A

LABORATORY MANAUL

On

"PROGRAMMING LAB I"

SEMESTER – (I)

Prepared by: Prof. Talekar S. A.

Checked by: Prof.Shinde.P.P

NAME OF LABORATORY : Lab Practice I

DEPARTMENT OF COMPUTER ENGINEERING

Nashik District Maratha Vidya Prasarak Samaj's Karmaveer

Adv. Baburao Ganpatrao Thakare College of Engineering

Nashik

LAB PRACTICALS

BASED ON

HIGH PERFORMANCE

COMPUTING

Assignment No.1

Title: a) Implement Parallel Reduction using Min, Max, Sum and Average operations.
b) Write a CUDA program that, given an N-element vector, find-

- The maximum element in the vector
- The minimum element in the vector
- The arithmetic mean of the vector
- The standard deviation of the values in the vector

Test for input N and generate a randomized vector V of length N (N should be large). The program should generate output as the two computed maximum values as well as the time taken to find each value.

Objective: To study and implementation of directive based parallel programming model. To study and implement the operations on vector, generate o/p as two computed max values as well as time taken to find each value

Outcome: Students will understand the implementation of sequential program augmented with compiler directives to specify parallelism. Students will understand the implementation of operations on vector, generate o/p as two computed max with respect to time.

Pre-requisites: 64-bit Open source Linux or its derivative, Programming Languages: C/C++,CUDA Tutorials.

Hardware Specification: x86_64 bit, 2 – 2/4 GB DDR RAM, 80 - 500 GB SATA HD, 1GB NIDIA TITAN X Graphics Card.

Software Specification: Ubuntu 14.04, GPU Driver 352.68, CUDA Toolkit 8.0, CUDNN Library v5.0

Theory:

Introduction:

OpenMP:

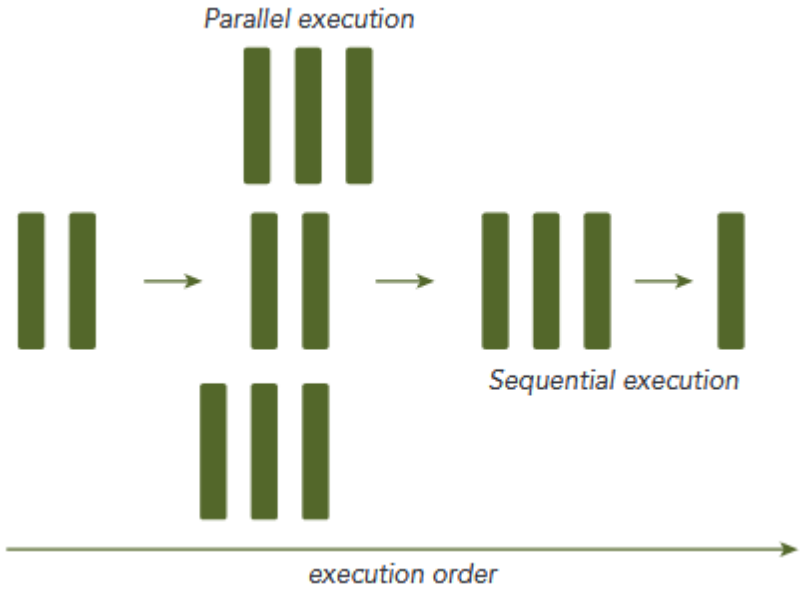
OpenMP is a set of C/C++ pragmas (or FORTRAN equivalents) which provide the programmer a high-level front-end interface which get translated as calls to threads (or other similar entities). The key phrase here is "higher-level"; the goal is to better enable the programmer to "think parallel," alleviating him/her of the burden and distraction of dealing with setting up and coordinating threads. For example, the OpenMP directive.

Parallel Programming:

There are two fundamental types of parallelism in applications:

➤Task parallelism
➤Data parallelism

Task parallelism arises when there are many tasks or functions that can be operated independently and largely in parallel. Task parallelism focuses on distributing functions across multiple cores.
Data parallelism arises when there are many data items that can be operated on at the same time. Data parallelism focuses on distributing the data across multiple cores.



CUDA:

CUDA programming is especially well-suited to address problems that can be expressed as data-parallel computations. Any applications that process large data sets can use a data-parallel model to speed up the computations. Data-parallel processing maps data elements to parallel threads. The first step in designing a data parallel program is to partition data across threads, with each thread working on a portion of the data.

CUDA Architecture:

A heterogeneous application consists of two parts:

- ➤Host code
- ➤Device code

Host code runs on CPUs and device code runs on GPUs. An application executing on a heterogeneous platform is typically initialized by the CPU. The CPU code is responsible for managing the environment, code, and data for the device before loading compute-intensive tasks on the device. With computational intensive applications, program sections often exhibit a rich amount of data parallelism. GPUs are used to accelerate the execution of this portion of data parallelism. When a hardware component that is physically separate from the CPU is used to accelerate computationally intensive sections of an application, it is referred to as a hardware accelerator. GPUs are arguably the most common example of a hardware accelerator. GPUs must operate in conjunction with a CPU-based host through a PCI-Express bus, as shown in Figure.

NVIDIA's CUDA nvcc compiler separates the device code from the host code during the compilation process. The device code is written using CUDA C extended with keywords for labeling data-parallel functions, called kernels . The device code is further compiled by Nvcc. During the link stage, CUDA runtime libraries are added for kernel procedure calls and explicit GPU device manipulation. Further kernel function, named helloFromGPU, to print the string of "Hello World from GPU!" as follows:

```
__global__ void helloFromGPU(void)
{
printf("Hello World from GPU!\n");
}
```

The qualifier __global__tells the compiler that the function will be called from the CPU and executed on the GPU. Launch the kernel function with the following code:

```
helloFromGPU <<<1,10>>>();
```

Triple angle brackets mark a call from the host thread to the code on the device side. A kernel is executed by an array of threads and all threads run the same code. The parameters within the triple angle brackets are the execution configuration, which specifies how many threads will execute the kernel. In this example, you will run 10 GPU threads.

A typical processing flow of a CUDA program follows this pattern:
1. Copy data from CPU memory to GPU memory.
2. Invoke kernels to operate on the data stored in GPU memory.
3. Copy data back from GPU memory to CPU memory.

Conclusion: We have implemented parallel reduction using Min, Max, Sum and Average Operations. We have implemented CUDA program for calculating Min, Max, Arithmetic mean and Standard deviation Operations on N-element vector.

Assignment NO: 02

Title: Design & implementation of Parallel (CUDA) algorithm to Add two large Vector, Multiply Vector and Matrix and Multiply two N × N arrays using n^2.

Outcome: To offload parallel computations to the graphics card, when it is appropriate to do so, and to give some idea of how to think about code running in the massively parallel environment presented by today's graphics cards.

Outcome: Students should understand the basic of GPU computing in the CUDA environment.

Prerequisites: Strassen's Matrix Multiplication Algorithm and CUDA Tutorials.

Hardware Specification: x86_64 bit, 2 – 2/4 GB DDR RAM, 80 - 500 GB SATA HD, 1GB NIDIA TITAN X Graphics Card.

Software Specification: Ubuntu 14.04, GPU Driver 352.68, CUDA Toolkit 8.0, CUDNN Library v5.0

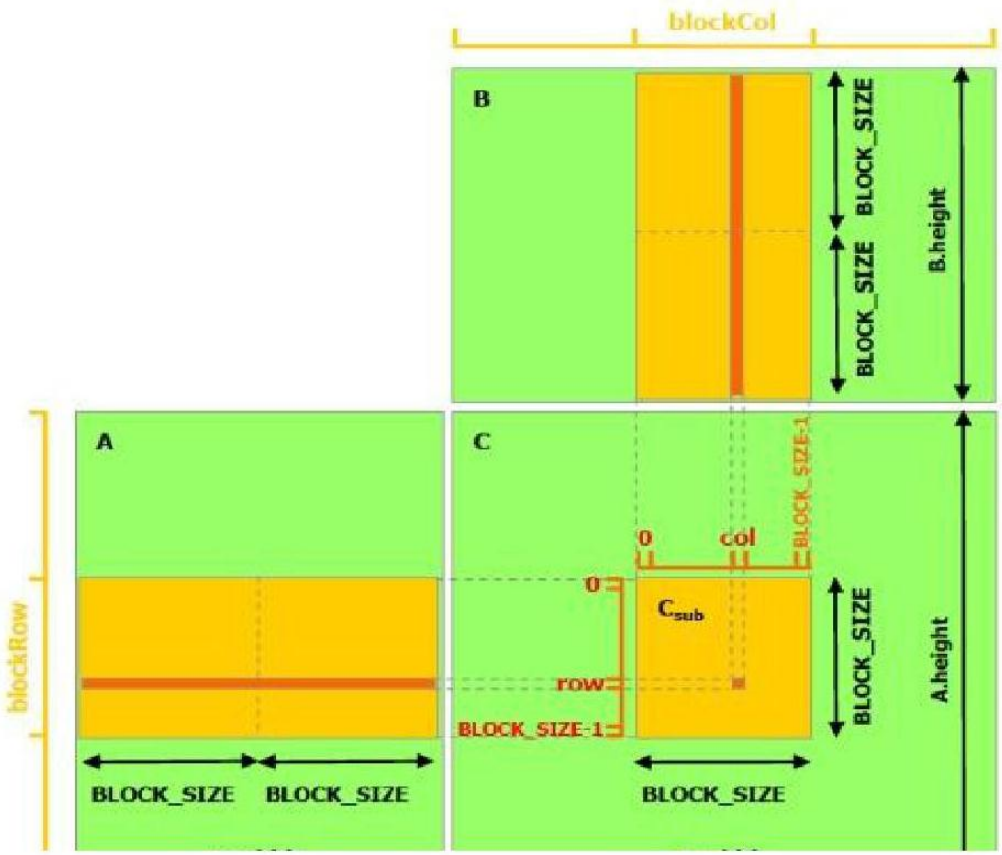
Introduction:

