**Module 4**

**Data Structures for SSRAI running on HPC CI**

**Module created by Sirwe Saeedi and Kyle Hammerberg**

**Outline**

* **Introduction**
* **Safe, Secure, and Reliable AI**
* **High Performance Computing**
* **Security in High Performance Computing**
* **Data Structures in AI**
  + - * **Why is data structure important in AI?**
      * **A few applications of data structure in AI/ML**
      * **Machine Learning Algorithms**
* **Problems for Learners**
* **Appendix**

**1. Introduction**

We first discuss the concept of Safe, Secure, and Reliable Artificial Intelligence (SSRAI). The safety, security, and reliability of AI systems help to make accurate and effective decisions. Not only does this learning module help you understand SSRAI, but also its importance on High-Performance Computing (HPC). Identifying HPC safety assurance components, security gaps, and data breaches, and the quality of being reliable for any AI-equipped systems are vital in commercial and industry use cases.

This module aims to offer an intermediate level of understating in SSRAI running on HPC and then emphasizes the applied fundamental data structures as the basis of developing AI algorithms. One should be fully aware of the significance of common data structures used in designing AI algorithms to deliver an effective AI system. So, this module wants to give you a generic view of how suitable and efficient data structures for a specific AI system can be a game changer. In this module, you will learn not only the importance of proper data structures for developing AI or Machine Learning (ML) algorithms with some examples, but also its relationship between SSRAI and HPC.

**2. Safe, Secure, and Reliable AI**

SSRAI stands for Safe, Secure, and Reliable Artificial Intelligence. All three are different but interrelated aspects of AI that lead to dependable or trustworthy AI. Trust is one of the most important and complex activities in human relationships, so proposing that AI should be trusted, is a very serious claim [1].

* Safe: Safety is a crucial part of building AI systems. Technical AI safety is a relatively nascent but rapidly evolving field. Understanding safety is important in the context of fairness, accountability, and explainability for intelligent systems. So, in personal safety, people can have confidence that AI tech will not cause them physical or mental harm.
* Secure: an AI secure system should be designed to prevent unauthorized access by using a secure infrastructure where data and access are locked down. Also, comprehending the possible threats enables people to design and implement changes to secure the application. Anticipating and detecting problems are as important as the designing process. So, people can trust AI tech to maintain privacy and do things like banking, healthcare, etc.
* Reliable: AI is something we can have a reliance on, it must be at least as robust and reliable as the traditional systems. The goal of reliability in AI is that people can rely on AI tech for important tasks. So, a reliable AI system needs to ensure their employed AI algorithms produce the right results for each new data set. There is also a notion of repeatability here. For example, people can have confidence that if an AI technology worked well yesterday, it would work well again when they use it tomorrow.

AI can meet safety, security, and reliability requirements individually but can fail to meet them all simultaneously. However, this is not a type of trust at all, but in fact, a form of reliance. It is noteworthy to mention that the SSR system is not equal to model robustness (fault-tolerant system). To evaluate the safety, security, and reliability of an AI system (or ML system), we need to consider the entire system, and SSR testing is beyond the model. In fact, safety, security, and reliability are the entire system properties, not just the model or software.

In other words, SSR’s concern focuses on how the system interacts with the environment based on outputs from trained models that can be unreliable. So, if you think about model robustness, remember that it does not guarantee the safety, security, and reliability of the system. Robustness can be a component of the entire SSRAI system. This means a reliable system can be robust but vice versa is not necessarily true.

Adversarial example research is a major challenge to robustness and so does the safety, security, and reliability of ML models. Image recognition, natural language processing, and any ML-equipped system can hardly fail in terms of safety, security, and reliability. See Figure 1.

A picture containing text

Description automatically generated

This research [2] shows that it’s possible to manipulate an image that is invisible to the human eyes that which can trick Google image recognition algorithms into classifying them as an entirely different thing. This study recognizes 3D-printed turtle images as a rifle nearly 100% of the time. This example shows how an almost robust system does not ensure safety, security, and reliability.

SSRAI systems are a challenging concept due to the nature of the data dependency of ML models. For instance, biased data and data manipulation can make AI technology totally unreliable. This problem is becoming one of the main focused areas for researchers to consider. Learners who are interested more in adversarial examples of ML and vulnerabilities of AI systems, modules 7, and 10 can give a general understanding of this topic.

