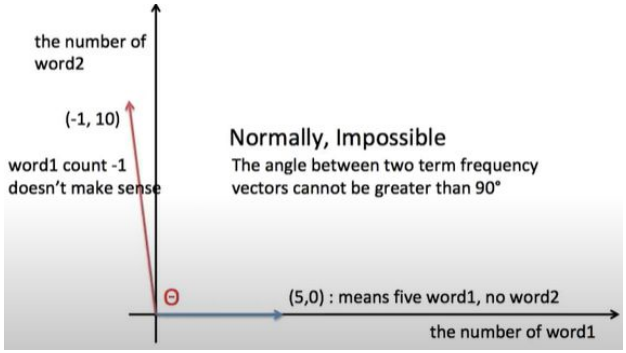
$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Taxicab geometry is a form of geometry in which the usual metric of Euclidean geometry is replaced by a new metric in which the distance between two points is the sum of the (absolute) differences of their coordinates.Its a special case of the minkowski distance and is used widely.

Cosine similarity

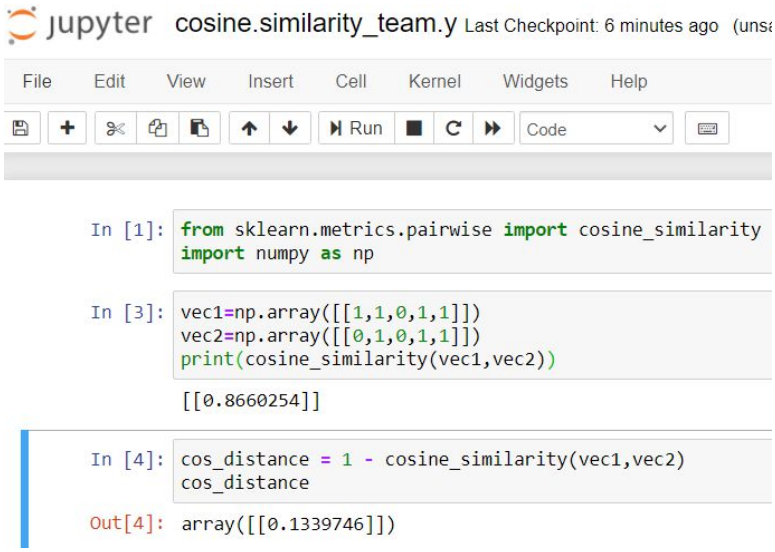
In the context of documents, we usually use this cosine similarity. This is because cosine similarity is really nice for metric documents and we get the similarity measure between 0 and 1 and this value doesn't suffer from dimensionality. And since the documents used tend to get very long, this dimensionality is a pretty reasonable issue and this mesure can avoid some of its complications. For understanding the maths behind this cosine similarity refer [Dot Product] .





$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

cosine distance = 1 - cosine similarity



Chebyshev distance

The Chebyshev distance between two vectors or points p and q, with standard coordinates and respectively, is:

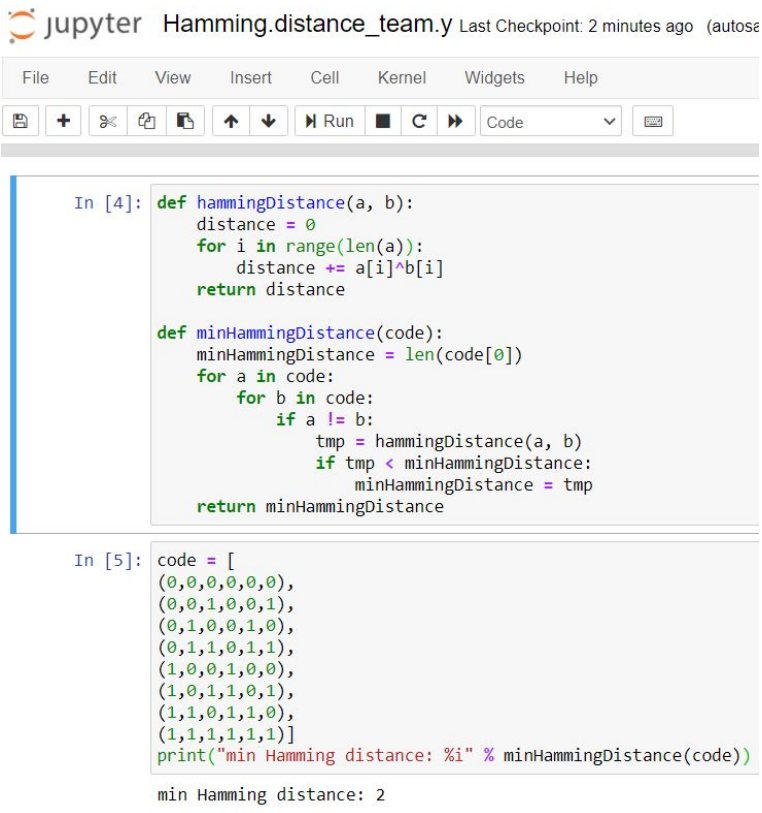
$$D_{\text{Chebyshev}}(p,q) := \max_i(|p_i - q_i|).$$

It is also known as chessboard distance, since in the game of chess the minimum number of moves needed by a king to go from one square on a chessboard to another equals the Chebyshev distance between the centers of the squares

Hamming distance

The Hamming distance between two integers is the number of positions at which the corresponding bits are different. This is the bitwise comparison distance that is used as frequently as the manhattan distance in data mining.In another way, it measures the minimum number of substitutions required to change one string into

the other. It is used in telecommunication to count the number of flipped bits in a fixed-length binary word as an estimate of error, and therefore is sometimes called the signal distance.



Jaccard similarity

Jaccard distance and coefficient are measurements of asymmetric information on binary and sometimes non-binary variables.

The Jaccard distance is complementary to the Jaccard coefficient.

Jaccard Similarity = (Intersection of A and B) / (Union of A and B)

Jaccard distance = 1- Jaccard coefficient

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

Just like cosine similarity, the range of Jaccard similarity is 0 to 1 i.e. if the score is 0, it means they are mutually exclusive and if the score is 1, it means they are identical.

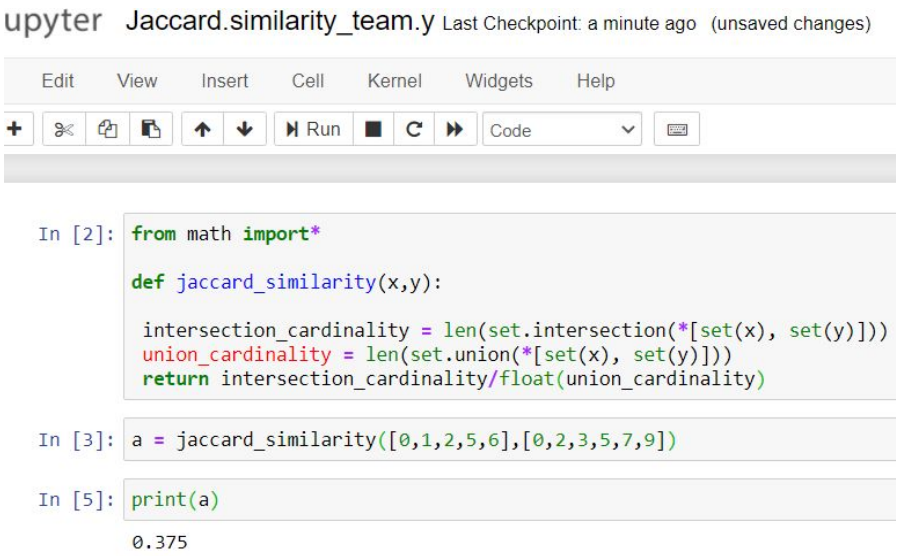
Refer to [Euclidean Distance, Cosine Similarity and Jaccard Similarity] for the implementation

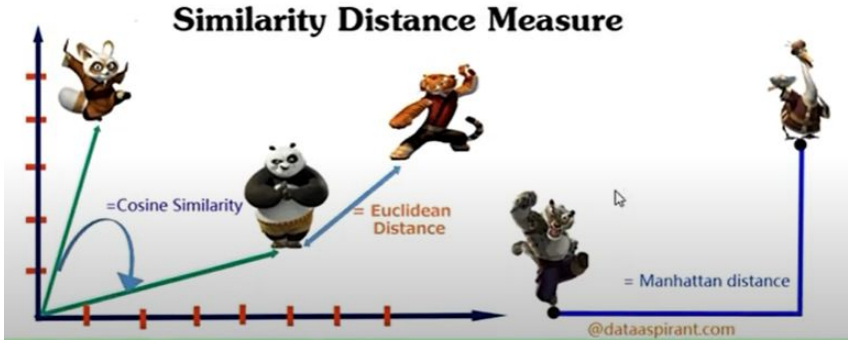
$$J_\mu(A,B) = \frac{\mu(A \cap B)}{\mu(A \cup B)},$$

$J_{\breve{u}}$ is the Jaccard coefficient where u is a measure on measurable space.

$$d_\mu(A,B) = 1 - J_\mu(A,B) = \frac{\mu(A \triangle B)}{\mu(A \cup B)}.$$

d is Jaccard distance and u is a measure of measurable space.





Euclidean distance is rarely preferred due to the curse of dimensionality and in those cases, the other type of distances are chosen over euclidean distance.

Small detour!! :)

Hangman:

- The selected word to be guessed has been taken from a known dictionary.
- You are given N lives, and thus have to maximise the probability of guessing all the letters in the word without making N mistakes (i.e. you may have an unlimited number of correct guesses).
- Each word in the dictionary has equal probability, for the sake of this exercise (i.e. the word is chosen randomly)

Algorithm:

- For a pattern such as ..e.. with tried letters: e,s,t
- Check against words of only n digits (in this case 5)
- Create a list of possible words
    - Create a regex from the supplied info
    - In this case it would be [^est][^est]e[^est][^est]
    - Parse your word list for words that match this regex
- Cycle through each word, counting up the number of times every letter appears
- Your optimal next guess is the most likely letter

3. Google search

Google search is a necessity nowadays for everyone. Whether a student, teacher, job holder, or old person almost everyone is using google to know things better.
Google uses so many algorithms to make this job. It has to satisfy users. Some will be described here

Google Search, or simply **Google**, is a web search engine developed by Google LLC. It is the most used search engine on the World Wide Web across all platforms, with 92.62% market share as of June 2019,handling more than 5.4 billion searches each day.

The order of search results returned by Google is based, in part, on a priority rank system called "PageRank". Google Search also provides many different options for customized search, using symbols to include, exclude, specify or require certain search behavior, and offers specialized interactive experiences, such as flight status and package tracking, weather forecasts, currency, unit, and time conversions, word definitions, and more.

Pagerank

When we search on google we get a lot of pages. But how google order pages it got. They should have some priority. Here comes the page rank algorithm.

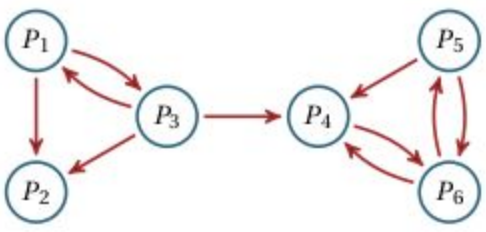
It gives ranks to web pages based on popularity score. Let's say we have gone to one website which has hyperlink to youtube. There is a probability to click on that link. The probability to go to that link depends on the probability of the website where we are.

Basic idea on algorithm

Make a directed graph with pages as nodes(n nodes) and hyperlinks as edges. Les assign probability to all nodes as 1/n. We change the probability of each node on each iteration. On each iteration every node takes probability from the node where it is getting pointed. Every node shares its probability to all the nodes it has hyperlinks. P(r) = sum(p(u)/outdegree(u)) where u has edge to r.

Hyperlink graph of web pages

A graph with web pages with hyperlinks between them is viewed as directed graph G, called hyperlink graph of web pages. The nodes of the graph are web pages. There will be an edge between two nodes u and v if there is a hyperlink of page v in page u. If there are multiple links from u to v, we consider them as one link.



Here is an example of a hyperlink graph. Here P1 has hyperlinks of P2 and P3, P2 has no hyperlinks, P3 has P1,P2 and P4, and so on for all nodes.

Page rank Equations

We assign a score to each node to assign ranks of pages. Nodes get scores from nodes which are pointing to this node. Nodes contribute to nodes which are pointed out by them.
Nodes which have more innerlinks get more probability. Nodes which have hyperlinks from a node which has a high score, will also get a high score.

Let the score of Pi is r(Pi) and the number of outlinks of Pi is o(Pi). Each node Pj that links to Pi contributes some of its score to the score of Pi. If Pi has link from Pj , then Pj contributes
r(Pj)/o(Pj). We get a score of Pi by adding all the contributing scores that have links.

r(Pi) = sigma(r(Pj)/o(Pj)) Pi has inner links from Pj.

Here is the example of pagerank equations for the previous graph.

$$r(P_1) = \frac{r(P_3)}{3},$$

$$r(P_2) = \frac{r(P_1)}{2} + \frac{r(P_3)}{3},$$

$$r(P_3) = \frac{r(P_1)}{2},$$

$$r(P_4) = \frac{r(P_3)}{3} + \frac{r(P_5)}{2} + \frac{r(P_6)}{2},$$

$$r(P_5) = \frac{r(P_6)}{2},$$

$$r(P_6) = \frac{r(P_4)}{1} + \frac{r(P_5)}{2}.$$

The Hyperlink Matrix

The hyperlink graph can be converted to hyperlink matrix A. A is a nxn matrix which has 1 at Aij if there is a hyperlink from Page i to Page j.

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Hyperlink matrix for given graph.

Solving equations using Matrices

All the equations can be summarised into a matrix H. To do this we normalize the row of matrix A (i.e H[i][j] = A[i][j]/ sum(A[i])).

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \quad \rightsquigarrow \quad H = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

Then our six equation will become

$$\begin{aligned} &\begin{pmatrix} r(P_1) & r(P_2) & r(P_3) & r(P_4) & r(P_5) & r(P_6) \end{pmatrix} \\ &\quad = \begin{pmatrix} r(P_1) & r(P_2) & r(P_3) & r(P_4) & r(P_5) & r(P_6) \end{pmatrix} \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}. \end{aligned}$$

v = v H

V is the score vector of nodes. By solving the above equation we get scores of each node. From there we can assign ranks to the websites. There is also a systematic method for solving such a system of simultaneous equations (the Gaussian algorithm) which can be carried out by a computer. But finding scores for the whole WWW which has 1.5 billion equations is not feasible. So we try to solve the equation in another method.

Power Iteration Method

The equation v = vH can be solved iteratively by calculating the approximation of vector v. For initial equations we give equal score to all.

$$\boldsymbol{v}^{(0)} = \left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right).$$

We can next v's by multiplying v with H.

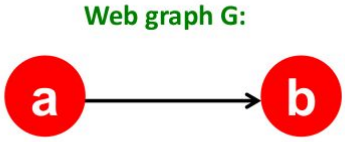
$$\boldsymbol{v}^{(1)} = \boldsymbol{v}^{(0)} H$$
$$\boldsymbol{v}^{(2)} = \boldsymbol{v}^{(1)} H$$
$$\boldsymbol{v}^{(3)} = \boldsymbol{v}^{(2)} H$$
$$\vdots$$

In general v(t+1) = v(t)H. We hope that the sequence will converge to v with v = vH. Normally we find v(t) where |v(t) - v(t-1)| < e for a small value e.

Issues in solution

By using the above discussed we are getting problems in some graphs.

- Example:

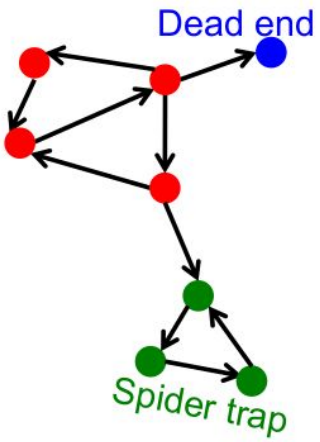


Sequence - (½,½), (0,½), (0,0),...,(0,0)

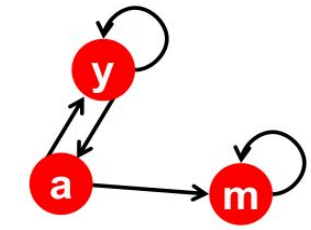
Is it converging to what we want? No. It is going to become 0 for all nodes.

Generally we observe two problems in practice.

- Some pages are dead ends (have no outlinks). All nodes contribute to this node. But this node does not contribute to any node. Such nodes leak out scores.
- Spider traps (all outlinks are within the group). Score will get stuck in the trap. Eventually spider trap absorbs all the score.



Example:



		Iteration i = 0,	1,	2,	3,	…	
r_y		1/3	2/6	3/12	5/24		0
r_a	=	1/3	1/6	2/12	3/24	…	0
r_m		1/3	3/6	7/12	16/24		1

Improved Solution

Add edges from deadend nodes to all nodes. And recompute Matrix H. This will solve issue with issue with deadend.

Spider tarp: Normally we are taking scores for a node u by taking scores from nodes which have hyperlinks to u. To solve the issue a node u gets score from its pointed nodes with a probability of bheta plus score from all nodes with probability of (1-bheta).

I.e v(t+1) = Gv(t) is called Google Matrix.

Where G = bheta * H(nxn) + (1-bheta) * R(nxn)

R[i][j] = 1/n for all i,j <= n

$$G = d\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} + (1-d)\begin{pmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{pmatrix}.$$

G for above given graph

Caffeine Indexing

Let's say now we are in 2020 Oct, if we search for “IPLl”, we get top results belonging to ipl 2020 matches. Why aren't we getting ipl 2019? From our pagerank algorithm, IPL 2019 has a higher score than IPL 2020. People need recent things. If you search for a conference which occurs annually you should get the recent conference results. Freshness of things makes a lot of difference in search. If you search for a book which has 15 editions, showing 4th or 5h edition is not desirable.

To order results based on freshness, Google uses an indexing system called Caffeine. It is the largest collection of web content. Whether it's a news story, a blog or a forum post, you can now find links to relevant content much sooner after it is published than was possible ever before. Every second Caffeine processes hundreds of thousands of pages in parallel. If this were a pile of paper it would grow three miles taller every second. Caffeine takes up nearly 100 million gigabytes of storage in one database and adds new information at a rate of hundreds of thousands of gigabytes per day. You would need 625,000 of the

largest iPods to store that much information; if these were stacked end-to-end they would go for more than 40 miles.

There is a data structure that is very efficient in carrying out an actual search as used very frequently for search applications. Its called trie and the next subsection would convey more about that data structure.

Trie Data structure

Trie is an efficient information re***Trie***val data structure. It is a type of graph and a special data structure with nodes and edges. Trie is used to store strings and is basically an n-ary tree.

Let us assume to have a string 'abc'. We initially take a null node and call it root. And now we start traversing the string abc. Here the weight of the edge from the root we create as a and the node on the other new node called N. now we arrive at b and create a new edge-weighted b and create another node N and then do the same for c.

Let us take a new string as abcd. For this we take the current pointer to be at R. Now we check if there is an edge-weighted a. Since it exists, we go through it to N and check if an edge-weighted b exists and traverse along with the graph until we meet a character of the string for which an edge does not previously exist. And then we can create new nodes for them.

A typical node for the trie would be

```
class PrefixTreeNode {
 constructor(value) {
   this.children = {};
   this.endWord = null;
   this.value = value;
 }
```

We can observe that nodes need not be repeated for all strings. This reduces the space complexity as previously abc and abcd take different memory locations but now shares similar memory locations and the space complexity in this example is the length of the longer string.

If we store keys in the binary search tree, a well-balanced BST will need time proportional to **M * log N**, where M is the maximum string length and N is the number of keys in the tree. Using Trie, we can search the key in O(M) time.

A trie is used for efficient retrieval of data associated with keys. If the key is of length n, then using trie worst-case time complexity for searching the record associated with this key is O(n). Insertion of the (key, record) pair also takes O(n) time in the worst case.

Algorithm

In this algorithm, elements are stored in an array in either increasing or non-increasing order. As in binary search, we divide the array in two equal parts (half-half) after finding the middle point index but here we are dividing the array list into three unequal parts after calculating two locations(P1 and P2) .

The first part of the first pass contains a 25% element, the second part contains a 50% element and the third part contains a 25% element. The element to be searched can be lying in any of these three parts. Similarly in the second pass array list is divided into the unequal parts as 35%,30%, and 35% respectively and in the third pass, array list divided into 45%,10%, and 45% respectively.

After performing these passes partially finished one iteration of the proposed algorithms. If till this iteration if searched element not found then suppose to recursively perform this
Procedure.

Implementation of Proposed Algorithm

//Output: hold the index of the searched item if found and -1 value
if not found
//A is array passed by the main function also TS received START
and END index value of the array list along with value to be
searched and 'I' is a global variable having initial value -10.

```
int TS(int ARR [],int START ,int END,int KEY)
{
int P1,P2;//contain the index of first part and second part of
```

```
array list
IF(START<=END){
I=I+10;//I is used for updating the index
IF(I==30) I=0;
P1=START+(25+I)*(END-START)/100;
P2=START+END-P1;
IF(ARR [P1]==KEY) return P1;
ELSE IF(ARR [P2]==KEY) return P2;
ELSE IF(KEY> ARR [P1] && KEY< ARR [P2])
{ START=P1+1;
END=P2-1;
return TS(ARR,START,END,KEY);}
ELSE IF(ARR [P1]>KEY)
{END=P1-1;return TS(ARR,START,END,KEY);}
ELSE IF(ARR [P2]<KEY)
{ START=P2+1; return TS(ARR,START,END,KEY);}
}//END IF
return -1;
}//end of function
```

If the searched element is not found then this function will return -1 otherwise return the index of the key if the searched element is found at P1 or P2. The proposed algorithms will check the flowing three conditions after finding P1 and P2 when the key is not found at the P1 and P2.

Condition 1: IF the SEARCH KEY is less than P2 and also greater than P1.Then (P1 + 1) will become the FIRST and (P2 - 1) will become the LAST and the procedure will be repeated for the sub list.
Condition 2: IF the SEARCH KEY is greater than P2. Then (P2+1) will become the FIRST and then the procedure will be repeated for the sub list.
Condition 3: IF the SEARCH KEY is less than P1. Then (P1 -1) will become the LAST procedure and will be repeated for the sub list.

After each pass, the array list is divided into three unequal parts. In the worst-case after completion of all three iterations of the kth pass, the total time taken is N(1/2)^k (7/20)^k (9/20)^k To terminate the recursive tree for this recursive procedure we have relation N(1/2)^k (7/20)^k (9/20)^k =1 In worst case and to terminate recursive tree we have the condition as N(63/800)^k =1 After computation the value of k is logB(n) where B=12.6

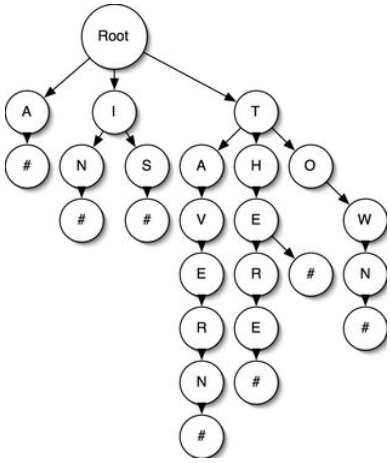
The code snippet to insert new words into trie data structure in python:

```
def add(root, word: str):
    node = root
    for char in word:
        found_in_child = False
        for child in node.children:
            if child.char == char:
                child.counter += 1
                node = child
                found_in_child = True
                break
```

```
        if not found_in_child:
            new_node = TrieNode(char)
            node.children.append(new_node)
            node = new_node
    node.word_finished = True
```

The function for finding a particular prefix in a trie in python:

```
def find_prefix(root, prefix: str) -> Tuple[bool, int]:
    node = root
    if not root.children:
        return False, 0
    for char in prefix:
        char_not_found = True
        for child in node.children:
            if child.char == char:
                char_not_found = False
                node = child
                break
        if char_not_found:
            return False, 0
    return True, node.counter
```



- Trie's retrieval/insertion time in the worst case is better than hashTable and binary search tree both of which take the worst-case time of O(n) for retrieval/insertion.
- Trie is useful when we want to search some string based on the character appearance of the characters within the string:
    - Autocomplete
    - Spell checker
    - IP routing (Longest prefix matching
    - T9 predictive text
    - Solving Boggle
- This trie data structure can be used in complex algorithms like google search to search and find the required page by searching with the least time complexity and also memory needed to store pointers causes it to be less space-efficient during implementations.
- Another advantage of Trie is, we can easily print all words in alphabetical order which is not easily possible with hashing.
- Trie is also used in features such as autocomplete in google search.

- Here the feature enables all the available options to be displayed just by entering the first few characters. By using the trie n-ary tree, we just need to traverse along the tree according to the given string and all the subtrees keeping the last character of that traversal string as root will be our options for auto-complete. Page rank algorithms can be further used to prioritize and organize these options.
- The main disadvantage of tries is that they need a lot of memory for storing the strings. Thus the final conclusion regarding *tries data structure* is that they are faster but require *huge memory* for storing the strings.

Small detour!! :)

The whole world of computer science is generally obsessed with efficiency and coming up with algorithms that are the most efficient and easy to implement. But there are some algorithms that are clearly almost the worst possible algorithms and have been created and started just fun and they are definitely interesting. Two of those fun and almost useless algorithms are the bogosort and stooge sort. These are pretty complicated as well as have bizarre worst case complexities.

- Bogosort simply shuffles a collection randomly until it is sorted.
    - Its average run-time is O(n!) because the chance that any given shuffle of a set will end up in sorted order is about one in *n* factorial, and the worst case is infinite since there's no guarantee that a random shuffling will ever produce a sorted sequence.
    - Pseudocode:
        - **while not InOrder(list) do**
        - **Shuffle(list)**
        - **done**
- Stooge Sort is a recursive sorting algorithm.
- Time complexities:
    - Average time complexity - **O(n ^ log(3) /log(1.5))**
    - Best-Case time complexity - **O(n ^ log(3) /log(1.5))**
    - Worst-Case time complexity - **O(n ^ log(3) /log(1.5))**
    - Worst-case space complexity - **O(n)**
- Algorithm

- If the value at the start is larger than the value at the end, swap them.
- Recursively sort the first 2/3 part of the array.
- Recursively sort the last 2/3 part of the array.
- Again sort the first 2/3 part of the array.
- The array becomes finally sorted.

4.Digital Maps

Before the digital Maps came along, most people were still using **paper maps** to navigate and then we quickly adapted and started using Digital maps.

Digital maps had several advantages over the print equivalent – different zoom levels, ability to add your own points of interest, etc but perhaps the most attractive feature was the ability to use the computer (or smartphone) to calculate the shortest distance from point A to B without needing to figure it out yourself or asking someone who has lived in the place long enough to know it by experience.

How it works

The coordinates and position as well as atomic time obtained by a terrestrial GPS receiver from GPS satellites orbiting earth interact together to provide the digital mapping programming with points of origin in addition to the destination points needed to calculate distance. This information is then analyzed and compiled to create a map that provides the easiest and most efficient way to reach a destination.

- GPS receivers collect data from at least four GPS satellites orbiting the Earth, calculating the position in 3d.
- The GPS receiver then utilizes a position to provide GPS coordinates or exact points of latitudinal and longitudinal direction from GPS satellites.
- The points, or coordinates, output an accurate range between approximately "10-20 meters" of the actual location.
- The beginning point, entered via GPS coordinates, and the ending point, (address or coordinates) input by the user, are then entered into the digital mapping software.
- The mapping software outputs a real-time visual representation of the route. The map then moves along the path of the user.
- If the user drifts from the designated route, the navigation system will use the current coordinates to recalculate a route to the destination location.

Digital Maps are necessary in our daily life. It finds a way to travel from your home to college. It suggests restaurants and hotels when you go for a tour. If we start to discuss them, there are a lot of features we are getting from digital maps. All use digital maps today and more often than not, it's for navigation. Dijkstra's algorithm made Navigation possible. In this chapter, we are going to cover some algorithms used in digital maps.

Shortest Path

Suppose you have an exam on a day at 9 o'clock. But you woke up late. You have

to find a way to reach the examination center as soon as possible. Here comes the A* algorithm to find out the shortest path between two nodes.

Dijkstra's Algo

Pseudocode

Let a graph G be having n vertices and e edges. We want to find a path from source node s to target t.

- Create a min heap having all vertices and the distances from s.
- Initialize the distances to INF expect for the source node. Because source to source distance is 0.
- Create a list parent of size n and parents of i equals i, i.e parent[i] = i
- While min heap is empty
    - Take the node with minimum distance from the heap Let the node be m. If the m is t come out of the loop
    - For every node p adjacent to m, check whether it is in the heap. If it is in the heap and if the distance of m plus distance of m-p edge is less than the distance of o, update the distance of o with lesser distance and change the parent of o to m.
- Create an empty stack l and let k be t. Push k to l.
- While k != s:
    - k = parent of k
    - Push k to stack l
- Print the stack from the top. Here is our shortest path from node s to t.

A* Algorithm

It is an improvement algo for Dijkstra algo. Dijsktras will check unnecessary nodes for me.

The heuristic function is the only difference between above algo and this. Lets define a function f(node) be g(node) + h(node) where g is distance of node from source and h be estimated distance of target form node. It is often referred to as heuristic or smart guess.

If h(node) is actual distance then the A* will take the shortest time to compute. If that distance is more than the actual distance then A* will fail to give the shortest path. If it is less than the actual distance then it will surely give the shortest path but takes more time when compared to the actual distance. For now assume that we have values h(i) for every node i

Pseudo code

- Create a list l and push s(source node) to it. It is the list to store the shortest path.
- While last node of l is not t:
    - Let the last node of l is k
    - Compute f for all the adjacent nodes of k.
    - Find the node which has the least value and let the node be m
    - Push the m to l
- Print the list from start to end. Here is our shortest path

R-tree data structure for Geographical mapping

R-tree

R-tree is a tree data structure used for storing spatial data indexes in an efficient manner. It is used for spatial data queries and storage. In R-trees we can use more than three children at each node.The idea of R-trees is to impose ordering on higher dimensional objects by using an ordering on minimum bounding rectangles(MBR).

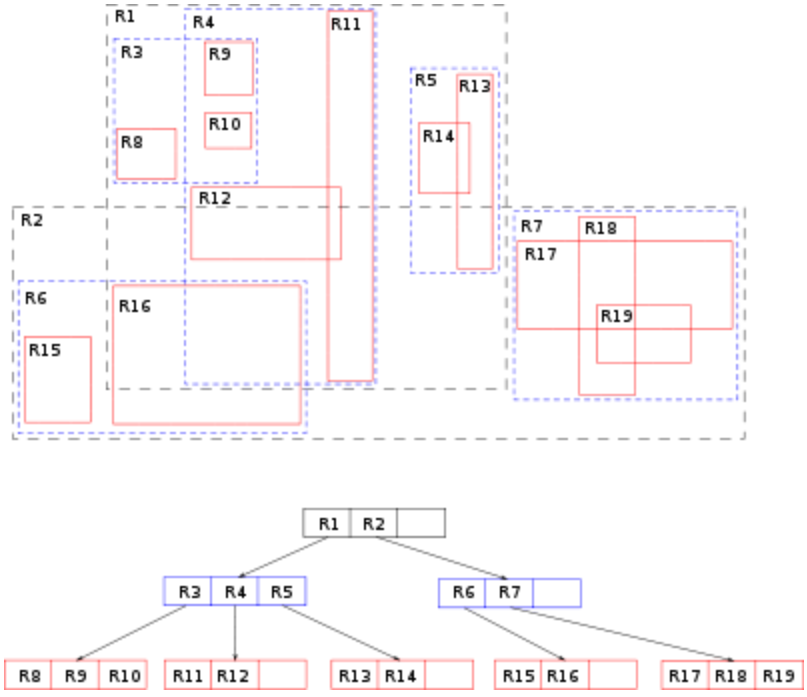
Properties of R-tree :-

- Every leaf node contains between m and M objects
- At each leaf node, for each object stored there, an MBR containing that object is stored.
- Every non-leaf node has between m and M children unless it is the root
- At each non-leaf node, for each of its children, an MBR which contains all MBR's of that child is stored.
- All leaves have the same depth.

- The root node has at least two children unless it is a leaf node

Visualisation of R-tree

The idea is to use the bounding boxes to decide whether or not to search inside a subtree. In this way, most of the nodes in the tree are never read during a search.



1. Insertion

The r-tree is traversed to locate an appropriate leaf to accommodate the new entry.The entry is inserted in the found leaf and the path from the root to that leaf are updated accordingly.If the leaf is not able to accommodate to its position then it

splits into 2 nodes.

Insert(Type Entry E, TypeNode RN)

Traverse the tree from root RN to the appropriate leaf.At each level,select the node L whose MBR will require the minimum are to cover E.mbr
Select the node whose mbr is minimum.
If the selected leaf L can accommodate E
Insert E into L
Update all MBR's in the path from the root L.so it can cover E.mbr
Else
Let e be the set consisting of all entries and the new entry E.select as seeds two entries e1,e2 belongs to e where the distance between them is the maximum of all other pairs.
See the remaining members of e one by one and assign them to L1 or L2
If a tie occurs
Then assign the entry of node whose MBR has smaller area
Endif
If a tie occurs again
Assign the entry to the node that contains the smaller no fof entries
Endif
If during the assignment of entries,there remain t entries to be assigned and the one node contains m-t entries
Assign all the remaining entries to this node without considering the aforementioned criteria.
Endif

```
Update the MBR of nodes that are in the path from root to L.
Splits if necessary
Create a new root,if root has ro be split
Increase the height of the tree.
endif
```

2. Search

For a node E,E.mbr denote the MBR and E.p the corresponding pointer to the next level.If the node is a leaf,then E.p denotes the corresponding object identifier.Below function finds all rectangles that are stored in an R-tree with root node RN,which are intersected by a query rectangle Q and the final answer were stored in the set A.

Search(TypeNodeRN,TypeRegionQ)

```
If RN is not a leaf node
Examine each entry e of RN to find those e.mbr that      intersect Q
Foreach such entry e call search(e.ptr,Q)
Else(RN is a leaf node)
Examine all entries e and find those for which e.mbr
Add these entries to the answer set A
 endif
```

3. Deletion

If we want to delete a particular map details.we can use this function to delete that nodes.Deleting an entry from a page may require updating the bounding rectangles of parent pages. However, when a page is underfull, it will not be balanced with its neighbors. Instead, the page will be dissolved and all its children (which may be subtrees, not only leaf objects) will be reinserted.