It has become increasingly common to see supercomputing applications harness the massive parallelism of graphics cards (Graphics Processing Units or GPUs) to speed up computations. One platform for doing so is NVIDIA's Compute Unified Device Architecture (CUDA). We use the example of Matrix Multiplication to introduce the basics of GPU computing in the CUDA environment.
Matrix Multiplication is a fundamental building block for scientific computing. Moreover, the algorithmic patterns of matrix multiplication are representative. Many other algorithms share similar optimization techniques as matrix multiplication. Therefore, matrix multiplication is one of the most important examples in learning parallel programming.
A kernel that allows host code to offload matrix multiplication to the GPU. The kernel function is shown below,

```
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C) {
  // Each thread computes one element of C
  // by accumulating results into Cvalue
  float Cvalue = 0;
  int row = blockIdx.y * blockDim.y + threadIdx.y;
  int col = blockIdx.x * blockDim.x + threadIdx.x;
  if(row > A.height || col > B.width) return;
  for (int e = 0; e < A.width; ++e)
    Cvalue += A.elements[row * A.width + e] *
                            B.elements[e * B.width + col];
  C.elements[row * C.width + col] = Cvalue;
}
```

The first line contains the **__global__** keyword declaring that this is an entry-point function for running code on the device. The declaration float **Cvalue = 0** sets aside a register to hold this float value where we will accumulate the product of the row and column entries. The next two lines help the thread to discover its row and column within the matrix. It is a good idea to make sure you understand those two lines before moving on. The ***if*** statement in the next line terminates the thread if its row or column place it outside the bounds of the product matrix. This will happen only in those blocks that overhang either the right or bottom side of the matrix. The next three lines loop over the entries of the row of **A** and the column of **B** (these have the same size) needed to compute the (row, col)-entry of the product, and the sum of these products are accumulated in the **Cvalue** variable. Matrices **A** and **B** are stored in the device's global memory in row major order, meaning that the matrix is stored as a one-dimensional array, with the first row followed by the second row, and so on. Thus to find the index in this linear array of the **(i, j)** entry of matrix **A.** Finally, the last line of the kernel copies this product into the appropriate element of the product matrix C, in the device's global memory.

In light of the memory hierarchy described above, each threads loads **(2 × A. width)** elements in Kernel .



From global memory — two for each iteration through the loop, one from matrix **A** and one from matrix **B**. Since accesses to global memory are relatively slow, this can bog down the kernel, leaving the threads idle for hundreds of clock cycles, for each access.
Matrix **A** is shown on the left and matrix **B** is shown at the top, with matrix **C**, their product, on the bottom-right. This is a nice way to lay out the matrices visually, since each element of **C** is the product of the row to its left in **A** and the column above it in **B**. In the above figure, square thread blocks of dimension **BLOCK_SIZE × BLOCK_SIZE** and will assume that the dimensions of **A** and **B** are all multiples of **BLOCK_SIZE**. Again, each thread will be responsible for computing one element of the product matrix **C**.

It decomposes matrices **A** and **B** into non-overlapping submatrices of size **BLOCK_SIZE × BLOCK_SIZE**. It shows in above figure in red row and red column. It passes through the same number of these submatrices, since they are of equal length. If it load the left-most of those submatrices of matrix **A** into shared memory, and the top-most of those submatrices of matrix **B** into shared memory, then it compute the first **BLOCK_SIZE** products and add them together just by reading the shared memory. But here is the benefit, as long as it have those submatrices in shared memory, every thread's thread block (computing the BLOCK_SIZE × BLOCK_SIZE submatrix of C) can compute that portion of their sum as well from the same data in shared memory. When each thread has computed this sum, it loads the next **BLOCK_SIZE × BLOCK_SIZE** submatrices from **A** and **B**, and continue adding the term-by-term products to our value in **C**.

Applications:

1. Fast Video Trans-coding & Enhancement.
2. Medical Imaging.
3. Neural Networks.
4. Gate-level VLSI Simulation.

Conclusion: Strassen's Matrix Multiplication Algorithm have been implemented parallel using GPU computing in the CUDA environment

Assignment NO: 03

Title: Design & implementation of Parallel (OpenMP) Sorting algorithms (Bubble Sort & Merge Sort) based on existing sequential algorithms.

Objectives: To offload parallel computations to the graphics card, when it is appropriate to do so, and to give some idea of how to think about code running in the massively parallel environment presented by today's graphics cards.

Outcome: Students should understand the basic of OpenMP Programming Module using existing Sorting techniques.

Prerequisites: Data Structure and Files, Design and Analysis of Algorithm, OpenMP Tutorials.

Hardware Specification: x86_64 bit, 2 – 2/4 GB DDR RAM, 80 - 500 GB SATA HD.

Software Specification: Ubuntu 14.04, G Edit, GCC Complier.

Introduction:

An Application Program Interface **(API)** that may be used to explicitly direct multithreaded, shared memory parallelism. An Open Multi-Processing (**OpenMP)** consists of a set of compiler ***#pragmas*** that control how the program works. The **pragmas** are designed so that even if the compiler does not support them, the program will still yield correct behavior, but without any parallelism. Sorting is the process of putting data in order; either numerically or alphabetically. It is necessary to arrange the elements in an array in numerical or lexicographical order, sorting numerical values in descending order or ascending order and alphabetical value like addressee key. Many existing sorting algorithms were observed in terms of the efficiency of the algorithmic complexity. All sorting algorithms are appropriate for specific kinds of problems. Some sorting algorithms work on less number of elements, some are used for huge number of data, some are used if the list has repeated values, and some are suitable for floating point numbers. Sorting is used in many important applications and there have been a plenty of performance analysis.

There are several sorting algorithms available to sort the elements of an array. Some of the sorting algorithms are,

A. **Bubble Sort**

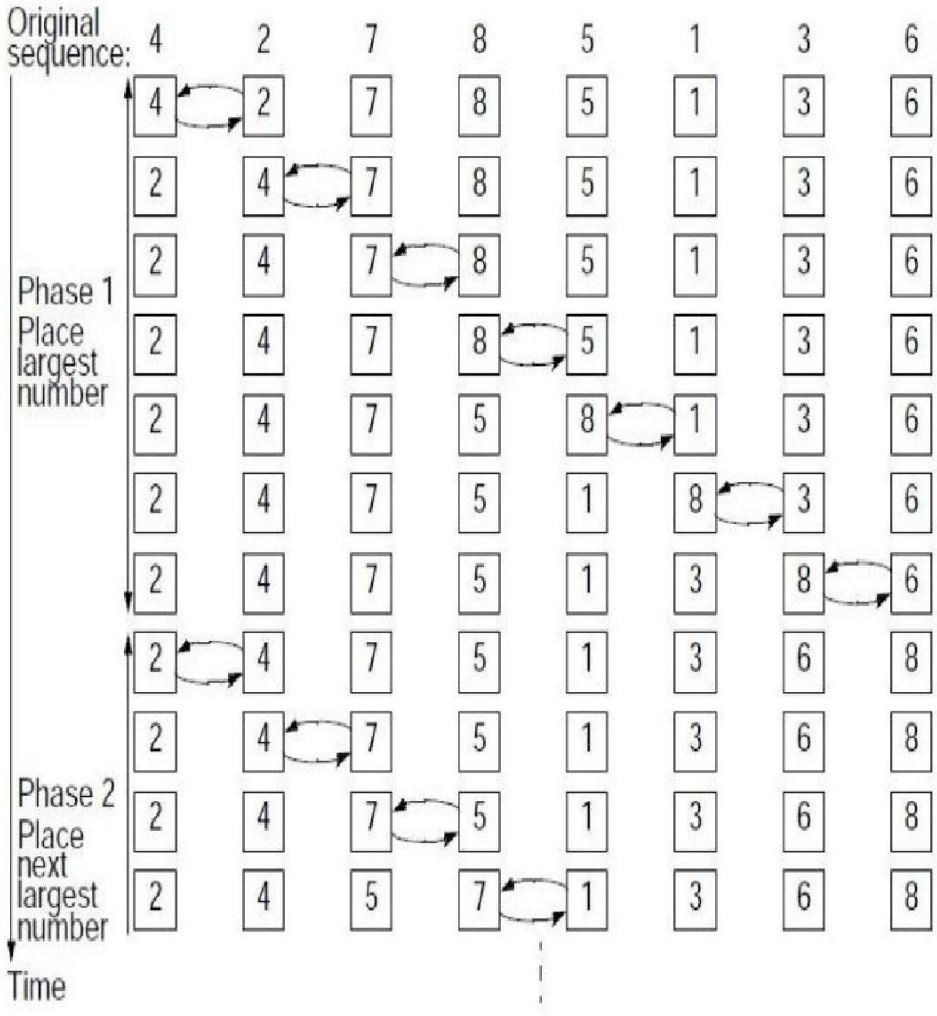
B. **Merge Sort.**

Introduction:

A. Bubble Sort:

In bubble sort, the largest number is first moved to the very end of the list by a series of

compare-and-exchange operations, starting at the opposite end. The procedure repeats, stopping just before the previously positioned largest number, to get the next-largest number. In this way, the larger numbers move (like a bubble) toward the end of the list.