**3. High Performance Computing**

Machine learning is a part of AI and computer science which focuses on computer algorithms and the use of data to imitate the way that humans think and learn, it can improve its accuracy gradually through experience and seeing new data. Basically, ML includes tons of computations and mathematical operations. Also, neural networks are another algorithmic approach from the early machine learning crowd that helped AI get closer to General AI and the “deep” in Deep earning (DL) describes all the layers in artificial neural networks which causes heavy computation [3].

The ML workflow consists of four primary steps: First, the preprocessing of input data, next training the DL model, and then storing the trained DL model to deploy the model. Among these steps, training the DL model is by far the most computationally intensive task. The training process often requires passing massive amounts of data down weighted cell gradients. These cell gradients comprise the ‘layers’ of the deep learning model. ML applications spend most of their execution time to computational operation such as computing activation functions for thousands of thousands of neurons and hundreds of neural network layers. Large computational operations and demanding a lot of hardware for ML make GPU (Graphics Processing Unit) an essential part of DL.

A GPU can contain thousands of processing cores that allows it to execute multiple tasks in parallel up to 50 times faster than traditional CPUs. This architecture helps ML/neural networks/DL wonderfully because they are simply a sequence of matrix computations. Your standard, off-the-shelf laptop typically will not include a GPU processor unless it is purchased specifically for something like gaming.

As a machine learning scientist/developer, you may recognize the limitation you will reach during the process of designing and implementing algorithms for resource-intensive tasks. A bottleneck-effect can appear during the training phase of even relatively simple DL models where it takes hours, days, or even years to train a model with a standard laptop. However, with the aid of GPUs or supercomputers the training process can be expedited to a significant degree. Therefore, the role of High-Performance Computing (HPC) in building the infrastructure of ML is just as important as the construction of the DL algorithms themselves when large amounts of data are involved.

A critical question: how many times have you been frustrated by the slow internet? In the world of DL research, the same question can be asked when we were stuck because of the insane number of iterations needed to train a model with many hyper-parameters. Furthermore, there is a continuous inflow of data that needs to be addressed for retraining models. HPC approaches are being used to develop, redesign, and model products, as well as analyze large datasets. An HPC system typically consists of hundreds to thousands of physical servers, each powered by powerful processors and developers implement parallel programming techniques (as opposed to sequential) to allow computationally intensive tasks to run concurrently. HPC servers have many GPU cores and CPU cores working in parallel, permitting complex workloads like a Depp Neural Network (DNN) for machine translation tasks in natural language processing.

Here is [the link](https://www.youtube.com/watch?v=c_55gZfUK1E) for a video from Dr. Andrew Ng talkng about HPC and DL for your reference. The following plot shows the reduction in time to train for a neural machine translation with an HPC approach.

Chart

Description automatically generated with low confidence

**Challenge yourself: Write your first GPU-based project**

So far you have learned why accelerating DL projects is getting more attention and the role of HPC in computationally intensive training is the core of DL project. It is time to practice your own GPU-based project and experience challenges and uses some famous platforms that provide GPU like Crestle, Paperspace, and Google cloud [17, 18, 19]. You can either choose your interested topic and search for available dataset or look up some well-known DL projects ideas, [here](https://data-flair.training/blogs/deep-learning-project-ideas/) and [here](https://www.upgrad.com/blog/exciting-deep-learning-project-ideas-for-beginners/).

If you are interested more in learning DL with HPC and bottlenecks discovery on general-purpose CPU architectures, module 5 can give you a fair understanding of this topic.

**4. Security in High Performance Computing**

As you saw in section 2 of this module, the safety, security, and reliability of an AI system determines whether its deployment will be successful. Providing security in HPC is one part of a secure AI system and it is a challenging task. Internet, operating systems, and distributed environments may suffer from poor security support and cannot resist common attacks. The threats to HPC often involve breaches of confidentiality (e.g., data leakage), integrity such as alteration of code or data, misuse of computing cycles, availability, and disruption of service against HPC systems or networks that connect them. HPC systems face many of the same challenges that standard IT systems face, but there are challenges unique to HPC. HPC cannot use certain security solutions, such as network firewalls, in the same way as other IT systems.