Pseudocode:

Delete(type entry E,typeNode RN)

```
/* (initially n=0) * L is leaf /
if RN is a leaf node
Search all entries of RN to find E.mbr
Else (if RN is an internal node)
Find all entries of RN that cover E.mbr
Follow the corresponding subtrees until the leaf L that contains E
Remove E from L
end if
/*condenseTree(L)*/
Set x=L;
Let n be the set of nodes that are going to be removed from the tree.
While x is not the root
Let the parentx be the father node of X
Let the Ex be the entry of parentx that corresponds to x
```

```
        If x contains less than m entries.
                Remove Ex from parentx
                Insert x into n
                Endif
        If x has not removed
                Adjust its MBR Ex.mbr to small
                Endif
                Set x=parentx
        Endwhile
        Reinsert all the entries of the node that are in the set n
If the root has only one child
Remove the root
Set as the new root its only child.
endif
```

Complexities :

1. For search algorithm the average time complexity is O($log_M n$)
2. For Insertion O(log n)
3. For Delete operation O(log n)

Small detour!! :)

Knight’s tour

We have all played or heard about the chess game at some point of the other. A piece in the game of chess called the knight has come to become the base of a problem that’s named knight’s tour and it goes as follows : A knight's tour is a sequence of moves of a knight on a chessboard such that the knight visits every square exactly once. If the knight ends on a square that is one knight's move from the beginning square (so that it could tour the board again immediately, following the same path), the tour is closed; otherwise, it is open. This problem can be solved in various methods. But for now, we will focus on one type of algorithm namely the Warnsdoff’s algorithm. Its explained below:

- A position Q is accessible from a position P if P can move to Q by a single Knight's move, and Q has not yet been visited.
- The accessibility of a position P is the number of positions accessible from P.
- Set P to be a random initial position on the board
- Mark the board at P with the move number “1”
- Do following for each move number from 2 to the number of squares on the board:
    - let S be the set of positions accessible from P.
    - Set P to be the position in S with minimum accessibility
    - Mark the board at P with the current move number

- Return the marked board — each square will be marked with the move number on which it is visited.

This problem has many variants, and the knight's tour problem can be solved by various algorithms starting from brute force algorithms to more complicated and fun filled ones.

5.Cryptography

The main goal of cryptography is to make the communication secure. Cybersecurity refers to a set of techniques used to protect the integrity of networks, programs and data from attack, damage or unauthorized access.

The Three main cryptographic Methods are:

1. Hash Functions
2. Symmetric key cryptography
3. Asymmetric key cryptography

Hash Functions

A hash function maps arbitrary strings of data mapped to a fixed length output in a deterministic method. These are public and random.

$H : \{0,1\}^* \rightarrow \{0,1\}^d$(Here d is fixed and length of the output string).

No secrecy or no secret keys in these functions. All operations are public and anyone can compute H. Poly-time computation (Not necessarily O(1)).

Property 1: Deterministic

This means that no matter how many times you parse a particular input through a hash function you will always get the same result. This is critical because if you get different hashes every single time it will be impossible to keep track of the input.

Property 2: Quick Computation

The hash function should be capable of returning the hash of an input quickly. If the process isn't fast enough then the system simply won't be efficient.

Property 3: Preimage Resistance

What preimage resistance states is that given H(A) it is infeasible to determine A, where A is the input and H(A) is the output hash. Notice the use of the word "infeasible" instead of "impossible". We already know that it is not impossible to determine the original input from its hash value.

But this only works when the given amount of data is very less. What happens when you have a huge amount of data? Suppose you are dealing with a 128-bit hash. The only method that you have to find the original input is by using the "brute-force method". Brute-force method basically means that you have to pick up a random input, hash it and then compare the output with the target hash and repeat until you find a match.

Best case scenario: You get your answer on the first try itself. You will seriously have to be the luckiest person in the world for this to happen. The odds of this happening are astronomical.

Worst case scenario: You get your answer after 2^128 – 1 times. Basically, it means that you will find your answer at the end of all the data.

Average scenario: You will find it somewhere in the middle so basically after 2^128/2 = 2^127 times. To put that into perspective, 2^127 = 1.7 X 10^38. In other words, it is a huge number.

So, while it is possible to break preimage resistance via brute force method, it takes so long that it doesn't matter.

Property 4: The Avalanche Effect.

Even if you make a small change in your input, the changes that will be reflected in the hash will be huge.

Classification of cryptography

1. **Symmetric Key Cryptography**

   Classical :

   - Transposition Cipher
   - Substitution Cipher

   Modern :

   - Stream Cipher
   - Block Cipher

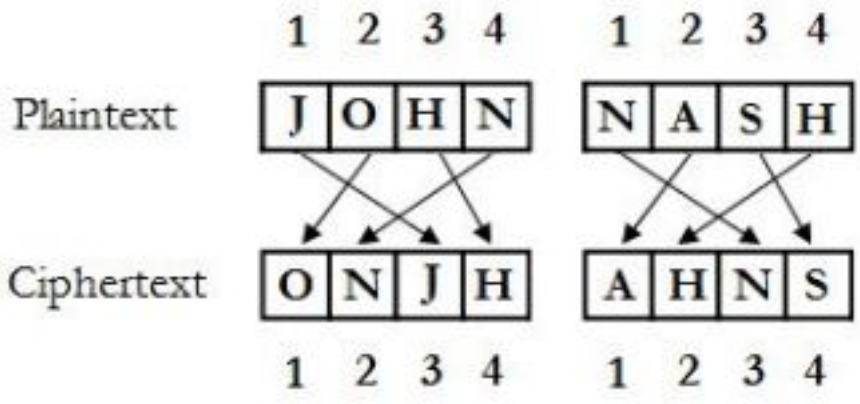
2. **Asymmetric Key Cryptography**

Symmetric Key Cryptography

An encryption system in which the sender and receiver uses the same common key that is used to encrypt and decrypt the message. The most popular **symmetric-key** is Data Encryption Standard (DES).

Transposition Cipher

In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plain text are shifted according to a regular system, so that the ciphertext constitutes the permutation of the plain text.



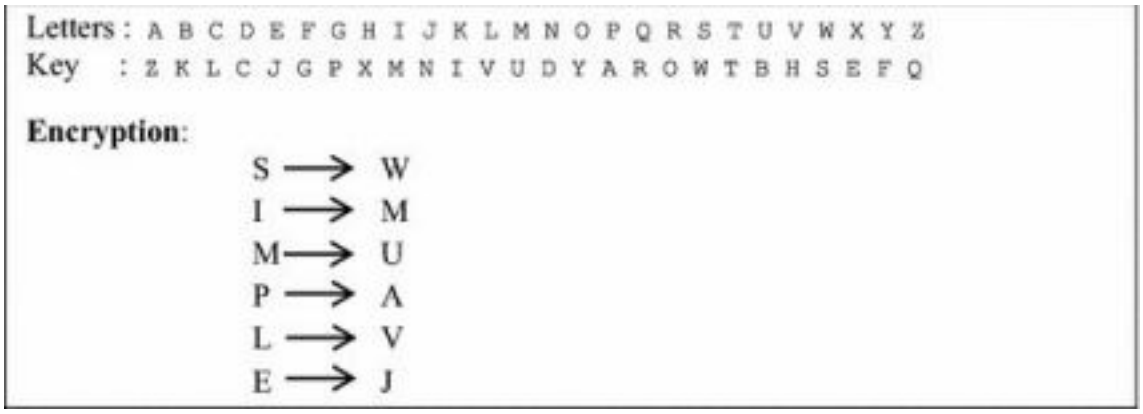
Pseudocode

```
set rows (message length divided by key, rounded up)
   set columns (same as key)
   initialize relevant variables


   create a 2D array as a table
```

```
for each row in the table
  for each column in the table
    if we haven't hit the end of the message yet
      fill each element with the next character in the message
    if we have hit the end of the message
      fill each element with a random character

for each column in the table
  for each row in the table
    combine each character into a string

return the encrypted message
```

Substitution Cipher

Method of encryption by which units of plaintext are replaced with ciphertext, according to a fixed system; the “units” may be single individual letters, pairs of letters or even triplets of letters, mixtures of the bove and so on...



Caesar cipher is an example of substitution cipher

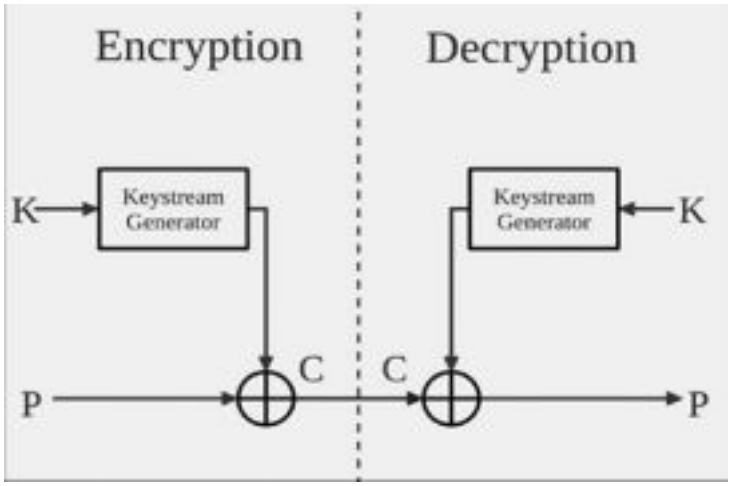
To implement this cipher technique, we need a few simple steps:

- Identify the character in the sentence.
- Find its position inside the alphabet. If not found, just repeat it.
- Substitute it by the character that it is in index + key inside our alphabet.
- Create a new sentence with the new characters instead of the old ones.
- Repeat until the end of the sentence.
- Return the result.

```
def caesar(plaintext, shift):
    alphabet = string.ascii_lowercase
    shifted_alphabet = alphabet[shift:] + alphabet[:shift]
    table = string.maketrans(alphabet, shifted_alphabet)
    return plaintext.translate(table)
```

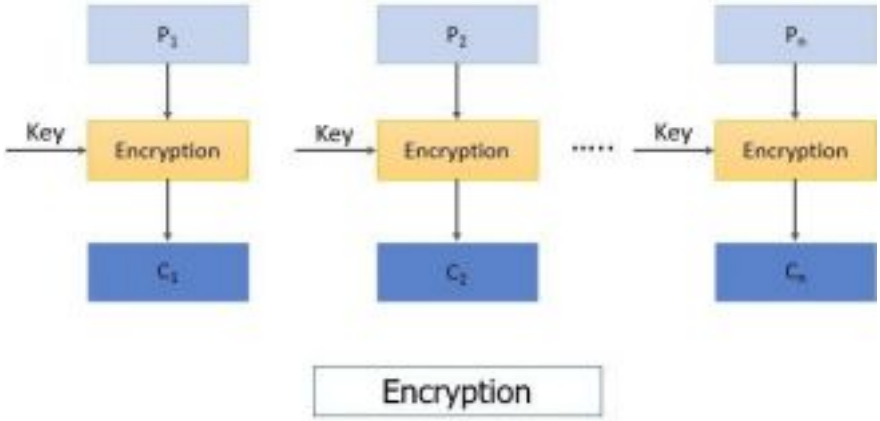
Stream Cipher

A symmetric or secret-key encryption algorithm that encrypts a single bit at a time. With a stream cipher, the same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted.



Block Cipher

An encryption method that applies a deterministic algorithm along with a symmetric key to encrypt a block of text, rather than encrypting one bit at a time as in stream ciphers.



Asymmetric Key Cryptography

It is also known as public key cryptography. Public-key cryptography or asymmetric cryptography, is a cryptographic system that uses pairs of keys: public keys, which may be disseminated widely, and private keys, which are known only to the owner. The generation of such keys depends on cryptographic algorithms based on mathematical problems to produce one-way functions. Effective security only requires keeping the private key private; the public key can be openly distributed without compromising security.

Main Features Of Cryptography are as follows

1. **Confidentiality:**

   Information can only be accessed by the person for whom it is intended and no other person except him can access it.

2. **Integrity:**

   Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.

3. **Non-repudiation:**

   The creator/sender of information cannot deny his or her intention to send information at a later stage.

4. **Authentication:**

The identities of sender and receiver are confirmed. As well as destination/origin of information is confirmed.

F^2(Fun and Fast) Learning

When was cryptography identified as to be used for the first time?

The first known evidence of the use of cryptography was found in an inscription craved around 1900 BC, in the main chamber of the tomb of the nobleman Khnumhotep II, in Egypt.

Is there any founder for this cryptography??

Is considered by many to be the father of mathematical cryptography. By his work he published an article namely " A mathematical theory cryptography ".

Why is this actually invented??

We human beings have to basic and important needs:

1. Communicate and share information
2. Communicate with selective

To satisfy the second need we invented the method of cryptography.

What are basic operations involved in cryptography?

There are three basic operations as follows:

1. Encryption
2. Decryption
3. Hashing

What is a cipher??

A cipher, or cryptographic algorithm, is the means of altering data from a readable form (also known as plaintext) to a protected form (also known as ciphertext), and back to the readable form. Changing plaintext to ciphertext is known as encryption, whereas changing ciphertext to plaintext is known as decryption.

Strongest Data encryption algorithms till date

RSA Algorithm in Cryptography:

PYTHON:

```
from decimal import Decimal
def gcd(a,b):
if b==0:
return a
```

```
else:
return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)
for e in range(2,t):
if gcd(e,t)== 1:
break
for i in range(1,10):
x = 1 + i*t
if x % e == 0:
d = int(x/e)
break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n
dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n
print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text =
'+str(ct)+' decrypted text = '+str(dt))
```

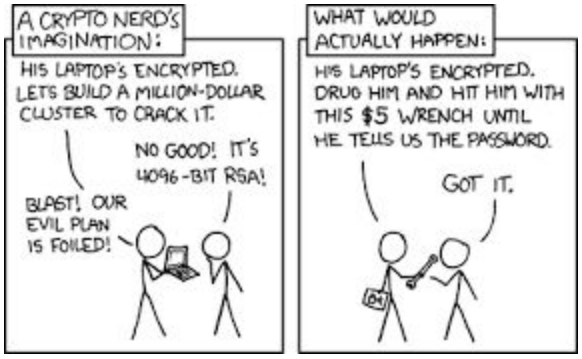
RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As

the name describes that the Public Key is given to everyone and the Private key is kept private.

C Implementation:

```
#include<stdio.h>
#include<math.h>
int gcd(int a, int h)
{
int temp;
while (1)
{
temp = a%h;
if (temp == 0)
return h;
a = h;
h = temp;
}
}
int main()
{
double p = 3, q = 7, e = 2;
double n = p*q, phi = (p-1)*(q-1);
while (e < phi)
{
if (gcd(e, phi)==1)
```

```
break;
else
e++;
}
int k = 2;
double d = (1 + (k*phi))/e, msg = 20;
printf("Message data = %lf", msg);
double c = pow(msg, e);
c = fmod(c, n);
printf("\nEncrypted data = %lf", c);
double m = pow(c, d);
m = fmod(m, n);
printf("\nOriginal Message Sent = %lf", m);
return 0;
}
```



Small detour!! :)

Soundex algorithm

English is a pretty confusing language , don't u think? And with it's various pronunciations and words that sound alike, it can get confusing very easily. What if there was an algorithm that could compare two words and their phonetics and tell us if they sound alike.

Well, of course I am not the first one to face this problem, and many algorithms have been concocted to face the mystery of english language. One of those is the soundex algorithm. Though the sundex algorithm by itself has many variations, we will try and explain the most basic and simple version of it, and leave it to the readers to explore more with the references below.

The Soundex algorithm applies a series of rules to a string to generate the four-character code. The encoding steps are as follows:

```
Ignore all characters in the string being encoded except
for the English letters, A to Z.
The first letter of the Soundex code is the first letter of
the string encoded.
After the first letter in the string, do not encode vowels
or the letters H, W, and Y. These letters may affect the
code by being present but are not encoded directly.
Assign a numeric digit between one and six to all letters
after the first using the following mappings:
1: B, F, P, or V
```

```
2: C, G, J, K, Q, S, X, Z
3: D, T
4: L
5: M, N
6: R
Where adjacent digits are the same, remove all but one of
those digits unless a vowel, H, W, or Y was found between
them in the original text. We can also use a temporary
placeholder for these non-encodable letters to avoid
incorrect removal of adjacent, matching digits.
Force the code to be four characters in length by padding
with zero characters or by truncation.
```

6.Epicycles

Introduction

"An **epicycle** is an orbit revolving around a point on the deferent epicycle is an orbit revolving around a point on the deferent". This is a formal definition of the epicycle. We will try to explore what they are and the significance that such a model has in understanding more complicated concepts like Fourier transformations.

Generally, this epicycle is used to understand the solar system model, or in general, any celestial object moving around and has been greatly used in Ptolemaic astronomy.

In the modern times, when we want to compute the positions of the Sun, Moon and planets, with respect to Earth (and to do astronomical observations from Earth you need exactly this!) we use long trigonometric series.

Mathematically, trigonometric series is the same as epicycles. Some coefficients of these series are obtained by solving differential equations, others are empirical.

So, in layman's words, epicycle a circle rolling over another circle. Circle moving on another circle is a frequent motif in the models of planetary models. For example, imagine describing the motion of the moon; in a helio-centric model, the moon circles earth while the earth circles around the sun.

Fourier showed that ANY curve can be described as a motion of a circle which is moving on another circle, which is moving on another circle and so on. In this lesson, we will explore the Fourier analysis using complex numbers.

We want to model a closed curve in two dimensions; as a simplistic example, consider a circle with center at origin given by $x^2+y^2=c^2$ with c being the radius of the circle. Of specific interest is the parametric description with parameter t, the circle has following parametric description: $x(t)=c.\cos(t)$ and $y(t)=c.\sin(t)$.

The complex number description of this parametrization is given by $z(t)=c.\exp(jt)$. Consider $z(t)=ce^{njt}$ this has $z(t)=z(t+2\pi/n)$, i.e. with a period of $2\pi/n$ evey point returns to itself. For the sake of having a period 1 unit of time and fractions of it, we will consider $z(t)=ce^{2\pi njt}$ which has a period of 1/n unit of time.

The topic of fourier transformations and fourier analysis is a massive one, and will be dealt more clearly in the next upcoming chapters of the book. For this chapter, we intend to discuss briefly about an epicycle approach to fourier analysis without delving deep into the concept.

Here, we take a small complex equation and convert it into a graph in another domain(specifically speaking, the frequency domain). And then in the next part of this code, we will try to explain more about how an ellipse can be converted into a dictionary of points and after a fourier and inverse fourier on it, we can get back the original image almost, pertaining to some irregularities due to small number of iteration, the more the iterations, the sharper the image.

At the end of this, we have also given a few excellent websites which explain these concepts and algorithm in better ways, and also a few website's links where one can draw closed curves and get their curves replicated to a certain level of approximation by epicycles method, just as how we explained in the code below.

We try to follow step by step so as to converting an image into a series of points for further analysis

jupyter Epicycles_team.y Last Checkpoint: 9 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

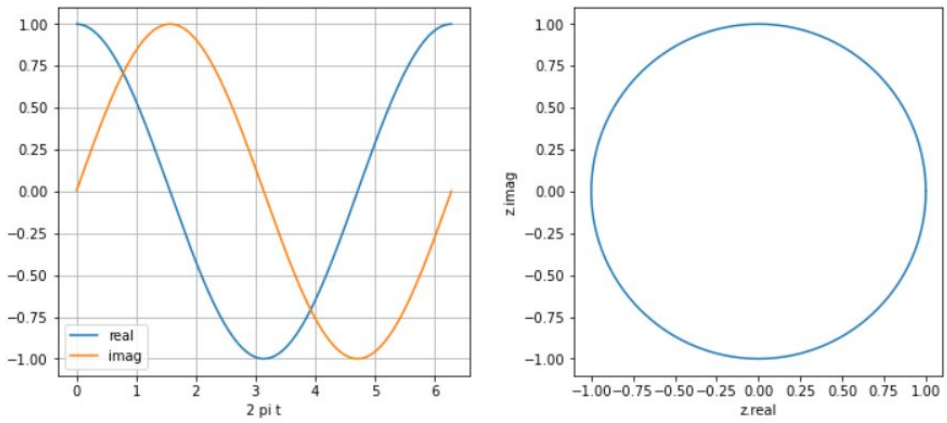
Run Code

In [16]:
```
%pylab inline
tvals = linspace(0,1,1000)[1:]
zvals_circle = np.exp(2j*pi*tvals)
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [4]:
```
figure(figsize=(12,5))

subplot(1,2,1)
plot(tvals*2*pi,zvals_circle.real,label='real')
plot(tvals*2*pi,zvals_circle.imag,label='imag')
xlabel('2 pi t')
grid()
legend()
#axis('scaled')

subplot(1,2,2)
plot(zvals_circle.real,zvals_circle.imag)
xlabel('z.real')
ylabel('z.imag')
axis('scaled')
```

Out[4]:
```
(-1.0999948080833182,
 1.0999997527658723,
 -1.0999986402114572,
 1.0999986402114572)
```

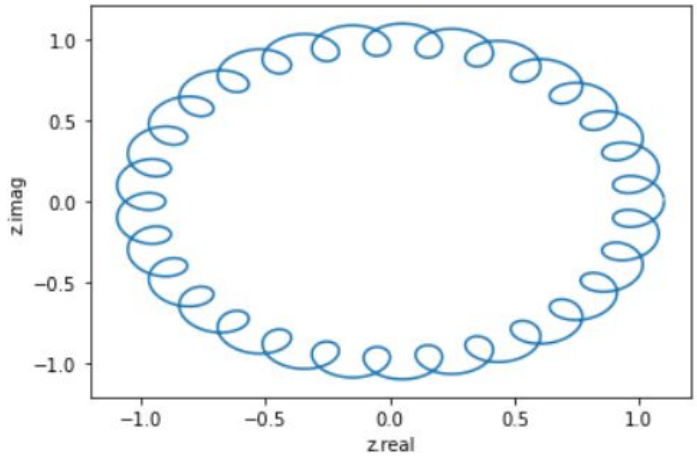


Consider an epicycle given by:

$z(t)=c_1e^{2\pi jt}+c_2e^{30\pi jt}$

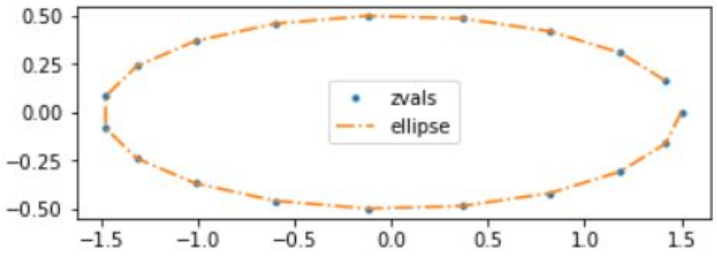
In [5]:
```
zvals_1_30 = np.exp(tvals*2j*pi) + 0.1*np.exp(30*tvals*2j*pi)
plot(zvals_1_30.real,zvals_1_30.imag)
xlabel('z.real')
ylabel('z.imag')
```

Out[5]:
```
Text(0, 0.5, 'z.imag')
```



In [6]:
```
tvals = linspace(0,1,20)[1:] # remove 0
zvals_2 = np.exp(tvals*2j*pi) + 0.5*np.exp(-tvals*2j*pi)
plot(zvals_2.real, zvals_2.imag,'.',label='zvals')
plot(1.5*np.cos(tvals*2*pi),0.5*np.sin(tvals*2*pi),'-.',label='ellipse')
axis('scaled')
legend()
```

Out[6]:
```
<matplotlib.legend.Legend at 0x2376155e520>
```

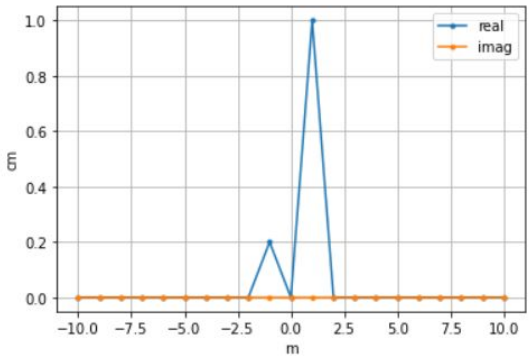


In [7]:
```
def fourier_coeff(zvals, m):
    numPoints = shape(zvals)[0]
    tvals = linspace(0,1,numPoints+1)[1:]
    c_m = 1.0/numPoints * np.sum( zvals * np.exp(-2*pi*1j*m*tvals) )

    return c_m
```

In [9]:
```
l1 = []
for m in range(-10,11):
    cm = fourier_coeff(zvals_ell,m)
    l1.append(cm)
```

In [11]:
```
l1 = array(l1)
xval = list(range(-10,11))
plot(xval,l1.real,'.-',label='real')
plot(xval,l1.imag,'.-',label='imag')
xlabel('m')
ylabel('cm')
grid()
legend()
```

Out[11]: <matplotlib.legend.Legend at 0x237615c4bb0>

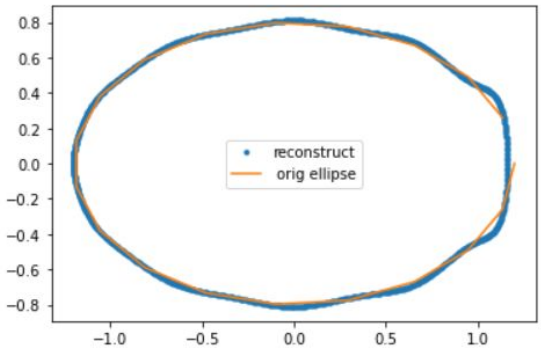


In [12]:
```
ellipse_dict = {}
for m in range(-10,11):
    cm = fourier_coeff(zvals_ell[:-1],m)
    ellipse_dict[m] = cm
```

In [13]:
```
z_reconstruct = zeros((1000,),dtype=complex)
dt = 1./1000
tvals = linspace(0,1,1001)[1:]
for key in ellipse_dict.keys():
    z_reconstruct += ellipse_dict[key] * np.exp(2*pi*1j*key*tvals)
```

In [14]:
```
plot(z_reconstruct.real,z_reconstruct.imag,'.',label='reconstruct')
plot(zvals_ell.real, zvals_ell.imag,label=' orig ellipse')
legend()
```

Out[14]: <matplotlib.legend.Legend at 0x23761716d00>



7.Fourier transformation

The whole idea of Fourier transformation comes from the Euler's formula of e to the power some x times i, then the value is equal to the distance/ the pt is from distance x units around a circle with radius 1 counter-clockwise starting from the right.

If we finish one cycle in 1 sec we get the distance as $e^{2i\ t}$ where t is the time passed (for a circle of radius 1, the circumference is 2)

The negative of 2it gives the same result but in the clockwise direction which is a standard convention for Fourier transformation

Multiplying the power of e by any f, we can get a slower version of it(needs to be explained more)

Thus we get the equation $e^{-\ 2\ tf}$

We now multiply the equation by any periodic function g(t) to get the function wound around a circle with the increasing and decreasing distances on the y-axis as values of the function.

This wound up graph can be visualized as a metal wire mesh where the winding frequency would be F . We can increase or decrease the frequency as we wise and we get a different arrangement of the wire mesh structure every time. If we imagine a hypothetical center of mass of this mesh, it keeps changing its position for every different winding frequency.

This point is the average of all the points in the wound-up graph. Now we notice that this point is almost near the center at all frequencies except when it nears the frequency of g(t) or when it nears the frequencies f1,f2,f3…. That together adds up to make the function g(t).

Thus summing up all these points would give us the frequency we need to extract from the graph.

This is how the Fourier transformation works on a pure intuition basis.

The graph now obtained from these functions can be subjected to inverse Fourier transformation(the reverse) to obtain the original wave. Ideally, the Fourier transformation is done from the negative infinity to positive infinity. The Fourier transformation equation now becomes

$$\hat{g}(f) = \int_{-\infty}^{+\infty} g(t) e^{-2\pi i f t} dt$$

A visualisation of the above can be seen in the following website:

Fourier Transform Visualization

The python code for all of the intuition would be as follows

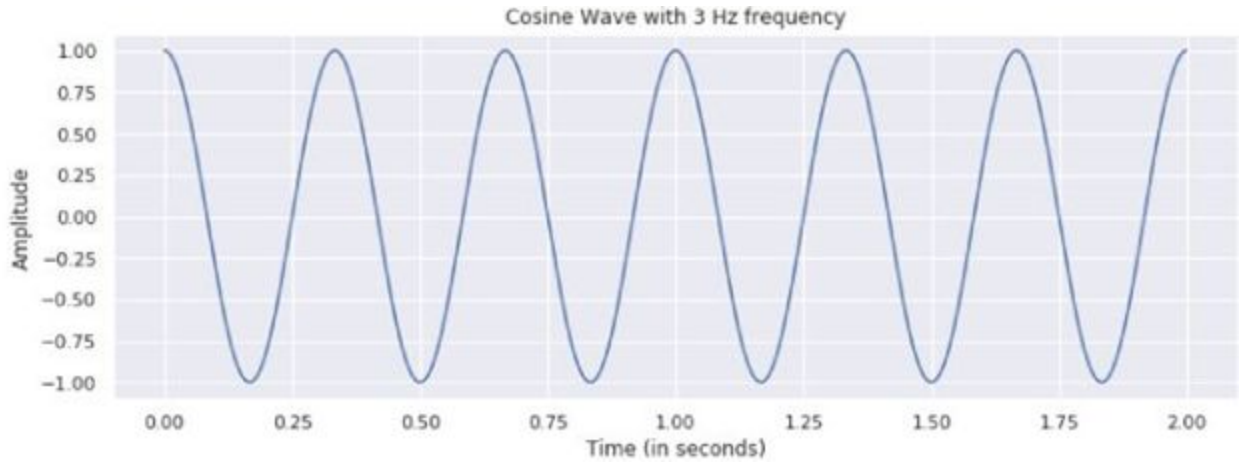
Example code

Note: its necessary to import matplotlib (as plt) and import numpy (as np) and import seaborn as sns for this

Matlplotlib and numpy are mathematical tools to visualize these processes. Seaborn is used to making the plots more picturesque and elegant.

```
sns.set()
f = 3.0 t = np.arange(0,2,0.001)  cos_wave = np.cos(2*np.pi*f*t)
# cos_wave = 2*m.cos(2*np.pi*f*t) + 5*m.cos(2*np.pi*f*2*t)

plt.rcParams["figure.figsize"] = (12,4) plt.plot(t,cos_wave) plt.title("Cosine Wave with 3 Hz frequency") plt.ylabel("Amplitude") plt.xlabel('Time (in seconds)');
```

Till here, this code now generates a signal of three Hz, more specifically a cosine wave of 3 Hz to implement our Fourier transform on it



```
r_cord = [] min_freq_range = 0.0 max_freq_range = 10.0 sf_list =
np.arange(min_freq_range, max_freq_range, 0.1) for sf  in  sf_list:
        r_cord.append( [(cos_wave[i], t[i]*sf*2*np.pi) for  i  in  range(len(t)) ] )
x_cord , y_cord = [], [] for l  in  range(len(r_cord)):
   x_cord.append( [amp*np.cos(theta) for  (amp,theta)  in  r_cord[l]] )
y_cord.append( [amp*np.sin(theta) for  (amp,theta)  in  r_cord[l]] )
```

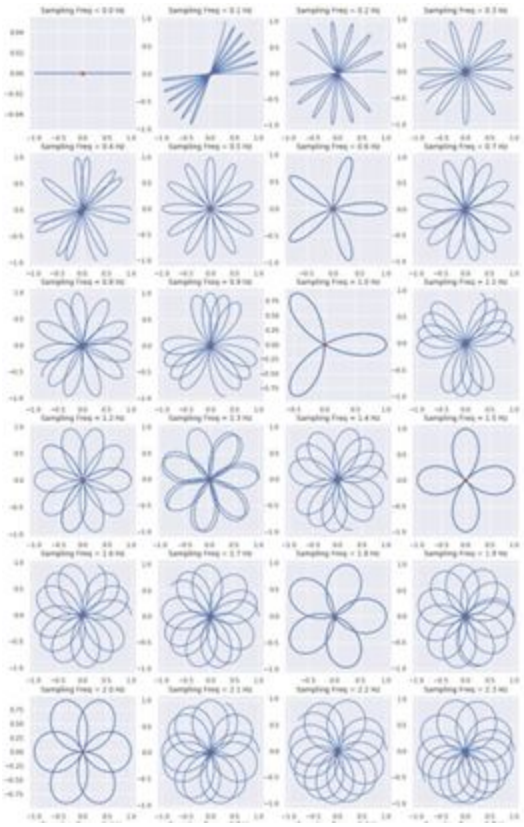
we need to try warping the input signal graph for a number of sampling frequencies and then by keeping the track of the x-coordinate of the center of mass, considering the warped graph as a piece made of metal wire, we can estimate the frequencies present in the input signal.

The range for frequency search is kept here as 0 to 10 Hz.

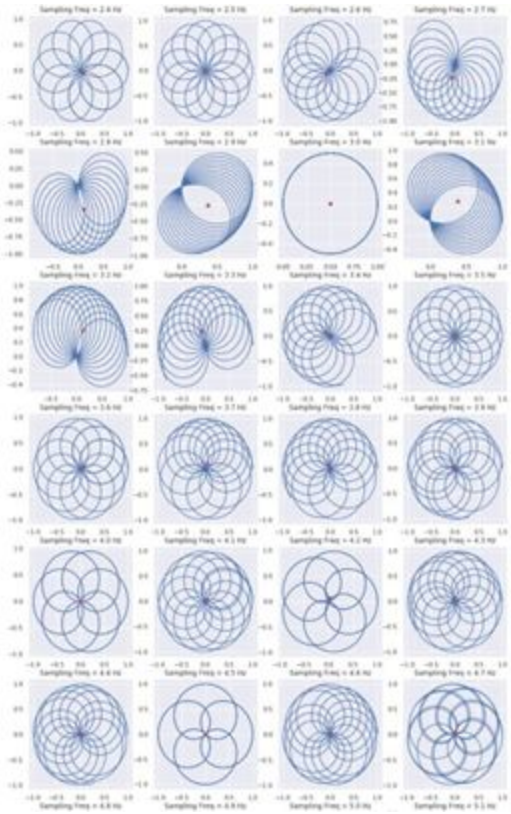
```
mean_list = []
plt.rcParams["figure.figsize"] = (15,110) for l  in range(len(r_cord)):
        plt.subplot(int(len(r_cord)/4)+1, 4, int(l+1))  plt.plot(x_cord[l], y_cord[l])
        plt.plot(np.mean(x_cord[l]), np.mean(y_cord[l]), 'or' )
plt.title("Sampling Freq = "+str(round(sf_list[l], 2))+" Hz")
        x_mean = np.sum(x_cord[l])     mean_list.append(x_mean)
```

This piece of code now plots many different varieties of the warped frequencies at different F as subplots. For every 0.1 Hz, a subplot is drawn that shows the function with that specific winding frequency.

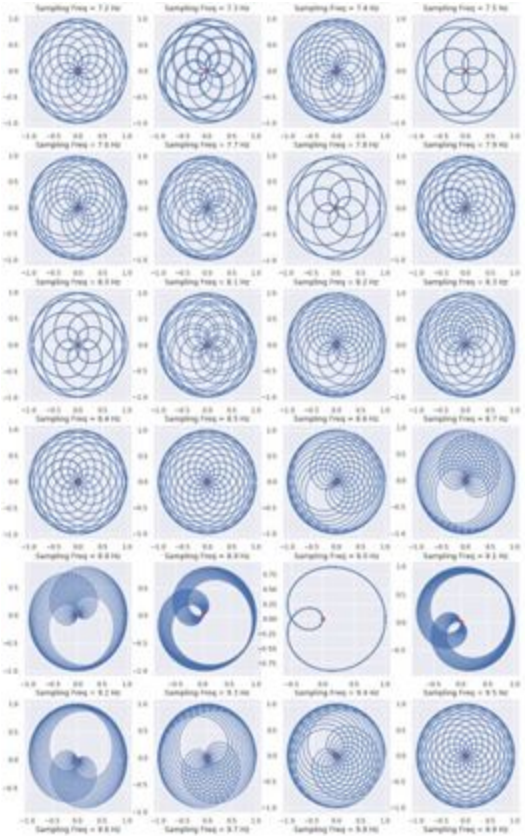
Given below are the subplots that result from this simulation of the warping up. They show the progression of the wounded graph as the frequency of the winding-up increases though the graphs are beautiful by themselves, these graphs intend to reveal some even more intriguing and some of the most fascinating secrets in the world of science and application.



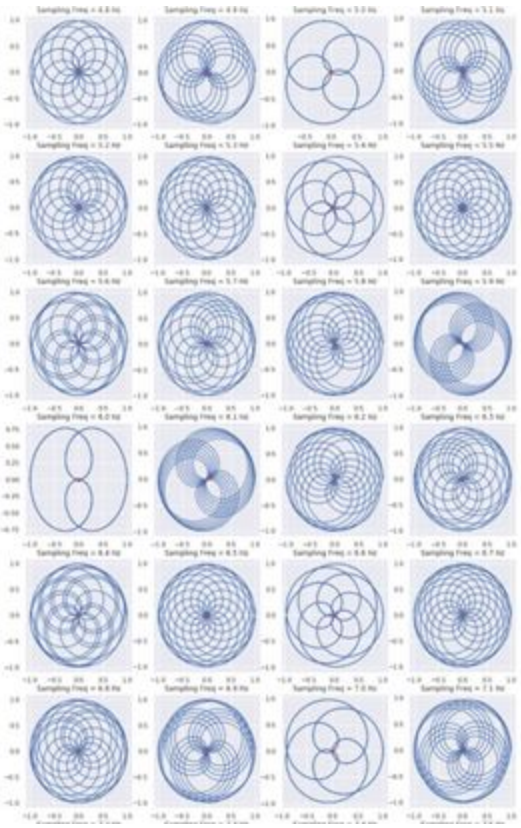
1st

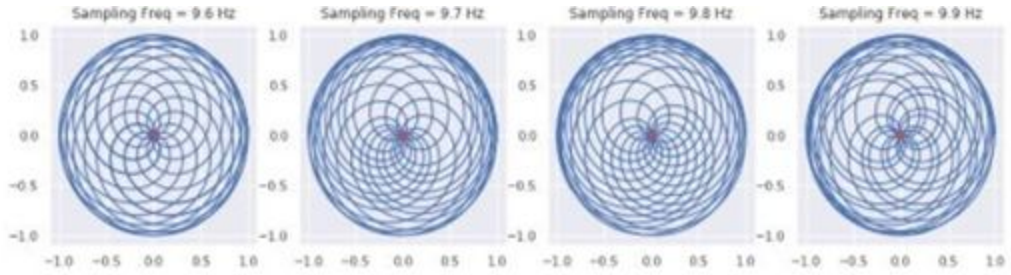


2nd



3rd 4th



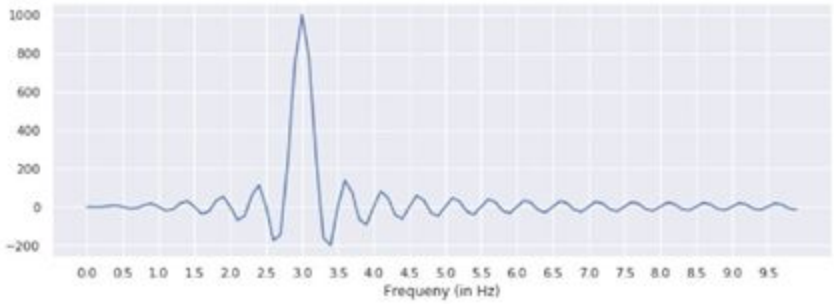


5th