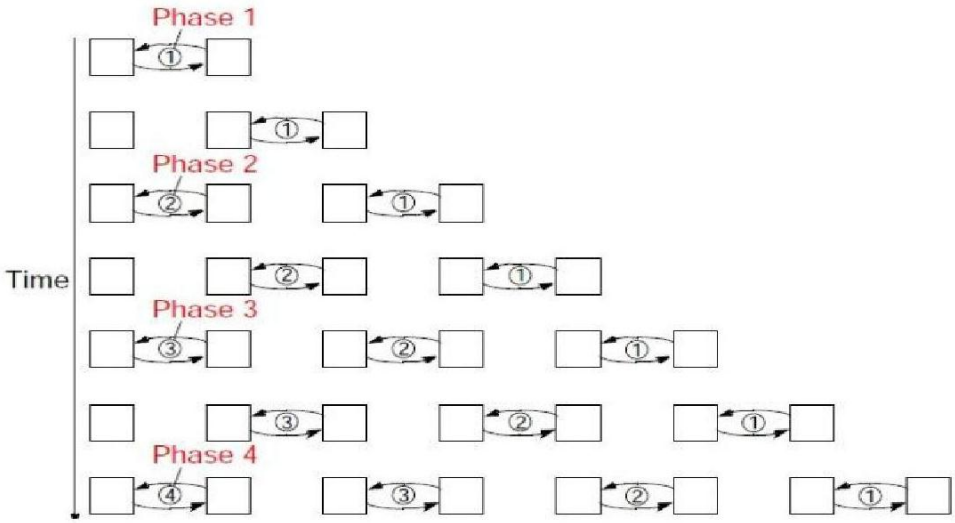



```
for (i = N - 1; i > 0; i--)
for (j = 0; j < i; j++) {
k = j + 1;
if (a[j] > a[k]) {
temp = a[j];
a[j] = a[k];
a[k] = temp;
}}
```

Parallel Bubble Sort

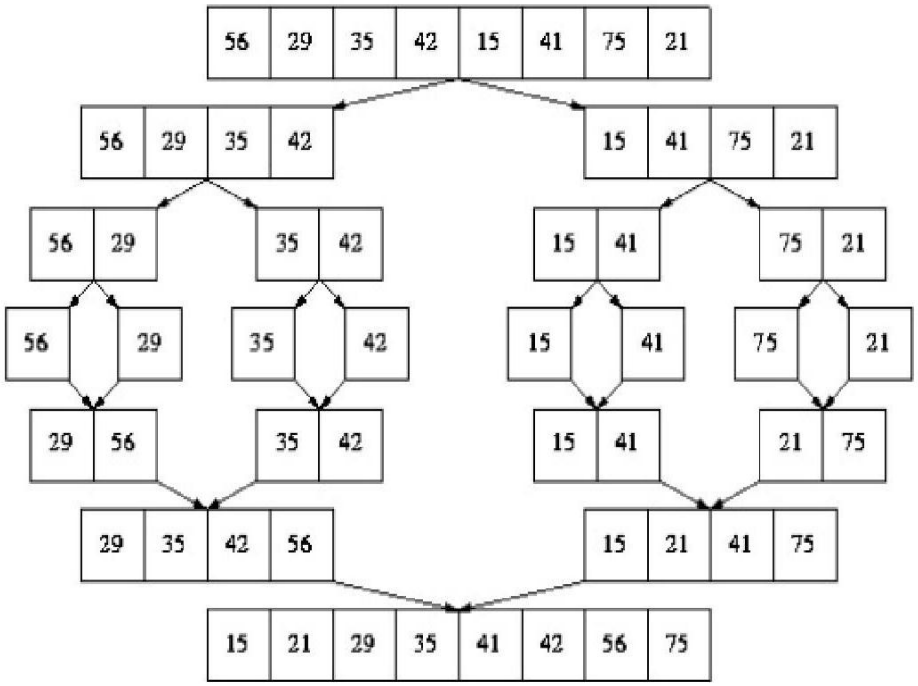
A possible idea is to run multiple iterations in a pipeline fashion, i.e., start the bubbling action of the next iteration before the preceding iteration has finished in such a way that it does not overtake it.

Introduction:



B. Merge Sort

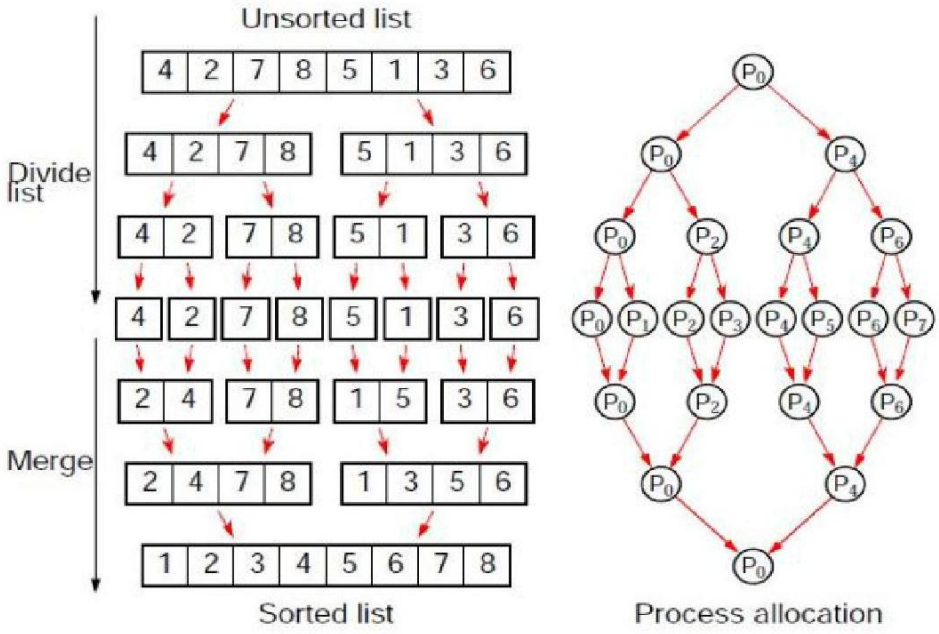
Merge Sort is a classical sorting algorithm using a divide-and-conquer approach. The initial unsorted list is first divided in half, each half sub list is then applied the same division method until individual elements are obtained. Pairs of adjacent elements / sub lists are then merged into sorted sub lists until the one fully merged and sorted list is obtained.



Introduction:

Parallel Merge Sort

The idea is to take advantage of the tree structure of the algorithm to assign work to processes. If communication time ignore, computations still only occur when merging the sub lists. But now, in the worst case, it takes ***2s − 1*** steps to merge all sub lists (two by two) of size s in a merging step.



Applications: 1. Bubble sort is used in programming TV remote to sort channels on the basis of longer viewing time.

2. Databases use an external merge sort to sort set of data that are too large to be loaded entirely into memory.

Conclusion: We have tested both the Sorting algorithms for different number elements with respective to time. As the number of elements increased time required for OpenMP is reduced.

Assignment NO: 04

Title: Design & implementation of Parallel (OpenMP) Searching algorithms for Binary search for Sorted Array and Depth-First Search (tree or an undirected graph) or Breadth-First Search (tree or an undirected graph) or Best-First Search that (traversal of graph to reach a target in the shortest possible path).

Objectives: To offload parallel computations to the graphics card, when it is appropriate to do so, and to give some idea of how to think about code running in the massively parallel environment presented by today's graphics cards.

Outcome: Students should understand the basic of OpenMP Programming Module using exiting Searching techniques.

Prerequisites: Data Structure and Files, Design and Analysis of Algorithm, OpenMP Tutorials.

Hardware Specification: x86_64 bit, 2 – 2/4 GB DDR RAM, 80 - 500 GB SATA HD.

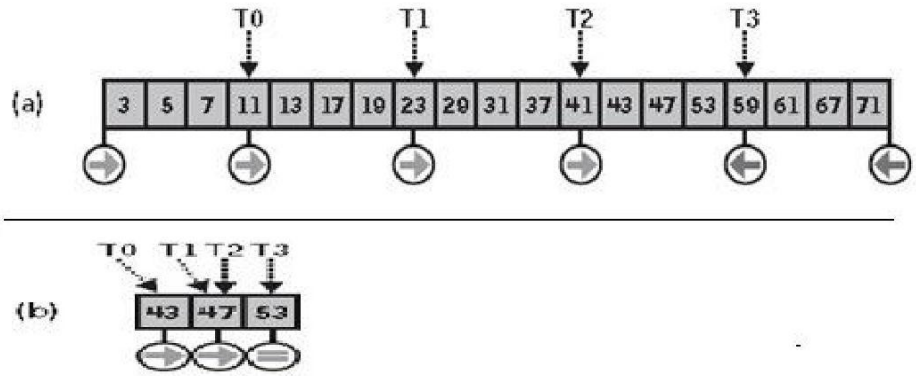
Software Specification: Ubuntu 14.04, G Edit, GCC Complier.

Introduction:

An Application Program Interface **(API)** that may be used to explicitly direct multithreaded, shared memory parallelism. An Open Multi-Processing (**OpenMP)** consists of a set of compiler ***#pragmas*** that control how the program works. The **pragmas** are designed so that even if the compiler does not support them, the program will still yield correct behavior, but without any parallelism. Searching identifies N well-spaced points within the search array bounds and compares the key of the corresponding records to the search key. Each thread does one of the N comparisons. There are three possible outcomes from these comparisons. The first is that the item of interest is found and the search is complete; the second is that the item key examined is less than the search key; the third is that the item key examined is greater than the search key. If no search key match is found, a new, smaller search array is defined by the two consecutive index points whose record keys were found to be less than the search key and greater than the search key.

The N-ary search is then performed on this refined search array. As with the serial version of binary search, the process is repeated until a match is found or the number of items in the search array is zero.