The nature of security for HPC is different because they tend to be used for very distinctive purposes, notably mathematical computations. HPC systems use different hardware and software which are widely open to users in commercial, industrial and consumer applications like product design, financial analytics, personalized medicine, marketing automation, fraud detection, and autonomous vehicle. ￼All these characteristics make HPC security challenging.

Many researchers investigate the importance of security in HPC [12, 13, 16] and some organizations work on HPC security and its solution [14, 15]. The threats to HPC systems are real like what happened before in [20, 21]. Therefore, monitoring HPC systems are important to detect abnormal behavior, unplanned manipulated programs, and misuse of cycles.

**5. Data Structures in AI**

In this module, we assume that learners have a basic knowledge of data structures, however, a few resources are provided [8, 9] for learners to do a quick recap of basic data structures in algorithm design.

Having machine learning skills alone for designing an efficient AI system is not enough, a good knowledge of data structures will help ML practitioners to understand under the hood of an AI system. But what exactly is a data structure? Data structures are a way to organize data into memory or other storage devices so that they can be efficiently manipulated by the target algorithm. This structure should be efficient in time/space complexity for the storage and retrieval of data. Data structures used in machine learning are not significantly different from those used in other areas of software development or algorithm design. Every problem needs input, and the input can be any type and size data which will be stored in memory (random access memory, cache, etc.) or external storage such as a local disk or NAS. Data needs to have the appropriate structure to allow for storage, as well as easy access to make the necessary manipulations.

A few of the most common data structures in ML include arrays, linked lists, stacks, queues, trees, and graphs. Because creating ML models often requires substantial amounts of complex mathematical computation, it is important that we understand how data structures can be used to solve problems efficiently. The main reason that makes data structure important to ML is that we can implement vectors, matrices, and tensors which are the critical components of training phase of learning models via all data structures. It would not be wrong to say that basic array is the most important data structure in ML. Clustering data or s statistical classification problems remind you of the importance of good working knowledge of data structures. The most common types of data structures in linear algebra (a branch of mathematics that is foundational to Machine Learning) are the one- and two-dimensional arrays, corresponding to vectors and matrices respectively, though you may occasionally encounter three- or four-dimensional arrays either for higher ranked tensors or to group examples of the former.

**5.1 Why is data structure important in AI?**

Although it is not crucial to learn data structures and algorithms to learn machine learning, it is important when you want to build and train your own neural network instead of using pre-built models like publicly available models. The practical and functional understanding of the models helps to digest cutting-edge machine learning algorithms.

If we deal with a huge dataset with millions to billions of data items working on thousands of lines of code, then data structures and algorithms play a key role in efficiency (storage space and time). With an insufficient knowledge of data structures, our AI application may take a lifetime to produce a result or even never complete. For example, pandas dataframe is a common library to use among data scientist for preprocessing time series datasets, however, using a data structure which is called hash table (supports time complexity of O(1) insert and search operation) can make an enormous difference in running time in this type of problems.

By having programming skills in Python, we can implement hash tables in Python by dictionaries that is many times faster than naïve implementation with no knowledge of data structures.

**5.2 A few applications of data structure in AI/ML**

**A\* Search Algorithm**

A\* algorithm is a searching algorithm in AI that searches for the shortest path between the start and the final state. It is used in various applications such as web-based maps and [pathfinding](https://en.wikipedia.org/wiki/Pathfinding" \o "Pathfinding) problems in video games. If you look at the implementation of the A\* algorithm (for instance in Python), you will notice that a variety of data structures are used. The A\* Algorithm works on vertices of a graph which start with the starting vertex as the starting point of the wanted shortest path and then repeatedly examines the next neighbor vertex which is unvisited. Each time, we add visited vertices to the set of vertices that are examined.  Lists, dictionaries, and sets are the basic data structures used when implementing the A\* algorithm in Python. The below GIF shows the execution of A\* algorithm which starting point is green vertex, goal point is blue vertex, and all orange vertices are visited vertices along the shortest path.

Chart

Description automatically generated

See Appendix for A\* program in Python. Challenge yourself and try to write your own code as the way of thinking leads to different programming and it can create more efficient ones.