```
plt.rcParams["figure
.figsize"] = (12,4)
plt.xlabel("Frequenc
y (in Hz)")
plt.xticks(np.arange(
```

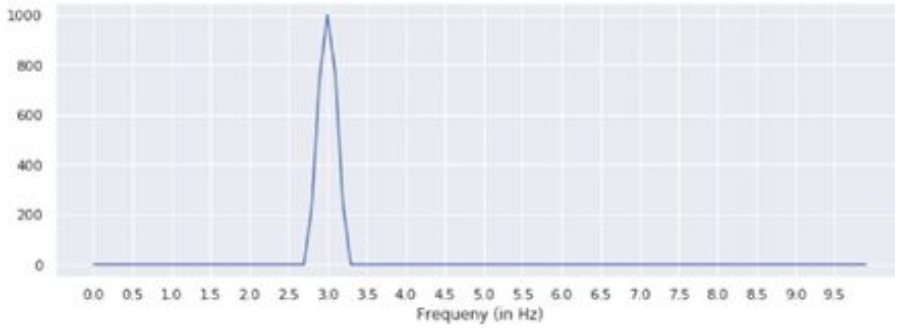
```
min(sf_list),
max(sf_list), 0.5))
sns.set()
plt.plot(sf_list,mean
_list);
```

The next piece of code intends to plot the center of mass of this graph points vs the sampling frequencies. The concept of center of mass for a group of points was explained earlier.



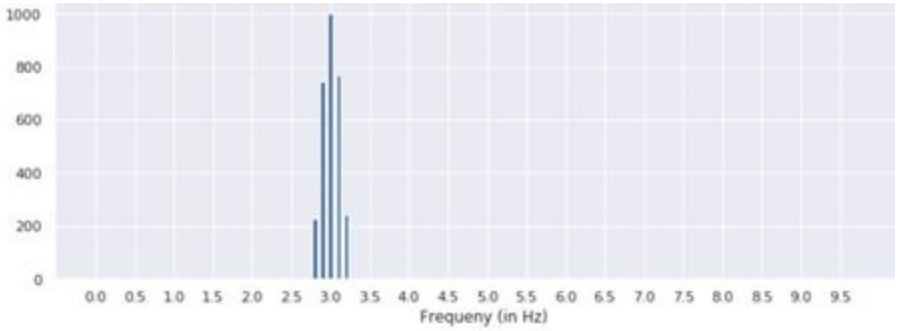
This graph has its frequencies at some points toggling around 0 and needs to be smoothed out. The next part of the code is optional but makes the overall implementation more efficient and neat. This also makes any other implementations of the Fourier transformation to be used in the future easier.

```
plt.rcParams["figure.figsize"] = (12,4) smoothed = [i if  i>0       and
i>0.2*max(mean_list)  else  0  for  i  in  mean_list]  plt.plot(sf_list, smoothed)
plt.xlabel("Frequency (in Hz)")
plt.xticks(np.arange(min(sf_list), max(sf_list), 0.5));
```



There is also an option of implementing the same in a bar graph method.

```
plt.rcParams["figure.figsize"] = (12,4) plt.xticks(np.arange(min(sf_list),
max(sf_list), 0.5)) plt.xlabel("Frequency (in Hz)")
plt.bar(sf_list, smoothed, width = 0.06);
```



Now, since we have understood the intuitive part of the Fourier transformations, we can move on to the algorithmic part of it.

This part involves two major algorithms. The first one is a simple brute force algorithm and implements the Fourier transformation using summation

The discrete Fourier transform (DFT) is a basic yet very versatile algorithm. The DFT overall is a function that maps a vector of n complex numbers to another vector of n complex numbers.

Using 0-based indexing, let x(t) denote the t^{th} element of the input vector and let X(k) denote the k^{th} element of the output vector. Then the basic DFT is given by the following formula:

$$X(k) = \sum_{t=0}^{n-1} x(t) e^{-2\pi i t k / n}.$$

The interpretation is that the vector x represents the signal level at various points in time, and the vector X represents the signal level at various frequencies. What the formula says is that the signal level at frequency k is equal to the sum of {the signal level at each time t multiplied by a complex exponential}.

The pseudo code for the above dft algorithm can be given as follows :

```
import cmath def compute_dft_complex(input):
n = len(input) output = [] for k in range(n):  # For each output element s =
complex(0) for t in range(n):  # For each input element angle = 2j * cmath.pi * t *
k / n s += input[t] * cmath.exp(-angle)
output.append(s)
return output
import math def compute_dft_real_pair(inreal, inimag):
```

```
assert len(inreal) == len(inimag) n = len(inreal) outreal = [] outimag = [] for k in
range(n):  # For each output element sumreal = 0.0 sumimag = 0.0 for t in
range(n):  # For each input element
angle = 2 * math.pi * t * k / n sumreal +=  inreal[t] * math.cos(angle) + inimag[t]
* math.sin(angle) sumimag += -inreal[t] * math.sin(angle) + inimag[t] *
math.cos(angle) outreal.append(sumreal) outimag.append(sumimag)
return (outreal, outimag)
```

There are many drawbacks to the discrete fourier transformation algorithm.Clearly the DFT is only an approximation since it provides only for a finite set of frequencies. But how correct are these discrete values themselves?

There are two main types of DFT errors: “aliasing” and “leakage” (these mainly occur in signal processing)

Aliasing

This is another manifestation of the phenomenon which we have now encountered several times in different fields. If the initial samples are not sufficiently closely spaced to represent high-frequency components present in the underlying function, then the DFT values will be corrupted by aliasing. The solution is either to increase the sampling rate (if possible) or to pre-filter the signal in order to minimize its high-frequency spectral content.

Leakage

Recall that the continuous Fourier transform of a periodic waveform requires the integration to be performed over the interval over an integer number of cycles of the waveform. If we attempt to complete the DFT over a non-integer number of cycles of the input signal, then we might expect the transform to be corrupted in some way usually resulting in discontinuities.

The solution is to use windows such as the Hamming or Hanning windows. These window functions taper the samples towards zero values at both endpoints, and so there is no discontinuity (or very little, in the case of the Hanning window) with a hypothetical next period.

Hence the leakage of spectral content away from its correct location is much reduced. For understanding what are hamming windows, please refer this link.

Though the Fourier transformation was a very useful tool, the real boom to its applications in varied fields of sciences and maths came with its more efficient counterpart, the fast

Fourier transformation algorithm, derived majorly from the Cooley-Tukey algorithm. A significant drawback in the mathematical sense of implementation of the DFT is that it involves an O(n^2) matrix multiplication. The FFT algorithm instead has

O(NlogN) time complexity that makes many problems comparatively very fast.

Say it took 1 nanosecond to perform one operation. It would take the Fast Fourier Transform algorithm approximately 30 seconds to compute the Discrete Fourier Transform for a problem of size N = 10^9. In contrast, the regular algorithm would need several decades(31.2 years approx.).

N	1000	10^6	10^9
N^2	10^6	10^{12}	10^{18}
$Nlog_2N$	10^4	$20\,x\,10^6$	$30\,x\,10^9$

We will now try and understand the fast Fourier transformation algorithm we will separate the Fourier Transform into even and odd indexed sub-sequences: n=2r if even and n=2r+1 if even ;

R =1,2,....,(N/2) -1

For doing this we tweak the DFT formula

$$x[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}$$

Here, we have changed the frequency to n/N and the rest of it is pretty much similar to what we dealt with earlier.

Now, we can summate the odd and even terms of the dft according to the above splitting and take advantage of the fact that the even and odd subsequences can be computed concurrently.

The below steps show the algebra for that

$$x[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] e^{\frac{-j2\pi k(2r)}{N}} \quad + \quad x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] e^{\frac{-j2\pi k(2r+1)}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] e^{\frac{-j2\pi k(2r)}{N}} \quad + \quad x[k] = e^{\frac{-j2\pi k}{N}} \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] e^{\frac{-j2\pi k(2r)}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] e^{\frac{-j2\pi k(r)}{N/2}} \quad + \quad x[k] = e^{\frac{-j2\pi k}{N}} \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] e^{\frac{-j2\pi k(r)}{N/2}}$$

$$x[k] = x_{even}[k] \; + \; e^{\frac{-j2\pi k}{N}} x_{odd}[k]$$

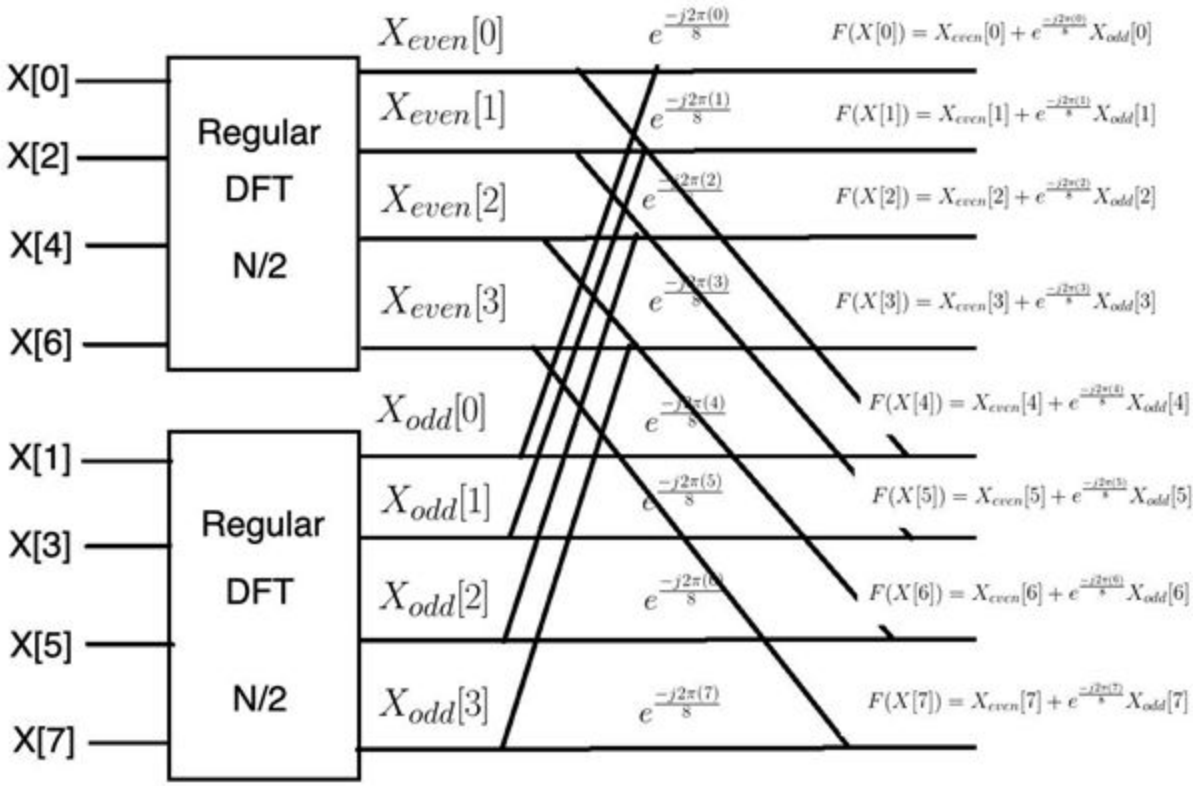
The next trick that the FFT used is called the butterfly diagram.

Butterfly Diagrams show where each element in the array goes before, during, and after the FFT. As mentioned, the FFT must perform a DFT. This means that even though we need to be careful about how we add elements together, we are still ultimately performing the summation operation.

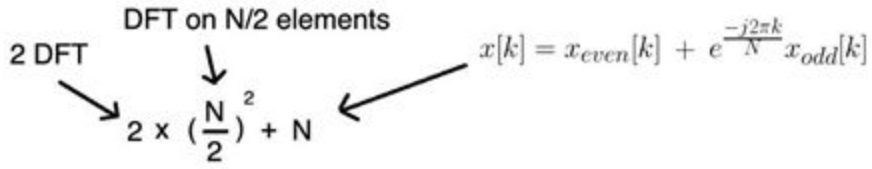
To visualize the working of these butterfly diagrams, let us assume N=8 for simplicity's sake.

We compute the Discrete Fourier Transform for the even and odd terms simultaneously.

Then, we calculate x[k] using the formula from above.



Here, with some observation, we can calculate the time complexity of this algorithm



$$= 2 \times \frac{N^2}{4} + N$$

$$= \frac{N^2}{2} + N$$

$$O(\frac{N^2}{2} + N) \sim O(N^2)$$

Even though this seems like apparently not of much change, we can notice that we have halved the time complexity.

And if we continue this splitting into subsequences further and further, we get to an interesting result.

$$\frac{N}{2} \longrightarrow 2\left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N$$

$$\frac{N}{4} \longrightarrow 2\left(2\left(\frac{N}{4}\right)^2 + \frac{N}{2}\right) + N = \frac{N^2}{4} + 2N$$

$$\frac{N}{8} \longrightarrow 2\left(2\left(2\left(\frac{N}{8}\right)^2 + \frac{N}{4}\right) + \frac{N}{2}\right) + N = \frac{N^2}{8} + 3N$$

$$\vdots$$

$$\frac{N}{2^P} \longrightarrow \frac{N^2}{2^P} + PN = \frac{N^2}{N} + (\log_2 N)N = N + (\log_2 N)N$$

$$\sim O(N + N\log_2 N) \sim O(N\log_2 N)$$

Lo and behold ! we get the complexity O(NlogN) using algebra and divide & conquer methods. The true beauty of the butterfly diagram is represented when we achieve this. And this result has radically changed the use of fourier transformation in many many fields of science and engineering.

Now we can move forward with the cooley-tukey algorithm for FFT that implements this.

We can also use a vector notation to make this algorithm work efficiently.

```
def fft_v(x):
x = np.asarray(x,dtype=float)
N = x.shape[0] if np.log2(N)% 1 > 0:
raise ValueError("must be a power of 2")
N_min = min(N,2)
n = np.arange(N_min)
k = n[:, None]
M = np.exp(-2j * np.pi * n * k / N_min)
X = np.dot(M, x.reshape((N_min, -1))) while X.shape[0] < N:
X_even = X[:, :int(X.shape[1] / 2)]
X_odd = X[:, int(X.shape[1] / 2):]
terms = np.exp(-1j * np.pi * np.arange(X.shape[0]) / X.shape[0])[:, None]
X = np.vstack([X_even + terms * X_odd, X_even - terms * X_odd])
return X.ravel()
```

So long as N is a power of 2, the maximum number of times you can split into two equal halves is given by p = log(N)

$$e^{ln(x)} = x$$

$$2^{log_2(N)} = N$$

8.Signal Processing

Laplace transform

The notation for the Laplace transform is $L\{f(t)\} = F(s)$

This transform gets an input which is some function of t, and then the laplace transform converts it into another function F which depends on another variable s.

Here the laplace transform can thus convert a function into a completely different type of function. Or in other words it changes a function from the time domain to the Laplace domain s. It is defined as

$$F(s) = \int_0^\infty e^{-st} f(t) dt$$

The function F of variable s is defined as the improper interval from 0 to infinity of e to the power of (-st) of f(t)*dt. Notice that the exponential term is where the new variable comes into the picture.

Since we are doing an integral with respect to t, the t variable would vanish away, and what we get as F(s) is determined largely by this negative exponential term. As this is an improper integral, we will have questions like when or when will it not converge, but nevertheless, this is the formal definition.

Some important Laplace transforms are given below

f	$\mathcal{L}_t\,[f\,(t)]\,(s)$	conditions
1	$\frac{1}{s}$	
t	$\frac{1}{s^2}$	
t^n	$\frac{n!}{s^{n+1}}$	$n \in \mathbb{Z} \geq 0$
t^a	$\frac{\Gamma(a+1)}{s^{a+1}}$	$\mathbb{R}\,[a] > -1$
$e^{a\,t}$	$\frac{1}{s-a}$	

$\cos(\omega\,t)$	$\frac{s}{s^2+\omega^2}$	$\omega \in \mathbb{R}$
$\sin(\omega\,t)$	$\frac{\omega}{s^2+\omega^2}$	$s > \lvert\mathbb{I}\,[\omega]\rvert$
$\cosh(\omega\,t)$	$\frac{s}{s^2-\omega^2}$	$s > \lvert\mathbb{R}\,[\omega]\rvert$
$\sinh(\omega\,t)$	$\frac{\omega}{s^2-\omega^2}$	$s > \lvert\mathbb{I}\,[\omega]\rvert$
$e^{a\,t}\sin(b\,t)$	$\frac{b}{(s-a)^2+b^2}$	$s > a + \lvert\mathbb{I}\,[b]\rvert$
$e^{a\,t}\cos(b\,t)$	$\frac{s-a}{(s-a)^2+b^2}$	$b \in \mathbb{R}$

$\delta\,(t-c)$	$e^{-c\,s}$	
$H_c\,(t)$	$\begin{cases} \frac{1}{s} & \text{for } c \leq 0 \\ \frac{e^{-c\,s}}{s} & \text{for } c > 0 \end{cases}$	
$J_0\,(t)$	$\frac{1}{\sqrt{s^2+1}}$	
$J_n\,(a\,t)$	$\frac{\left(\sqrt{s^2+a^2}-s\right)^n}{a^n\sqrt{s^2+a^2}}$	$n \in \mathbb{Z} \geq 0$

Here, $J_0\,(t)$ is the zeroth-order Bessel function of the first kind, $\delta\,(t)$ is the delta function, and $H_c\,(t)$ is the Heaviside step function.

Note that the functions *f(t)* and *F(s)* are defined for time greater than or equal to zero

The Laplace transform also has some interesting properties

1. If $\mathcal{L}_t\,[f\,(t)]\,(s) = F\,(s)$ for $s > \alpha$, then

   $$\mathcal{L}_t\left[e^{a\,t}\,f\right](s) = F\,(s-a) \text{ for } s > a + \alpha.$$

   The proof of this is as follows,

$$F(s-a) = \int_0^\infty f\, e^{-(s-a)t}\, dt$$
$$= \int_0^\infty \left[f(t)\, e^{at}\right] e^{-st}\, dt$$
$$= \mathcal{L}_t \left[e^{at} f(t)\right](s).$$

2. If $f(t)$ is piecewise continuous and $|f(t)| \leq M e^{at}$, then,

$$\mathcal{L}_t \left[\int_0^t f(t')\, dt'\right] = \frac{1}{s} \mathcal{L}_t [f(t)](s).$$

For more information about transfer functions, please refer [Transfer Functions | Dynamics and Control](#) .

The biggest advantage of Laplace transform, and the reason it is so widely used in the mathematics behind signals and systems is that, similar to Fourier domains, we can transform input signal x(t) to the Laplace or s-domain as X(s), and we can model the system in the s-domain using its response H(s). This is also called the Transfer Function. If you knew X(s) and H(s), then the output in the s-domain Y(s) = H(s) X(s) – very similar to the Fourier analysis we did before.

So, in simple terminology, Laplace transform is more general and Fourier transform is a special case of Laplace transform. When s = jω, Laplace transform becomes equivalent to Fourier transform. Laplace transform, just like Fourier, obeys the law of linearity – it is a linear transform.

H(s) is known as the Transfer function, and it characterizes the system in the s-domain is a 2nd order polynomial function in the complex Laplace

variable s. This is an algebraic equation. Since Y(s) = H(s) X(s), a simple multiplication, we can predict the output by simple algebraic calculations.

This is a huge advantage in mathematical calculations as algebraic equations are way more understandable than fiddling with differential equations.

There are two types of Laplace transforms. Unilateral and bilateral Laplace transforms.

Bilateral

$$F(s) = \int_{-\infty}^{\infty} f(t) e^{-(st)} \, \mathrm{d}t$$

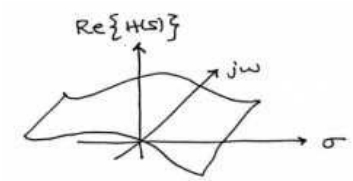
Unilateral

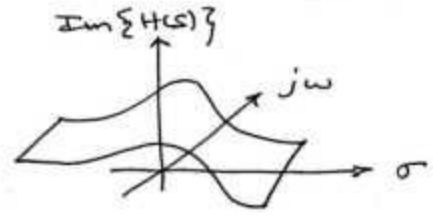
$$F(s) = \int_{0}^{\infty} f(t) e^{-(st)} \, \mathrm{d}t$$

The inverse Laplace transform function can be used to convert the solution from the s-domain back into the time domain. Given below is the inverse laplace transform for the bilateral type.

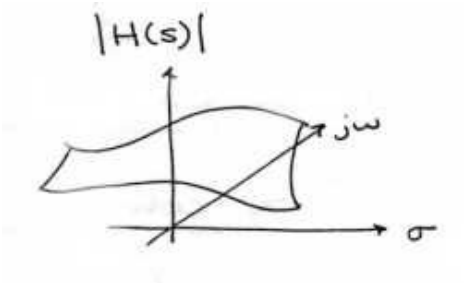
$$f(t) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} F(s) e^{st} \, \mathrm{d}s$$

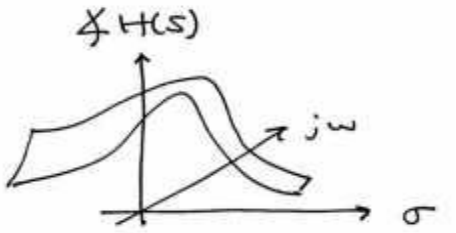
The envisioning of plots in Fourier was easy since we had the provision of a complex-valued function of a purely imaginary variable.
With Laplace, we have a complex-valued function of a complex variable. In order to examine the magnitude and phase or real and imaginary parts of this function, we must examine 3-dimensional surface plots of each component.





The Real part of H(s) The Imaginary part of H(s)

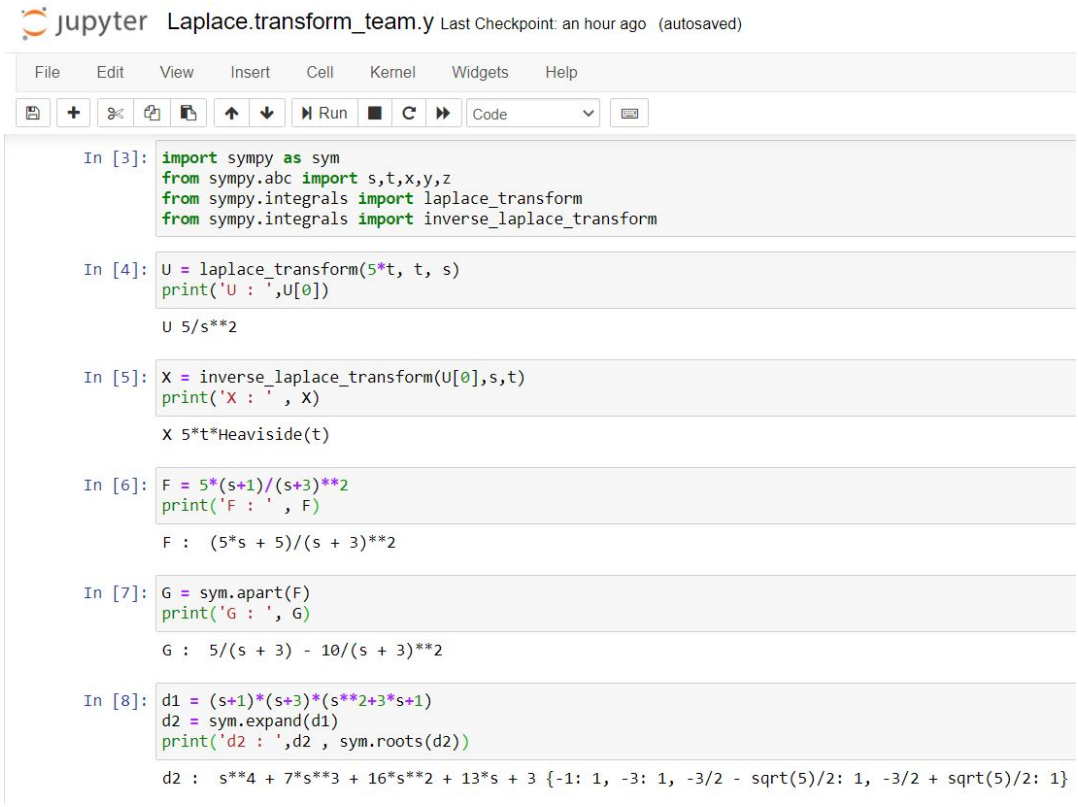




The Magnitude of H(s) The Phase of H(s)

The second kind of representation is more common than the real and imaginary parts.

A simple demonstration of Laplace and inverse Laplace transform using python code is given below. A module named sympy used to compute the Laplace transform



If x(t) and its 1st derivative is Laplace transformable, the initial and final values of x(t) have special equations called as :

Initial value theorem

$$x(0^+) = \lim_{s\to\infty} SX(S)$$

Final value theorem

$$x(\infty) = \lim_{s\to\infty} SX(S)$$

The code for simple laplace transform :

```
#include<stdio.h>
#include<math.h>
#define S 4
```

```
typedef float newvar[S+1][S+1];
void entrow(int i,newvar u)
{
   int j;
   printf("\n Enter the value of u[%d,j],j=1,%d+-
+\n",i,S);
   for(j=1;j<=S;j++)
   scanf("%f",&u[i][j]);
}
void entcol(int j, newvar u)
{
int i;
printf("Enter the value of u[i,%d],""i=2,%d\n",j,S-1);
for(i=2;i<=S-1;i++)
scanf("%f",&u[i][j]);
}
void oput(newvar u, int wd, int prsn)
{
   int i,j;
   for(i=i;i<=S;i++)
   {
      for(j=1;j<=S;j++)
      printf("%d,%d,%f",wd, prsn, u[i][j]);
      printf("\n");

   }
```

```
}
main()
{
   newvar u;
   float mer, ar, e, t;
  int i,j,itr, maxitr;
  for(i=1;i<=S;i++)
  for(j=1;j<=S;j++)
  u[i][j]=0;
  printf("\n  Enter the Boundary Condition\n");
  entrow(1,u); entrow(S,u);
  entcol(1,u); entcol(S,u);
  printf(" Enter the allowed error and maximum number of iteration : ");
  scanf("%f%f",&ar,&maxitr);
  for(itr=1;itr<=maxitr;itr++)
  {
     mer=0;
     for(i=2;i<S-1;i++)
     {
     for(j=2;j<=S-1;j++)
     {
        t=(u[i-1][j]+u[i+1][j]+u[i][j+1]+u[i][j-1])/4;
        e=fabs(u[i][j]-t);
        if(e>mer)
        mer=e;
        u[i][j]=t;
```

```
    }
    printf(" Iteration Number %d\n",itr);
    oput(u,9,2);
    if(mer<=ar)
    {
       printf(" After %d iteration \n The solution : \n",itr);
        oput(u,8,1);
       return 0;