Given the sorted a rray of prime numbers in Figure (a), let's say we want to determine whether the value 53 is in the array and where it can be found. If there are four threads (T0 to T3), each computes an index into the array and compares the key value found there to the search key. Threads T0, T1, and T2 all find that the key value at the examined position is less than the search key value. Thus, an item with the matching key value must lie somewhere to the right of each of these thread's current search positions (indicated by the circled arrows).



Thread T3 determines that the examined key value is greater than the search key, and the matching key can be found to the left of this thread's search position (left-pointing circled arrow). Notice the circled arrows at each end of the array. These are attached to "phantom" elements just outside the array bounds. The results of the individual key tests define the subarray that is to become the new search array. Where we find two consecutive test results with opposite outcomes, the corresponding indexes will be just outside of the lower and upper bounds of the new search array.

Figure (a) shows that the test results from threads T2 (less than search key) and T3 (greater than search key) are opposite. The new search array is the array elements between the elements tested by these two threads.

Figure (b) shows this sub array and the index positions that are tested by each thread. The figure shows that during this second test of element key values, thread T3 has found the element that matches the search key.

Consider the case, where find a composite value, like 52, in a list of prime numbers. The individual key results by the four threads shown in Figure (a) would be the same. The sub array shown in Figure (b) would have the same results, except that the test by T3 would find that the key value in the assigned position was greater than the search key (and the equals sign would be a left-pointing arrow). The next round of key comparisons by threads would be from a sub array with no elements, bounded by the array slots holding the key values of 47 and 53. When threads are confronted with the search of an empty search space, then the key is not to be found.

This is obviously more complex than a simple binary search. The algorithm must coordinate the choices of index positions that each thread needs to test, keep and store the results of each test such that multiple threads can examine those results, and compute the new search array bounds for the next round of key tests. From this quick description, it's clear that it needs some globally accessible data and, more importantly, it needs a barrier between the completion of the key tests and the examination of the results of those tests.

Applications: 1. Information retrieval in the required format is the central activity in all computer applications.

2. Searching methods are designed to take advantage of the file organization and optimize the search for a particular record or to establish its absence.

Conclusion: We have tested both the Searching algorithms for different number elements with respective to time. As the number of elements increased time required for OpenMP is reduced.

Assignment NO: 05

Title: Design & implementation of Parallel (OpenMP) algorithm for K - Nearest Neighbors Classifier.

Objectives: To offload parallel computations to the graphics card, when it is appropriate to do so, and to give some idea of how to think about code running in the massively parallel environment presented by today's graphics cards.

Outcome: Students should understand the basic of OpenMP Programming Module using exiting K - Nearest Neighbors Classifier.

Prerequisites: Data Structure and Files, Design and Analysis of Algorithm, OpenMP Tutorials.

Hardware Specification: x86_64 bit, 2 – 2/4 GB DDR RAM, 80 - 500 GB SATA HD.

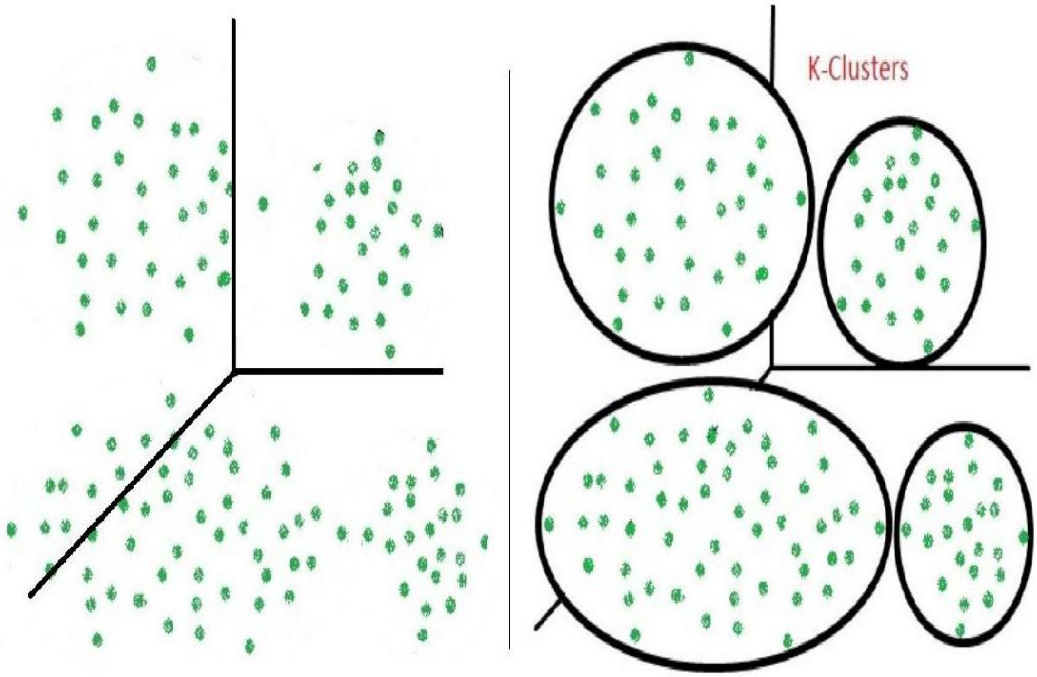
Software Specification: Ubuntu 14.04, G Edit, GCC Complier.

Introduction:

An Application Program Interface **(API)** that may be used to explicitly direct multithreaded, shared memory parallelism. An Open Multi-Processing (**OpenMP)** consists of a set of compiler ***#pragmas*** that control how the program works. The **pragmas** are designed so that even if the compiler does not support them, the program will still yield correct behavior, but without any parallelism.

K-means is a popular clustering method because it is simple to program and is easy to compute on large samples . The k-means clustering algorithm classifies N vectors with d dimension into k clusters by assigning each vector to the cluster whose average value is nearest to it by some distance measure on that set. The algorithm computes iteratively, until reassigning points and recompiling averages reaches optimal value. Initially given k, total number of clusters, the training samples are assigned to k clusters randomly. Alternately the first k training sample can be considered as single-element cluster and then each of the remaining (N-k) training samples is assigned to the nearest cluster.

After each assignment, the cluster centers are recomputed as referred as centroid of the cluster. Again each sample is taken in a sequence and computed its distance from the centroid If a sample is not currently in the cluster with the closest centroid, switch this sample to that cluster and update the centroid of the cluster gaining the new sample and the cluster losing the sample.



Repeat step 3 until convergence is achieved, that is until sample causes no new assignments. We calculate the distance to all centroid and get the minimum distance. At the termination of the K means algorithms all the ponts will belong to a cluster with shortest distance from its centroid. Above Figure, shows the clusters formed with number of points and 4 clusters.

Conclusion: We have tested both the Searching algorithms for K - clusters with respective to time. As the number of Clusters increased time required for OpenMP is reduced.

Applications:

1. In Wireless Sensor Network's based Application.
2. In Search Engines and Drug Activity Prediction.

LAB PRACTICALS

BASED ON

ARTIFICAL INTELLIGENCE

&

ROBOTICS

Assignment No. 01

Aim: Implement Tic-Tac-Toe using A* Algorithm

Software Requirements: NetBeans IDE

Theory

A* is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes. It is *complete* and will always find a solution if one exists. It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

f (n) = g(n) + h(n) .

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have f (n) = estimated cost of the cheapest solution through n. Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of g(n) + h(n). The algorithm is identical to UNIFORM-COST-SEARCH except that A* uses g + h instead of g.

A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest known heuristic cost, keeping a sorted priority queue of alternate path segments along the way.

It uses a distance-plus-cost heuristic function (usually denoted $f(x)$) to determine the order in which the search visits nodes in the tree. The distance-plus-cost heuristic is a sum of two functions: the path-cost function, which is the cost from the starting node to the current node (usually denoted $g(x)$) an admissible "heuristic estimate" of the distance to the goal (usually denoted $h(x)$).

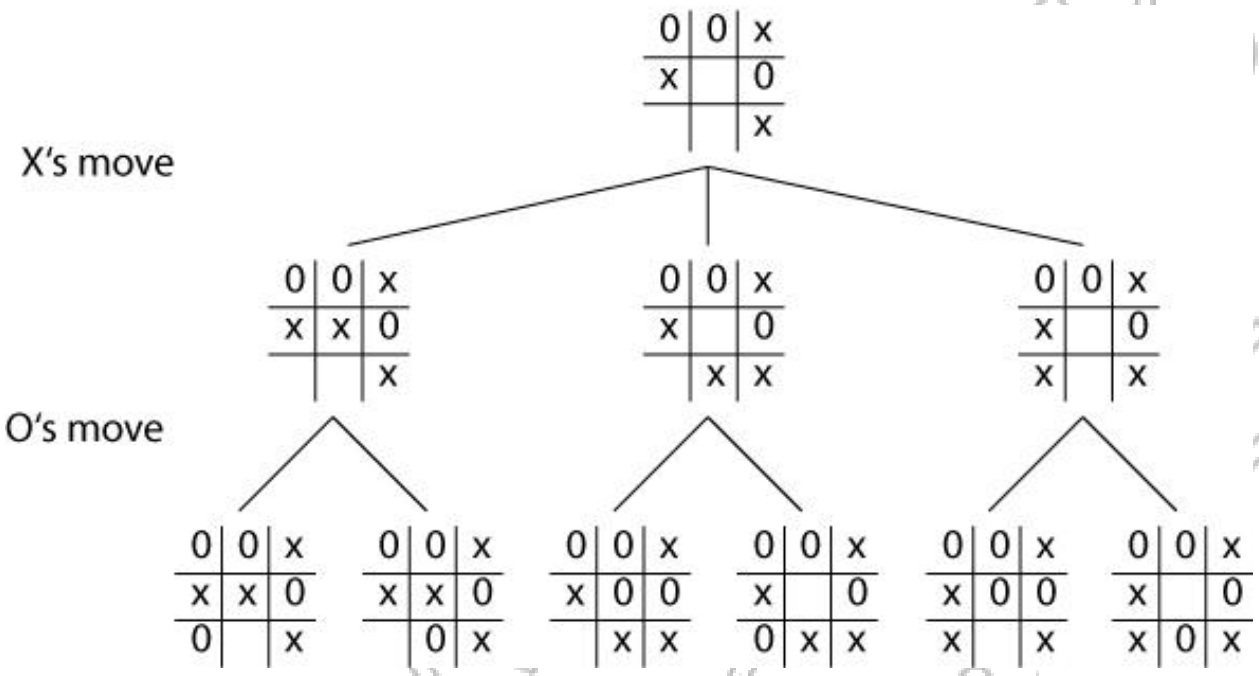
The $h(x)$ part of the $f(x)$ function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for an application like routing, $h(x)$ might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points or nodes.If the heuristic *h* satisfies the additional condition $h(x) \leq d(x,y) + h(y)$ for every edge *x*, *y* of the graph (where *d* denotes the length of that edge), then *h* is called monotone, or consistent. In such a case, A* can be implemented more efficiently—roughly speaking, no node needs to be processed more than once (see closed set below)—and A* is equivalent to runningDijkstra's algorithm with the reduced cost.