**Graphs/Trees**

Another example of data structure’s application is in designing algorithms like dynamic programming, which is used mainly in reinforcement learning, randomized algorithms. Dynamic programming in dropout for deep learning, training linear classifier, and gradient/ stochastic algorithms is essential specially for optimization problems. Dynamic programming is also suitable for applications where decision processes are critical in a highly uncertain environment like Markov decision process.

Furthermore, graphs can be ingested by various models of neural networks such as classification, and clustering (see section 5.2 for more details). Sophisticated AI applications might use directed or undirected graphs, which could be thought of as the generalizations of trees and linked lists. Knowledge graphs in Natural Language Processing are used to represent the relationship between a group of real-world entities. The nodes of the knowledge graph represent the individual entities being modeled (these could be living beings, concepts, ideas, locations, etc.) and the connections between the nodes represent the ways the entities relate to each other. A knowledge graph is a good representation for unstructured data such as texts in the question answering task (QA is a computer science discipline within the field of NLP, which is the task of answering a question by building systems in a [natural language](https://en.wikipedia.org/wiki/Natural_language" \o "Natural language)), information extraction, and commonsense reasoning. The figure below is an example of graph knowledge in a commonsense reasoning problem that connects entities of an event (“X repels Y’s attack”) [].

Diagram

Description automatically generated

Also, graphs including nodes and edges can be converted into a vector of real numbers (like an array, list, etc. which are basic data structures) and then we need to traverse the graph to arrange the vectors. These ordered vectors are stored into a matrix (high-dimension matrix called tensor). Matrix multiplication then becomes tensor contraction. Tensors represent important information within the graph, and they can function as inputs for DNN or ML models.

**Tensors**

There are many powerful deep learning libraries and frameworks - Tensorflow, Pytorch, Keras, Apache MXNet, to name a few. These frameworks use different data structures in their built-in functions. One of the common data structures in most of these frameworks is tensors. As mentioned above, Tensors are a generalization of scalars, vectors, and matrices to higher dimensions. If you are familiar with basic linear algebra, you have encountered tensors of at least one (vector) or two (matrix) dimensions. It is worth mentioning that the tensor is dynamic data structure (i.e., a structure of data that has the flexibility to grow or shrink in size to control exactly how much memory is utilized). Matrices, on the other hand, are simply two-dimensional tensors.

Google’s machine learning framework Tensorflow uses tensors as the basic unit for calculation. Information in an image of video can be represented in a tensor. In TensorFlow, computation is described using “data flow graphs” kind of a computational graph. Each node of the graph represents a mathematical operation (like addition or multiplication), and each edge is a tensor. So, the combination of nodes and edges represents a mathematical expression that again produces a tensor.

The following table shows commonly used data structures and their application in ML.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Structures | Definition | ML Applications | Visualization |
| Array | An assemblage of items that are randomly accessible by an index of integers. All elements of an array must be of the same type. | Used to represent images, text documents, and many other types of data. Likely the most common data structure used in Machine Learning pipelines. | Diagram  Description automatically generated |
| Tree-structure | Tree-based data structures are non-linear, hierarchical data structures that contain a series of nodes that are connected by things called edges. In tree-based structure, if we have N nodes, we must have (N-1) edges (i.e., one edge for each parent-child relationship) | Trees have found widespread use in the ML community. They are excellent for representing hierarchical clusters and classifications. | A picture containing necklet  Description automatically generated |
| Graphs | Graphs are unidirectional generalizations of trees and linked lists. | Graphs are excellent choices for representing linked relationships between elements. Neural networks are typically represented using a graph structure. | Diagram  Description automatically generated |
| Dictionaries | A general-purpose data structure for storing a group of objects. A dictionary has a set of keys, and each key has a single associated value. When presented with a key, the dictionary will return the associated value. | Dictionary is a good representative for hash table, for example, in preprocessing time series datasets. | Diagram  Description automatically generated |
| Stacks | Stacks are linear data structures that have a particular order in which operations are performed on the data. Stacks can be conceptualized by imaging the removal of a poker chip from the top of a stack and replacing it with a new chip. | For instance, the libAGF library uses a **recursive control language** to generalize binary classification to multi-class. A special character is used to repeat a previous option, but because the language is recursive, the option must be taken from the same hierarchical level or higher. | Diagram  Description automatically generated |
| Queues | Queues are almost just like stacks except that the insertion and deletion of elements occur on different ends of the linear structure (e.g., taking a poker chip off the bottom of a stack of chips and putting a new one on the top). | One application example in the context of big data is scheduling queues. These enable different end users of large computing systems to use computing systems in a fair manner. End users submit their computing job to a queue and depending on the scheduling policy will see their job soon on the computing system. | Timeline  Description automatically generated |
| Linked list | The linked list is a linear data structure whose order is not given by their physical placement in memory, rather each element ‘points’ to the next element in the list. | Linked lists are useful **for parsing lists of indeterminate length.** They can be converted to fixed-length arrays for fast access. A fast and efficient way for conversion to an array. | Diagram, schematic  Description automatically generated |
| Sets | Sets are unordered lists of non-repeating elements. Duplicate elements cannot be added to a set, but the same element can belong to multiple different sets. | If you add an element that’s already in the set, there will be no change. Since much of the mathematics of machine learning deals with sets, they are very useful data structures. | Diagram, venn diagram  Description automatically generated |