    }
  }
  printf(" Sorry! The number of iteration is not sufficient");
  return 1;
}
}
```

We have understood a very essential mathematical tool in signals and systems, which is the Laplace transform, and also fourier transform earlier.

Now we will move on to another essential concept in this field, modulation, We will go on about trying to explain the overall idea of modulation and then try to generalize overall standard operations on signals as well as try to algorithmize these processes.

Modulation

We communicate signals from one place to another place in daily life, and most of those include long-range signal transmission. Sound energy or light energy usually spreads in our atmosphere in a spherical wave and thus we can notice that the energy comes down by inverse square law(surface area of the sphere is proportional to r^2 > thus when we want to transmit energy, it is very inefficient to depend on inverse square law.

Given the usual frequencies, it gets very complicated to transmit the proper signal without a proper medium i.e the signals need to be properly isolated and efficiently transmitted to the appropriate target.

Thus attenuation(Understanding the attenuation of a signal) should be reduced over the long-range. Now engineers have noticed that attenuation(signal strength coming down) is directly proportional to wavelength.

And the higher the frequency, the more efficient the transmission. The effective use of bandwidth(What is Bandwidth - Definition, Meaning & Explanation) is a necessary implementation. Bandwidth can be an analogy for the width of the road and the lesser the width of the vehicle, the more vehicles would fit, but in order to avoid a collision, vehicles should be properly distanced.

Since our audio frequency is very less, pioneers have developed a way where a higher frequency wave carries our lower frequency audio wave for transmission.

This carrier can solve most of the above-mentioned problems. It reduces attenuation, increases bandwidth due to higher frequency, and also allows our antenna height to be far less than the original antenna needed. It also gives us an opportunity to transmit similar frequency audio signals to two different targets without any overlap using two different carrier signals.(Wideband audio)

For example, cable tv bandwidth is approx 6 megHz. but we can get almost above 300 channels. This is possible because a six mega Hz carrier in a one-second slot lets say, we split the time duration to different channels since the frequency is very high.

And the destination electronics can be hardwired in such a way that each channel is properly separated. This is called multiplexing, or more specifically time-division multiplexing. Surprisingly this is just how the human ear usually works.

There is another type of multiplexing called frequency division multiplexing. This is very efficiently seen in Fm radio signals. Here these signals are propagated along with the same bandwidth.

Analog to digital conversion is very advantageous but the digitalization causes loss of smoothness this is compensated by increasing the bandwidth to 200 kHz gap between channels thus resulting in good quality of the channels, and the disadvantage being that it only works in line of sight propagation and would be usually 60-70km.

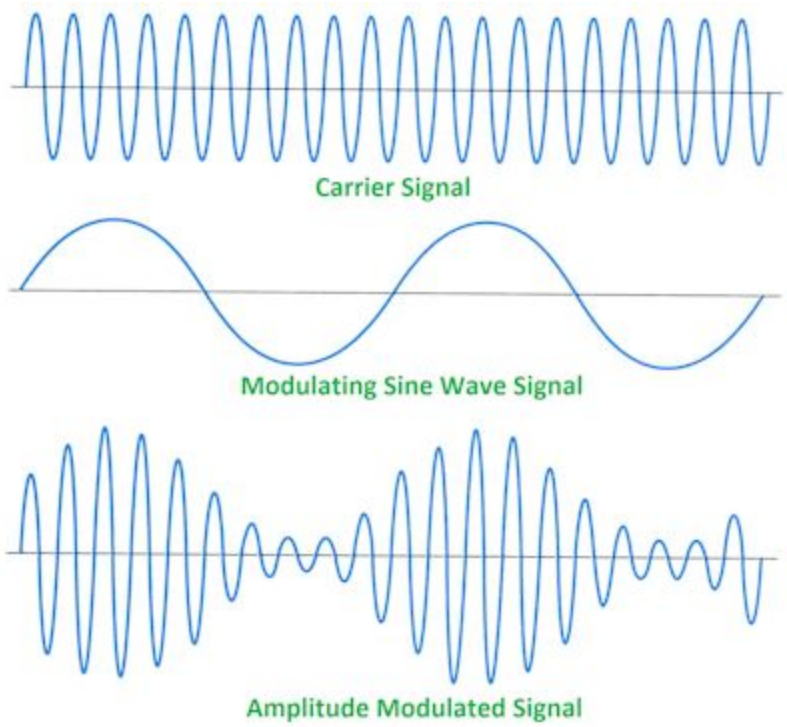
This shortcoming can become an advantage by utilizing the same frequency in different parts of the world for different channels (as the channels are transmitted only over a city-wide radius).

We have also noticed that some signal frequencies are transmitted by reflection from the ionosphere, and this type of reflection has enabled the signal to travel across large distances around the globe.

But this ionosphere cannot transmit video signals(they are too high) and can thus transmit only radio signals reflected from the ionosphere at appropriate frequencies.

Video frequencies thus require satellites to be transmitted across long distances. The satellites act as the ionosphere.

Now coming back to the carrier signal concept, let us assume a sine wave to be taken as our audio signal. And let us multiply this with a high-frequency carrier (modulate). Due to the high difference in the frequencies of the signals. The amplitude of the carrier signal modifies it to look like a baseband signal.



This is the most simple kind of modulation. At the destination, the signal needs to be demodulated to retrieve the original baseband signal. This type of modulation can be used efficiently only when the frequency is less viz. Audio signals.

Thus, all the signals share the same bandwidth and can be used to send different signals using such carrier waves. Most noise changes affect the amplitude of the signals. noise(unpleasant unrequired sound signals existing in the medium. Once we place a proper filter in the destination end to filter out the required audio signal after demodulation.

Amplitude modulation was found to be very easily affected by noise thus frequency modulation was introduced. Here the carrier signal is of very very high

frequencies in the order of 100MHz and the base and signal are in a few hundred Khz or sometimes thousands of Kz at maximum.

This modulation appears like small ripples in a big ocean. Thus usually frequency-modulated waves are short range line of sight transmissions as mentioned earlier.

The next type is phase modulation. In layman's terms, the phase is the observable lag in the wave. The theoretical wave that we are familiar with has 0 phase.

But waves usually have phases ranging from values 0 to 2π. Here we change the amplitude and frequency changes in the baseband signal are transmitted as phase changes into the carrier signal. Phase modulation makes the bandwidth more efficient to use.

Thus, in more formal categorisation of modulation, we have broadly four types of it:
Amplitude modulation, frequency modulation, pulse modulation ,spread spectrum.

Amplitude modulation is also referred to as multiplication of signals. Here is a picture that gives an idea of it.

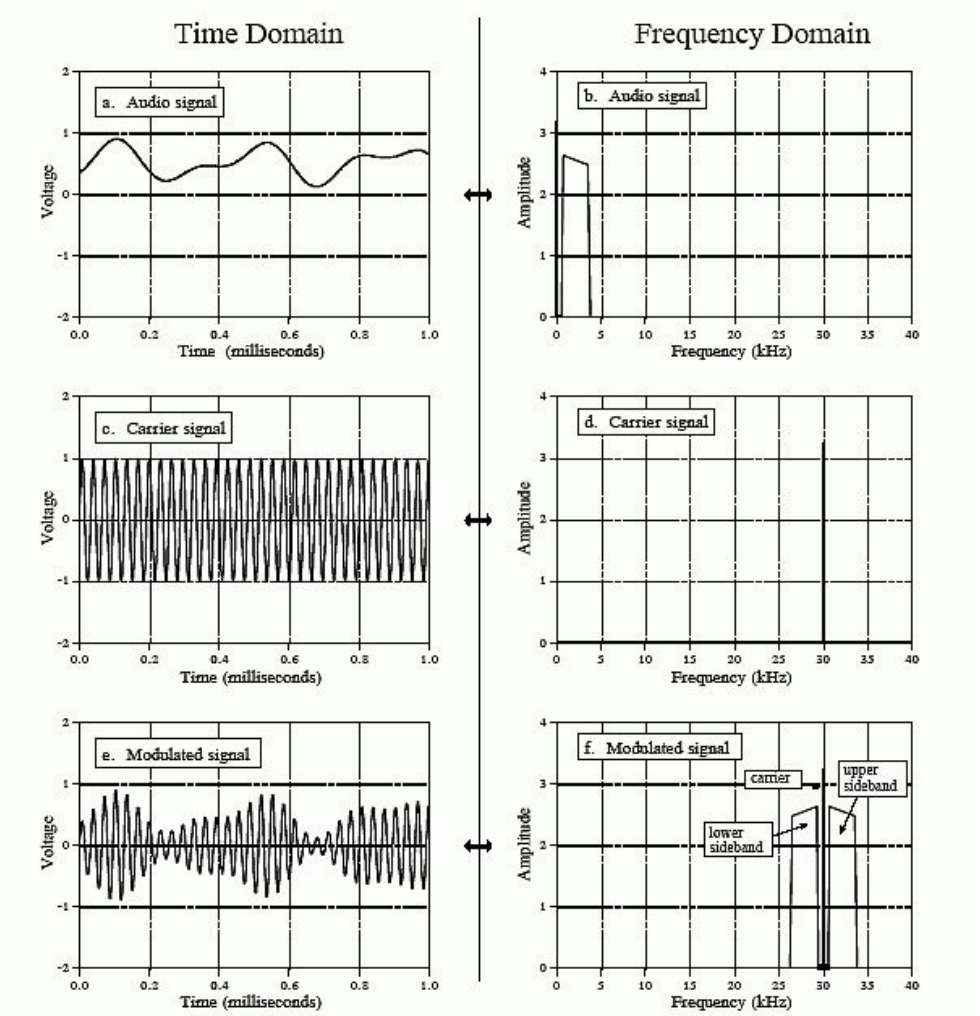


FIGURE 10-14
Amplitude modulation. In the time domain, amplitude modulation is achieved by multiplying the audio signal, (a), by the carrier signal, (c), to produce the modulated signal, (e). Since multiplication in the time domain corresponds to convolution in the frequency domain, the spectrum of the modulated signal is the spectrum of the audio signal shifted to the frequency of the carrier.

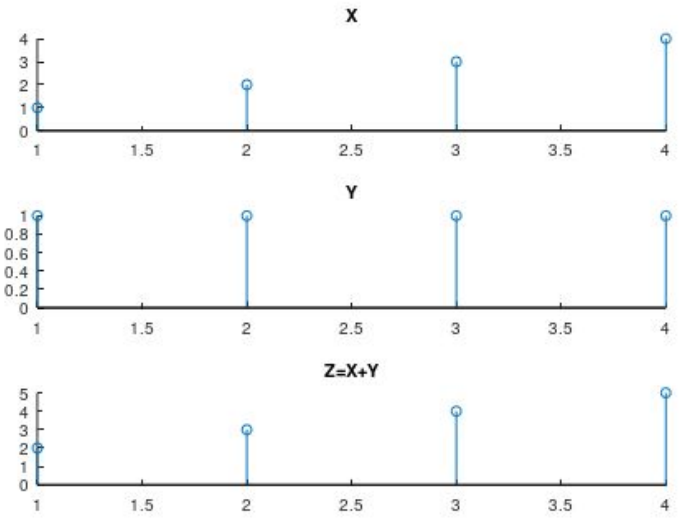
Ref: The Scientist and Engineer's Guide to Digital Signal Processing By Steven W. Smith, Ph.D.

A very useful reference:

A. Abdelgawad, S. Abdelhak, S. Ghosh and M. Bayoumi, "A low-power multiplication algorithm for signal processing in wireless sensor networks," 2009 52nd IEEE International Midwest Symposium on Circuits and Systems, Cancun, 2009, pp. 695-698, doi: 10.1109/MWSCAS.2009.5236001.

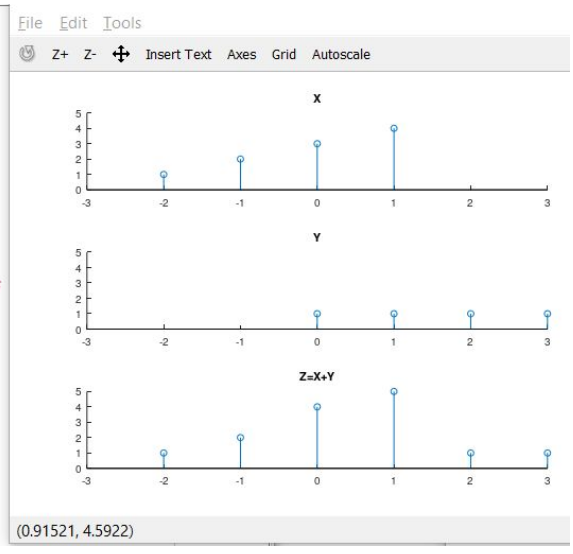
Addition of signals

```
x=[1 2 3 4];
subplot(3,1,1);
stem(x);
title('X');
y=[1 1 1 1];
subplot(3,1,2);
stem(y);
title('Y');
z=x+y;
subplot(3,1,3);
stem(z);
title('Z=X+Y');
```



(If the index values of x and y are different we have to find the range of output by comparing the index values of both signals.)

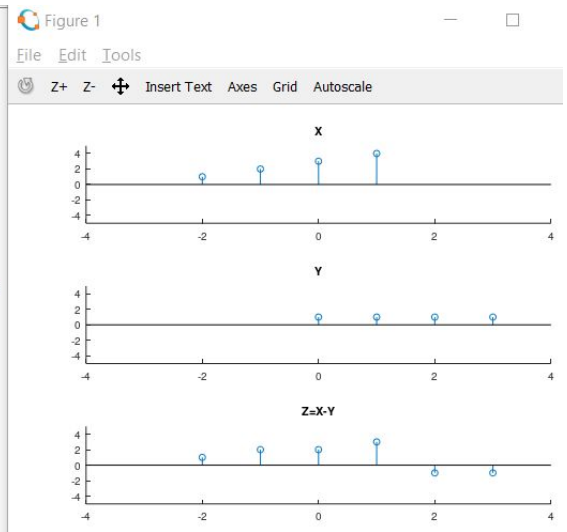
```
n1=-2:1;
x=[1 2 3 4];
subplot(3,1,1);
stem(n1,x);
title('X') ;
axis([-3 3 0 5]);
n2=0:3;
y=[1 1 1 1];
subplot(3,1,2);
stem(n2,y);
title('Y');
axis([-3 3 0 5]);
n3 =min (min(n1) ,min( n2 ) ) : max ( max ( n1 ) , max ( n2 ) );
s1 =zeros(1,length (n3) );
s2 =s1;
s1 (find ( ( n3>=min( n1 ) ) & ( n3 <=max ( n1 ) )==1 ) )=x;
% signal x with the duration of  output signal add
s2 (find ( ( n3>=min ( n2 ) ) & ( n3 <=max ( n2 ))==1) )=y;
 % signal y with the duration of  output signal add
add=s1 +s2; % addition
subplot(3,1,3)
stem(n3,add)
title('Z=X+Y');
axis([-3 3 0 5]);
```



Here, the min() and max() functions are used.

Subtraction of signals

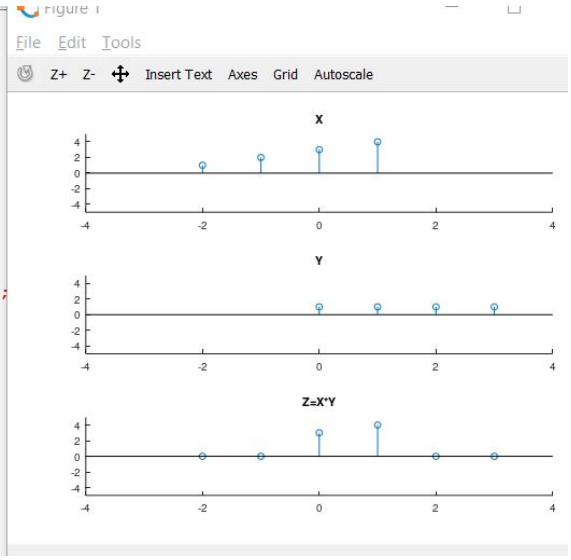
```
n1=-2:1;
x=[1 2 3 4];
subplot(3,1,1);
stem(n1,x);
title('X') ;
axis([-4 4 -5 5]);
n2=0:3;
y=[1 1 1 1];
subplot(3,1,2);
stem(n2,y);
title('Y');
axis([-4 4 -5 5]);
n3 =min (min(n1) ,min( n2 ) ) : max ( max ( n1 ) , max ( n2 ) );
s1 =zeros(1,length (n3) );
s2 =s1;
s1 (find ( ( n3>=min( n1 ) ) & ( n3 <=max ( n1 ) )==1 ) )=x;
% signal x with the duration of output signal 'sub'
s2 (find ( ( n3>=min ( n2 ) ) & ( n3 <=max ( n2 ))==1) )=y;
% signal y with the duration of output signal 'sub'
sub=s1 - s2; % subtraction
subplot(3,1,3)
stem(n3,sub)
title('Z=X-Y');
axis([-4 4 -5 5]);
```



Multiplication of signals

Multiplication operation results in the generation of a signal whose values can be obtained by multiplying the corresponding values of the original signals. This is true irrespective of whether we are dealing with a continuous-time or discrete-time signal.

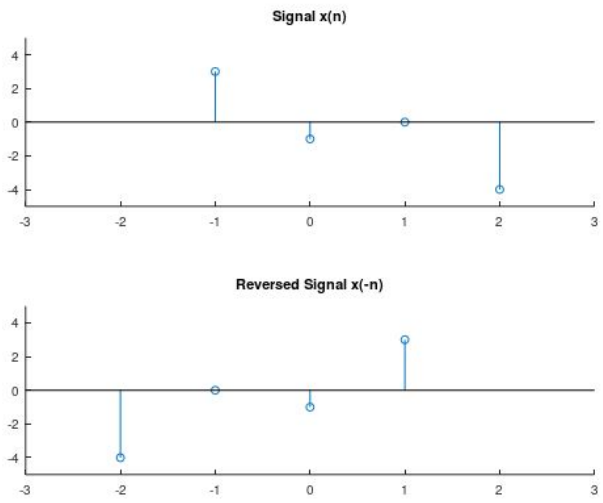
```
n1=-2:1;
x=[1 2 3 4];
subplot(3,1,1);
stem(n1,x);
title('X') ;
axis([-4 4 -5 5]);
n2=0:3;
y=[1 1 1 1];
subplot(3,1,2);
stem(n2,y);
title('Y');
axis([-4 4 -5 5]);
n3 =min (min(n1) ,min( n2 ) ) : max ( max ( n1 ) , max ( n2 ) );
s1 =zeros(1,length (n3) );
s2 =s1;
s1 (find ( ( n3>=min( n1 ) ) & ( n3 <=max ( n1 ) )==1 ) )=x;
% signal x with the duration of output signal 'mul'
s2 (find ( ( n3>=min ( n2 ) ) & ( n3 <=max ( n2 ))==1) )=y;
% signal y with the duration of output signal 'mul'
mul=s1 .* s2; % multiplication
subplot(3,1,3)
stem(n3,mul)
title('Z=X*Y');
axis([-4 4 -5 5]);
```



Reversing of signal

If the independent variable t is replaced by '−t' , this operation is known as time reversal of the signal about the y-axis or amplitude axis. This can be achieved by taking mirror image of the signal x(t) about y-axis or by rotating x(t) by 180° about y-axis

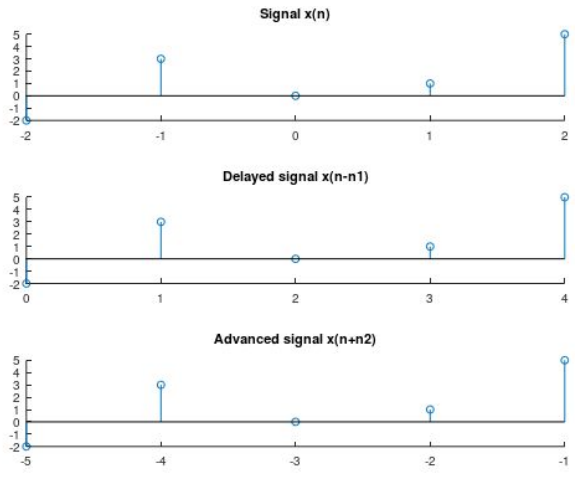
```
n=-1:2;
x=[3 -1 0 -4];
subplot(2,1,1)
stem(n,x);
axis([-3 3 -5 5]);
title('Signal x(n)');
c=fliplr(x);
y=fliplr(-n);
subplot(2,1,2);
stem(y,c);
axis([-3 3 -5 5]);
title('Reversed Signal x(-n)')
```



Shifting a signal

Shifting means movement of the signal, either in time domain aroundY−axis or in amplitude domain aroundX−axis. It can either be done by delaying or advancing the signal.

```
n1=input('Enter the amount to be delayed = ');
n2=input('Enter the amount to be advanced = ');
n=-2:2;
x=[-2 3 0 1 5];
subplot(3,1,1);
stem(n,x);
title('Signal x(n)');
m=n+n1;
y=x;
subplot(3,1,2);
stem(m,y);
title('Delayed signal x(n-n1)');
t=n-n2;
z=x;
subplot(3,1,3);
stem(t,z);
title('Advanced signal x(n+n2)');
```

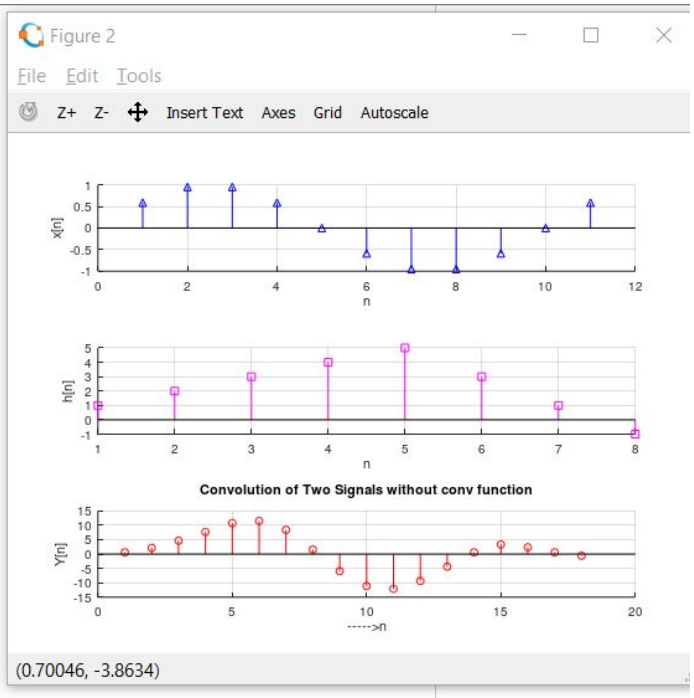


```
>> run test1
Enter the amount to be delayed = 2
Enter the amount to be advanced = 3
>>
```

Linear convolution of two signals

Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Convolution is important because it relates the three signals of interest: the input signal, the output signal, and the impulse response.

```
x=sin(2*pi*0.1.*(1:1:11));
%h=input('Enter h:   ')
h=[1 2 3 4 5 3 1 -1];
% convolution
m=length(x);
n=length(h);
X=[x,zeros(1,n)];
H=[h,zeros(1,m)];
for i=1:n+m-1
    Y(i)=0;
    for j=1:m
        if(i-j+1>0)
            Y(i)=Y(i)+X(j)*H(i-j+1);
        else
        end
    end
end
% plot results
figure;
subplot(3,1,1); stem(x, '-b^'); xlabel('n');
ylabel('x[n]'); grid on;
subplot(3,1,2); stem(h, '-ms');
xlabel('n'); ylabel('h[n]'); grid on;
subplot(3,1,3); stem(Y, '-ro');
ylabel('Y[n]'); xlabel('----->n'); grid on;
title('Convolution of Two Signals');
```



Small detour!! :)

Hill climbing algorithm

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

Hill climbing algorithm is a technique which is used for optimizing mathematical problems. One of the widely discussed examples of Hill climbing algorithm is the Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.

There are many variants of hill climbing but for now, let us look at its most simplest type

Algorithm for Simple Hill Climbing

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    i. If it is goal state, then return success and quit.

ii. else if it is better than the current state then assign a new state as a current state.

iii. else if not better than the current state, then return to step 2.

- **Step 5:** Exit.

9.Monte Carlo Simulation

What Is a Monte Carlo Simulation?

Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables. It is a technique used to understand the impact of risk and uncertainty in prediction and forecasting models**.**

Why is it named so??

Monte Carlo simulations are **named** after the popular gambling destination in Monaco, since chance and random outcomes are central to the modeling technique, much as they are to games like roulette, dice, and slot machines.

Used for:

Monte Carlo simulation performs risk analysis by building models of possible results by substituting a range of values—a probability distribution—for any factor that has inherent uncertainty. It then calculates results over and over, each time using a different set of random values from the probability functions.

A Monte Carlo simulation can be used to tackle a range of problems in virtually every field such as finance, engineering, supply chain, and science. It is also referred to as a multiple probability simulation.

KEY TAKEAWAYS

- A Monte Carlo simulation is a model used to predict the probability of different outcomes when the intervention of random variables is present.
- Monte Carlo simulations help to explain the impact of risk and uncertainty in prediction and forecasting models.
- A variety of fields utilize Monte Carlo simulations, including finance, engineering, supply chain, and science.
- The basis of a Monte Carlo simulation involves assigning multiple values to an uncertain variable to achieve multiple results and then to average the results to obtain an estimate.
- Monte Carlo simulations assume perfectly efficient markets.
- It can also be categorized as a method of estimating the value of an unknown quantity using the principles of inferential statistics.

Inferential Statistics:

- With inferential statistics you take samples from the total population and make generalizations.
- Population: A set of examples.
- Sample: A **proper subset** of a population.
- KEY FACT:
    - A random sample exhibits the same properties as the population from which it is drawn

Monte Carlo experiments are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle.

They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three problem classes:optimization, numerical integration, and generating draws from a probability distribution.

Monte carlo simulations are produced by following these steps:

1. Define a domain of possible inputs
2. Generate inputs randomly from a probability distribution over the domain
3. Perform a deterministic computation on the inputs
4. Aggregate the results

Calculating pi is one of the most popular approach using monte carlo simulations

There are quite similar approaches which uses monte carlo simulations and helps us in most ways :))

Monte carlo simulation in Algorithms

MCS based bat algorithm

The Bat Algorithm which is based on Monte Carlo simulation is used for Solving Stochastic Multi-Objective Optimization Problems.The traditional

Bat algorithm requires the deterministic equivalence of the problem which is only possible if the random variables follow specific distributions.

However by using Monte Carlo simulation, the algorithm is used to obtain the optimal solution.

Bat Algorithm is inspired from the behavior of microbats, in which they use an echolocation which is a kind of sonar. It is a very useful capability that enables them to notice their prey, turn over obstacles, and locate their paths in the dark.

The echo plays an important role in the search process, where these bats produce a very high sound pulse and then wait to listen for the echo that returns back from

the neighboring objects that they hit. In a magical way, they have the ability to determine the distance and the location of their prey or target.

The basic rules of the BA are used to solve unconstrained optimization problems. However, for implementation point of view, nonlinear uncertain constraints should be handled.

In this section, a stochastic multi-objective BA is proposed in which the Monte Carlo simulation approach is used for constraint handling. This algorithm is capable of handling any constraints where the random variables involved in the parameters follow an arbitrary continuous distribution with known probability density functions.

The stochastic constraints are represented as chance constraints with some determined level of probability. The feasibility of the generated solutions is checked by applying the technique of the stochastic Monte Carlo simulation

Error distribution

The probability of error distribution can be estimated using Monte-Carlo simulation

Genetic algorithms and Monte Carlo simulation for optimisation.
Using empirical rule

This rule under some assumptions states the following

I. ~68% of the data lies within one standard deviation(mean - standard deviation) and (mean + standard deviation).

II. ~95% of the data lies within 2 standard deviations(mean - 2*standard deviation) and (mean + 2*standard deviation).

III. ~99.7 of the data lies within 3 standard deviations.

Examples to understand **MCS** better:

Given a single coin, estimate the fraction fraction of head you would get if you flipped the coin infinitely large a number of times.

Now let's make it a small number. Let it be 100 and let all show up heads. Now do you actually think that head is more probable to appear on the next flip..

Now let's change it a bit more, let the first 50 outcomes be heads, next 48 are tails and last two are heads.

Now do you say that the probability that the next flip turns up to be heads is 52/100..

Yes you might, because that was typically the best guess we could make, but isn't actually right.

Confidence is also prior in these situations and that mainly depends on 2 major parameters in our mind:

I. Size of sample (100 vs 2)
II. Variance (All heads vs 52 heads)

As variance grows we need larger numbers of samples to have the same degree of confidence.

Small detour!! :)

Calculating the Digits of π

The π value is one of the most used and common constants in maths and many other fields of science and application. To calculate the value of pi in a fast and efficient manner is a challenge when there is extensive computation done around this value. In such cases, special theorems from maths are employed to implement a shortcut in finding the digits of π.

Legendre-Gauss Quadrature is a famous method to find the digits of π using very minimal iterations. This is used in many softwares and applications and is called the Gauss-Legendre algorithm. The algorithm is as follows:

- Here, some initial values are set for four different variables. (Initialisation)
  - a = 1
  - b = 1 / √2
  - t = 1/4
  - x = 1
- Repeat the following statements until the difference between a and b is within the desired accuracy;(Iterating n times and updating)
  - y = a
  - a = (a+b) / 2
  - b = √(b*y)
  - t = t - x * $(y-a)^2$
  - x = 2 * x
- Pi is approximated with a, b, and t as;
  - Pi = $(a+b)^2$ / (4*t)

Python code for Gauss-legendre

```
def pi_gauss_legendre():
    D = decimal.Decimal
    with decimal.localcontext() as ctx:
        ctx.prec += 2
        a, b, t, p = 1, 1/D(2).sqrt(), 1/D(4), 1
        pi = None
        while 1:
            an    = (a + b) / 2
            b     = (a * b).sqrt()
            t    -= p * (a - an) * (a - an)
            a, p  = an, 2*p
            piold = pi
            pi    = (a + b) * (a + b) / (4 * t)
            if pi == piold:  # equal within given precision
                break
    return +pi
```

The first three iterations give (approximations are given up to and including the first incorrect digit):

3.140...

3.14159264...

3.1415926535897932382…

Algorithms like the Chudnovsky are employed for an even faster calculation of digits of $\boldsymbol{\pi}$.

10.Password generation

Are password generators safe?

The short answer is that it is safer to have a password generated by an online random password generator than to use a password even a toddler or weak hacking software can figure out. ... The concept behind online password generators is noble and they definitely help you create strong passwords.

Can you trust password generators?

No. It is not safe to generate passwords online. The average user has no way to verify that the web site is not keeping a copy of your password. The average user has no way to verify that the password generation code is using good entropy (and Javascript's Math).

What is the best password encryption algorithm?

Passwords should be hashed with either PBKDF2, bcrypt or scrypt, MD-5 and SHA-3 should never be used for password hashing and SHA-½ (password+salt) are a big no-no as well. Currently the most vetted hashing algorithm providing most security is bcrypt.

Have u ever thought of passwords you create during any signup?

What I observe is that user-generated passwords are easy to remember, not secure and it is not all complex and the passwords which algorithms create are most secure, complex, and not easy to remember.

Is there any method Easy to remember.?Difficult to crack?

Let’s see the methods for generating algorithms.

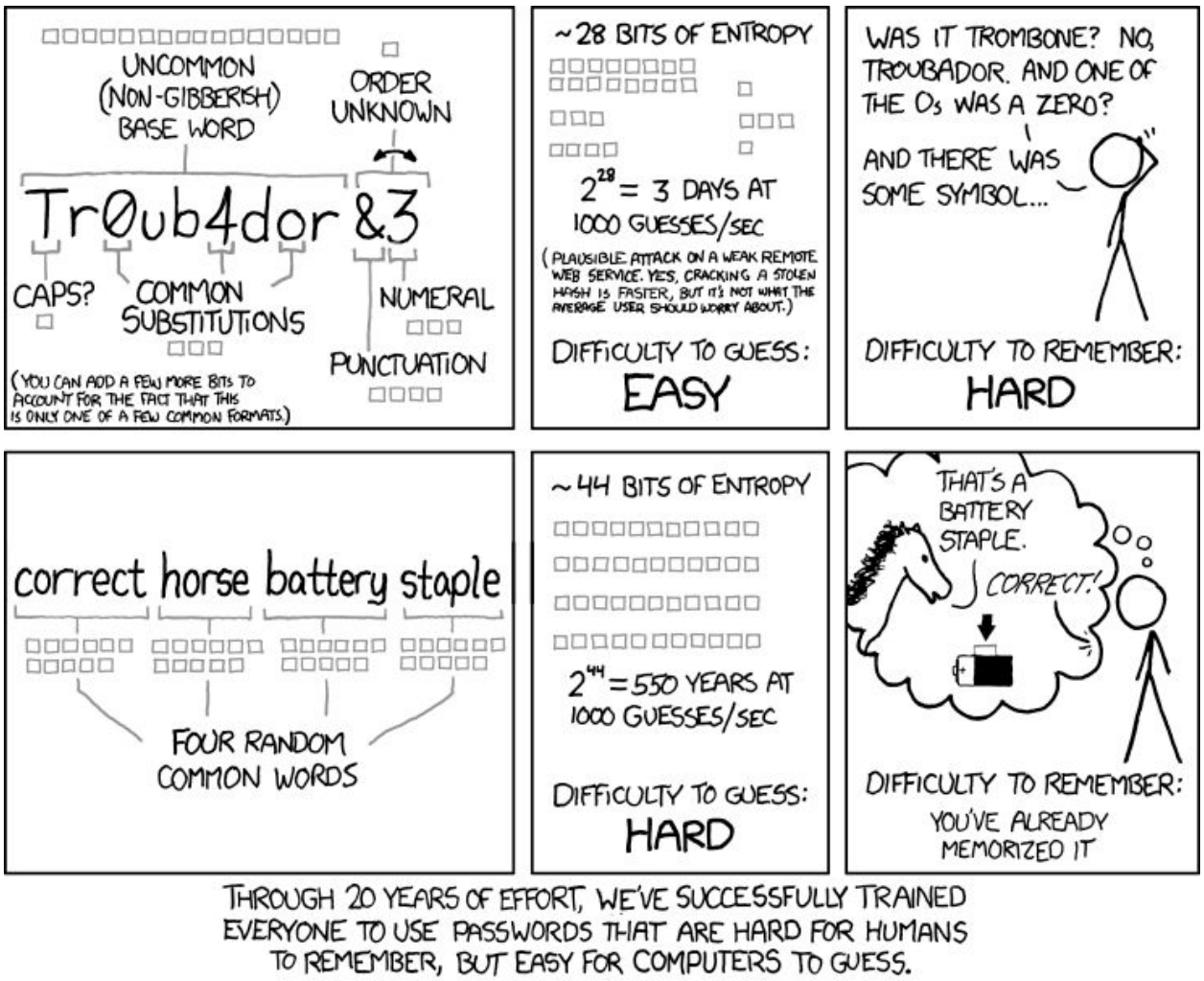
1. XKCD Baseline
2. Letter MnemonicFirst
3. All letter Method
4. Frequency Method
5. Poetry

1. XKCD Baseline

The XKCD password scheme is as good as it ever was. The security doesn't derive from it is unknown, but from it being a good way to generate memorable passwords from a large search space. If you select the words to use rather than generate them randomly, though, this advantage is lost, humans aren't good at being random.

The bit about memory is poorly stated, but it *is* a concern: if password-stealing malware ever gets on your computer, it'll sweep up everything text-like from RAM and the hard drive to use in a directed attack on your accounts.

Observe an Interesting Comic which says about this xkcd method



This comic compares Tr0ub4dor&3 at an assumed 28-bit entropy (though I

calculate it as 34.6) to correcthorsebatterystaple and its assumed 44 bits of entropy (a four-word diceware code is 51.7 bits … but one of those words isn't diceware. Using a simple 100k-word spelling dictionary, I calculate it to be 66.4 bits).

First, let's make this easier to understand.

There are 94 printable characters. A one-character password has **$\log_2(94) = 6.55$ bits** of entropy. Two characters have **$\log_2(94^2) = 13.10$** bits of entropy. You can divide the final entropy of a password scheme by 6.55 in order to determine the equivalent complexity of a purely random password as measured in characters.

Therefore:

- 28 bits of entropy ≈ 4.3 character password
- 44 bits of entropy ≈ 6.7 character password
- 66.4 bits of entropy ≈ 10.1 character password

2. Letter MnemonicFirst

XKCD passwords are short but nonsensical, This is the method that creates longer but fluent English sentences. We might think to guaran-tee fluency by selecting sentences from an already-existing text corpus, but no corpus is large enough to contain 260(~1018) distinct sentences.
Therefore, we must be able to synthesize new English strings.

In our first sentence generation method (First Letter Mnemonic), we store our input 60-bit code in the first letters of each word. We divide the 60-bit code into 4-bit sections, e.g., ‘0100-1101-1101-...’. Every4-bit sequence type corresponds to an English letter.