$$d'(x,y) := d(x,y) - h(x) + h(y)$$

For example, since airline distance never overestimates actual highway distance, and manhattan distance never overestimates actual moves in the gliding tile.

Tic-Tac-Toe:

Tic-tac-toe (also known as noughts and crosses or Xs and Os) is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.



f(n)=g(n)+h(n)

Where, f(n) is evaluation function g(n) is path cost

h(n) is heuristic function

A* is commonly used for the common path finding problem in applications such as games, but was originally designed as a general graph traversal algorithm.

Program Execution:

Choose X|O : X

| | | |

| | | |

| | | |

Your Move (Use Numpad) : 5

| | | |

| | X | |

| | | |

Your Move (Use Numpad) :

| O | | |

| | X | |

| | | |

AI Player marked O at position 1-1
Your Move (Use Numpad) : 9

| O | | X |

| | X | |

| | | |

AI Player marked O at position 1-1
Your Move (Use Numpad) :

| O | | X |

| | X | |

| O | | |

AI Player marked O at position 3-1
Your Move (Use Numpad) : 4

| O | | X |

| X | X | |

| O | | |

AI Player marked O at position 3-1
Your Move (Use Numpad) :

| O | | X |

| X | X | O |

| O | | |

AI Player marked O at position 2-3

Your Move (Use Numpad) : 2

| O | | X |

| X | X | O |

| O | X | |

AI Player marked O at position 2-3
Your Move (Use Numpad) :

| O | O | X |

| X | X | O |

| O | X | |

AI Player marked O at position 1-2
Your Move (Use Numpad) :
3

| O | O | X |

| X | X | O |

| O | X | X |

Draw!

Conclusion: : Using A* algorithm we are able to solve Tic Tac Toe problem.

Assignment No. 2

Aim: Solve 8-puzzle problem using A* algorithm. Assume any initial configuration and define goal configuration clearly.

Software Requirements: NetBeans IDE

Objective: Student will learn:

i) The Basic Concepts of A Star : Evaluation function, Path Cost, Heuristic function, Calculation of heuristic function

ii) General structure of eight puzzle problem.
iii) Logic of A star implementation for eight puzzle problem.

Theory:

Introduction

In computer science, **A*** (pronounced as "A star") is a computer algorithm that is widely used in path finding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes. The A* algorithm combines features of uniform-cost search and pure heuristic search to efficiently compute optimal solutions.

A* algorithm is a best-first search algorithm in which the cost associated with a node is $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the initial state to node n and $h(n)$ is the heuristic estimate or the cost or a path from node n to a goal.

Thus, f(n) estimates the lowest total cost of any solution path going through node n. At each point a node with lowest f value is chosen for expansion. Ties among nodes of equal f value should be broken in favor of nodes with lower h values. The algorithm terminates when a goal is chosen for expansion.

-

A* algorithm guides an optimal path to a goal if the heuristic function h(n) is admissible, meaning it never overestimates actual cost. For example, since airline distance

never overestimates actual highway distance, and manhattan distance never overestimates actual moves in the gliding tile.

For Puzzle, A* algorithm, using these evaluation functions, can find optimal solutions to these problems. In addition, A* makes the most efficient use of the given heuristic function in the following sense: among all shortest-path algorithms using the given heuristic function h(n). A* algorithm expands the fewest number of nodes.

The main drawback of A* algorithm and indeed of any best-first search is its memory requirement. Since at least the entire open list must be saved, A* algorithm is severely space-limited in practice, and is no more practical than best-first search algorithm on current machines. For example, while it can be run successfully on the eight puzzles, it exhausts available memory in a matter of minutes on the fifteen puzzles.

A star algorithm is very good search method, but with complexity problems

To implement such a graph-search procedure, we will need to **use two lists of node:**

1) **_OPEN:** nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined (i.e., had their successors generated). OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.
2) **_CLOSED:** Nodes that have already been examined. We need to keep these nodes in memory if we want to search a graph rather than a tree, since whether a node is generated; we need to check whether it has been generated before.

A * Algorithm:

1. Put the start nodes on OPEN.
2. If OPEN is empty, exit with failure
3. Remove from OPEN and place on CLOSED a node n having minimum f.
4. If n is a goal node exit successfully with a solution path obtained by tracing back the pointers from n to s.
5. Otherwise, expand n generating its children and directing pointers from each child node to n.

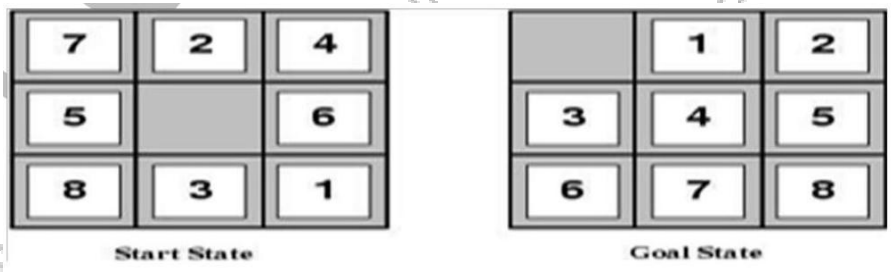
- For every child node n' do
- Evaluate h(n') and compute f(n') = g(n') +h(n')= g(n)+c(n, n')+h(n)
- If n' is already on OPEN or CLOSED compare its new f with the old f and attach the lowest f to n'.
- put n' with its f value in the right order in OPEN

6. Go to step 2.

Example of calculation of heuristic values for 8-puzzle problem:

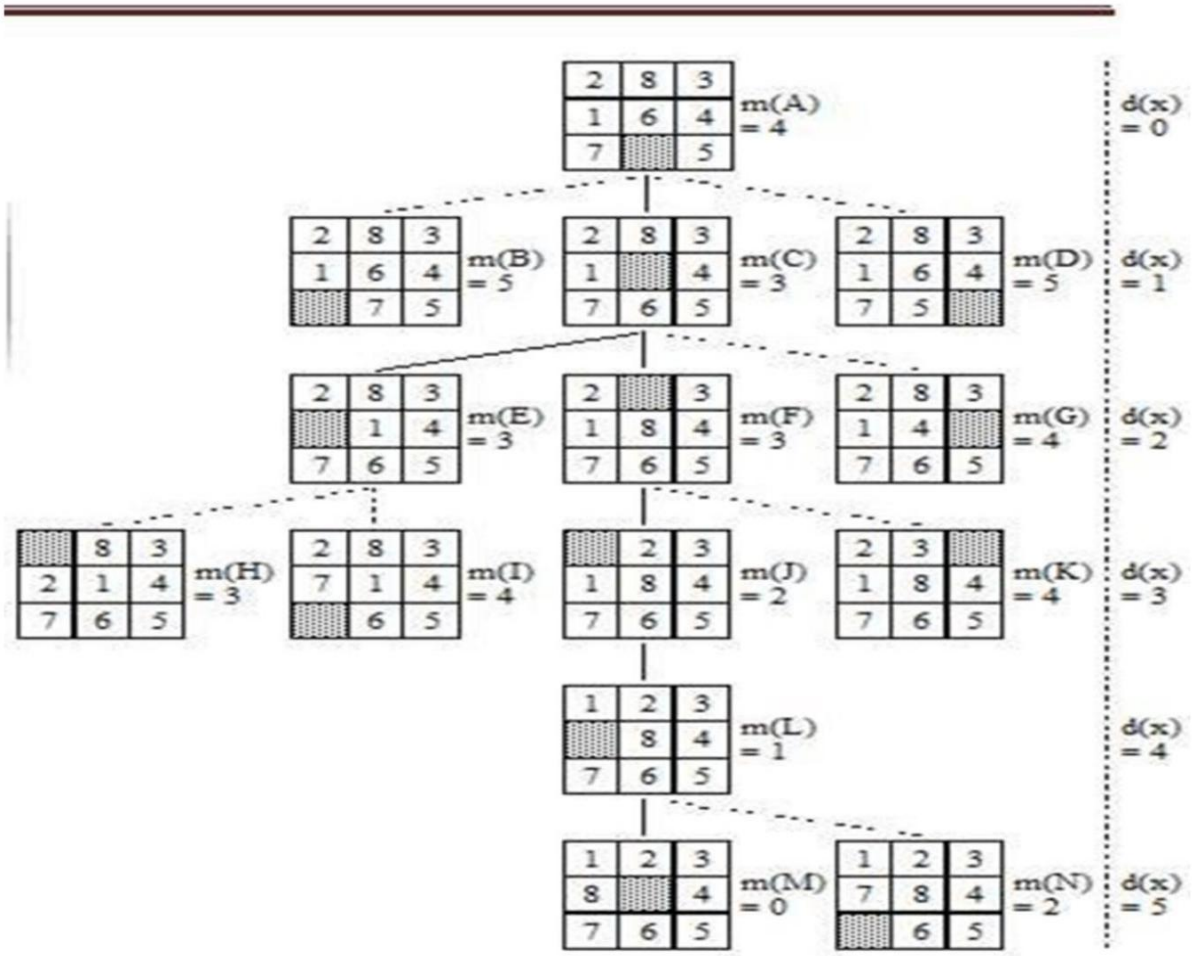
- h1(n) = number of misplaced tiles
- h2(n) =no. of squares from desired location of each tile

.