**5.3 Machine Learning Algorithms**

Artificial intelligence algorithms can be broadly classified as:

Classification Algorithms :  
 1. Naive bayes  
 2. Decision Tree  
 3. Random Forest  
 4. Support Vector Machines  
 5. K Nearest Neighbors

Regression Algorithms:  
 1. Linear regression

2. Logistic Regression

3. Multivariate Regression

4. Multiple Regression Algorithm

Clustering Algorithms:  
 1. K-Means Clustering  
 2. Fuzzy C-means Algorithm  
 3. Agglomerative Clustering

4. Gaussian Mixture Modeling (GMM)

**Regression Algorithm Data Structures**

Perhaps the most used data structure for performing linear regression is the **array**. Arrays are typically either 1 or 2 dimensional, ‘non-primitive’ structures (called vectors and matrices, respectively). Arrays have numerous advantages that make them a suitable choice for the construction of regression-based models. Arrays are easy to understand and simple to implement. Over the years, the array has proven itself to be one of the most effective and reliable data structures because it allows for the easy storage of indexed values. Arrays are simple to navigate, sort, and manipulate. Arrays are not without their disadvantages, however. The data structure is homogenous, meaning only one element type can be stored per array, so arrays must be entirely composed of, for example, integers or strings, but never a mix of both. [23]

A few examples of array-specific vulnerabilities in Javascript include out of bound array indexing, zero length array vulnerability, and negative length array vulnerability.

* **Out of bound array indexing** results from the use of the array length function in the initialization or in any condition checking of a loop. Improper indexing may lead to buffer overflow which creates breaching opportunities. When buffer overflow occurs, Java applets gain increased privileges, opening the possibility of attackers gaining access to sensitive files.
* **Zero length array vulnerabilities** arise when an array of length 0 is executed. This can lead to unreleased resource attacks.
* **Negative length array vulnerability** canoccur when an array with a negative value or negative length is initialized. This can lead to buffer / integer overflow or denial of service attacks. [22]

**Clustering Algorithm Data Structures**

The most common distinction made when discussing clustering techniques is that of hierarchical versus partitional clustering. Partitional clustering involves the splitting of the data set into non-overlapping clusters. In the figure below, (b) represents 2 instances of partitional clustering, (c) represents 4 instances of partitional clustering, and (d) represents 6 instances of partitional clustering. The key distinction to be made when conceptualizing partitional clustering is that each cluster is its own subset, with no sets within itself [25].

Chart, scatter chart

Description automatically generated

**Hierarchical Tree-based Clustering**

Nesting clusters within clusters is referred to as hierarchical clustering**.** Hierarchical clustering can be represented with a tree data structure (also referred to as a *hierarchical* tree structure). Each node represents the set of all its subclusters, with the base or ‘root’ of the tree containing the set of all sets, and the ‘leaves’ of the tree containing all of the ‘singletons’, or individual sets. In the figure above, (d) is partitioned off into 6 clusters, but those 6 clusters could be represented as two nested clusters, each containing 3 subclusters, thus creating a hierarchical tree structure. [25]

Pictured below is a simple hierarchical tree structure with variables *x, y, z,* and *t* representing individual sets. The root node at the top of the tree structure represents the set of all sets (remember tree diagrams are typically inverted, with the root node at the top and the leaves at the bottom), and then the sets are partitioned off into smaller sets of sets until the data has been clustered to a sufficient degree.