```
import random

import fire
import math
import sys

if sys.version_info[0] < 3 or sys.version_info[1] < 6:
        raise Exception("Python version is %. Must be using at least Python 3.6."
% sys.version_info)

def run(passwords=10,length=3,wordlist="list.txt",exclude="exclude.txt"):
        words=list(set((a[:len(a)-1]
        for a in open(wordlist).readlines())).difference(
        set((a[:len(a)-1]
        for a in open(exclude).readlines()))
        ))
        sys.stderr.write("Word list from {wordlist} minus {exclude} contains {}
strings.\nLength {length} passwords\nis equivalent to {:2} characters ASCII
33-127\n".format(
                len(words),
```

```
                math.log(len(words)**length,(127-33)),
                **locals()
                ))
        for x in range(0,passwords) :
        print("-".join((random.choice(words) for a in range(0,length))))


if __name__ == "__main__" :
        fire.Fire(run)
```

3. All letter Method

In this technique, mapping a 1 to certain letters and a 0 to others, then creates sentences of meaningful English. This yields 4·1084 possible output strings per input, 3·1056 of which consists of legal English words.

From those3·1056strings, we choose the one that yields the best word 5-gram score. It is not immediately clear how to process a letter-based lattice with a word-based language model.

We solve this search problem by casting it as one of ma-chine translation from bit-strings to English. We create a phrase translation table by pairing each English word with a corresponding “bit phrase”.

Sample entries include:

din ||| 1 0 1

through ||| 1 0 0 0 0 0 0

yields ||| 1 0 0 1 1 1

We then use the Moses machine translation toolkit(Koehn et al., 2007) to search for the 1-best translation of our input 60-bit string, using the phrase table and a 5-gram English LM, disallowing re-ordering.

4. Frequency Method

This technique modifies the phrase table by assigning short bit codes to frequent words, and long bit codes to infrequent words.

For example:

din ||| 0 1 1 0 1 0 1 0 0

through ||| 1 1 1 1

yields ||| 0 1 0 1 1 1 0 1

5. Poetry

In this technique we create poems of two eight-syllable lines, in iambic pentameter, complete with rhymes.Poetry passwords were among the easiest to memorize.

Subjects tended to prefer the sentences, but they weren't actually that good at remembering them. They would recall the gist of the sentence, but mis-remember the exact wording.The rhyming and rhythm of poetry could be what makes poem-passwords easier to remember.

After all, as the authors point out, this helped ancient bards memorize epic poems so long that people today are reluctant to even read them.

Here's some example output:

01000110100000110010011110110010100011001001010000011110000

The Proctor faltering Chinese
applied retention guarantees.

Code in python for poetry technique password generation

```
if not password:
    try:
        import keyring
        if keyring.get_keyring():
            password = keyring.get_password(repository_name, username)
    except ImportError:
        print("Install keyring to store passwords securely")
        keyring = None
```

```
if not password:
    password = self._io.ask_hidden("Password:")
    if keyring and keyring.get_keyring():
        keyring.set_password(repository_name, username, password)
```

Fun Facts:

What are the 5 most common passwords?

The top 5 most common passwords were:

- qwerty.
- password.
- 111111.
- 12345678.
- abc123.

What is the most hacked password?

Here are the most commonly used passwords revealed in data breaches, according to the report:

123456 (23.2 million)
123456789 (7.7 million)
qwerty (3.8 million)

password (3.6 million)
111111 (3.1 million)

QR Code Generation

What are QR codes?

QR Codes,which stand for “quick response” codes, are little black and white squiggly barcodes that usually look something like this:

They are often found on direct mail, signage, billboards and in commercials.

These were originally designed in Japan for the automotive industry, marketers

adopted the barcodes because of their large storage capacity and ability to translate additional information to consumers beyond what creative and/or packaging could convey.

If a consumer sees a QR code somewhere, they can take out their mobile device, open up a QR code scanner, and "scan" the barcode to gain access to additional information.

How to create a QR Code for a website

1. **Selecting a QR Code Generator:**
   There are a lot of QR code generators available online, but a few of the post popular include Kaywa, GOQR.me, Visualead, and QR Stuff. Some things to look for when choosing a QR code generator are whether you can track and analyze performance, if it allows you to design a code that's unique to your brand, and if it is compatible with common QR code readers.

2. **Design and linking the website with the QR Code:**
   In this article,http://goqr.me/ is used for generating the QR Code.
   1. Select what type of content you want your QR code to represent — we'll choose a URL for this article.

2. Insert the URL of your respective website. And gives the respective QR with the data you entered embedded in it.

How does QR code work?

The main advantage is that you can store up to a hundred times more information on a QR code than on a conventional horizontal barcode. In addition, QR codes can be scanned from any direction for 360 degrees. ... An Android user can use something like QR Code Reader, and an iPhone user can download the Quick Scan app.

Why are QR Codes bad?

1) QR codes and 2D Tags in general are ugly, generic and mess with a brand's aesthetic, destroying much of the investment made by brands to develop distinct brand identities.

2) The codes have limited uses and are only capable of translating into a text string that sends users to a website, phone number or SMS.

What are the two types of QR code?

Micro QR Code:

This QR Code is usually found on product packaging. It only has one orientation making it easier to print on smaller surfaces. This code is viable even as a 2 module whereas a QR Code requires at least 4 modules. The largest version of this QR Code is M4 (17x17 modules) and is capable of storing 35 modules.

iQR Code:

This can be printed as a square or a rectangular QR Code. It can be printed as dot pattern code, inversion code or a turned over code. The maximum version is 61 (422x422 modules) which can store about 40,000 numerals.

SQRC Code:

This looks like a regular QR Code except it is restricted and is used to store confidential information.

FrameQR:

This type of QR Code has a frame area wherein you can place letters and images and is used for promotional activities.

HCC2D Code:

The High Capacity Colored 2-Dimensional (HCC2D) Code is still in the prototyping phase and has been proposed by researchers to preserve QR robustness to distortions. It uses colors to increase data density and to cope with chromatic distortions, HCC2D codes use an additional field called Color Palette Pattern.

As everything we do in our daily life has a set of rules and guidelines, let's talk about Do's and Don'ts of the QR's. I know it looks a bit funny. Do this QR's also have do's and don'ts. But they aren't what you are thinking. So let's dive through,

Do's:

1. Put QR codes in places where scanning is easy, and there's enough time for the consumer to actually scan the code.
2. Do mobile optimize the page to which you're sending people. Consumers will be on their phone when scanning the QR code, so they should be brought to a page with a positive mobile experience.

Don'ts:

1. Don't require a special QR code scanner.The QR code should be app-agnostic so that anyone can scan the code with any reader.

The process (and high-level algorithm) for generating a QR Code symbol is as follows:

1. Choose the text (Unicode string) or binary data (byte string) to encode.

2. Choose one of the 4 error correction levels (ECL). A higher ECC level will yield a barcode that tolerates more damaged parts while preserving the payload data, but will tend to increase the version number (i.e. more modules in width and height).

3. Encode the text into a sequence of zero or more segments. A segment in byte mode can encode any data, but using alphanumeric or numeric mode is more compact if the text falls into these subsets.

4. Based on the segments to be encoded and the ECL, choose a suitable QR Code version to contain the data, preferably the smallest one.

5. Concatenate the segments (which have headers and payload) and add a terminator. The result is a sequence of bits.

6. Add padding bits and bytes to fill the remaining data space (based on the version and ECL).

7. Reinterpret the bitstream as a sequence of bytes, then divide it into blocks. Compute and append error correction bytes to each block. Interleave bytes from each block to form the final sequence of 8-bit codewords to be drawn.

8. Initialize a blank square grid based on the version number.

9. Draw the function patterns (finders, alignment, timing, version info, etc.) onto the appropriate modules. This is formatting overhead to support the QR Code standard, and does not encode any user data.

10. Draw the sequence of (data + error correction) codewords onto the QR Code symbol, starting from the bottom right. Two columns at a time are used, and the scanning process zigzags going upward and downward. Any module that was drawn for a function pattern is skipped over in this step.

11. Either manually or automatically choose a mask pattern to apply to the data modules. If masking automatically, then all 8 possibilities are tested and the one with the lowest penalty score is accepted. Note that the format

information is redrawn to reflect the mask chosen.

12. We are now finished with the algorithmic parts of QR Code generation. The remaining work is to render the newly generated barcode symbol as a picture on screen, or save it as an image file on disk.

Note that my QR Code generator library provides the logic to perform steps 3 through 11. The other steps must be performed by the user of the library.

Small detour!! :)

Maze algorithm

Even from our childhood , everyone of us had our fun with mazes and getting frustrated with some pretty tough ones that make us go round and round in circles. Well, what if there was an algorithm that could solve those mazes that the childhood version of you longed to solve! Here is the Lee algorithm that gives the shortest path/ fastest path out of a maze

The algorithm is a breadth-first based algorithm that uses queues to store the steps. It usually uses the following steps:

1. Choose a starting point and add it to the queue.
2. Add the valid neighboring cells to the queue.
3. Remove the position you are on from the queue and continue to the next element.
4. Repeat steps 2 and 3 until the queue is empty.

11.Shopping malls

Shopping is the biggest business ever which makes customers buy things from the seller where the seller makes a lot of money. Although the producer of the goods gets Money , it's different from the money made by the seller. An entire sector is there for this which is called service sector which includes sellers selling goods to buyers.

Our approach is to explain the algorithms used by sellers to make their profits high and also to sell all their goods , simply playing some tricks on buyers to attract them and to make them buy it.

Here are the few algorithms...

Customer segmentation

Customer segmentation is grouping customers based on different factors like age, gender, interests,income etc. Nowadays ecommerce stores also consider browsing behaviour, purchase history, lifetime value etc..

All these additional factors make it significantly more complex to group customers.

However, the upside is that there is far more data to analyse and the powerful algorithms now available make it possible for online stores to find new groupings based on data patterns that, previously, were not known to exist.

Understanding these new segments opens up important new marketing opportunities for eCommerce stores.

Recommending products to customers

Algorithms known as ‘product recommendation engines’ are used extensively in eCommerce in order to find the most suitable products for customers. These engines use a range of data to show relevant products which are more likely to be purchased.

This helps improve both user-experience by cutting down the amount of time a customer searches and, consequently, improves sales.

Importantly, the algorithm learns how long customers will spend looking for a product before they get fed up and look elsewhere and will aim to deliver the right choices to the customer within that time frame.

They make customers spend more time on their sites and attract them with the recommendation products and make them buy those things also.

Intelligent site search

Although virtually every website now offers customers a site search facility, the standard search that comes built into your theme or is added through a plugin is, by modern standards, rather basic. These tend to work by matching products in your database with the key terms inputted by the user's search query.

While these can provide accurate results, this is not always the case and often leads to customers having to modify or filter their query before they find what they are actually looking for.

Modern machine learning algorithms are able to provide more relevant results by taking into consideration other kinds of data, such as purchase histories and add to cart behaviour that can list items that similar shoppers have purchased.

Dynamic pricing

The retail sector is enormously competitive and margins are increasingly slim. Getting the pricing right can sometimes be the difference between staying in

business or going under. Today, many eCommerce sites rely on dynamic pricing algorithms to help them achieve the optimum price for their products, ensuring they remain both competitive and profitable.

Dynamic pricing applications do a number of jobs, not only do they tell companies the best prices, they can also be used to automate the setting of those prices on the website, saving the company the hassle of having to do it manually.

This can be very helpful as prices may change often to take into account such things as purchasing costs, competitor price changes, seasonal fluctuations, customer demand and product availability. In addition, the algorithm can be used to split test pricing points in order to fathom what customers are willing to pay.

Personalisation

Personalisation is one of today's most important marketing techniques as it provides each user with a shopping experience tailored to their own needs and wants. Personalisation is far more than putting the customer's name on the website when they log in and showing them products related to the last items they bought or looked at.

Using algorithms, companies are now able to communicate over the user's preferred channel (email, app notification, text message, etc.) the products can be displayed in the way the user prefers, the actual products shown can be far more

relevant and they can be given offers and deals that are more likely to be taken up. Not only does this increase sales; it can have a significant impact on customer loyalty.

Discount Sales:

Discount sales are the most used terms in shopping platforms to attract customers to buy things . In most of the cases, they just use a high price as mrp and a normal price as a discount price which feels like it is a very costly product and it is at its best price.

They create an illusion to people which makes them buy it even if they don't need it.

They also use the terms festival discount sale, year end sale to attract more people and to make high profits.

Small detour!! :)

The first formal algorithm in sorting is usually the bubble sort. And though that's the least used sort, it is a basic algorithm. And then there is the insertion sort which is not so frequently used, nevertheless important. Many people have tried and are

trying to come up with new and different types of sorting algorithms. Though there are some that are really very efficient, some algorithms just don't come into limelight because they neither have different techniques nor efficiency. We just want to introduce some of those to you. Just have fun knowing them!!There is an algorithm called the cocktail shaker sort which is just bidirectional sort and the gnome sort, which is a variation of insertion sort.

➔ Cocktail shaker sort

- ❑ The first stage loops through the array from left to right, just like the Bubble Sort. During the loop, adjacent items are compared and if the value on the left is greater than the value on the right, then values are swapped. At the end of the first iteration, the largest number will reside at the end of the array.
- ❑ The second stage loops through the array in the opposite direction- starting from the item just before the most recently sorted item, and moving back to the start of the array. Here also, adjacent items are compared and are swapped if required.
- ❑ As the cocktail shaker sort goes bidirectionally, the range of possible swaps, which is the range to be tested, will reduce per pass, thus reducing the overall running time slightly.

➔ Gnome sort

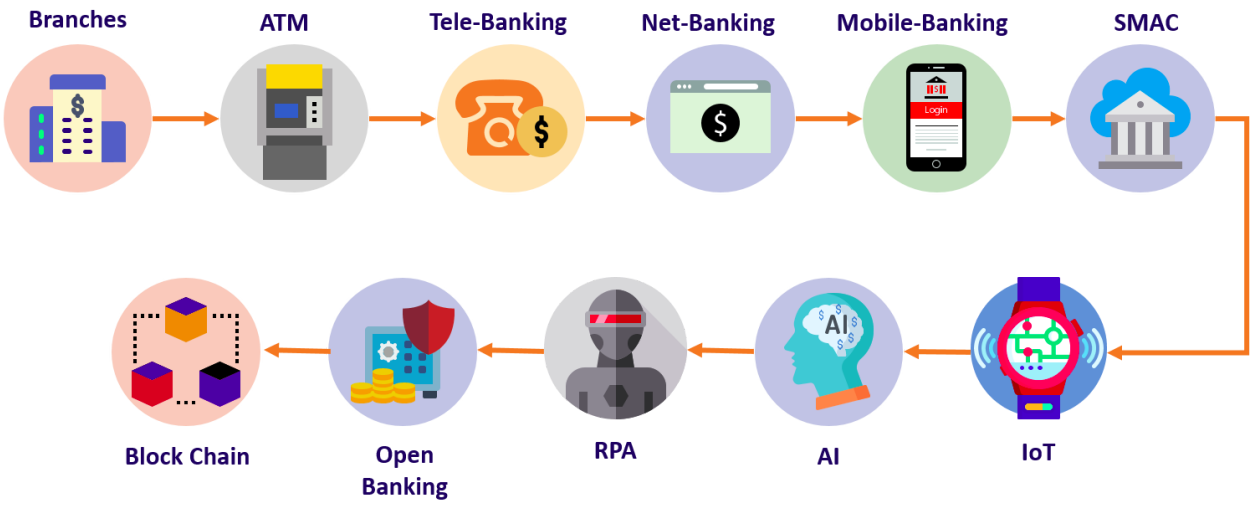
- ❑ Gnome Sort is a simple sorting algorithm with time complexity O(N^2) where the key idea is to swap adjacent elements (if not in order) to sort the entire list. It is similar to Insertion sort except the fact that in this case, we swap adjacent elements.
- ❑ It is inspired by the standard Dutch Garden Gnome sorting his flower pots. A garden gnome sorts the flower pots by the following method:

- ➢ If the flower pot just before and after him are in the correct order, then he moves one step forward.
- ➢ If it is not in correct order, he swaps the pots and moves back one step.
- ➢ At the starting when there is no pot before him, he steps forward and on reaching the end of the pot line, the list is sorted.

12.Banking and transactions

In 1998, ICICI bank in India started online banking services for the first time in India.

History



Evolution of Banking Technology

A bank must support many types of transactions with its customers, but we will examine a simple model where customers wish to open accounts, close accounts,

and add money or withdraw money from accounts. We can consider this problem at two distinct levels:

(1) The requirements for the physical infrastructure and workflow process that the bank uses in its interactions with its customers, and

(2) the requirements for the database system that manages the accounts

The typical customer opens and closes accounts far less often than accessing the account. Customers are willing to spend many minutes during the process of opening or closing the account, but are typically not willing to wait more than a brief time for individual account transactions such as a deposit or withdrawal.

These observations can be considered as informal specifications for the **time constraints** on the problem.

It is common practice for banks to provide two tiers of service. Human tellers or automated teller machines (**ATMs)** support customer access to account balances and updates such as deposits and withdrawals.

Special service representatives are typically provided (during restricted hours) to handle opening and closing accounts. Teller and ATM transactions are expected to take little time. Opening or closing an account can take much longer.

From a database perspective, we see that ATM transactions do not modify the database significantly. For simplicity, assume that if money is added or removed, this transaction simply changes the value stored in an account record.

Adding a new account to the database is allowed to take several minutes. Deleting an account need have no time constraint, because from the customer's point of view all that matters is that all the money be returned (equivalent to a withdrawal). It can be removed at some time period.

When considering the choice of data structure to use in the database system that manages customer accounts, we see that a data structure that has little concern for the cost of deletion, but is highly efficient for search and moderately efficient for insertion, should meet the resource constraints imposed by this problem.

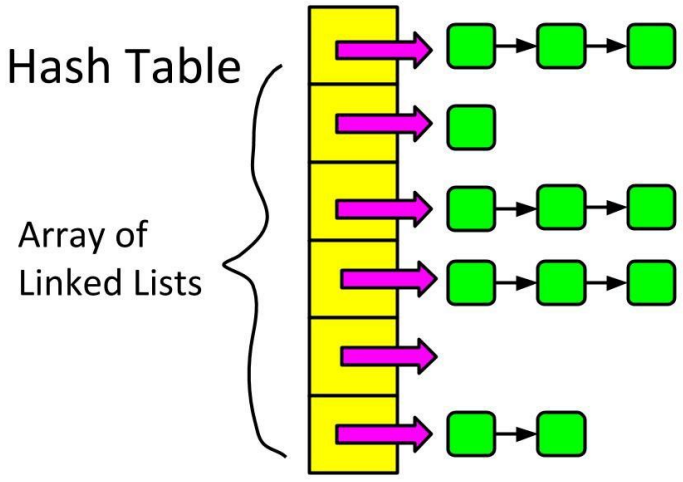
Records are accessible by unique account number. One data structure that meets these requirements is the **hashtable**. Hash tables allow for extremely fast **exact-match search.** A record can be modified quickly when the modification does not affect its space requirements. Hash tables also support efficient insertion of new records.

While deletions can also be supported efficiently, too many deletions lead to some degradation in performance for the remaining operations. However, the hash table can be reorganized periodically to restore the system to peak efficiency. Such reorganization can occur offline so as not to affect ATM transactions.

Now,we will see the hash tables pseudocode:

Separate chaining is one of the most commonly used collision resolution techniques. It is usually implemented using linked lists. In separate chaining, each element of the hash table is a linked list.

To store an element in the hash table you must insert it into a specific linked list. If there is any collision (i.e. two different elements have the same hash value) then store both the elements in the same linked list.



Hash function

A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table.

The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.

The hash function will compute the same index for all the strings and the strings will be stored in the hash table in the following format. As the index of all the strings is the same, you can create a list on that index and insert all the strings in that list.

```
vector <string> hashTable[20];
 int hashTableSize=20;
```

```
Pseudocode:
1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find the hash key.
4. If chain[key] == NULL
        Make chain[key] points to the key node.
5. Otherwise(collision),
        Insert the key node at the end of the chain[key].
```

Code in c for Insertion using separate chaining

```
void insert(string s)
{
 // Compute the index using Hash Function
        int index = hashFunc(s);
        // Insert the element in the linked list at the particular index
        hashTable[index].push_back(s);
    }
```

Insert using linear probing

```
void insert(string s)
    {
        //Compute the index using the hash function
        int index = hashFunc(s);
        //Search for an unused slot and if the index will exceed the hashTableSize
then roll back
        while(hashTable[index] != "")
            index = (index + 1) % hashTableSize;
        hashTable[index] = s;
    }
```

Search

Each slot stores a key/value pair. As you're searching through each slot, check whether the key is equal to the key you're searching for. Stop searching and return the value when you find an equal key.

With **separate chaining**, you can do a linear search through the list, checking the key against each key in the list.

Pseudocode:

```
Search()

1. Get the value
2. Compute the hash key.
3. Search the value in the entire chain. i.e. chain[key].
4. If found, print "Search Found"
5. Otherwise, print "Search Not Found"
```

Code in c using separate chaining

```
void search(string s)
   {
      //Compute the index by using the hash function
      int index = hashFunc(s);
      //Search the linked list at that specific index
      for(int i = 0;i < hashTable[index].size();i++)
      {
         if(hashTable[index][i] == s)
         {
            cout << s << " is found!" << endl;
            return;
         }
      }
      cout << s << " is not found!" << endl;
   }
```

Search using linear probing

```
void search(string s)
   {
      //Compute the index using the hash function
      int index = hashFunc(s);
       //Search for an unused slot and if the index will exceed the hashTableSize
then roll back
```

```
    while(hashTable[index] != s and hashTable[index] != "")
        index = (index + 1) % hashTableSize;
    //Check if the element is present in the hash table
    if(hashTable[index] == s)
        cout << s << " is found!" << endl;
    else
        cout << s << " is not found!" << endl;
}
```

Delete in hash tables

An easy technique is to delete is

```
Pseudocode:
Find and remove the desired element using search function
We have its index so put the -1 in that index
Then the data will be deleted
```

```
void delete(string s)
  {
            search(s) // find the removing data
      for(int i = 0;i < hashTable[index].size();i++)
```

```
    {
        if(hashTable[index][i] == -1)
        {
            cout << s <<"deleted" << endl;
            return;
        }
    }
    cout << s << "given data not found!" << endl;
}
```

Time complexity in hash tables

Hash tables are with complexity of O(1) average case complexity,however it suffers from O(n) worst case time complexity.Hash tables suffer from O(n) worst time complexity due to two reasons:

1. If too many elements were hashed into the same key: looking inside this key may take O(n) time.

2. Once a hash table has passed its load balance,it has to create a new bigger table, and re-insert each element to the table.

Concurrency in Banking:

We will go through some functions in banking and see a few pseudocodes how the concurrency concept solves the problem.

Let's list what operations a bank has:

- deposit(amount)
- withdraw(amount)
- getbalance(amount)
- setbalance(amount)

The withdraw function can be written as:

```
Pseudocode:
Withdraw(int amount)
        double b=getbalance()
        if (amount > b)
                invalid_argument() // no balance
        else
                setBalance(b - amount)
```

Suppose let’s visualise a problem.

Suppose we have a bank account x with a balance of Rs.1500. Suppose further that thread T1 calls x.withdrawal(1000) and thread T2 calls x.withdrawal(1000) right afterwards.

These two transactions are attempting to happen on the same account, and what SHOULD happen is that one of the transactions succeeds in withdrawing 1000, and the other throws an exception because the remaining balance of 500 is insufficient to satisfy the withdrawal.

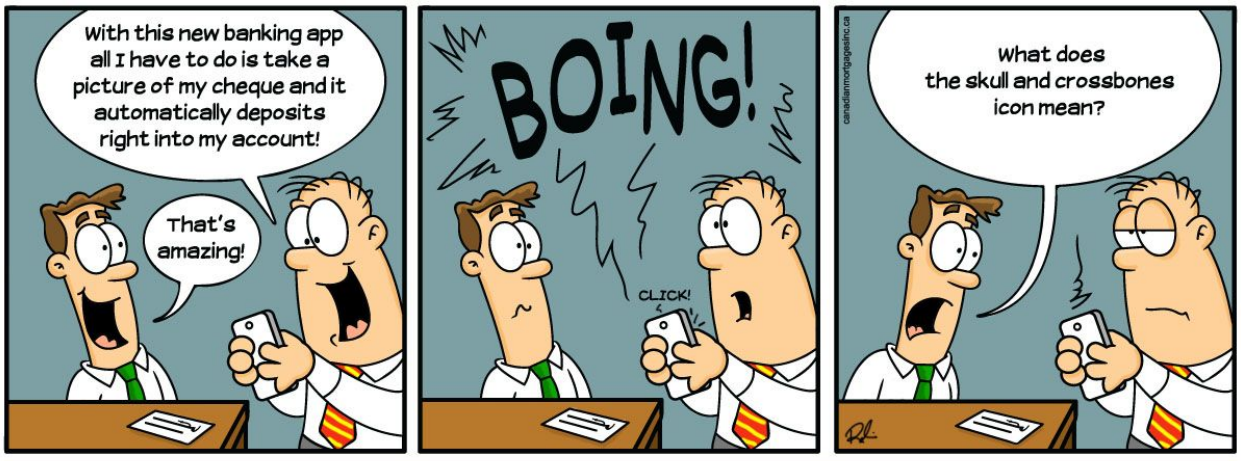
```
Thread T1
while(busy){}
busy =true
double  b=getBalance()
if(amount>b)
        Invalid argument()
Else
        setbalance(b-amount)

Thread T2
b=getbalance()
if(amount>b)
```

```
        Invalid argument()
Else
        setbalance(b-amount)
```

We still have a problem in threads. We can solve this problem by using **mutex locks**. A lock is an object that can be locked and unlocked atomically. If you have called lock() successfully, you can be sure that NO ONE ELSE owns the lock.

```
Mutex m
withdraw(double amount) {
    m.lock();
    if (getBalance() > b)
        invalid_argument()
    setBalance(getBalance() - amount);
    m.unlock()
Do the same for all the functions.then we can get rid of this problem.

setBalance(double amount)
 m.lock();
 setBalanceUnderLock(amount);
 m.unlock()

setBalanceUnderLock(double amount)
 Balance = amount
```

Small detour!! :)



13.Stock Market

A stock market, equity market or share market is the aggregation of buyers and sellers of stocks (also called shares), which represent ownership claims on businesses; these may include *securities* listed on a public stock exchange, as well as stock that is only traded privately, such as shares of private companies which are sold to investors through equity crowdfunding platforms. Investment in the stock market is most often done via stock brokerages and electronic trading platforms. Investment is usually made with an investment strategy in mind.

Stock Exchange:

A stock exchange is an exchange (or bourse) where stockbrokers and traders can buy and sell shares (equity stock), bonds, and other securities. Many large companies have their stocks listed on a stock exchange. This makes the stock more liquid and thus more attractive to many investors. The exchange may also act as a guarantor of settlement. These and other stocks may also be traded "over the counter" (OTC), that is, through a dealer. Some large companies will have their stock listed on more than one exchange in different countries, so as to attract international investors.

Shares:

Shares are units of equity ownership interest in a corporation that exist as a financial asset providing for an equal distribution in any residual profits, if any are declared, in the form of dividends. Shareholders may also enjoy capital gains if the value of the company rises.

In simpler terms - A **share** is the single smallest denomination of a company's **stock**. So if you're diving up **stock** and referring to specific characteristics, the proper word to use is **shares**. Technically speaking, **shares** represent units of **stock**.

Difference between Shares and Stocks:

Out of the two, "**stocks**" is the more general, generic term. It is often used to describe a slice of ownership of one or more companies. In contrast, in common parlance, "**shares**" has a more specific meaning: It often refers to the ownership of a particular company.

There are actually five types of stocks namely:

1. Common stocks:

- **Common stock** is a security that represents ownership in a corporation. In a liquidation, **common** stockholders receive whatever assets remain after creditors, bondholders, and preferred stockholders are paid.

2. Preferred stocks

- **Preferred stock** (also called **preferred shares**, **preference shares** or simply preferreds) is a form of **stock** which may have any combination of features not possessed by common **stock** including properties of both an **equity** and a debt instrument, and is generally considered a hybrid instrument.

3. Growth stocks

- A **growth stock** is any share in a company that is anticipated to grow at a rate significantly above the average **growth** for the market. ... This is because the issuers of **growth stocks** are usually companies that want to reinvest any earnings they accrue in order to accelerate **growth** in the short term.

4. Value stocks

- A **value stock** is a **stock** with a price that appears low relative to the company's financial performance, as measured by such fundamentals as the company's revenue, dividends, yield, earnings and profit margins.

5. Income stocks

- An **income stock** is an equity security that pays regular, often steadily increasing dividends. ... While there is no specific breakpoint for classification, most **income stocks** have lower levels of volatility than the overall **stock** market, and offer higher-than-market **dividend** yields.

What Is a Stock Dividend?

A **stock dividend** is a **dividend** payment to shareholders that is made in shares rather than as cash. The **stock dividend** has the advantage of rewarding shareholders without reducing the company's cash balance, although it can dilute earnings per share.

Quick Que's:

What is the 3 day rule in stocks?

The 'Three Day Rule' tells investors and **stock** traders to wait a full **three days** before buying a **stock** that has been slammed due to negative news. By using this **rule**, investors will find their profit expansion and losses contract.

How to analyze stocks?

A common method to **analyze a stock** is studying its price-to-earnings ratio. You calculate the P/E ratio by dividing the **stock's** market value per share by its earnings per share. To determine the value of a **stock**, investors compare a **stock's** P/E ratio to those of its competitors and industry standards.

Now let's dive back,

History and Origin of trading:

Algorithmic Trading:

Algorithmic trading is a process for executing orders utilizing automated and pre-programmed **trading** instructions to account for variables such as price, timing and volume. An **algorithm** is a set of directions for solving a problem. Computer **algorithms** send small portions of the full order to the market over time.

Fun Fact:

Foreign exchange markets also have active **algorithmic trading**, measured at about 80% of orders in 2016 (up from about 25% of orders in 2006).

Is Algorithmic trading profitable?

Yes! **Algorithmic trading** is profitable, provided that **you** get a couple of things right. These things include proper backtesting and validation methods, as well as correct risk management techniques. Unfortunately, many never get this completely right, and therefore end up losing **money**.

Who uses this Algorithmic trading?

Algorithmic trading is mainly used by institutional investors and big brokerage houses to cut down on costs associated with **trading**. According to research, **algorithmic trading** is especially beneficial for large order sizes that may comprise as much as 10% of overall **trading** volume.

Do Banks also use algorithmic trading?

Banks regularly **use algorithmic trading** strategies and have high-frequency **trading** firms as clients. As these markets become more interconnected due to **algorithmic trading**, the effects of errors or attacks could amplify risk in the financial system, the report said.

Small detour!! :)

Odds algorithm

This is an algorithm that can solve a specific type of problem. A template of the problem is called the secretary problem that is stated below.

A manager needs to hire a secretary and has n applicants, who are interviewed one at a time. The manager's only concern is to maximize the probability of picking the best of the n candidates. The decision of whether to accept or reject an applicant has to be made straight after the interview. What is the optimal strategy?
The odds-strategy is the rule to observe the events one after the other and to stop on the first interesting event from index s onwards (if any), where s is the stopping threshold of output a.

The odds-algorithm sums up the odds in reverse order

$$r_n+r_n-1+r_n-2+\cdots,$$

until this sum reaches or exceeds the value 1 for the first time. If this happens at index s, it saves s and the corresponding sum

$$R_s=r_n+r_n-1+r_n-2+\cdots+r_s.$$

If the sum of the odds does not reach 1, it sets $s = 1$. At the same time it computes

$$Q_s=q_nq_n-1\cdots q_s.$$

The output is

1. s, the stopping threshold

2. $w=Q_sR_s$, the win probability.

The odds-theorem states that

1. The odds-strategy is *optimal*, that is, it maximizes the probability of stopping on the last 1.
2. The win probability of the odds-strategy equals $w=Q_sR_s$
3. If $R_s \geq 1$, the win probability w is always at least $1/e=0.368\ldots$, and this lower bound is *best possible*.
4.

Reference: Bruss, F. & Louchard, Guy. (2009). The odds algorithm based on sequential updating and its performance. Advances in Applied Probability - ADVAN APPL PROBAB. 41. 10.1239/aap/1240319579.

14. Econometrics

What is econometrics?

As the name suggests it is interpreted that econometrics means "economic measurement". Although measurement is an important part of econometrics, the scope of econometrics is much broader.

Linear regression

An interesting thing to know is that it is one of the easiest algorithms in machine learning (ML). It is a statistical module used to show the relationship between two variables with a linear equation.

Reading the heading and the above lines many of you get an important doudt, What does regression actually mean?

What is regression?

Regression analysis is a form of modelling technique used to predict the relationship between a dependent and independent variable.

There are two types of regressions:

1. Linear

2. Logistic

Where do we use it?

1. Evaluating trends and estimate the sales

2. Analyzing the impact of price changes

3. Assessment of risk in financial services and insurance domain.

What is the use of it?

There are three major uses:

1. Determining the strength of predictors

2. Forecasting an effect

3. Trend forecasting

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2.x$$

While training the model we are given :

x: Input training data (univariate – one input variable(parameter))

y: Labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.

θ1: Intercept

θ2: Coefficient of x

Finding the best θ1 and θ2 values:

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

The values of θ1 and θ2 are decided at which the sum of distances minimizes from all known points to the observed points on the lines or simply variance minimizes.

In mathematical terms it is Root Mean Squared Error (RMSE) between predicted y value (pred) and true y value (y).

Algo in Python:

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
```

```
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return(b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()
```

```
def main():
    # observations
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

Logistic Regression

In statistics, the logistic model is used to model the probability of a class or event in the form pass/fail or win/lose or to be straight forward the events that have probabilities either 1 or 0.

Logistic regression models data using the Sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}}$$

The dataset has ‘p’ feature variables and ‘n’ observations. The feature matrix is represented as:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

Here, x_{ij} denotes the values of j^{th} feature for i^{th} observation.

Here, we are keeping the convention of letting $x_{i0} = 1$. (Keep reading, you will understand the logic in a few moments).

The i^{th} observation x_i is represented as follows:

$$x_i = \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ . \\ . \\ x_{ip} \end{bmatrix}$$

$h(x_i)$ is represented as the predicted response of i^{th} observation.

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}$$

Algo in Python:

```
import csv
import numpy as np
import matplotlib.pyplot as plt

def loadCSV(filename):
    '''
    function to load dataset
    '''
    with open(filename,"r") as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for i in range(len(dataset)):
            dataset[i] = [float(x) for x in dataset[i]]
    return np.array(dataset)

def normalize(X):
    '''
    function to normalize feature matrix, X
    '''
    mins = np.min(X, axis = 0)
    maxs = np.max(X, axis = 0)
```

```
    rng = maxs - mins
    norm_X = 1 - ((maxs - X)/rng)
    return norm_X

def logistic_func(beta, X):
    '''
    logistic(sigmoid) function
    '''
    return 1.0/(1 + np.exp(-np.dot(X, beta.T)))

def log_gradient(beta, X, y):
    '''
    logistic gradient function
    '''
    first_calc = logistic_func(beta, X) - y.reshape(X.shape[0], -1)
    final_calc = np.dot(first_calc.T, X)
    return final_calc

def cost_func(beta, X, y):
    '''
    cost function, J
    '''
    log_func_v = logistic_func(beta, X)
    y = np.squeeze(y)
    step1 = y * np.log(log_func_v)
    step2 = (1 - y) * np.log(1 - log_func_v)
```

```
    final = -step1 - step2
    return np.mean(final)

def grad_desc(X, y, beta, lr=.01, converge_change=.001):
    '''
    gradient descent function
    '''
    cost = cost_func(beta, X, y)
    change_cost = 1
    num_iter = 1

    while(change_cost > converge_change):
        old_cost = cost
        beta = beta - (lr * log_gradient(beta, X, y))
        cost = cost_func(beta, X, y)
        change_cost = old_cost - cost
        num_iter += 1
    return beta, num_iter

def pred_values(beta, X):
    '''
    function to predict labels
    '''
    pred_prob = logistic_func(beta, X)
    pred_value = np.where(pred_prob >= .5, 1, 0)
    return np.squeeze(pred_value)
```

```
def plot_reg(X, y, beta):
    '''
    function to plot decision boundary
    '''
    # labelled observations
    x_0 = X[np.where(y == 0.0)]
    x_1 = X[np.where(y == 1.0)]

    # plotting points with diff color for diff label
    plt.scatter([x_0[:, 1]], [x_0[:, 2]], c='b', label='y = 0')
    plt.scatter([x_1[:, 1]], [x_1[:, 2]], c='r', label='y = 1')

    # plotting decision boundary
    x1 = np.arange(0, 1, 0.1)
    x2 = -(beta[0,0] + beta[0,1]*x1)/beta[0,2]
    plt.plot(x1, x2, c='k', label='reg line')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.show()


if __name__ == "__main__":
    # load the dataset
```

```
dataset = loadCSV('dataset1.csv')

# normalizing feature matrix
X = normalize(dataset[:, :-1])

# stacking columns with all ones in feature matrix
X = np.hstack((np.matrix(np.ones(X.shape[0])).T, X))

# response vector
y = dataset[:, -1]

# initial beta values
beta = np.matrix(np.zeros(X.shape[1]))

# beta values after running gradient descent
beta, num_iter = grad_desc(X, y, beta)

# estimated beta values and number of iterations
print("Estimated regression coefficients:", beta)
print("No. of iterations:", num_iter)