- h1(S) = 8
- h2(S) = 3+1+2+2+2+3+3+2 = 18

Implementation logic for 8-puzzle problem using A* algorithm



f(n)=g(n)+h(n)

Where, f(n) is evaluation function g(n) is path cost

h(n) is heuristic function

A* is commonly used for the common path finding problem in applications such as games, but was originally designed as a general graph traversal algorithm.

Program Execution:

The given start board is :

- a c

h b d

g f e

Enter goal Board :

Enter one tile as '-' ie. Blank tile

Enter the value of tile [0][0] : a

Enter the value of tile [0][1] : b

Enter the value of tile [0][2] : c

Enter the value of tile [1][0] : h

Enter the value of tile [1][1] : -

Enter the value of tile [1][2] : d

Enter the value of tile [2][0] : g

Enter the value of tile [2][1] : f

Enter the value of tile [2][2] : e

The given goal board is :

a b c

h - d

g f e

The board is solved as :

Board after 0 moves :

- a c

h b d

g f e

Possible moves are :

For Fn = 3 :

a - c

h b d

g f e

For Fn = 5 :

h a c

- b d

g f e

Board after 1 moves :

a - c

h b d

g f e

Possible moves are :

For Fn = 5 :

- a c

h b d

g f e

For Fn = 5 :

a c -

h b d

g f e

For Fn = 2 :

a b c h - d

g f e

Board after 2 moves :

a b c

h - d

g f e

Goal state achieved.

BUILD SUCCESSFUL (total time: 36 seconds)

Conclusion: **A star algorithm is implemented for eight puzzle problem.**

Assignment No. 03

Aim: Implement Expert System for Medical Diagnosis of 10 diseases based on adequate symptoms.

Software Requirements: SWI-Prolog for Windows, Editor.

Theory: A system that uses human expertise to make complicated decisions. Simulates reasoning by applying knowledge and interfaces. Uses expert's knowledge as rules and data within the system. Models the problem solving ability of a human expert.

Components of an ES:

1. Knowledge Base
    i. Represents all the data and information imputed by experts in the field.
    ii. Stores the data as a set of rules that the system must follow to make decisions.
2. Reasoning or Inference Engine
    i. Asks the user questions about what they are looking for.
    ii. Applies the knowledge and the rules held in the knowledge base.
    iii. Appropriately uses this information to arrive at a decision.
3. User Interface
    i. Allows the expert system and the user to communicate.
    ii. Finds out what it is that the system needs to answer.
    iii. Sends the user questions or answers and receives their response.
4. Explanation Facility
    i. Explains the systems reasoning and justifies its conclusions.

Program Execution:

```
?- go.
```

Does the patient has the symptom headache? :|:y

Does the patient has the symptom runny_nose? : |: n.

Does the patient has the symptom sore_throat? : |: n.

Does the patient has the symptom abdominal_pain? : |: y.

Does the patient has the symptom poor_appetite? : |: y.

Does the patient has the symptom fever? : |: y.

Advices and Sugestions:

1: Chloramphenicol
2: Amoxicillin
3: Ciprofloxacin
4: Azithromycin

Please do complete bed rest and take soft diet because It is suggested that the patient has typhoid true.

Conclusion: Medical Diagnosis based on adequate symptoms is implemented for Expert System

Assignment No.04

Aim: Develop elementary chatbot for suggesting investment as per the customer needs.

Software Requirements: NetBeans, program-ab library.

Theory:

A **chatbot** (also known as a **talkbot**, **chatterbot**, **Bot**, **IM bot**, **interactive agent**, or **Artificial Conversational Entity**) is a computer program or an artificial intelligence which conducts a conversation via auditory or textual methods. Such programs are often designed to convincingly simulate how a human would behave as a conversational partner, thereby passing the Turing test. Chatbots are typically used in dialog systems for various practical purposes including customer service or information acquisition. Some chatterbots use sophisticated natural language processing systems, but many simpler systems scan for keywords within the input, then pull a reply with the most matching keywords, or the most similar wording pattern, from a database.

The term "ChatterBot" was originally coined by Michael Mauldin (creator of the first Verbot, Julia) in 1994 to describe these conversational programs. Today, most chatbots are either accessed via virtual assistants such as Google Assistant and Amazon Alexa, via messaging apps such as Facebook Messenger or WeChat, or via individual organizations' apps and websites. Chatbots can be classified into usage categories such as conversational commerce (e-commerce via chat), analytics, communication, customer support, design, developer tools, education, entertainment, finance, food, games, health, HR, marketing, news, personal, productivity, shopping, social, sports, travel and utilities.

Chatbots Work on Pattern Recognition and a Set of Algorithms

Most chatbots work on a basic model of:

1. **Entities:** The thing the user is talking about.
2. **Intents:** The question the user asks the chatbot.
3. **Responses:** The answer the chatbot provides. It identifies the appropriate predefined responses from its repository. It then selects the most appropriate answer based on the intent and context. The repository is often confined to linguistic building blocks, but the real value comes from structured information it can mine from elsewhere.

Creating a Chatbot that Uses AI

These systems use machine learning and natural language processing (NLP) to interpret text like a person and find the right answer. They can also use visual information, such as facial recognition and images. Companies such as Amazon, Google, IBM and Microsoft, offer machine learning AI platforms you can use to power your own AI-driven chatbot.

Natural Language Processing

Natural language processing involves breaking down sentences and other parts of language, into components. It processes the semantics of the content to identify things like the entities and intents of the user. This has been a hard problem for computer science to tackle, but recent advances in the field have made this feasible.

Machine Learning

These systems use a corpus (a body of content) to retrieve the correct answer. Machine learning works best with massive repositories. So it will often include open-source repositories, such as structured Wikipedia. It uses algorithms to manipulate the data - particularly, linguistic pattern matching to find and score the information in the corpus. It continues to improve its responses based on its interactions with users. In other words, the machine learns.

Feeding the Bots

To help the machine learn, subject matter experts train it by uploading pairs of questions and answers. This enables the system to check the results of its learning for accuracy against the real world. This is called the ground truth. It can apply this learning to any new information added to the corpus or through user feedback.

You can also use unstructured content to help the system learn, where it contains relevant information. This might be in articles and blog posts. Platforms such as IBM Watson can process a corpus of unstructured content. However, you do need structured content when you're developing the system - as those question and answer key pairs.

- **Set-up steps:**

  Download the program-ab file.

  Copy the bots folder into project folder.

  Add the Ab.jar file to the project libraries.

  Create the .aiml file in bots/super/aiml folder.

  Create the .aiml.csv file in bots/super/aimlif folder.

Program Execution:

Human : Hello

Robot : Hi nice to see you!

Human : My name is Swaraj

Robot : Hello Swaraj! How can I help you?

Human : What are different investment options

Robot : Some of the investment options are Public provident fund Mutual fund Equity shares Real estate investment etc.

Human : I am interested in mutual fund

Robot : Yes we can discuss mutual fund

Human : Minimum investment period

Robot : Only applicable in case of close-ended and ELSS schemes

Human : Let us discuss equity shares

Robot : Yes we can discuss equity shares

Human : Maximum investment amount

Robot : No limit

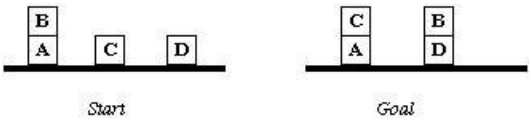
Human : Bye

Robot : Goodbye Swaraj! It was nice to talk to you

Conclusion: Chatbot is the functionality used for conversation in a textual format.

Assignment No. 05

Aim: Implement goal stack planning for the following configurations from the blocks world,



Objective: Student will learn

i) The Basic Concepts of Goal Stack Planning.
ii) General Algorithm for Goal Stack Planning.
iii) Logic of goal stack planning implementation.

Theory:

Introduction:

The planning in Artificial Intelligence is about the decision making tasks performed by the robots or computer programs to achieve a specific goal.The execution of planning is about choosing a sequence of actions with a high likelihood to complete the specific task.

Components of Planning System

The planning consists of following important steps:

Choose the best rule for applying the next rule based on the best available heuristics.

Apply the chosen rule for computing the new problem state.

Detect when a solution has been found.

- Detect dead ends so that they can be abandoned and the system's effort is directed in more fruitful directions.
- Detect when an almost correct solution has been found.

Blocks-World planning problem

The blocks-world problem is known as Sussman Anomaly.

Non inter leaved planners of the early 1970s were unable to solve this problem, hence it is considered as anomalous.