Bubble chart

Description automatically generated with medium confidence

**Contiguity-based clustering**

Contiguity-based clustering, sometimes referred to as ‘nearest neighbor’ clustering can be represented by an abstract graph structure where the nodes are various objects or sets of objects, and the links between the objects represent the connections between them. A common technique for partitioning data in this manner employs Euclidean distance as a measure of similarity. [25]

Chart, diagram

Description automatically generated

**Classification Algorithm Data Structures**

Classification algorithms frequently make use of various n-dim arrays and tree structures. The data to be classified may initially present itself in the form of various arrays or matrices. For example, when classifying the famous iris data set, the analyst is presented with a matrix consisting of four feature columns and a species column. These individual columns could all be thought of as vectors, each containing 150 elements.

One way to go about beginning the classification process is to use the Pandas library in Python to convert the data into a special matrix called a *Pandas DataFrame.* This allows the data to be separated into a feature matrix of independent (X) variables and a target vector with the dependent (Y) variable. In this case, the target vector contains integer elements (0, 1, or 2) representing each of the three iris species, and the feature matrix is composed of multiple column vectors containing float values representing measurements for each of the four chosen features (sepal length, sepal width, petal length, and petal width).

You may be asking yourself why we are taking raw data in the form of a matrix and then converting it into essentially another matrix. What we are discussing here is simply one method for preparing the data for analysis. Converting the data from a raw CSV file to a Pandas DataFrame simply puts it into a form that Python can understand for the specific task being performed. Although the data structures are the same in an abstract sense, they have now taken on a form that allows us to use another Python library *sklearn* to train the classification model. The actual classification of the data can then be represented with an abstract tree structure with nodal splits representing the classification splits.

Applied Machine Learning often requires the use of multiple data structures, sometimes even moving between different forms of the same abstract structure (e.g., moving from a CSV matrix to a Pandas DataFrame) as the analyst moves through the process of using ML to glean insights from sets of data.

**6. Problems for learners**

Try to implement a program and solve a real problem to practice and realize data structures for ML algorithm yourself. Here are some ideas:

1. Implement your favorite data structure in your favorite language.
2. Implement a TreeSort and a HeapSort. Now use the same data structures to find the top *k* elements. This data structure can be used in which machine learning algorithm
3. Implementing a simple linear regression. Determine data structures that you used.
4. What type of data structure is better suited for representing hierarchical data, linear or non-linear? List 3 examples of that type of data structure.
5. Try out the iris classification exercise included with this module to familiarize yourself with the implementation of some of the data structures studied in this module

**References**

1. “In AI We Trust: Ethics, Artificial Intelligence, and Reliability”, Mark Ryan

Science and Engineering Ethics, 2020.

2. “A\* search algorithm”*,* [*http://www.ccpo.odu.edu/~klinck/Reprints/PDF/wikipediaNav2018.pdf*](http://www.ccpo.odu.edu/~klinck/Reprints/PDF/wikipediaNav2018.pdf)

3. “CS224W: Machine Learning with Graphs | 2021 | Lecture 1.1 - Why Graphs,’’

4. “Graph Matching Networks for Learning the Similarity of Graph Structured Objects”, [Yujia Li](https://arxiv.org/search/cs?searchtype=author&query=Li%2C+Y), [Chenjie Gu](https://arxiv.org/search/cs?searchtype=author&query=Gu%2C+C), [Thomas Dullien](https://arxiv.org/search/cs?searchtype=author&query=Dullien%2C+T), [Oriol Vinyals](https://arxiv.org/search/cs?searchtype=author&query=Vinyals%2C+O), [Pushmeet Kohli](https://arxiv.org/search/cs?searchtype=author&query=Kohli%2C+P)