# predicted labels
y_pred = pred_values(beta, X)

# number of correctly predicted labels
print("Correctly predicted labels:", np.sum(y == y_pred))
```

```
# plotting regression line
plot_reg(X, y, beta)
```

Small detour!! :)

A seemingly impossible problem that actually has a very simple and elegant solution. And once again, finding the solution has to do with relying on the overall properties of the problem, not with any specific data within it.

Given only:

- ➔ An arbitrary linked list.
- ➔ A `list.next(node)` method that returns the next node in the list (or null if there is no next node, or the first node if the given node is null).
- ➔ A large but finite amount of memory.

➔ Determine whether:

- ➔ The list has a last node.
- ➔ The list contains a loop.

➔ There are many approaches to this, but they all rely on storing previously visited nodes so that the loop can be detected. And since the input can be arbitrarily large, there will always be a test case that will require more memory than available.

- ➔ If you try walking along the list, you will eventually enter the loop and keep going around in a circle. But you have no way of knowing at which node you entered the loop, nor any way of marking any nodes as having been visited already.
- ➔ The key to this problem is not in thinking about the list itself, but of what it means to have a list with a loop.
- ➔ Now suppose you are in a foot race with someone that is faster than you. They will slowly keep getting farther and farther away from you. Eventually, you will either meet them again at the finish line, or they will continue around the circuit and lap you. This provides the solution to the original problem.
- ➔ It has an official name, referred to as the "Hare and Tortoise" solution. Start with a "hare" and a "tortoise", each initialized to the starting node. At each iteration:
  - ➔ Move the hare ahead two nodes.
  - ➔ Move the tortoise ahead one node.
- ➔ Eventually one of these results will happen:
  - ➔ The hare will reach the last node of the list, meaning the list has an end.
  - ➔ The hare and the tortoise will end up on the same node, meaning there is a loop in the list.
- ➔ Simple, elegant, efficient, and it is based on the meaning of the data, not on the data itself.

15. Smart cities

A smart city is an urban area that uses different types of electronic methods and sensors to collect data and use that to improve the operations across the city. Nowadays where technology is ruling the world, cities using these technologies are becoming smart. Thus called smart cities .

Smart city technology

Smart cities use a combination of the internet of things (IOT) devices, software solutions, user interfaces (UI) and communication networks to manage resources and function efficiently.

The IoT is a network of connected devices -- such as vehicles, sensors or home appliances -- that can communicate and exchange data. Data collected and delivered by the IoT sensors and devices is stored in the cloud or on servers.
A firewall security system is also necessary for the protection, monitoring and control of network traffic within a computing system. Firewalls ensure that the data

constantly being transmitted within a smart city network is secure by preventing any unauthorized access to the IoT network or city data.

It also uses these technologies to improve the city's functionality.

- application programming interfaces (APIs)
- artificial intelligence (AI)
- Cloud computing
- Dashboards
- machine learning (ML)
- machine to machine (M2M)
- Mesh network

Features of a smart city

- Dynamic vehicle rerouting
    - A basic *routing problem* looks for the best path for a delivery *vehicle* around a set of customers
- Bus arrival times
    - bus arrival time is calculated under traffic conditions using GPS data from buses. The analysis was using a model based algorithm and uses Kalman filtering method for the prediction of travel time.
- Available parking

    - Available parking uses a sensor information to know the info of parking slots and uses that data to implement the available parking system.
- Crime monitoring
    - Crime Prediction & Monitoring uses various visualization techniques to analyze the data in a better way. This framework is implemented in a GUI based tool using R programming and its various libraries.
- Police patrol routing (to empty houses identified through zero energy use)
    - Empty houses that are identified through zero energy use that are obtained from clouds are watched by police patrol to prevent roborries.
- Reporting erratic driving
    - Speedometers which senses the speed of the vehicle that is going in a road, checks its speed and if it's greater than a limit, they are fined or immediate action is taken by the police.
- Machine learning applications
    - So many things like image recognition, speech recognition etc use machine learning.
- Being watched
    - In smart cities where no one can’t escape from being watched from cc camara’s , it helps to know what is done when an incident occurs and it also gives security to people because if something happens to them, then sensors immediately respond and give signals to respective members or officers to rescue or help people .

Algorithms in Smart city development

- Cloud computing.
    - Most important part of a smart city is to store data and use it for better purposes. Cloud computing includes storing data in the cloud and sending it to authorised devices.
    - Storage algorithms in Cloud computing.
    - It includes storing,accessing and processing data in cloud environments.
    - Security algorithms in cloud computing.
    - Prevents data from being attacked or keeps it safe, access only to authorised ones.
    - Some of the security algorithms are Public Key / Asymmetric Algorithms,Signature algorithms(RSA,DH) which verifies users signature,hash algorithms(MD5.SHA) to compress data and into fixed sizes etc.

- Encryption algorithms
    - The *data* must be encrypted before *sending* them *to the* cloud. It *uses* the. symmetric encryption *algorithm* AES *in* order to benefit *from* its advantages *in* terms.

AI

- All the services that include vehicle parking,security systems etc use AI.

- Advance Security Camera & Surveillance System

  AI-enabled cameras and sensors can keep an eye on the surroundings to enhance the security level in the city's neighborhoods. Such cameras can recognize the people and their faces or track the unusual activities done by them in restricted areas.

- Vehicle Parking and Traffic Management System

  Using the road surface sensors or CCTV cameras incorporated into parking spots allow cities to create real-time parking and traffic maps, helping drivers to save their time by avoiding waiting to find an empty space to move smoothly or be in traffic.

- Smart Waste and Disposal Management System

  AI-enabled installing sensors on waste bins makes the waste collection more efficient. Authorities receive notifications when the bins are about to be filled and ensure reducing operational costs by eliminating unnecessary pickups, providing dynamic collection routes, and schedules for optimization of waste management.

- Optimized Energy & Water Management

  The power generating grid and smart water management help to produce energy with less pollution and to get clean drinking water to keep our environment clean.

- Advance Security and Safety in the Public

  The safety of the people in the cities is at topmost priority in such cities. The AI-enabled development of smart cities allows municipalities to better monitor their citizen's thanks to CCTV cameras with facial recognition.

Machine Learning

- Machine learning is also a part of the AI
- Facial recognition
  - Your face is detected and a picture of it is captured from a photo or video.the software reads your facial features. Key factors that play a role in the detection process can differ from each other based on what mapping technique the database and algorithm use.The algorithm verifies your face by encoding it into a facial signature.
- Speech recognition
  - A microphone records a person's **voice** and the hardware converts the signal from analog sound waves to digital audio. The audio data is then processed by software, which interprets the sound as individual words.
  - The algorithms used in this form of technology include PLP features, Viterbi search, deep neural networks, discrimination training, WFST framework, etc.

Small detour!! :)

There is a conference related to algorithms named the The Fun with Algorithms Conference . The Fun with Algorithms conference is dedicated to the use, design, and analysis of algorithms and data structures, focusing on results that provide amusing, witty but nonetheless original and scientifically profound contributions to the area. The covered topics include:

- FUN with biological algorithms
- FUN with cryptographic algorithms
- FUN with game-theoretic algorithms
- FUN with graph algorithms
- FUN with optimization algorithms
- FUN with robotics algorithms
- FUN with string algorithms
- FUN with combinatorial algorithms
- FUN with distributed algorithms
- FUN with geometrical algorithms
- FUN with mobile algorithms
- FUN with parallel algorithms
- FUN with space-conscious algorithms
- FUN with visualization of algorithms
- FUN with web algorithms
- FUN with machine learning algorithms

16. Games

Introduction

Video games employ algorithms to respond intelligently to players' inputs. From the predictable patterns of the alien ships in "Space Invaders", to the much more responsive ghosts in "Pac Man", the earliest video games used mathematical processes to mimic the behaviour of thinking beings. Over time, these technologies have evolved to become extremely sophisticated and produce incredibly complex behaviours. Just like the data manipulation techniques we've seen previously, at their core are some surprisingly simple tricks of geometry.

The creation of a video game is no easy task for anyone to undertake. When it comes to programming, a lot of knowledge needs to be obtained to create an efficient and solid product. The game industry is a very competitive business, where games are defined heavily by the products that came before them.
This combination is the key to creating real-time simulations that push the envelope on what is considered cutting-edge.

Game physics

Well, We all enjoy the hypes and collisions and the anti-gravity gimmicks in games. But at some point, one wonders how much actual physics is involved in these games. Well, since video games are just simulations, physics can be both used or just ignored according to the needs.

Many games rely entirely on physics simulation to be fun. That means these games require a stable simulation that will not break or slow down, and this is usually not trivial to achieve.

Physics in games can make explosions and enemy deaths more spectacular and entertaining than canned animations, and it's great for things like projectile weapons and fire propagation. But physics can do so much more than provide eye candy--it can be a major tool for game design.

We'll see situations of both physics being an advantage and disadvantage and try to understand those.

- Collisions

    Collisions are the most common physics phenomenon that's used in gaming. There can be broadly two types of these simulations: rigid body and free body dynamics.

In the rigid body collision idea, the game makers deal more with the interaction of the object whose shape or size is usually unchangeable and most of its dynamics depend on realistic motion, collisions, friction, and resting contact between multiple bodies.

Soft body simulations are based on objects that are more deformable and these simulations are usually more realistic in their interaction than rigid body simulation.

These collisions are to be visualized with an algorithm based coding. The objects have an invisible box around them that's usually used to identify if that object has been in contact or “collided” with another object. More precise this box is , more precise and proper is our collision simulation.

Often the impact of these collisions is dynamically simulated and changes with the properties of objects that we are going to learn about beow.

- Particle simulation

In particle simulation, first, a specific object is initialized usually as a vector. And an array of such particles come together to form a particle system. These particles are then assigned mass, velocity, and allowed to run in the designed loop to get more informative information like

force, acceleration, etc.

We can also simulate this force into being the effect of gravity, magnetism, electrical, mechanical. These forces can be local or globalized as intended.

As a collection of particles, these systems have center of mass, and other properties assigned to a system of particles.

The group of particles could also have a rotating movement besides linear motion. This rotating motion brings in angular momentum and torque that expands the game maker's horizon to include even more complicated dynamics and simultaneously making the game visualizations look realistic to the players.

These particles once they come together could have 2D or 3D properties as intended by the maker. Both 2D and 3D dynamics can be visualized using calculus and many more physics laws that can very well include many different types of motion and interaction.

These particle systems can be of many types like smoke, fire, water as rain or snow, big rocks, generally it's an object moving in a precalculated motion.

- Ragdoll physics

Death is pretty awful and unfortunate. But in video games these days, the changing world sees games that involve the killing of characters in games. It could be the death of a character moving within a computer simulation or could be another player in a multiplayer game or in some cases the player's character itself.

Sloppy death simulation can make the game look weird. Without ragdoll physics, the object once died would just fall down and maybe disappear in the next frame. Since most of the objects are designed for flat surfaces, these deaths in the game occurring on any other surface would give a totally awkward look to the game.

With ragdoll physics, a gamer is able to involve the skeletal structure, a muscular nature. This enables the character that dies to behave and look just the way a realistic death occurs. The skeletal structure once simulated can make it look as if the death that occurred is just as real as the rest of the game.

Ragdolls have been implemented using Featherstone's algorithm and spring-damper contacts. An alternative approach uses constraint solvers and idealized contacts. The constrained-rigid-body approach to ragdolls is the most common.

- Projectiles

Projectiles are generally used in games to visualise bullets these type of projectiles can be of two types

The first type of projectiles is where the bullet is simulated using actual laws of physics. That bullet has speed and has a trajectory that is defined by the initial velocity and position. The movement of such bullets are usually seen and they take some time to reach the target.

The other type of bullets are where they are just locked onto a specific target. These bullets hit anything on their way and they are instantaneous. These types of bullets have no actual existence and can't be seen on the screen .These types of projectiles are called hitscans.

In most implementations of a hitscan weapon, when the player shoots a bullet, the physics engine will:Figure out the direction the gun is pointing at,Cast a ray from the muzzle of the gun until a defined range,Use raycasting to determine if the ray hit an object. (raycasting is used to render 3D environments onto 2D images).

Hitscan is simple at its core, but a lot of different modifications can be made to support other logic: If we continue the ray past the first object that it hit, we can penetrate multiple objects in a line, like the railgun in Quake, Removing the maximum range of the ray means that we can

shoot out a laser that will continue forever until we hit something, Programming certain surfaces to be reflective, to bounce bullets off .

Hitscans are pretty fast and in games that are rather more interested in slow motion techniques, this method would be a no-go.

Also, there is no travel time after you fire a bullet and hit an object. This means it's impossible to dodge a bullet if a ray is on target, even if the target is miles away.This makes it a very inefficient simulation to use in games where dodging projectiles plays an important role.

- Are bullets in games real?

  This may sound absurd but the amount of violence in video games these days has created big debates. These bullets can only damage the mental state or to the maximum the processor of the gamer if he plays too much. Besides that there is no damage to the physical world as such.

  There is an emerging idea in modern physics that says that we live in a Multiverse, and because of this limitless possibility of other planes of existence and universes beyond our own, it is possible that every video game world is in fact a portal to a smaller, temporary universe.

  So if you believe that, and you take the leap of faith required to think that video games are portals to other universes, and you accept the fact that the bullet on the screen is just a hologram on a 2D field

existing in that parallel universe, and echoed on the 2D holographic field of our own universe, then yes, “real” is arbitrary and the multiverse is crowded so, in effect, real bullets are killing real people in the video game universe.

- Disadvantages

  There are also many games and many instances where physics is not used in games directly. We have enumerated some of those reasons here.

  In games that involve blowing up things or flying cars or grazing gliders, following physics laws exactly would make the game more dull.Physics in a game should boost the fun, not eliminate it! If making your game more realistic (i.e. with more realistic physics) makes it less fun, then it’s probably not the right design move. In those games the fun part of it is conserved by tweaking a little bit. This is understandable since the computer world is pure imagination and a little bit of creativity doesn't hurt there.

  When calculating power or the amount of fuel needed to move , following physics can be another restriction. The designer might be interested in customising the amount of fuel the gamer consumes or may want to impose conditions that alter the power of a character or object drastically. These unrealistic moves don't follow laws of physics but nevertheless make the game more engaging.

A helicopter in a game, for example, isn't designed to actually take off against gravity. It's more likely that the gravity simulation in the game turns off when you "take off" in the helicopter.The rotors, which need to lift and stabilize the vehicle in real life, aren't actually countering any acting forces in the game. The programmers more likely just write code to simulate it.

Aside from these common physics implementations, games have many scientific and science fiction ideas embedded in them, It could be aliens or space time portals or even zombie apocalypse. Though the whole ideas behind these illustrations are fictitious, the underlying physics laws in them make them more realistic and engaging to a gamer.

The implementation of such complicated physics equations in games require sophisticated software to match the video game experience to the real world. These softwares are generally called the physics engines an play a crucial role in game-programming

- Physics engine

  Physics engines is a software that allows computers to create physics simulations from the real world like forces of gravity wtc and apply them to 2d or 3d created objects in games.

  These physics engines help the designers to create a life-like environment that interacts and responds to actions that mimic reality.

Even 3d rendering is done using these softwares.

These softwares also generally have customisation options that allow a game maker to tweak their physics a little so that it fits better into the game design.

These software s reduce the time taken to configure and calculate the exact equations and concepts that work in the physics side of the game. Some games might have even hundreds of characters interacting in ways that might be almost impossible to comprehend if calculated by the programmer.

To qualify as a physics engine, a software must simulate a variety of physical systems (rigid body dynamics, soft body dynamics, fluid dynamics, etc.), apply those systems to 3D objects and environments, work in tandem with other software systems to create a cohesive experience.

Key Benefits of Physics Engine Software are easing workload, empowering creativity, limiting possibility for human error

Without physics engine software, game design is bogged down by the prospect of either building an in-house solution, or using individual animations to represent physics. The former is a monumental task that would prove difficult for even a large, well funded studio. The latter is

tedious and time consuming, especially in games with hundreds if not thousands of interactive elements.

Multiplayer games

A **multiplayer game** is a **game** in which more than one person can play in the same **game** environment at the same time, either locally or online over the internet. In multiplayer games, there are at least three players partitioned into two teams, To and T1. A multiplayer game is specified by a set of positions, a relation defining the possible next moves, division of teams, and access rights of players to view or modify certain components of a game position.

Here are the few algorithms that are used in multiplayer games.

Artificial Intelligence(AI)

Artificial intelligence (AI), is intelligence demonstrated by machines, unlike the natural intelligence displayed by humans and animals. Leading AI textbooks define the field as the study of "intelligent agents": any device that perceives its

environment and takes actions that maximize its chance of successfully achieving its goals.

(AI) is used to generate responsive, adaptive or intelligent behaviors primarily in non player characters (NPCs) similar to human-like intelligence. Artificial intelligence has been an integral part of video games since their inception in the 1950s AI in video games is a distinct subfield and differs from academic AI.

It serves to improve the game-player experience rather than machine learning or decision making. During the golden age of arcade video games the idea of AI opponents was largely popularized in the form of graduated difficulty levels, distinct movement patterns, and in-game events dependent on the player's input. Modern games often implement existing techniques such as pathfinding and decision trees to guide the actions of NPCs. AI is often used in mechanisms which are not immediately visible to the user, such as data mining and procedural content generation.

Generic algorithms

Nowadays we can see bots in every multiplayer games.Bots are designed to make players feel like they are playing with another real component.

genetic algorithms for training the bots in offline multiplayer games. It makes the bot more intelligent and tough to fight with .games will be more competitive with AI. It will be boring to play a game without a tough competition. AI makes it possible in making intelligent and interesting gameplay. AI also helps in designing the graphics of the game like physics in-game, fluid simulations, fog and air simulations etc which should look real.

Examples:

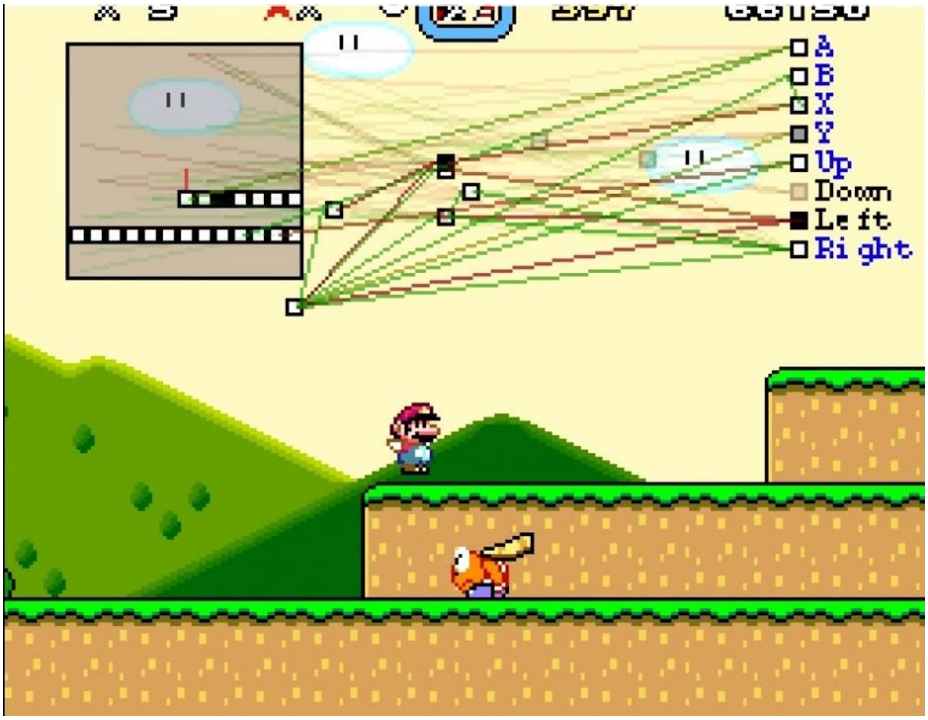
Super mario,ludo ,call of duty etc.

Strategies

A multiplayer game can be specified by a set of positions, a relation defining possible next moves,and an assignment of the rights of players to view and modify certain components of positions.
A strategy for player i defines a single next move for player i from any given sequence of previous moves ending in a position for which it is player i's turn to move. Such strategies are called pure.A strategy of player i must depend only on the visible components of previous positions for which player i made moves.

Neural networks

Neural networks can be used in a variety of different ways in computer games. They can be used to control one game agent, several game agents, or several neural networks can be used to control a single game agent. A game agent can be a non-player character or it can be used to represent the game environment.



DECISION ALGORITHMS FOR GAMES

We assume that the perfect information game G = (POS, ~-, vis, T1) has a space bound S(n) > log n. We will let POS(p0) be the set of positions in POS reachable from P0 by some sequence of moves, as defined by the relation ~-, with space bound S(n).

The win, non loss, and Markov (m(n)) outcome of any deterministic game with space bound S(n) greater or equal to log(n), can be decided in deterministic space S(n).

Decision for Games with Time and Branch Bounds

a game G has a time bound T(n) if the outcome problem can be solved in at most T(n) moves for every play sequence in the game tree. Analogously, a game G has a branch bound b if for each position p E POS : [{ptlp }- p'}[_< b. In other words, at any position in POS, there are at most b choices for the next transition. Consider a game G of incomplete information with time bound Tin). In order to decide the win outcome of G, we only need to choose each strategy a for Team T1, and verify that Team T1 wins for any play 7r induced by a. Observe that each such play has at most T(n) moves.

function Decide-Existential(p, a): returns Boolean

Input: Position p E POS(po), and Alternation bound a

Output: TRUE if and only if T1 has a winning strategy from p with an alternation bound of a and a space bound S(n).

Spatial Partition

Spatial partition is one of the optimisation techniques used in gaming.This technique provides us to locate objects efficiently by storing them in a data structure organized by their positions.

When making a game, you will often have a need to find enemies that are close to

the player. The common way is to store all enemies in a list, search through the list of all enemies, and for each enemy calculate the distance to the player.
This is the code in C to find the closest enemy.

```
GameObject FindClosestEnemySlow(GameObject soldier)
{
	GameObject closestEnemy = null;

	float bestDistSqr = Mathf.Infinity;

	//Loop through all enemies
	for (int i = 0; i < enemySoldiers.Count; i++)
	{
		//The distance sqr between the soldier and this enemy
		float distSqr = (soldier.transform.position -
enemySoldiers[i].transform.position).sqrMagnitude;

		//If this distance is better than the previous best distance, then
we have found an enemy that's closer
		if (distSqr < bestDistSqr)
		{
			bestDistSqr = distSqr;

			closestEnemy = enemySoldiers[i];
		}
	}
```

```
        return closestEnemy;
}
```

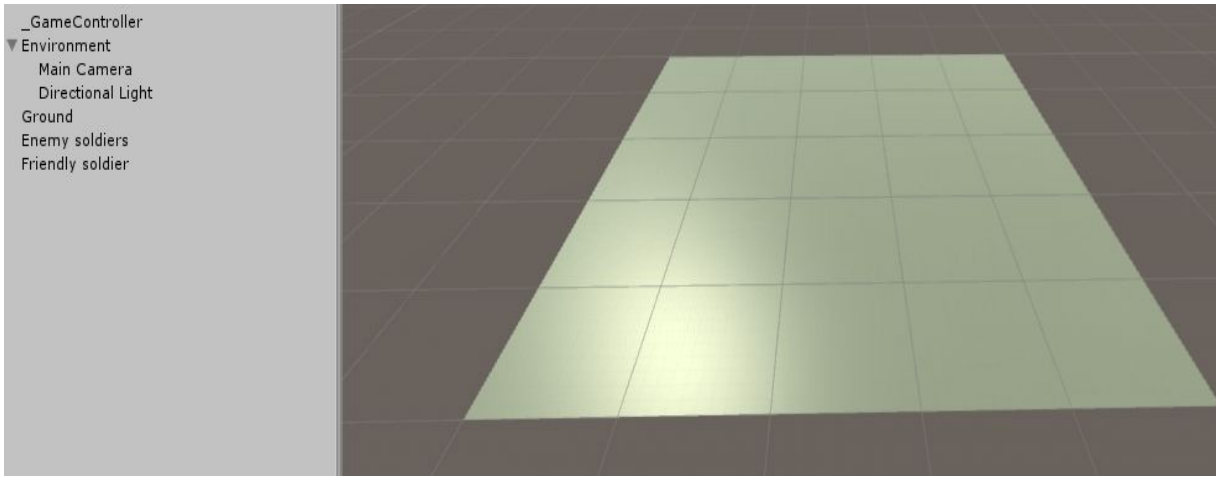
A better way to store their positions is using geometryThis data structure lets you **efficiently query for objects at or near a location**. When an object's position changes, **update the spatial data structure** so that it can continue to find the object.A common way is to divide the area into a 2D grid and then you associate each cell with the enemies in it. Then to find the closest enemy, you have to figure out which cell you are standing in to get the enemies in that cell, and then you just figure out which of those enemies in the cell is the closest enemy.

Spatial partition uses additional memory as it stores all the moving objects in the games.Like many optimizations, it trades memory for speed. If you're shorter on memory than you are on clock cycles, that may be a losing proposition.

In Unity

The idea here is that we will have spheres chasing cubes. The spheres are friendly while the cubes are called enemies. So in Unity you should add an empty gameobject called GameController, and two empty gameobjects called Enemy soldiers and Friendly soldiers to which we will parent the cubes and spheres to get a clean workspace. Also add a ground plane. The reason is that we have to translate

coordinates of the cubes and spheres to positions in the grid, and it's easier if everything begins at (0,0,0).



Does the partitioning depend on the set of objects?

Many partition techniques are adaptable — they pick partition boundaries based on the actual set of objects and where they are in the world.

The goal is have a *balanced* partitioning where each region has roughly the same number of objects in order to get the best performance.Consider in our grid example if all of the units were clustered in one corner of the battlefield. They'd all be in the same cell, and our code for finding attacks would regress right back to the original $O(n^2)$ problem that we're trying to solve.

If the partitioning is object-independent

- Objects can be added incrementally. Adding an object means finding the right partition and dropping it in, so you can do this one at a time without any performance issues.

- Objects can be moved quickly. With fixed partitions, moving a unit means removing it from one and adding it to another. If the partition boundaries themselves change based on the set of objects, then moving one can cause a boundary to move, which can in turn cause lots of other objects to need to be moved to different partitions.

- The partitions can be imbalanced. Of course, the downside of this rigidity is that you have less control over your partitions being evenly distributed. If objects clump together, you get worse performance there while wasting memory in the empty areas.

If the partitioning adapts to the set of objects

Spatial partitions like BSPs and k-d trees split the world recursively so that each half contains about the same number of objects. To do this, you have to count how many objects are on each side when selecting the planes you partition along. Bounding volume hierarchies are another type of spatial partition that optimizes for the specific set of objects in the world.

- You can ensure the partitions are balanced. This gives not just good performance, but consistent performance: if each partition has the same

number of objects, you ensure that all queries in the world will take about the same amount of time. When you need to maintain a stable frame rate, this consistency may be more important than raw performance.

It's more efficient to partition an entire set of objects at once. When the set of objects affects where boundaries are, it's best to have all of the objects up front before you partition them. This is why these kinds of partitions are more frequently used for art and static geometry that stays fixed during the game.

spatial partition as the place where the objects in your game live, or you can consider it just a secondary cache to make look-up faster while also having another collection that directly holds the list of objects.

- **If it is the only place objects are stored**

  It avoids the memory overhead and complexity of two collections. Of course, it's always cheaper to store something once instead of twice. Also, if you have two collections, you have to make sure to keep them in sync. Every time an object is created or destroyed, it has to be added or removed from both.

- **If there is another collection for the objects**

  Traversing all objects is faster. If the objects in question are "live" and have some processing they need to do, you may find yourself frequently needing to visit every object regardless of its location. Imagine if, in our earlier example, most of the cells were empty. Having to walk the full grid of cells to find the non-empty ones can be a waste of time. A second collection that just stores the objects gives you a way to walk all them directly. You have

two data structures, one optimized for each use case.

Board Game Algorithms

Chess

We know chess is one of a popular game which is useful to improve our IQ level. Two people are needed to play chess. But nowadays people are having no time. Then the need of playing chess with only one person arrived. You will wonder how a single person plays chess. It's possible if we can replace another person with a computer which knows how to play. Lets know some algos which are useful in developing chess.

Static Evaluation function

First we define a function which evaluates the position of the game statically as opposed to dynamically. Search algorithms evaluate dynamically. It means a position is evaluated statistically without considering future positions which arrive after some moves.

SEF just counts the value of all white an dblack pieces on the board at present(with say pawn=1, knight=3,bishop=3,rook=5,queen=9) and subtract the

total value of black pieces from white pieces and return the result. If it returns a positive score then white has more power and positive then black has more power.

Using the above function, we can generate the next move which is useful for us. Here we need another function which gives all possible moves from the present position. SEF will be applied to every move and we choose the best move which is in our favour.

This type of algorithm wouldn't play well. For example the algorithms would capture a defended rook with its queen because it would not see that the queen would be recaptured on the next turn, losing queen for a rook!.

British Museum Algorithm

In the above algorithm we just evaluated the score of just the next positions. Here we generate the next positon's position's ….. position all the way to end the game. We evaluate all of them to find the best move. But the issue is "Can we evaluate scores for all possible positions?". Is it practically possible ? no. This algorithm is not going to work.

Min Max algorithm

It goes down the game tree until some depth and evaluates scores of leaf nodes. Then it goes up assigning values to intermediate nodes. Max value of child nodes will be assigned if it is algo chance to move otherwise min of children will be assigned.

Alpha Beta Algorithm

It is the advancement of the Min max algorithm. It cuts the unnecessary parts of the search tree. That's why it is more efficient than the min max algorithm.

Collision Detection

Collision Detection is an important part in game development.

2D collision detection

Algorithms to detect collision in 2D games depend on the type of shapes that can collide (e.g. Rectangle to Rectangle, Rectangle to Circle, Circle to Circle).

Axis-Aligned Bounding Box

One of the simpler forms of collision detection is between two rectangles that are axis aligned — meaning no rotation. The algorithm works by ensuring there is no gap between any of the 4 sides of the rectangles. Any gap means a collision does not exist.

```
rect1 = {'x': 5, 'y': 5, 'width': 50, 'height': 50}
rect2 = {'x': 20, 'y': 10, 'width': 10, 'height': 10}

if (rect1['x'] < rect2['x'] + rect2['width'] and
  rect1['x'] + rect1['width'] > rect2['x'] and
  rect1['y'] < rect2['y'] + rect2['height'] and
  rect1['y'] + rect1['height'] > rect2['y']) {
   // collision detected!
}
```

```
// filling in the values =>

if (5 < 30 and
   55 > 20 and
   5 < 20 and
   55 > 10) {
   // collision detected!
}
```

Circle Collision

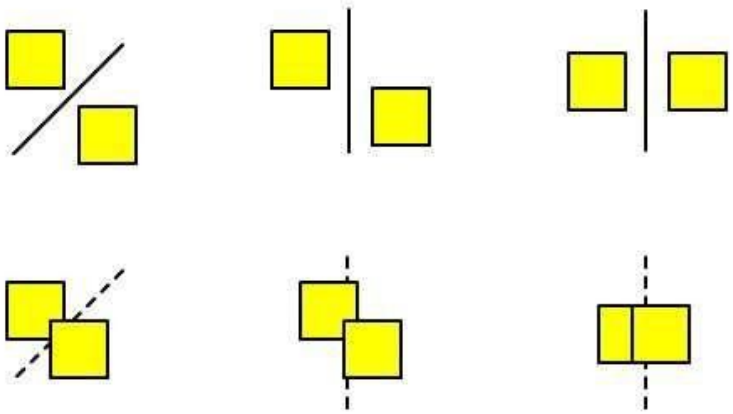
Another simple shape for collision detection is between two circles. This algorithm works by taking the centre points of the two circles and ensuring the distance between the centre points are less than the two radii added together.