When two sub goals G1 and G2 are given, a non inter leaved planner produces either a plan

for G1 concatenated with a plan for G2, or vice-versa. In blocks-world problem, three blocks labeled as 'A', 'B', 'C' are allowed to rest on the flat surface. The given condition is that only one block can be moved at a time to achieve the goal.

Goal stack planning

This is one of the most important planning algorithms, which is specifically used by STRIPS.

The stack is used in an algorithm to hold the action and satisfy the goal. A knowledge base is used to hold the current state, actions.

Goal stack is similar to a node in a search tree, where the branches are created if there is a choice of an action.

The important steps of the algorithm are as stated below:

i. Start by pushing the original goal on the stack. Repeat this until the stack becomes empty. If stack top is a compound goal, then push its unsatisfied subgoals on the stack.
ii. If stack top is a single unsatisfied goal then, replace it by an action and push the action's precondition on the stack to satisfy the condition.
iii. If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.
iv. If stack top is a satisfied goal, pop it from the stack.

Basic Idea to handle interactive compound goals uses goal stacks, here the stack contains

- goals,
- operators -- ADD, DELETE and PREREQUISITE lists
- a database maintaining the current situation for each operator used.

Goal Stack Planning Example

Consider the following where wish to proceed from the start to goal state.

Start state:

ON(B, A) ^ ONTABLE(A) ^ ONTABLE(C) ^ ONTABLE(D) ^ ARMEMPTY

Goal state:

ON(C, A) ^ ON(B,D) ^ ONTABLE(A) ^ ONTABLE(D)

Conclusion: **Hence we have implemented Goal Stack Planning Algorithm.**

Assignment No. 06

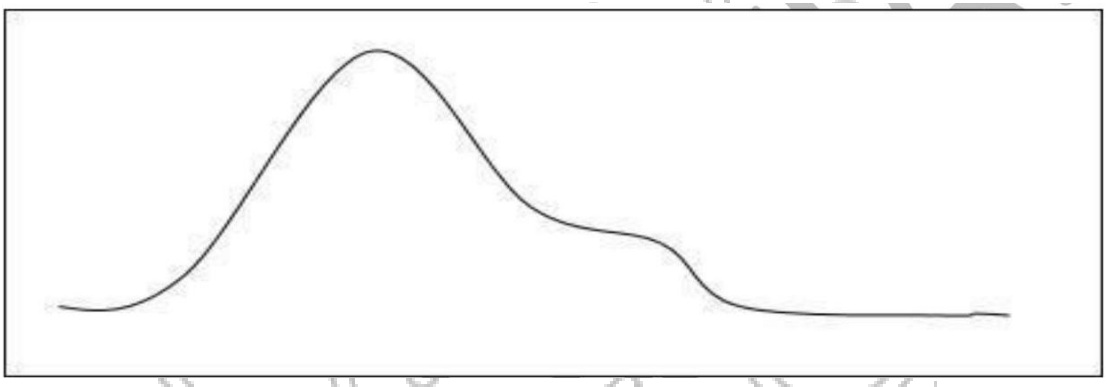
Aim: Use Heuristic Search Techniques to Implement Hill-Climbing Algorithm.

Software Requirements: NetBeans IDE

Objective: Student will learn:

1. The basic concept of constraint satisfaction problem and backtracking.
2. General structure of N Queens problem.

Theory: Hill climbing is a technique to solve certain optimization problems. In this technique, we start with a suboptimal solution and the solution is improved repeatedly until some condition is maximized.



The idea of starting with a sub-optimal solution is compared to starting from the base of the hill, improving the solution is compared to walking up the hill, and finally maximizing some condition is compared to reaching the top of the hill.

Hence, the hill climbing technique can be considered as the following phases −

- Constructing a sub-optimal solution obeying the constraints of the problem
- Improving the solution step-by-step
- Improving the solution until no more improvement is possible

Input:

As we are using Random function in java , we will get different input states each time we execute the code. Queens are placed on the board depending upon the generated random number. Function for placing Queens:

```
public void placeQueens()

{

        queenPositions = generateQueens(); for (int i = 0;
            i < board.length; i++)

            {

            board[queenPositions[i]][i] = 1;

            }

}
```

queenPositions[i] is the random number generated.

Algorithm:

1) Start in the leftmost column
2) If all queens are placed return true
3) Try all rows in the current column. Do following for every tried row.
a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing the queen in [row, column] leads to a solution then return true.
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
4) If all rows have been tried and nothing worked, return false to trigger backtracking.

Therefore, for the following random numbers, the queens will be placed on the board like this:

Random Numbers generated: 7

3

5

7

1

5

1

1

Corresponding

Board:

0 0 0 0 0 0 0

0 0 0 0 1 0 1 1

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 1 0 0 1 0 0

0 0 0 0 0 0 0 0

1 0 0 1 0 0 0 0

Conclusion: N-queens problem is implemented using **Hill-Climbing Algorithm**.

LAB PRACTICALS

BASED ON

DATA ANALYTICS

Assignment No. 01

Aim: Download the Iris flower dataset or any other dataset into a DataFrame. (eg https://archive.ics.uci.edu/ml/datasets/Iris) Use Python/R and Perform following –

- How many features are there and what are their types (e.g., numeric, nominal)?
- Compute and display summary statistics for each feature available in the dataset. (eg. minimum value, maximum value, mean, range, standard deviation, variance and percentiles
- Data Visualization-Create a histogram for each feature in the dataset to illustrate the feature distributions. Plot each histogram.
- Create a boxplot for each feature in the dataset. All of the boxplots should be combined into a single plot. Compare distributions and identify outliers.

Prerequisites: Fundamentals of R -Programming Languages

Objectives: To learn the concept of how to display summary statistics for each feature available in the dataset.

Implement a dataset into a dataframe. Implement the following operations:

1. Display data set details.
2. Calculate min, max ,mean, range, standard deviation, variance.
3. Create histograph using hist function.
4. Create boxplot using boxplot function.

Theory:

How to Find the Mean, Median, Mode, Range, and Standard Deviation

Simplify comparisons of sets of number, especially large sets of number, by calculating the center values using mean, mode and median. Use the ranges and standard deviations of the sets to examine the variability of data.

Calculating Mean

The mean identifies the average value of the set of numbers. For example, consider the data set containing the values 20, 24, 25, 36, 25, 22, 23.

Formula

To find the mean, use the formula: Mean equals the sum of the numbers in the data set divided by the number of values in the data set. In mathematical terms: Mean=(sum of all terms)÷(how many terms or values in the set).

Adding Data Set

Add the numbers in the example data set: 20+24+25+36+25+22+23=175.

Finding Divisor

Divide by the number of data points in the set. This set has seven values so divide by 7.

Finding Mean

Insert the values into the formula to calculate the mean. The mean equals the sum of the values (175) divided by the number of data points (7). Since 175÷7=25, the mean of this data set equals 25. Not all mean values will equal a whole number.

Calculating Range

Range shows the mathematical distance between the lowest and highest values in the data set. Range measures the variability of the data set. A wide range indicates greater variability in the data, or perhaps a single outlier far from the rest of the data. Outliers may skew, or shift, the mean value enough to impact data analysis.

Identifying Low and High Values

In the sample group, the lowest value is 20 and the highest value is 36.

Calculating Range

To calculate range, subtract the lowest value from the highest value. Since 36-20=16, the range equals 16.

Calculating Standard Deviation

Standard deviation measures the variability of the data set. Like range, a smaller standard deviation indicates less variability.

Formula

Finding standard deviation requires summing the squared difference between each data point and the mean $[\sum(x-\mu)^2]$, adding all the squares, dividing that sum by one less than the number of values (N-1), and finally calculating the square root of the dividend.

Mathematically, start with calculating the mean.

Calculating the Mean

Calculate the mean by adding all the data point values, then dividing by the number of data points. In the sample data set, 20+24+25+36+25+22+23=175. Divide the sum, 175, by the number of data points, 7, or 175÷7=25. The mean equals 25.

Squaring the Difference

Next, subtract the mean from each data point, then square each difference. The formula looks like this: $\sum(x-\mu)^2$, where $\sum$ means sum, x represents each data set value and μ represents the mean value. Continuing with the example set, the values become: 20-25=-5 and $-5^2=25$; 24-25=-1 and $-1^2=1$; 25-25=0 and $0^2=0$; 36-25=11 and $11^2=121$; 25-25=0 and $0^2=0$; 22-25=-3 and $-3^2=9$; and 23-25=-2 and $-2^2=4$.

Adding the Squared Differences

Adding the squared differences yields: 25+1+0+121+0+9+4=160. The example data set has 7 values, so N-1 equals 7-1=6. The sum of the squared differences, 160, divided by 6 equals approximately 26.6667.

Standard Deviation

Calculate the standard deviation by finding the square root of the division by N-1. In the example, the square root of 26.6667 equals approximately 5.164. Therefore, the standard deviation equals approximately 5.164.

Evaluating Standard Deviation

Standard deviation helps evaluate data. Numbers in the data set that fall within one standard deviation of the mean are part of the data set. Numbers that fall outside of two standard deviations are extreme values or outliers. In the example set, the value 36 lies more than two standard deviations from the mean, so 36 is an outlier. Outliers may represent erroneous data or may suggest unforeseen circumstances and should be carefully considered when interpreting data.

Facilities: Windows/Linux Operating Systems, RStudio, jdk.

Application:

1. The histogram is suitable for visualizing distribution of numerical data over a continuous interval, or a certain time period. The histogram organizes large amounts of data, and produces visualization quickly, using a single dimension.

2. The box plot allows quick graphical examination of one or more data sets. Box plots may seem more primitive than a histogram but they do have some advantages. They take up less space and are therefore particularly useful for comparing distributions between several groups or sets of data. Choice of number and width of bins techniques can heavily influence the appearance of a histogram, and choice of bandwidth can heavily influence the appearance of a kernel density estimate.