5. “A Comprehensive Survey on Graph Neural Networks”, [Zonghan Wu](https://arxiv.org/search/cs?searchtype=author&query=Wu%2C+Z), [Shirui Pan](https://arxiv.org/search/cs?searchtype=author&query=Pan%2C+S), [Fengwen Chen](https://arxiv.org/search/cs?searchtype=author&query=Chen%2C+F), [Guodong Long](https://arxiv.org/search/cs?searchtype=author&query=Long%2C+G), [Chengqi Zhang](https://arxiv.org/search/cs?searchtype=author&query=Zhang%2C+C), [Philip S. Yu](https://arxiv.org/search/cs?searchtype=author&query=Yu%2C+P+S)

6. “Deep Neural Networks for Learning Graph Representations”, [Shaosheng Cao](https://www.semanticscholar.org/author/Shaosheng-Cao/2105817), [Wei Lu](https://www.semanticscholar.org/author/Wei-Lu/143844110), [Qiongkai Xu](https://www.semanticscholar.org/author/Qiongkai-Xu/3101288)

7. “Deep Feature Learning for Graphs”, [Ryan A. Rossi](https://arxiv.org/search/stat?searchtype=author&query=Rossi%2C+R+A), [Rong Zhou](https://arxiv.org/search/stat?searchtype=author&query=Zhou%2C+R), [Nesreen K. Ahmed](https://arxiv.org/search/stat?searchtype=author&query=Ahmed%2C+N+K)  
8. Introduction to Algorithms by Thomas H. Cormen9. Algorithms by Robert Sedgewick & Kevin Wayne

10. <https://pandas.pydata.org/pandas-docs/stable/user_guide/enhancingperf.html#using-numba>

11. <https://docs.scipy.org/doc/scipy/reference/sparse.html>

12. <https://science.osti.gov/-/media/ascr/pdf/program-documents/docs/Cyber_security_science_dec_2008.pdf>

13. https://www.nsf.gov/pubs/2021/nsf21512/nsf21512.htm

14. <https://science.osti.gov/-/media/ascr/pdf/programdocuments/docs/ASCR_Cybersecurity_20_Research_Pathways_and_Ideas_Workshop.pdf>

15. <https://csrc.nist.gov/Events/2016/NSCI-High-Performance-Computing-Security-Workshop>

16. <https://cacm.acm.org/magazines/2017/9/220422-security-in-high-performance-computing-environments/fulltext>

17. <https://crestle.ai/>

18. <https://console.paperspace.com/machines>

19. <https://cloud.google.com/>

20. Stoll, C. Stalking the wily hacker. *Commun. ACM 31*, 5 (May 1988), 484497.  
  
21. Marko, J. and Bergman, L. Internet attack is called broad and long lasting. *New York Times* (May 10, 2005).

Image Sources:

Array: <https://javatutorial.net/java-arrayhttps://javatutorial.net/java-array>

Tree structure: <https://phil.lavin.me.uk/tag/dns-tree-structure/>

Graph structure: https://i.stack.imgur.com/gjJkX.jpg

Dictionary: <https://medium.com/python-pandemonium/python-dictionaries-45cacc2b76aa>

Stack and queue: <https://tutsmaster.org/how-to-represent-stack-in-c-data-structure/>

Set: <https://archive.lib.msu.edu/crcmath/math/math/v/v054.htm>

Linked list: <https://vitalflux.com/blockchain-linked-list-like-data-structure/>

**Appendix**

**Pseudo-code of A\* algorithm**

let openList equal empty list of nodes

let closedList equal empty list of nodes

put startNode on the openList (leave its f at zero)

while openList is not empty

let currentNode equal the node with the least f value

remove currentNode from the openList

add currentNode to the closedList

if currentNode is the goal

You've found the exit!

let children of the currentNode equal the adjacent nodes

for each child in the children

if child is in the closedList

continue to begin of for loop

child.g = currentNode.g + distance b/w child and current child.h = distance from child to end

child.f = child.g + child.h

if child.position is in the openList's nodes positions

if child.g is higher than the openList node's g

continue to begin of for loop

add the child to the openList

f(n) = g(n) + h(n)  
Where  
g(n) : The actual cost path from the start node to the current node.   
h(n) : The actual cost path from the current node to goal node.  
f(n) : The actual cost path from the start node to the goal node.

Diagram

Description automatically generated

Path found: ['A', 'B', 'D']