```
circle1 = {radius: 20, x: 5, y: 5};
circle2 = {radius: 12, x: 10, y: 5};

dx = circle1['x'] - circle2['y'];
dy = circle1['y'] - circle2['y'];
distance = math.sqrt(dx * dx + dy * dy);

if (distance < circle1['radius'] + circle2['radius']) {
   // collision detected!
}
```

Separate Axis Theorem

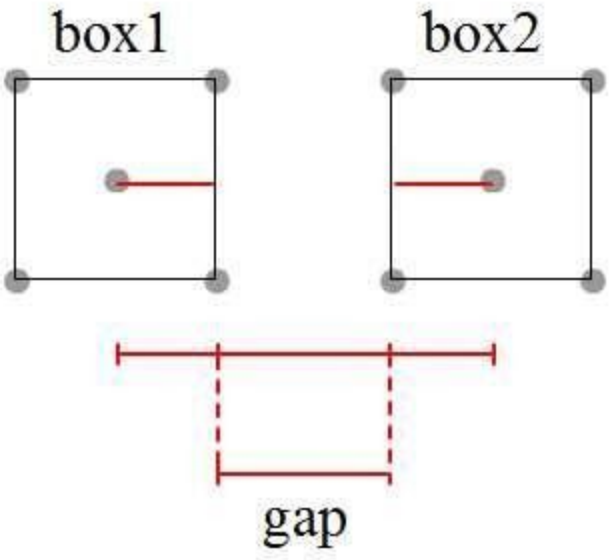
This is a collision algorithm that can detect a collision between any two *convex*

polygons.
Statement: If we can draw a seperate line between two polygons, then they do not collide.



Let's observe the above picture, the first three boxes are not colliding that's why we are able to draw lines between them. But the below figures are colliding in which the drawing line between them fails.

Projection along Arbitrary Axis



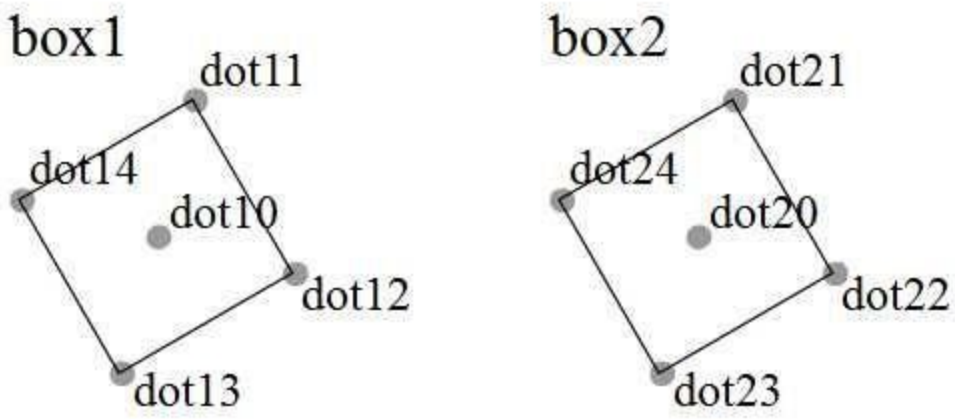
Let's assume for now that the polygons we refer to are squares: box1 on the left

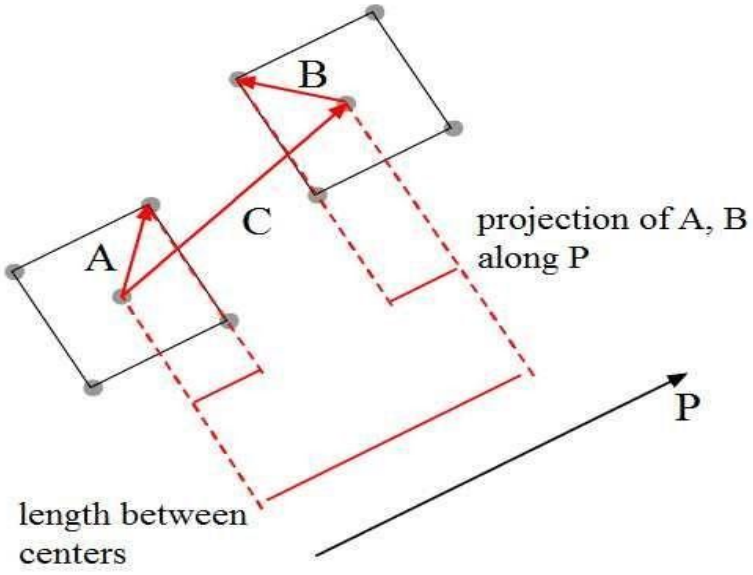
and box2 on the right. It's easy to see that these squares are horizontally separated. A straightforward approach to determine this in code is to calculate the horizontal distance between the two squares, then subtract the half-widths of box1 and box2:

```
//Pseudo code to evaluate the separation of box1 and box2
length = box2.x - box1.x;
half_width_box1 = box1.width*0.5;
half_width_box2 = box2.width*0.5;
gap_between_boxes = length - half_width_box1 -
half_width_box2;
if(gap_between_boxes > 0):
   print("It's a big gap between boxes")
elif(gap_between_boxes == 0)
   print("Boxes are touching each other")
elif(gap_between_boxes < 0)
   print("Boxes are penetrating each other")
```

What if boxes are not oriented nicely ?

Before we proceed, I'd like to clarify the naming convention used to denote the four corners of both boxes. This will be reflected in the code later:





Let's say both boxes are oriented 45° from the horizontal axis. We must calculate the following lengths in order to determine the gap between the boxes.

- Projection of A on axis P
- Projection of B on axis P
- Projection of C on axis P

While projection of A and C onto P will give a positive value, projection of B onto P will actually produce a *negative* value as the vectors are pointing in opposite directions.

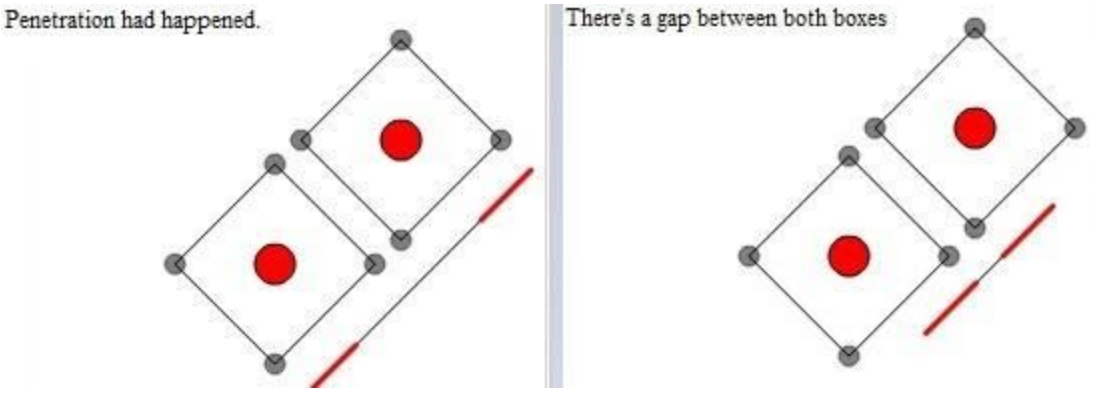
```
var dot10:Point = box1.getDot(0);
var dot11:Point = box1.getDot(1);
var dot20:Point = box2.getDot(0);
var dot24:Point = box2.getDot(4);
//Actual calculations
var axis:Vector2d = new Vector2d(1, -1).unitVector;
var C:Vector2d = new Vector2d(
   dot20.x - dot10.x,
   dot20.y - dot10.y
```

```
var A:Vector2d = new Vector2d(
   dot11.x - dot10.x,
   dot11.y - dot10.y
)
var B:Vector2d = new Vector2d(
   dot24.x - dot20.x,
   dot24.y - dot20.y
)
var projC:Number = C.dotProduct(axis)
var projA:Number = A.dotProduct(axis);
var projB:Number = B.dotProduct(axis);
var gap:Number = projC - projA + projB; //projB is expected
to be a negative value
if (gap > 0) t.text = "There's a gap between both boxes"
else if (gap > 0) t.text = "Boxes are touching each other"
else t.text = "Penetration had happened."
```

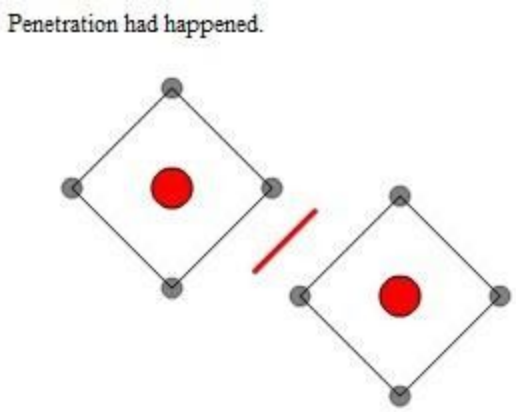
The Flaws

Well, we can go with the above implementation. But there are a few problems -- let me point them out:

First, vectors A and B are fixed. So when you swap the positions of box1 and box2, the collision detection fails.

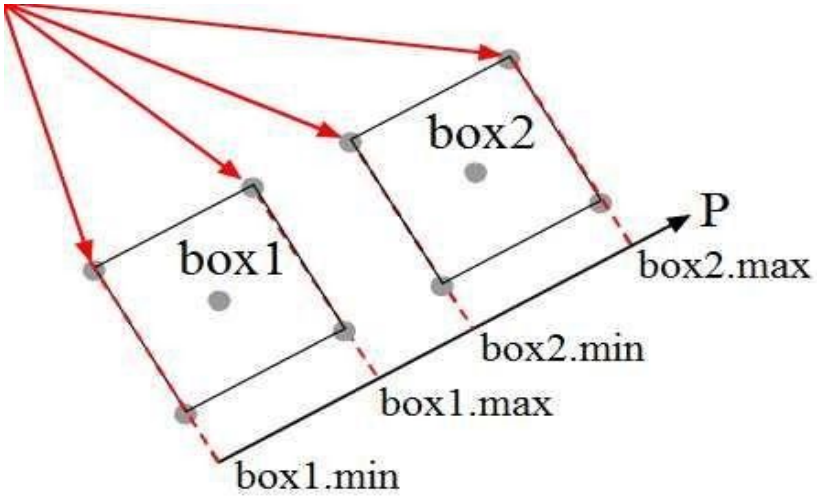


Second, we only evaluate the gap along one axis, so situations like the one below will not be evaluated correctly:



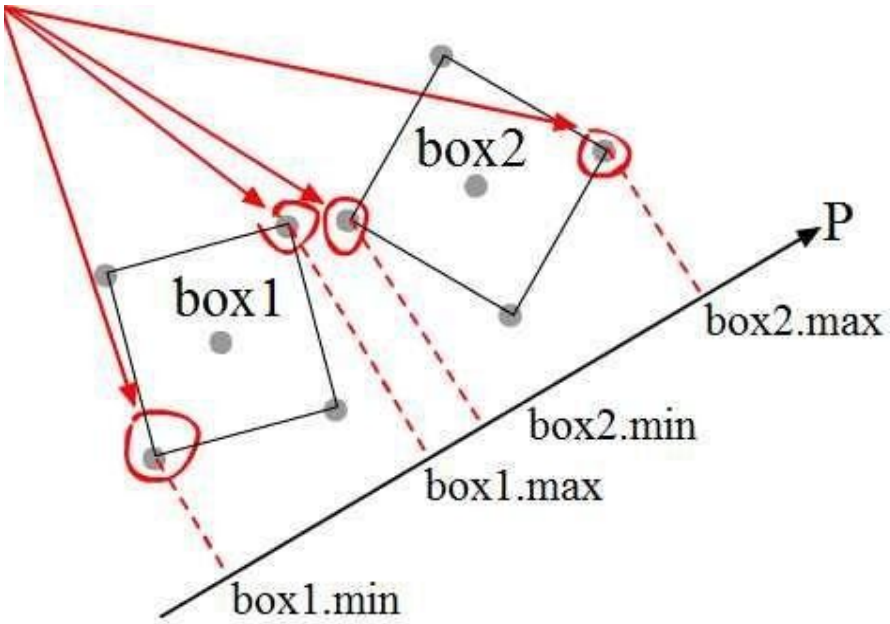
Solving First Flaw

So first of all, we'll need to get the minimum and maximum projections of corners (specifically the vectors from the origin to the boxes' corners) onto P.



The diagram above shows the projection of the minimum and maximum corners onto P when the boxes are oriented nicely along P.

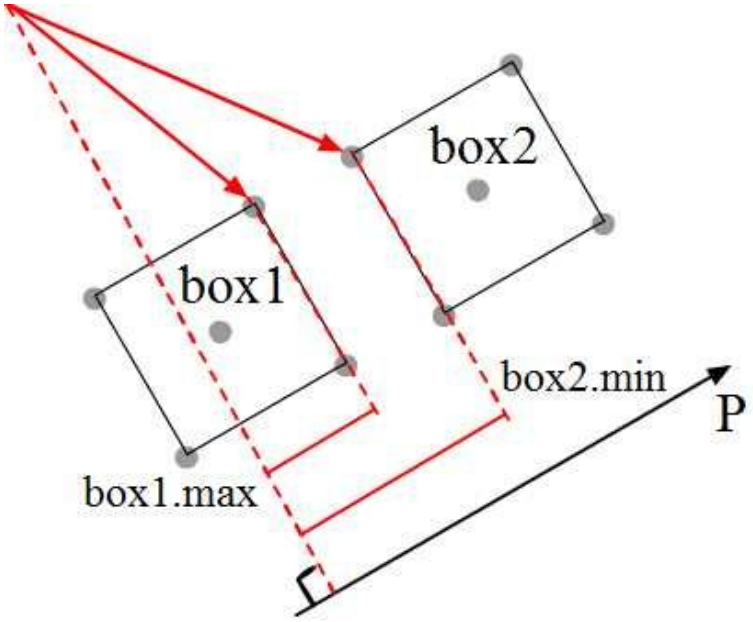
But what if box1 and box2 are not oriented accordingly?



The diagram above shows boxes which are not neatly oriented along P, and their corresponding min-max projections.

In this situation, we'll have to loop through each corner of each box and select the correct ones as appropriate.

Now we'll evaluate whether the boxes are colliding with each other. But How?



By observing the diagram above, we can clearly see the geometrical representation for projection of box1.max and box2.min onto axis P.

As you can see, when there's a gap between the two boxes, box2.min-box1.max will be more than zero -- or in other words, box2.min > box1.max. When the position of the boxes are swapped, then box1.min > box2.max implies there's a gap between them.

```
if(box2.min>box1.max || box1.min>box2.max){
   trace("collision along axis P happened")
}
else{
   trace("no collision along axis P")
}
```

Initialisation Code

Let's look at some more detailed code for figuring this out. Note that the AS3 code here is not optimised. Although it's long and descriptive, the advantage is that you can see how the math behind it works.

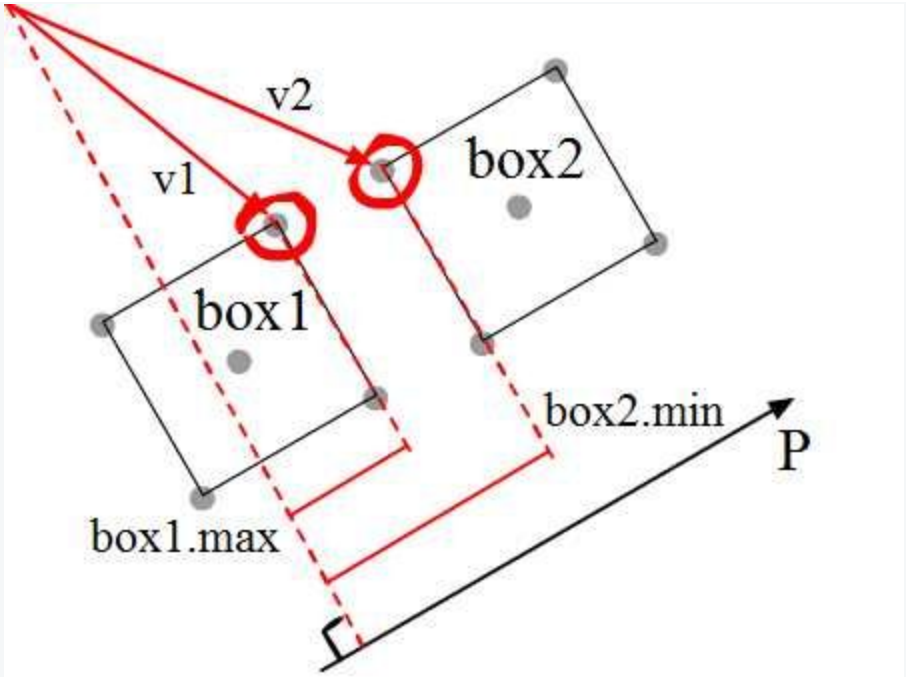
```
//preparing the vectors from origin to points
//since origin is (0,0), we can conveniently take the
coordinates
//to form vectors
var axis:Vector2d = new Vector2d(1, -1).unitVector;
var vecs_box1:Vector.<Vector2d> = new Vector.<Vector2d>;
var vecs_box2:Vector.<Vector2d> = new Vector.<Vector2d>;

for (var i:int = 0; i < 5; i++) {
    var corner_box1:Point = box1.getDot(i)
    var corner_box2:Point = box2.getDot(i)

    vecs_box1.push(new Vector2d(corner_box1.x,
corner_box1.y));
    vecs_box2.push(new Vector2d(corner_box2.x,
corner_box2.y));
}

//setting min max for box1
var min_proj_box1:Number = vecs_box1[1].dotProduct(axis);
var min_dot_box1:int = 1;
var max_proj_box1:Number = vecs_box1[1].dotProduct(axis);
var max_dot_box1:int = 1;

for (var j:int = 2; j < vecs_box1.length; j++)
{
    var curr_proj1:Number = vecs_box1[j].dotProduct(axis)
    //select the maximum projection on axis to
corresponding box corners
```

```
    if (min_proj_box1 > curr_proj1) {
        min_proj_box1 = curr_proj1
        min_dot_box1 = j
    }
    //select the minimum projection on axis to
corresponding box corners
    if (curr_proj1> max_proj_box1) {
        max_proj_box1 = curr_proj1
        max_dot_box1 = j
    }
}
var isSeparated:Boolean = max_proj_box2 < min_proj_box1 ||
max_proj_box1 < min_proj_box2
if (isSeparated) t.text = "There's a gap between both
boxes"
else t.text = "No gap calculated."
```

Optimisation

If you'd like to speed up the process, there's no need to calculate for the unit vector of P. You can therefore skip quite a number of expensive Pythagoras's theorem calculations which involve Math.sqrt():

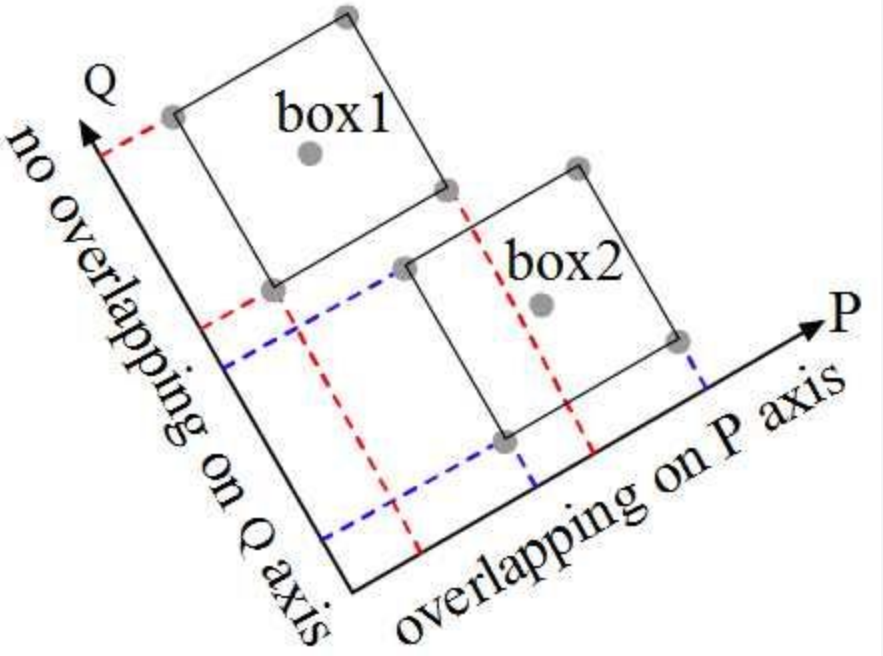
$$\begin{bmatrix} A_x \\ A_y \end{bmatrix} \cdot \begin{bmatrix} P_x / P_{magnitude} \\ P_y / P_{magnitude} \end{bmatrix} = A_{magnitude} * cos(theta)$$



whether it's a unit vector or not doesn't actually matter -- the result is the same.

Solving the Second Flaw

So we solved the issue for one axis, but that's not the end of it. We still need to tackle other axes -- but which?

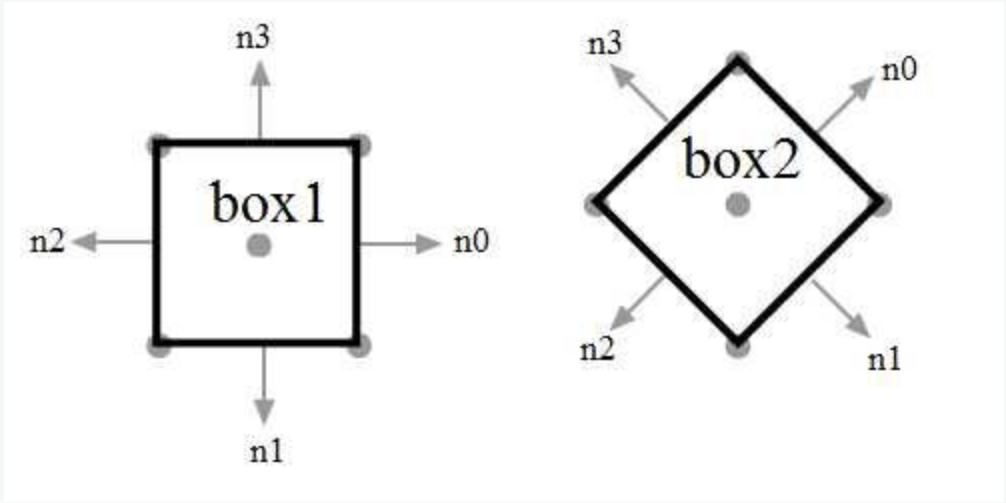


The analysis for boxes is quite straightforward: we compare two axes P and Q. In order to confirm a collision, overlapping on *all* axes has to be true -- if there's any axis without an overlap, we can conclude that there's no collision.

What if the boxes are oriented differently?

So of the P, Q, R, and S axes, there's only one axis that shows no overlapping between boxes, and our conclusion is that there's no collision between the boxes.

But the question is, how do we decide which axes to check for overlapping?

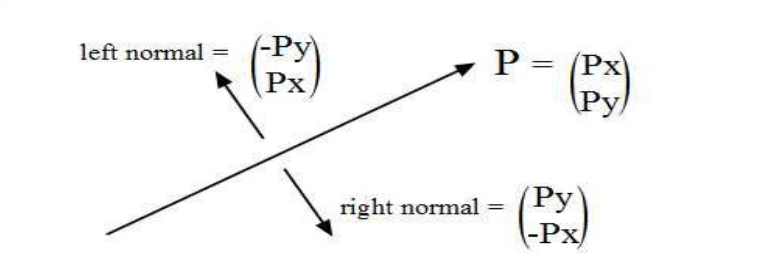


In a generalised form, with two boxes, we'll have to check along eight axes: n0, n1, n2 and n3 for each of box1 and box2. However, we can see that the following lie on the same axes:

- n0 and n2 of box1
- n1 and n3 of box1
- n0 and n2 of box2
- n1 and n3 of box2

So we don't need to go through all eight; just four will do. And if box1 and box2 share the same orientation, we can further reduce it to only evaluate two axes.

Calculating Normals

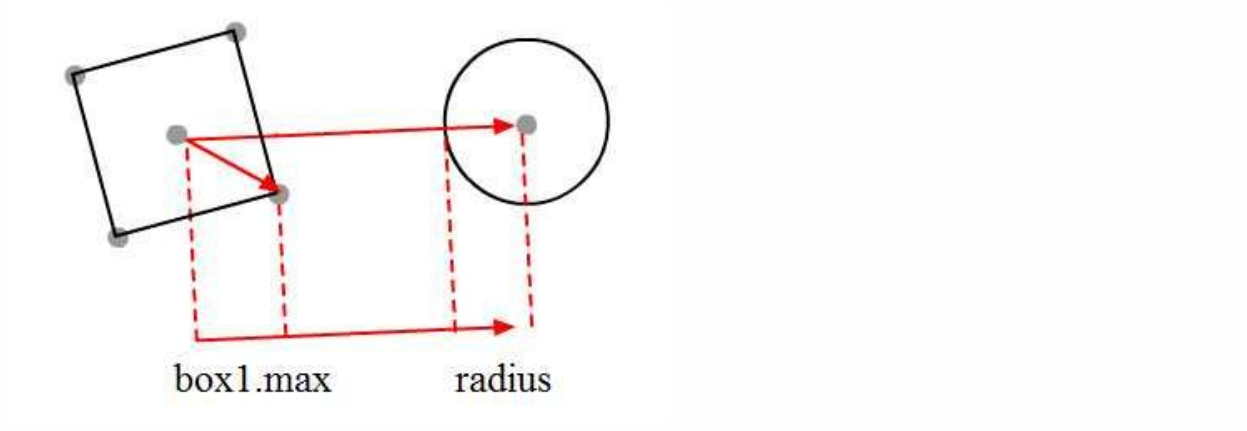


The diagram above shows the left and right normal of P. Note the switched components of the vector and the sign for each.

For my implementation, I'm using a clockwise convention, so I use the *left* normals.

```
public function getNorm():Vector.<Vector2d> {
    var normals:Vector.<Vector2d> = new Vector.<Vector2d>
    for (var i:int = 1; i < dots.length-1; i++)
    {
        var currentNormal:Vector2d = new Vector2d(
            dots[i + 1].x - dots[i].x,
            dots[i + 1].y - dots[i].y
        ).normL //left normals
        normals.push(currentNormal);
    }
    normals.push(
        new Vector2d(
            dots[1].x - dots[dots.length-1].x,
            dots[1].y - dots[dots.length-1].y
        ).normL
    )
    return normals;
}
```

Box-Circle Collision Detection




```
private function refresh():void {
    //prepare the vectors
    var v:Vector2d;
    var current_box_corner:Point;
    var center_box:Point = box1.getDot(0);

    var max:Number = Number.NEGATIVE_INFINITY;
    var box2circle:Vector2d = new Vector2d(c.x -
center_box.x, c.y - center_box.y)
    var box2circle_normalised:Vector2d =
box2circle.unitVector

    //get the maximum
    for (var i:int = 1; i < 5; i++)
    {
        current_box_corner = box1.getDot(i)
        v = new Vector2d(
            current_box_corner.x - center_box.x ,
            current_box_corner.y - center_box.y);
        var current_proj:Number =
v.dotProduct(box2circle_normalised)

        if (max < current_proj) max = current_proj;
```

```
    }
    if (box2circle.magnitude - max - c.radius > 0 &&
box2circle.magnitude > 0) t.text = "No Collision"
    else t.text = "Collision"
}
```

17. Metaphor based Heuristics

A heuristic function is one that ranks all the potential alternatives in a search algorithm based on the information available. It helps the algorithm to select the best route to its solution.
Metaphor based heuristics are the algorithms that have been inspired from various artificial and natural but physical processes. The efficiency and ease in those natural processes is what drives the core of these algorithms.

Some of those metaphor based heuristic algorithms are given below.

Harmony search

The process of music-making is a very precise one and this perfectness in harmony has inspired a population-based metaheuristics algorithm.
In music, the pitch of the instrument determines its aesthetics, the fitness function determines the quality of variables.

All the players with different pitches in a certain range to make a single harmony. When that harmony is good, it is stored in memory and the possibility of making that specific harmony is increased for next time.

Just in a similar way, a solution is randomly generated in a specific range of possible values. If the objective function of these variables is an optimized solution, then the possibility to make a good solution is increased.
The algorithm could be explained in the following way.

let the initial population, harmony memory set generates m possible vectors,let xi=xij , i=1,....m, j=1,...n
The rule for this is that a new random number say r1 is generated in [0,1].
Let h be harmony memory consideration rate
If r1 < h, the first variable in the new x_{ij} is chosen randomly from existing values.

The variable now obtained is checked to see if its pitch needs to be adjusted. A new random number r2 is generated in [0,1].
Pitch adjusting rate is p and the bandwidth factor is b. If r2<p, then the pitch adjustment is done as
$x_{ij}(new) = x_{ij} \pm rand(0,1).b$
B controls the local search around the already selected variable.

If r1 < h, then the new $x_{ij} = l_{ij} + (u_{ij} - l_{ij}).rand(0,1)$
l - lower bound
u - upper bound
After we get the new x, it will replace the old and worse harmony vector in the memory if its function value is better than the old function value.

If ($x_{new} < x_{old}$) then
Update the memory as $x_{old} = x_{new}$

end if

This process is repeated until a termination criterion is satisfied.

The final x matrix is the best solution.

A simple idea of musical harmony has thus created an algorithm that is used in many applications such as Engineering optimization, Data fusion in wireless sensor networks, manufacturing scheduling, etc.

Reference:

Askarzadeh, Alireza & Rashedi, Esmat. (2017). Harmony Search Algorithm. 10.4018/978-1-5225-2322-2.ch001.

Joong Hoon Kim, Harmony Search Algorithm: A Unique Music-inspired Algorithm, Procedia Engineering, Volume 154, 2016, ISSN 1877-7058.

Particle Swarm Optimization

- This is an optimization algorithm developed based on the foraging behaviour of birds.
- Which is achieved by random initialization of the population and the iterative update in the search process.
- In the process of searching for the optimal solution, each bird is considered to be a particle of no mass and no volume.

- During the search process, the particles are able to record their current best position (pbest) and the global best position (gbest). Velocity and position of each particle are calculated as follows:

$v_i^{t+1} = w* v_i^t + c_1 * rand * (pbest – x_i^t) + c_2 * rand * (gbest – x_i^t)$

$x_i^{t+1} = x_i^t + v_i^{t+1}$

whereas x_i^t and v_i^t are the current position and velocity of i^{th} agent on the t^{th} iteration, c_1 and c_2 acceleration coefficients that control the influence of pbest and gbest respectively on the search process. rand is a number in [0,1]. pbest is the current best position of all the particles at the t^{th} iteration. gbest is the best position among all the particles at whole iterations, and w is the inertia weight.

Gravitational Search Algorithm

- This is proposed by **Rashedi et al.**
- This algorithm is developed based on the law of gravitation and mass interactions.
- GSA consists of objects assigned masses named as agents.
- Agents are vectors.
- As these are massive, as per the laws of gravitation these attract among themselves according to our physical world gravitation laws.
- The greater the quality, the stronger is the attraction(gravity) between them.

- Therefore, location of the agent with the largest mass is the optimal solution.

Suppose let there be N d-dimensional agents.

$X_i = (x^1_i, x^2_i,.....x^d_i)$, where i = { 1,2,3,.....,N }

At time t the force acting on i^{th} agent by j^{th} agent is as follows:

$F^d_{ij} = G(t) (M_i(t) M_j(t)) / R_{ij}(t) + E (x^d_j(t) – x^d_i(t))$

where $M_j(t)$ and $M_i(t)$ are the masses of the jth agent and the ith agent, respectively, G(t) is the gravitational constant at the t time, E is a small constant, and $R_{ij}(t)$ is the Euclidean distance between the ith agent and the jth agent.

At time t, total force acting upon the i^{th} agent is as follows:

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} rand_j F_{ij}^d(t) \qquad (9)$$

Where rand is a uniform random variable in the interval [0,1].

Using the laws of motion, acceleration of i^{th} agent at time t is as follows,

$a^d_i(t) = F^d_i / M_i(t)$.

Further moving on in determining it's position and velocities using the laws of motion goes on as follows:

$v^d_i(t+1) = rand * v^d_i(t) + a^d_i(t)$

$x^d_i(t+1) = x^d_i(t) + v^d_i(t)$

where Rand is a uniform random variable in the interval [0,1] and $x^d_i(t)$ and $v^d_i(t)$ are

its current position and velocity, respectively.

The Combination Approach of PSO and GSA

- In GSA, agents do not share population information with each other and have a weak capability of development.
- By use of the global optimal searching ability of PSO and the local searching ability of GSA, each agent is updated by the velocity of PSO and the acceleration of GSA.

- The approach is called PSO-GSA.
- Thus, the exploration ability and exploitation ability are better combined with the unceasing change of parameters.
- The velocity and the position of the i^{th} agent are updated by the following two equations:

$$v_i^{t+1} = w * v_i^t + c_1' * rand * ac_i(t) + c_2' * rand * (gbest - x_i^t)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

Where w is inertia weight, x_i^t, v_i^t and $ac_i(t)$ are the velocity, current position and acceleration of the i^{th} particle at t^{th} iteration. c_1' and c_2' are constant acceleration coefficients, respectively.

Bees algorithm

In computer science and operations research, the bees algorithm is a population-based search algorithm, modelled on the foraging behaviour of honey bees.

This algorithm combines global explorative search with local exploitative search. A small number of artificial bees randomly explore the solution space (environment) for solutions of high fitness (highly profitable food sources), while

the bulk of the population search (harvest) the neighbourhood of the fittest solutions looking for the fitness optimum. A deterministics recruitment procedure which simulates the waggle dance of biological bees is used to communicate the scouts' findings to the foragers, and distribute the foragers depending on the fitness of the neighbourhoods selected for local search. Once the search in the neighbourhood of a solution stagnates, the local fitness optimum is considered to be found, and the site is abandoned. In summary, the Bees Algorithm searches concurrently the most promising regions of the solution space, while continuously sampling it in search of new favourable regions.

The bees algorithm mimics the foraging strategy of honey bees to look for the best solution to an optimisation problem. Each candidate solution is thought of as a food source (flower), and a population (colony) of n agents (bees) is used to search the solution space. Each time an artificial bee visits a flower (lands on a solution), it evaluates its profitability (fitness).

The bees algorithm consists of an initialisation procedure and a main search cycle which is iterated for a given number T of times, or until a solution of acceptable fitness is found. Each search cycle is composed of five procedures: recruitment, local search, neighbourhood shrinking, site abandonment, and global search.

Basic bee algorithm

Generate the initial population size as n, set the best patch size as m, set the elite patch size as e, set the number of forager bees recruited to the of elite sites as nep, set the number of forager bees around the non-elite best patches as nsp, set the neighborhood size as ngh, set the maximum iteration number as $MaxIter$, and set the error limit as $Error$.

$i = 0$

Generate initial population.

Evaluate Fitness Value of initial population.

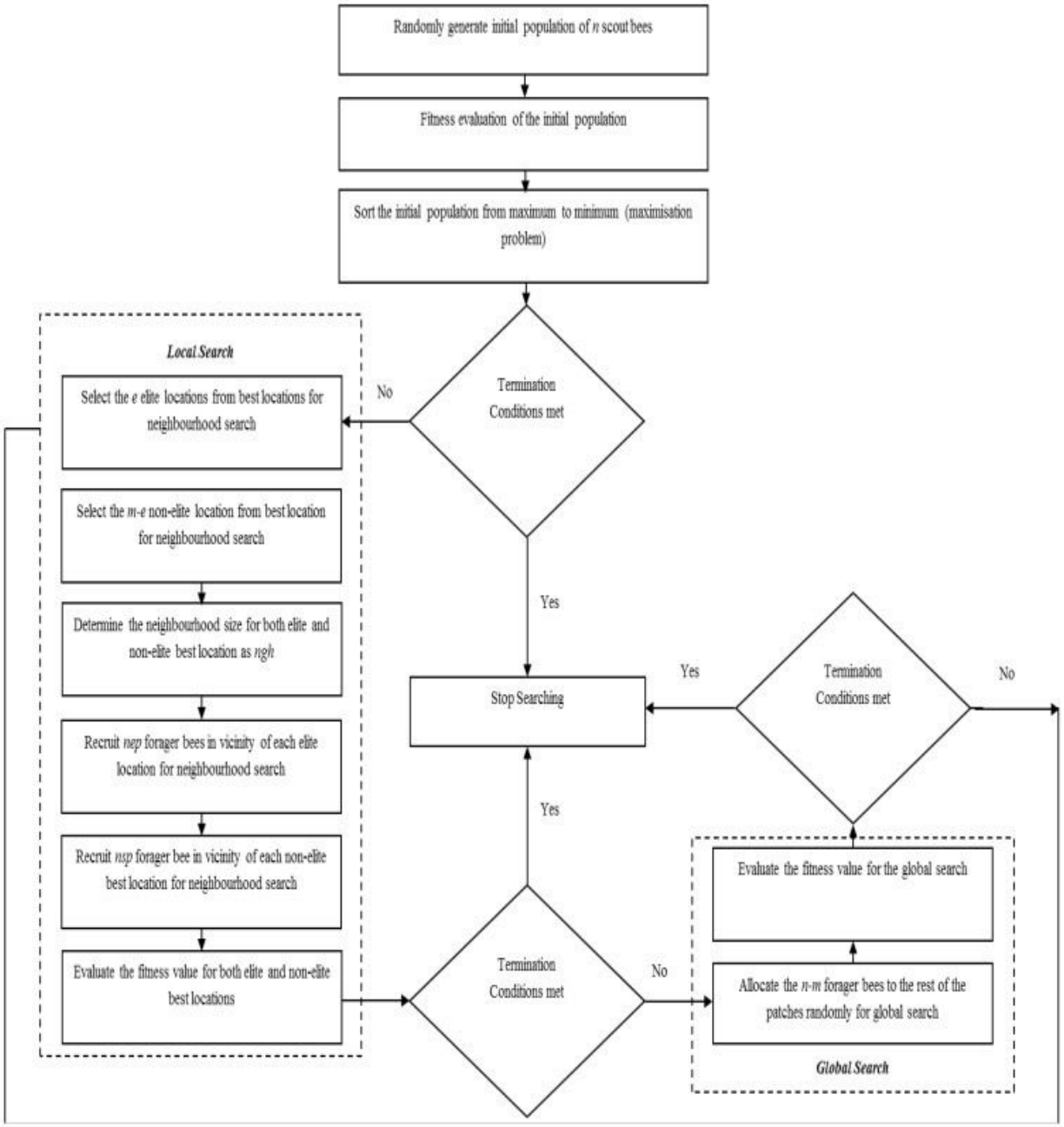
Sort the initial population based on the fitness result.

While $i \leq MaxIter$ or $FitnessValue_i - FitnessValue_{i-1} \leq Error$

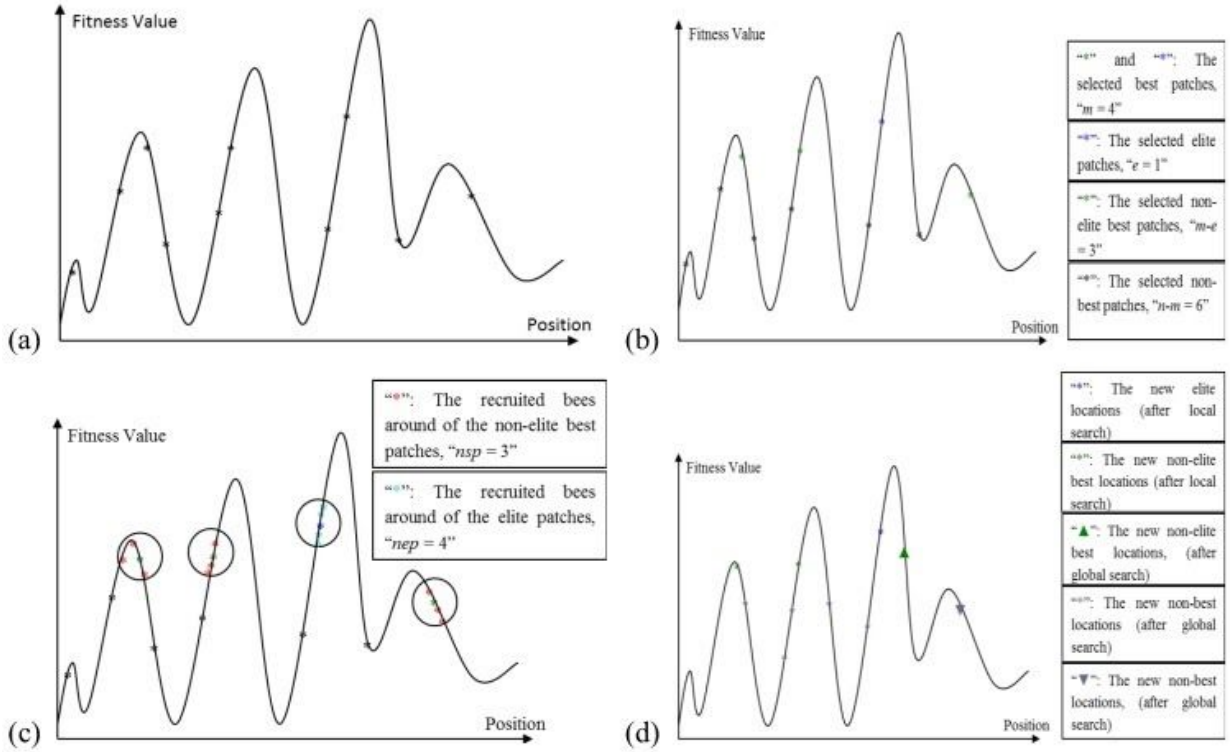
1. $i = i + 1$;
2. Select the elite patches and non-elite best patches for neighborhood search.
3. Recruit the forager bees to the elite patches and non-elite best patches.
4. Evaluate the fitness value of each patch.
5. Sort the results based on their fitness.
6. Allocate the rest of the bees for global search to the non-best locations.
7. Evaluate the fitness value of non-best patches.
8. Sort the overall results based on their fitness.
9. Run the algorithm until termination criteria met.

End

Flow chart of Bees algorithm



The Algorithm starts with sending *n* scout bees randomly to selected sites . The fitness values of each site are evaluated and sorted from the highest to the lowest (a maximization problem). The local search step of the algorithm covers the best locations (sites), which are the *m* fittest locations. The *m* best sites are also classified into two sub-groups; elite and non-elite best sites, as given in. The number of elite sites is set as "*e*" and the number of the non-elite best sites is "*m-e*". The local search process starts with recruiting forager bees in the neighborhood of the best sites. The neighborhood size is set to "*ngh*". The number of recruited bees in the neighborhood of each elite site is set to "*nep*" and the number of recruited bees in the neighborhood of the non-elite best sites is set to "*nsp*", as given in . The global search process is a random search process in the *n-m* "non-best" sites. Finally, the overall locations are sorted according to their fitness value and the process runs until the global optimum is found.



Bees inspired optimization methods

Optimization algorithms are search methods where the goal is to find an optimal solution to a problem, in order to satisfy one or more objective functions, possibly subject to a set of constraints.Studies of social animals and social insects have resulted in a number of computational models of swarm intelligence.

Swarm Intelligence (SI) is defined as the collective problem-solving capabilities of social animals

Swarm Optimization Algorithms (SOAs) mimic the collective exploration strategy of the swarms in the nature of optimization problems.These algorithms utilize a population based approach to the problems. This group of algorithms is known as population based stochastic algorithms.

In nature, honey bees have several complicated behaviors such as mating, breeding and foraging. These behaviors have been mimicked for several honey bee based optimization algorithms.

One of the famous mating and breeding behaviors of honey bees inspired algorithm is Marriage in Honey Bees Optimization (MBO). The algorithm starts from a single queen without family and passes on to the development of a colony with family having one or more queens.In the literature, several versions of MBO have been proposed such as Honey-Bees Mating Optimization (HBMO), Fast

Marriage in Honey Bees Optimization (FMHBO) and The Honey-Bees Optimization (HBO)

The other type of bee-inspired algorithms mimics the foraging behavior of the honey bees. These algorithms use standard evolutionary or random explorative search to locate promising locations. Then the algorithms utilize the exploitative search on the most promising locations to find the global optimum. The following algorithms were inspired from foraging behavior of honey bees; Bee System (BS), Bee Colony Optimization (BCO), Artificial Bee Colony (ABC) and The Bees Algorithm (BA).

Killer Whale Algorithm

- The algorithm is based on the life of a killer whale.
- A group of Killer Whales (Orcinus Orca) called Matriline that consist of a leader and members. The leader's duty searches prey position and the optimum direction to chase the prey, meanwhile chase the prey is performed by the members.
- Optimum direction means minimum distance and maximum velocity.
- Global optimum is generated by the results of all member's actions.

- These killer whales are the highest in the peak of marine ecological systems.
- There are three kinds of killer whales determined by their body morphological forms, they are Type A, Type B and Type C. In which Type A has the largest body shape than the other types of killer whales.
- Killer whales are also classified among their prey and hunting pattern, Fish-Feeding Residents and Mammal-Hunting Transients.
- Fish-Feeding Residents is Killer Whale with hunting patterns in the same area, meanwhile, the Mammal-Hunting Transients will hunt following the prey migration season.
- The prey scanning is carried out using echolocation vocalizations. Killer Whales have 3 types of sounds namely Clicks, Whistles and Pulsed Calls.

Let's dive into some mathematical stuff of it's hunting:

let the distance between top of the water and eyes whale be d_o and prey's distance from the top be d_F. Distance between the eyes of the whale and the prey is R.

$\theta = \sin^{-1} (d_F - d_o / R) = \tan^{-1} (d_F - d_o / x)$

where x is the horizontal distance between whale and prey.

- Every search agent requires a velocity to move from recent location to prey location, in terms of magnitude and direction of movement.

V_i <-- V_i + U((0, $\varnothing_1$)$\otimes$(p_i – x_i)) + U((0, $\varnothing_2$)$\otimes$(p_g – x_i)),

x_i <-- x_i + V_i*t

x_i --> current position, p_i --> previous best position, V_i --> velocity, t --> time

- The optimization problem consists of N-dimensional space and [-x,x] boundary. Optimization techniques will be assigned to find out the optimized variables within –x to x for each N.
- In Killer Whale Algorithm, as a group of search agents, the range of optimization can be clustered to become some range based on the number of search agents.
- Each search agent will seek local optimum value within the cluster from the centroid of the cluster. Clustering process in this algorithm requires matrix input of M points and K initial cluster centres in N dimensions.
- Number of points in cluster L is denoted by NC(L), D(I,L) is the Euclidean distance between point I and cluster L, the general procedure is to search for a K-partition with locally optimal within-cluster sum of squares by moving points from one cluster to another.

The clustering process can be adopted using method available in and can be explained as follows:

1) Each point I (I = 1,2,...,M), identified the closest and Second closest cluster centres IC1(I) and IC2(I) respectively. Denote point I to cluster IC1(I).
2) Update the centre of clusters to be the mean of points contained within them.

3) All clusters are appropriate to the live set, initially.
4) Utilize the optimal-transfer (OPTRA) stage: Consider each point I (I = 1,2,...,M) in turn. If cluster L (L = 1,2,...,K) is updated in the last quick-transfer (QTRAN) stage, therefore, it belongs to the live set throughout the current stage. Otherwise, it is not included in the live set if there are no updates in the last M OPTRA steps. Let the point I be in cluster L1. If L1 is included in the live set, perform Step 4a; otherwise, carry out Step 4b.
   a) Calculate the quantity of minimum, R2 = [NC(L) * D(I,L) 2] / [NC(L)+1], overall clusters L($L \neq L1$, L = 1,2,...,K). Let L2 clustered with the smallest R2. If this value greater than or equal to [NC(L1) * D(I,L1) 2] / [NC(L1)-1], reallocation is not required and L2 is the new IC2(I). (Should be noted that the value [NC(L1) * D(I,L1) 2] / [NC(L1)-1] is memorized and will save at the same value for point I until cluster L1 is renewed). Otherwise, point I is assigned to cluster L2 and L1 is the updated IC2(I). Centre of clusters are renewed to be the averages of points assigned to them if restructuring has taken place. The two involved clusters in the transfer of point I at this particular step are now in the live set.
   b) This step is similar to Step 4a, except that the minimum R2 is calculated over the cluster in the live set lonely.
5) Break if the live set is blank. Otherwise, continue to Step 6, after a single pass through the data set.

6) Perform the QTRAN stage: Take each point I (I = 1,2,...,M) in turn. Put L1 = IC1(I) and L2 = IC2(I). It is not required to check the point I if both the

clusters L1 and L2 have the same values in the last M steps. Calculate the values R1 = [NC(L1) * D(I,L1) 2] / [NC(L1)-1] and R2 = [NC(L2) * D(I,L2) 2] / [NC(L2)+1]. (As mentioned before, R1 is memorized and will save the same values until cluster L1 is renewed). If R1 less than R2, point I rests in cluster L1. Otherwise, over change IC1(I) and IC2(I) and renew the centre of clusters L1 and L2. The two areas of clusters are recognized for their involvement in a relocation at this step.

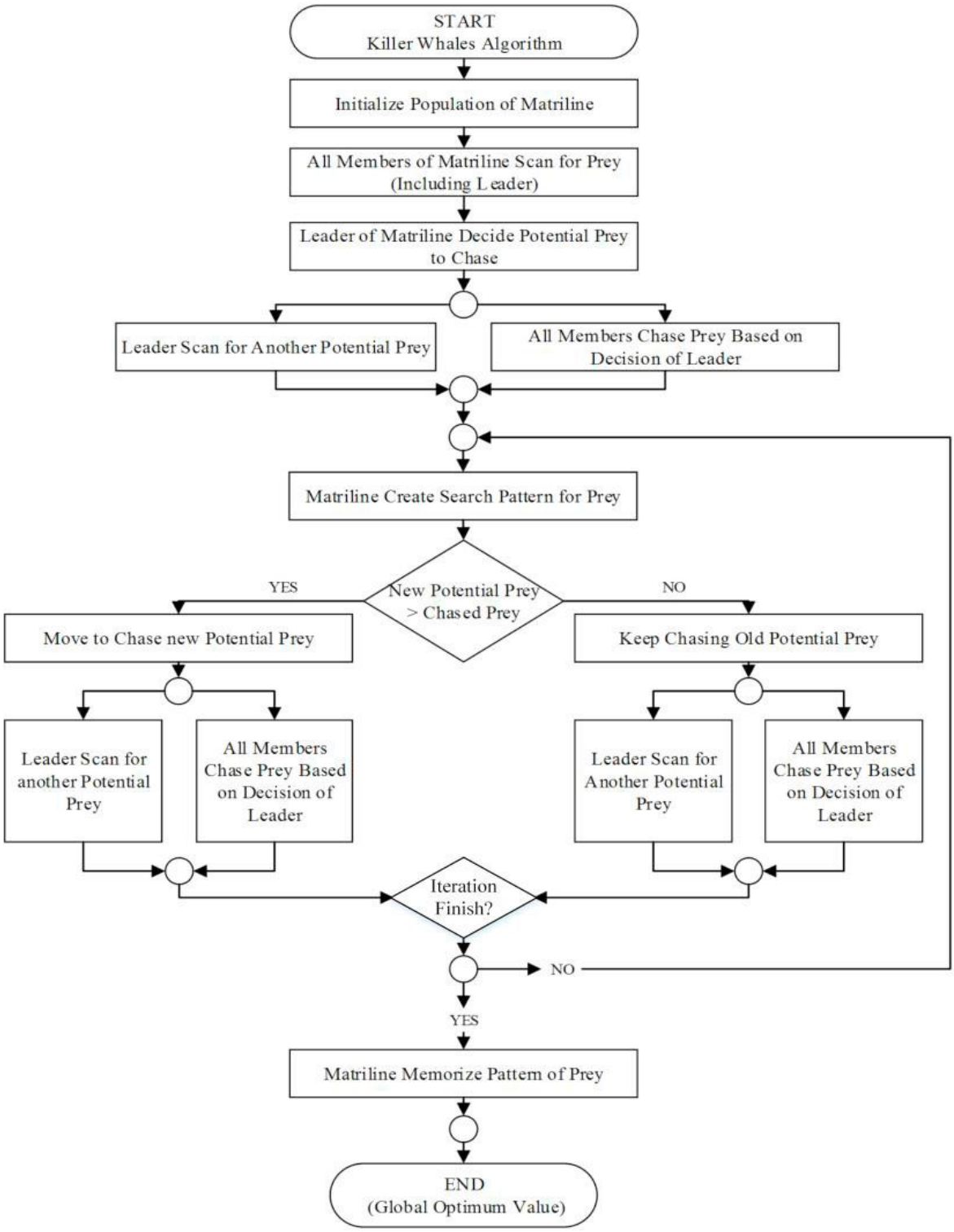
Experimental Results of the Algorithm:

The performance of Killer Whale Algorithm can be measured using some real-parameter in COCO especially BBOB. BBOB is a standard function to examine a new algorithm proposed by N. Hansen, et al. [16]. In this paper used 6 benchmark functions with noise. The purpose of the benchmark is to prove the capability of the proposed algorithm to solve optimization problems.

Conclusion:

In this paper, a new algorithm inspired by the life of killer whales was proposed. A group of Killer Whales called Matriline that consist of a leader and members. The leader’s duty searches prey position and the optimum direction to chase the prey, meanwhile prey hunting is performed by the members. Global optimum is obtained by comparing the results of a member's actions. Testing was performed using the 6 noisy benchmark functions that contain Gaussian noise, Uniform noise and Seldom Cauchy noise. The optimization results show the performance of proposed algorithms surpasses others compared to algorithms and robust.

Flow chart of Algorithms:



Intelligent water drops algorithm

In nature, flowing water drops are observed mostly in rivers, which form huge moving swarms. The paths that a natural river follows have been created by a swarm of water drops. For a swarm of water drops, the river in which they flow is the part of the environment that has been dramatically changed by the swarm and will also be changed in the future. We are using the concept of the water path in the river. How they prefer a path with less soil than a path with more soil. The water drop prefers an easier path to a harder path when it has to choose between several branches.

Intelligent Water Drops algorithm, or the IWD algorithm, is a Swarm based nature-inspired optimization algorithm. This algorithm contains a few essential elements of natural water drops and actions and reactions that occur between river's bed and the water drops that flow within. The IWD algorithm can be used for Combinatorial optimization.

Almost every IWD algorithm is composed of two parts:

A graph that plays the role of distributed memory on which soils of different edges are preserved, and the moving part of the IWD algorithm, which is a few number of Intelligent water drops.

These intelligent water drops (IWDs) both compete and cooperate to find better solutions and by changing soils of the graph, the paths to better solutions become more reachable. It is mentioned that the IWD-based algorithms need at least two IWDs to work.

The IWD algorithm has two types of parameters

static and dynamic parameters.

Static parameters are constant during the process of the IWD algorithm.

Dynamic parameters are reinitialized after each iteration of the IWD algorithm. The pseudo-code of an IWD-based algorithm may be specified in eight steps:

```
1) Static parameter initialization
a) Problem representation in the form of a graph
b) Setting values for static parameters
2) Dynamic parameter initialization: soil and velocity of IWDs
```

```
3) Distribution of IWDs on the problem's graph
4) Solution construction by IWDs along with soil and velocity
updating
a) Local soil updating on the graph
b) Soil and velocity updating on the IWDs
5) Local search over each IWD's solution (optional)
6) Global soil updating
7) Total-best solution updating
8) Go to step 2 unless termination condition is satisfied
```

Appendix

Linear Search

```
# If x is present then return its location,
# otherwise return -1
def search(arr, n, x):
    for i in range(0, n):
        if (arr[i] == x):
            return i
    return -1
 # Driver Code
arr = [2, 3, 4, 10, 40]
x = 10
n = len(arr)
result = search(arr, n, x)
if(result == -1):
    print("Element is not present in array")
else:
    print("Element is present at index", result)
```

Binary Search

```
# It returns location of x in given array arr
# if present, else returns -1

def binarySearch(arr, l, r, x):
```

```
  while l <= r:
      mid = l + (r - l) // 2
      if arr[mid] == x:
          return mid
      elif arr[mid] < x:
          l = mid + 1
      else:
          r = mid - 1
  return -1
#Driver code
arr = [ 2, 3, 4, 10, 40 ]
x = 10
result = binarySearch(arr, 0, len(arr)-1, x)
if result != -1:
  print ("Element is present at index % d" % result)
else:
  print ("Element is not present in array")
```

DFS

```
N = 2**5+5
visited = [0]*N
adj = [[] for i in range(N)]
# dfs function
def dfs(node):
  print(node)
  global visited
```

```
  visited[node] = 1
  for i in adj[node]:
      if visited[i] != 1:
          dfs(i)

#driver code
n = 5 # no of nodes
#edge 1
adj[0].append(1)
adj[1].append(0)

#edge 2
adj[2].append(3)
adj[3].append(2)

#edge 3
adj[4].append(1)
adj[1].append(4)

#edge 4
adj[0].append(3)
adj[3].append(0)

for i in range(n):
  if visited[i] != 1:
      dfs(i)
```

BFS

```
N = 2**5+5
visited = [0]*N
adj = [[] for i in range(N)]

def bfs(start):
  temp = [start]
  visited = [0]*N
  visited[start] = 1
  while(len(temp) > 0):
    print(temp[0])
    for u in adj[temp[0]]:
      if visited[u] != 1:
        visited[u] = 1
        temp.append(u)
    temp.pop(0)


#driver code
n = 5 # no of nodes
#edge 1
adj[0].append(1)
adj[1].append(0)

#edge 2
adj[2].append(3)
adj[3].append(2)

#edge 3
adj[4].append(1)
```

```
adj[1].append(4)

#edge 4
adj[0].append(3)
adj[3].append(0)

bfs(0)
```

Krushkal

```
from operator import attrgetter
class node:
  def __init__(self,u,v,w):
    self.u = u
    self.v = v
    self.w = w
N = 100005
parent = [i for i in range(N)]
height = [1] * N
def root(u):
  while u != parent[u]:
    u = parent[u]
  return u
def isformcycle(u,v):
  global parent
  a = root(u)
  b = root(v)
  if a==b:
    return 1
  elif height[a] > height[b]:
```

```
        parent[b] = a
    elif height[b] > height[a]:
        parent[a] = b
    else:
        parent[a] = b
        height[b] += 1
    return 0
nodes,edges = 5,7

l = [node(0,1,5),node(0,2,9),node(0,4,3),node(1,2,4)
    ,node(1,3,2),node(2,3,7),node(2,4,6)]

l.sort(key=attrgetter('w'))
mst_cost,k = 0,0
mst_edges = []
for i in range(edges):
    if isformcycle(l[i].u,l[i].v) == 0:
        mst_cost += l[i].w
        k += 1
        mst_edges.append(l[i])
    if k == edges - 1:
        break
print(mst_cost)
for ed in mst_edges:
    print(ed.u,ed.v,ed.w)
```

Floyd Warshell

```
nodes = 4
```

```
inf = 999999999
def print_dist(dist):
  for i in range(nodes):
    for j in range(nodes):
      if dist[i][j] == inf:
        print("    INF",end = "")
      else:
        print("%7d" % dist[i][j],end="")
    print()
def floyd_warshall(graph):
  dist = graph.copy()
  for k in range(nodes):
    for i in range(nodes):
      for j in range(nodes):
        dist[i][j] = min(dist[i][j],dist[i][k]+ dist[k][j])
  print_dist(dist)

graph = [[0,5,inf,10], [inf,0,3,inf], [inf,inf,0,1],[inf,inf,inf,0]]
floyd_warshall(graph)
```

Bellman-Ford

```
inf = 999999999
# Class to represent a graph
class Graph:

  def __init__(self, vertices):
    self.V = vertices # No. of vertices
```

```
    self.graph = []

# function to add an edge to graph
def addEdge(self, u, v, w):
    self.graph.append([u, v, w])

# utility function used to print the solution
def printArr(self, dist):
    print("Vertex Distance from Source")
    for i in range(self.V):
        print("{0}\t\t{1}".format(i, dist[i]))

# The main function that finds shortest distances from src to
# all other vertices using Bellman-Ford algorithm. The function
# also detects negative weight cycle
def BellmanFord(self, src):

    # Step 1: Initialize distances from src to all other vertices
    # as INFINITE
    dist = [inf] * self.V
    dist[src] = 0


    # Step 2: Relax all edges |V| - 1 times. A simple shortest
    # path from src to any other vertex can have at-most |V| - 1
    # edges
    for _ in range(self.V - 1):
        # Update dist value and parent index of the adjacent vertices of
        # the picked vertex. Consider only those vertices which are still in
```

```
        # queue
        for u, v, w in self.graph:
            if dist[u] != inf and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

    # Step 3: check for negative-weight cycles. The above step
    # guarantees shortest distances if graph doesn't contain
    # negative weight cycle. If we get a shorter path, then there
    # is a cycle.

        for u, v, w in self.graph:
                if dist[u] != inf and dist[u] + w < dist[v]:
                        print("Graph contains negative weight cycle")
                        return

        # print all distance
        self.printArr(dist)

g = Graph(5)
g.addEdge(0, 1, -1)
g.addEdge(0, 2, 4)
g.addEdge(1, 2, 3)
g.addEdge(1, 3, 2)
g.addEdge(1, 4, 2)
g.addEdge(3, 2, 5)
g.addEdge(3, 1, 1)
g.addEdge(4, 3, -3)

# Print the solution
g.BellmanFord(0)
```

Prims

```
# A Python program for Prim's Minimum Spanning Tree (MST) algorithm.
# The program is for adjacency matrix representation of the graph
```

```
inf = 999999999
class Graph():

  def __init__(self, vertices):
     self.V = vertices
     self.graph = [[0 for column in range(vertices)]
                 for row in range(vertices)]

  # A utility function to print the constructed MST stored in parent[]
  def printMST(self, parent):
     print("Edge \tWeight")
     for i in range(1, self.V):
        print(parent[i], "-", i, "\t", self.graph[i][ parent[i] ])

  # A utility function to find the vertex with
  # minimum distance value, from the set of vertices
  # not yet included in shortest path tree
  def minKey(self, key, mstSet):

     # Initialize min value
     min = inf

     for v in range(self.V):
        if key[v] < min and mstSet[v] == False:
           min = key[v]
           min_index = v

     return min_index
```

```
# Function to construct and print MST for a graph
# represented using adjacency matrix representation
def primMST(self):

    # Key values used to pick minimum weight edge in cut
    key = [inf] * self.V
    parent = [None] * self.V # Array to store constructed MST
    # Make key 0 so that this vertex is picked as first vertex
    key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1 # First node is always the root of

        for cout in range(self.V):

            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minKey(key, mstSet)

            # Put the minimum distance vertex in
            # the shortest path tree
            mstSet[u] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shortest path tree
            for v in range(self.V):

                # graph[u][v] is non zero only for adjacent vertices of m
                # mstSet[v] is false for vertices not yet included in MST
                # Update the key only if graph[u][v] is smaller than key[v]
                if self.graph[u][v] > 0 and mstSet[v] == False and key[v] >
self.graph[u][v]:
```

```
                        key[v] = self.graph[u][v]
                        parent[v] = u

        self.printMST(parent)

g = Graph(5)
g.graph = [ [0, 2, 0, 6, 0],
            [2, 0, 3, 8, 5],
            [0, 3, 0, 0, 7],
            [6, 8, 0, 0, 9],
            [0, 5, 7, 9, 0]]

g.primMST();
```

Lex-BFS

```
from queue import PriorityQueue
nodes = 5
edges = 5
adj = [[] for i in range(nodes)]
def printLexoSmall(l,n):
  visited = [0] * n
  q = PriorityQueue()
  visited[0] = 1
  q.put(0)
  while q.empty() == False:
     now = q.get()
     print(now,end = " ")
     for p in l[now]:
        if visited[p] == 0:
           q.put(p)
```

```
            visited[p] = True
    print()
def insert_edge(u,v):
    global adj
    adj[u].append(v)
    adj[v].append(u)

insert_edge(0,3)
insert_edge(2,3)
insert_edge(4,3)
insert_edge(2,1)
insert_edge(0,4)
printLexoSmall(adj,nodes)
```

Binary Gcd (Stein's Algo)

```
def gcd(a, b):
    if (a == 0) :
        return b
    if (b == 0) :
        return a
    # Finding K, where K is the
    # greatest power of 2 that
    # divides both a and b.
    k = 0
    while(((a | b) & 1) == 0) :
        a = a >> 1
        b = b >> 1
        k = k + 1
```

```
    # Dividing a by 2 until a becomes odd
    while ((a & 1) == 0) :
        a = a >> 1

    # From here on, 'a' is always odd.
    while(b != 0) :

        # If b is even, remove all
        # factor of 2 in b
        while ((b & 1) == 0) :
            b = b >> 1

        # Now a and b are both odd. Swap if
        # necessary so a <= b, then set
        # b = b - a (which is even).
        if (a > b) :

            # Swap u and v.
            temp = a
            a = b
            b = temp

        b = (b - a)

    # restore common factors of 2
    return (a << k)

# Driver code
```

```
a = 34
b = 17

print("Gcd of given numbers is ", gcd(a, b))
```

Best First Search

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

# Function For Implementing Best First Search
# Gives output path having lowest cost
def best_first_search(source, target, n):
  visited = [0] * n
  visited[source] = True
  pq = PriorityQueue()
  pq.put((0,source))
  while pq.empty() == False:
    u = pq.get()[1]
    # Displaying the path having lowest cost
    print(u,end=" ")
    if u == target:
      break

    for v,c in graph[u]:
      if visited[v] == False:
        visited[v] = True
        pq.put((c,v))
```

```
  print()

# Function for adding edges to graph
def addedge(x,y,cost):
  graph[x].append((y,cost))
  graph[y].append((x,cost))

# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source,target,v)
```

Merge sort

```
def merge(arr, l, m, r):
  n1 = m - l + 1
  n2 = r- m

  # create temp arrays
  L = [0] * (n1)
  R = [0] * (n2)

  # Copy data to temp arrays L[] and R[]
  for i in range(0 , n1):
     L[i] = arr[l + i]

  for j in range(0 , n2):
     R[j] = arr[m + 1 + j]

  # Merge the temp arrays back into arr[l..r]
  i = 0    # Initial index of first subarray
  j = 0    # Initial index of second subarray
  k = l    # Initial index of merged subarray

  while i < n1 and j < n2 :
     if L[i] <= R[j]:
        arr[k] = L[i]
        i += 1
     else:
        arr[k] = R[j]
        j += 1
```

```
        k += 1

    # Copy the remaining elements of L[], if there are any
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Copy the remaining elements of R[], if there are any
    while j < n2:
            arr[k] = R[j]
        j += 1
        k += 1

def mergeSort(arr,l,r):
    if l < r:
        # Same as (l+r)//2, but avoids overflow for large l and h
        m = (l+(r-1))//2
        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)


# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print ("Given array is")
for i in range(n):
```

```
  print ("%d" %arr[i]),

mergeSort(arr,0,n-1)
print ("\n\nSorted array is")
for i in range(n):
  print ("%d" %arr[i]),
```

Heap sort

```
def heapify(arr, n, i):
  largest = i  # Initialize largest as root
  l = 2 * i + 1     # left = 2*i + 1
  r = 2 * i + 2     # right = 2*i + 2
   # See if left child of root exists and is greater than root
  if l < n and arr[i] < arr[l]:
     largest = l
   # See if right child of root exists and is greater than root
  if r < n and arr[largest] < arr[r]:
     largest = r
   # Change root, if needed
  if largest != i:
     arr[i],arr[largest] = arr[largest],arr[i]  # swap
      # Heapify the root.
     heapify(arr, n, largest)
# The main function to sort an array of given size
def heapSort(arr):
  n = len(arr)
```

```
    # Build a maxheap.
    # Since the last parent will be at ((n//2)-1) we can start at that location.
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    # One by one extract elements
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]   # swap
        heapify(arr, i, 0)
# Driver code to test above
arr = [ 12, 11, 13, 5, 6, 7]
heapSort(arr)
n = len(arr)
print ("Sorted array is")
for i in range(n):
    print ("%d" %arr[i])
```

Fibonacci search

```
# Python3 program for Fibonacci search.
from bisect import bisect_left

# Returns index of x if present, else
# returns -1
def fibnocian_Search(arr, x, n):
    # Initialize fibonacci numbers
    fibMMm2 = 0 # (m-2)'th Fibonacci No.
    fibMMm1 = 1 # (m-1)'th Fibonacci No.
    fibM = fibMMm2 + fibMMm1 # m'th Fibonacci

    # fibM is going to store the smallest
```

```
# Fibonacci Number greater than or equal to n
while (fibM < n):
    fibMMm2 = fibMMm1
    fibMMm1 = fibM
    fibM = fibMMm2 + fibMMm1

# Marks the eliminated range from front
offset = -1;

# while there are elements to be inspected.
# Note that we compare arr[fibMm2] with x.
# When fibM becomes 1, fibMm2 becomes 0
while (fibM > 1):

    # Check if fibMm2 is a valid location
    i = min(offset+fibMMm2, n-1)

    # If x is greater than the value at
    # index fibMm2, cut the subarray array
    # from offset to i
    if (arr[i] < x):
        fibM = fibMMm1
        fibMMm1 = fibMMm2
        fibMMm2 = fibM - fibMMm1
        offset = i

    # If x is less than the value at
    # index fibMm2, cut the subarray
    # after i+1
```

```
    elif (arr[i] > x):
        fibM = fibMMm2
        fibMMm1 = fibMMm1 - fibMMm2
        fibMMm2 = fibM - fibMMm1

    # element found. return index
    else :
        return i

  # comparing the last element with x */
  if(fibMMm1 and arr[offset+1] == x):
     return offset+1;

  # element not found. return -1
    return -1

# Driver Code
arr = [10, 22, 35, 40, 45, 50,
   80, 82, 85, 90, 100]
n = len(arr)
x = 85
print("Found at index:",
   fibnocian_Search(arr, x, n))
```

BUBBLE SORT

```
# Python program for implementation of Bubble Sort

def bubbleSort(arr):
  n = len(arr)
```

```
  # Traverse through all array elements
  for i in range(n-1):
      for j in range(0, n-i-1):

          # traverse the array from 0 to n-i-1
          # Swap if the element found is greater than the next element
          if arr[j] > arr[j+1] :
              arr[j], arr[j+1] = arr[j+1], arr[j]

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
   print ("%d" %arr[i]),
```

INSERTION SORT

```
# Python program for implementation of Insertion Sort
def insertionSort(arr):
    # Traverse through 1 to len(arr)
   for i in range(1, len(arr)):
```

```
        key = arr[i]
        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >=0 and key < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = key

# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i])
```

QUICKSORT

```
def partition(arr, low, high):
    i = (low-1)         # index of smaller element
    pivot = arr[high]     # pivot
    for j in range(low, high):
        # If current element is smaller than or equal to pivot
        if arr[j] <= pivot:
            # increment index of smaller element
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]
```

```
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)
# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low  --> Starting index,
# high  --> Ending index

def quickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:
        # pi is partitioning index, arr[p] is now at right place
        pi = partition(arr, low, high)
        # Separately sort elements before partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)
# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr, 0, n-1)
print("Sorted array is:")
for i in range(n):
    print("%d" % arr[i])
```

Hamming distance

```

# Function to calculate hamming distance
```

```
def hammingDistance(n1, n2) :

  x = n1 ^ n2
  setBits = 0

  while (x > 0) :
     setBits += x & 1
     x >>= 1

  return setBits

if __name__=='__main__':
  n1 = 9
  n2 = 14
  print(hammingDistance(9, 14))
```

Topological sort

```
#Python program to print topological sorting of a DAG
from collections import defaultdict

#Class to represent a graph
class Graph:
  def __init__(self,vertices):
     self.graph = defaultdict(list) #dictionary containing adjacency List
     self.V = vertices #No. of vertices
```

```
# function to add an edge to graph
def addEdge(self,u,v):
    self.graph[u].append(v)

# A recursive function used by topologicalSort
def topologicalSortUtil(self,v,visited,stack):

    # Mark the current node as visited.
    visited[v] = True

    # Recur for all the vertices adjacent to this vertex
    for i in self.graph[v]:
        if visited[i] == False:
            self.topologicalSortUtil(i,visited,stack)

    # Push current vertex to stack which stores result
    stack.insert(0,v)

# The function to do Topological Sort. It uses recursive
# topologicalSortUtil()
def topologicalSort(self):
    # Mark all the vertices as not visited
    visited = [False]*self.V
    stack =[]

    # Call the recursive helper function to store Topological
    # Sort starting from all vertices one by one
    for i in range(self.V):
        if visited[i] == False:
```

```
            self.topologicalSortUtil(i,visited,stack)

        # Print contents of stack
        print stack

g= Graph(6)
g.addEdge(5, 2);
g.addEdge(5, 0);
g.addEdge(4, 0);
g.addEdge(4, 1);
g.addEdge(2, 3);
g.addEdge(3, 1);

print "Following is a Topological Sort of the given graph"
g.topologicalSort()
```

References

★ Introduction

- Gale–Shapley algorithm simply explained | by Alexander Osipenko
- Gale–Shapley algorithm
- Plagiarism tools and detection techniques
- External and Intrinsic Plagiarism Detection Using Vector Space Models
- Tokenization algorithms in Natural Language Processing (NLP) | by Mehul Gupta | Data Science in your pocket
- Overview of tokenization algorithms in NLP | by Ane Berasategi
- What is Tokenization | Tokenization In NLP
- https://github.com/mutux/kmp/blob/master/kmp.py
- C# Implementation of Knuth–Morris–Pratt Algorithm
- KMP Algorithm for Pattern Searching
- Stemming algorithms
- Porter Stemming Algorithm
- The Porter stemming algorithm
- rolling hash cpp
- DAA Boyer-Moore Algorithm - javatpoint
- Rabin-Karp Algorithm
- Implement Trie (Prefix Tree) in Python
- A Simpler way to implement Trie Data Structure in Python
- Implementing Trie in Python
- Implement Trie (Prefix Tree)
- thinking-tower/Fourier-and-Images: Fourier and Images

- Drawing with epicycles and the Fourier transform
- Epicycles
- Drawing/Rendering 3D objects with epicycles and fourier transformations [Animation]
- How to implement the discrete Fourier transform
- C++ Program to Compute Discrete Fourier Transform Using Naive Approach
- cooley tukey
- Understanding the FFT Algorithm
- algorithm archive
- Signals and Systems Lecture 7: Laplace Transform
- THE LAPLACE TRANSFORM
- Inverse Laplace Transform source code
- https://github.com/Vladimir-Kryzhniy/inversion-of-real-valued-Laplace-transforms/blob/master/InvertLT.ipynb
- A small program to display epicycles with a given image using Fourier Transform.
- Epicycle Demonstrator
- Drawing-with-Fourier-Epicycles: MATLAB GUI
- (PDF) Predictive Model of Graduate-On-Time Using Machine Learning Algorithms
- 10 Clustering Algorithms With Python
- K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks
- Page Rank Algorithm and Implementation
- Dijsktra's algorithm

- A* search algorithm
- R-trees in Data Structure
- Columnar Transposition Cipher
- Simple Substitution Cipher
- RSA (cryptosystem)
- (PDF) Bat Algorithm: Literature Review and Applications
- QR code
- What is dynamic pricing? - Definition from WhatIs.com
- Basics of Hash Tables Tutorials & Notes | Data Structures
- Concurrency in banking
- Regression Analysis: Step by Step Articles, Videos, Simple Definitions
- Stocks and Shares | Concepts and Important Definitions with Examples
- What is a smart city? Technology and examples
- Understanding the Role of AI in Gaming
- Artificial neural network
- Space partitioning algorithm
- Introduction to Minimax Algorithm with a Java Implementation
- What is Alpha Beta Pruning in Artificial Intelligence?