3. Data Visualization Application lets you quickly create insightful data visualizations, in minutes.

Data visualization tools allow anyone to organize and present information intuitively. They enables users to share data visualizations with others.

Input:

Structured Dataset: Iris

Dataset File: iris.csv

Output:

1. Display Dataset Details.
2. Calculate Min, Max, Mean, Variance value and Percentiles of probabilities also Display Specific use quantile.
3. Display the Histogram using Hist Function.
4. Display the Boxplot using Boxplot Function.

Conclusion:

Hence, we have studied using dataset into a dataframe and compare distribution and identify outliers.

Questions:

1. What is Data visualization?
2. How to calculate min,max,range and standard deviation?
3. How to create boxplot for each feature in the daset?
4. How to create histogram?
5. What is dataset?

Assignment No. 02

Aim: Download Pima Indians Diabetes dataset. Use Naive Bayes‟ Algorithm for classification

- Load the data from CSV file and split it into training and test datasets.
- Summarize the properties in the training dataset so that we can calculate probabilities and make predictions.
- Classify samples from a test dataset and a summarized training dataset.

Prerequisites: Fundamentals of Python -Programming Languages

Objectives: To learn the concept of Naïve Bayes classification algorithm, Bayes theorem.

Implement a classification algorithm that is Naïve Bayes. Implement the following operations:

1. Split the dataset into Training and Test dataset.
2. Calculate conditional probability of each feature in training dataset.
3. Classify sample from a test dataset.
4. Display confusion matrix with predicted and actual values.

Theory:

- Bayes Theorem: Bayes theorem is a way of finding a probability when we know certain other probabilities.

  Formula is:

$$P(A|B) = P(A)\,P(B|A)\mathbf{P(B)}$$

P(A|B): how often A happens given that B happens, written **P(A|B)**,

P(B|A): how often B happens given that A happens, written **P(B|A)**

P(A): and how likely A is on its own, written **P(A)**

P(B): and how likely B is on its own, written **P(B)**

Let us say P(Fire) means how often there is fire, and P(Smoke) means how often we see smoke, then:

P(Fire|Smoke) means how often there is fire when we can see smoke P(Smoke|Fire) means how often we can see smoke when there is fire

So the formula kind of tells us "forwards" P(Fire|Smoke) when we know "backwards" P(Smoke|Fire)

Example: If dangerous fires are rare (1%) but smoke is fairly common (10%) due to barbecues, and 90% of dangerous fires make smoke then:

$$P(Fire|Smoke) = P(Fire)\ P(Smoke|Fire)\mathbf{P(Smoke)}$$

$$=1\% \times 90\%\mathbf{10\%}$$

$$=9\%$$

So the "Probability of dangerous Fire when there is Smoke" is 9%

Naive Bayes Classification:

Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting. It can also be represented using a very simple Bayesian network. Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection. Windows/Linux Operating Systems, RStudio, jdk.

Applications:

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)

• **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

Confusion Matrix:

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. It allows easy identification of confusion between classes e.g. one class is commonly mislabeled as the other. Most performance measures are computed from the confusion matrix.

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

Here,

• Class 1 : Positive

• Class 2 : Negative

Definition of the Terms:

• Positive (P) : Observation is positive (for example: is an apple).

• Negative (N) : Observation is not positive (for example: is not an apple).

• True Positive (TP) : Observation is positive, and is predicted to be positive.

• False Negative (FN) : Observation is positive, but is predicted negative.

• True Negative (TN) : Observation is negative, and is predicted to be negative.

• False Positive (FP) : Observation is negative, but is predicted positive.

Classification Rate/Accuracy:

Rate or Accuracy is given by the relation:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

Recall:

Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN).

Recall is given by the relation:

Recall=TP / (TP + FN)= Recall=TP / (TP + FN)

Precision:

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (small number of FP).

Precision is given by the relation:

Precision = TP / (TP + FP)

High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)

F-measure:

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.

The F-Measure will always be nearer to the smaller value of Precision or Recall.

Fmeasure=(2*Recall*Precision)/(Recall+Presision)

nput:

Structured Dataset: PimaIndians

Diabetes Dataset File: PimaIndiansDiabetes.csv

Output:

1.Splitted dataset according to Split ratio.

2.Conditional probability of each feature.

3.visualization of the performance of an algorithm with confusion matrix

Conclusion: Hence, we have studied classification algorithm that is Naïve Bayes classification.

Questions:

1. What is Bayes Theorem?
2. What is confusion matrix?
3. Which function is used to split the dataset in R?
4. What are steps of Naïve Bayes algorithm?
5. What is conditional probability?

Assignment No. 03

Aim: Write a Hadoop program that counts the number of occurrences of each word in a text file.

Prerequisites: Fundamentals of Python -Programming Languages

Objectives: To understand the installation and concepts related to Hadoop.

Theory:

Hadoop is an open source distributed processing framework that manages data processing and storage for big data applications running in clustered systems. It is at the center of a growing ecosystem of big data technologies that are primarily used to support advanced analytics initiatives, including predictive analytics, data mining and machine learning applications. Hadoop can handle various forms of structured and unstructured data, giving users more flexibility for collecting, processing and analyzing data than relational databases and data warehouses provide.

- **Hadoop Architecture**

  Hadoop framework includes following four modules:

  1. **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
  2. **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.
  3. **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.
  4. **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.

MapReduce

Hadoop **MapReduce** is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

The term MapReduce actually refers to the following two different tasks that Hadoop programs perform:

- **The Map Task:** This is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).
- **The Reduce Task:** This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

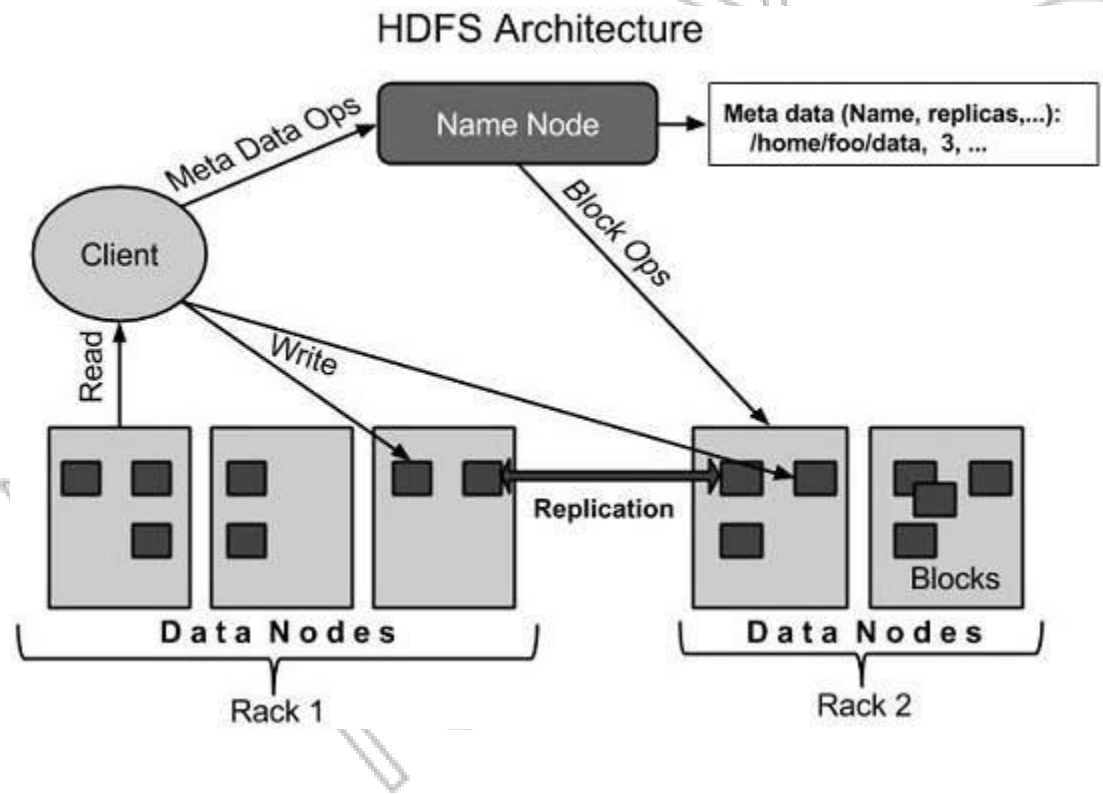
Hadoop Distributed File System

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single **NameNode** that manages the file system metadata and one or more slave **DataNodes** that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.



Conclusion: Hence, we have studied concepts of Hadoop and performed word count.

Assignment No.04

Title: Trip History Analysis: Use trip history dataset that is from a bike sharing service in the United States. The data is provided quarter-wise from 2010 (Q4) onwards. Each file has 7 columns. Predict the class of user.

Objective: To classify and predicate the data using KNN and Decision Tree

Prerequisites: K-Nearest Neighbors Algorithm,Decision Tree, DecisionTreeClassifier, sklearn, numpy, pandas

Theory:

Introduction:

There are two forms of data analysis that can be used for extracting models describing important classes or to predict future data trends. These two forms are as follows −

- Classification
- Prediction

Classification models predict categorical class labels; and prediction models predict continuous valued functions. For example, we can build a classification model to categorize bank loan applications as either safe or risky, or a prediction model to predict the expenditures in dollars of potential customers on computer equipment given their income and occupation.

What is classification?

Following are the examples of cases where the data analysis task is Classification −

- A bank loan officer wants to analyze the data in order to know which customer (loan applicant) are risky or which are safe.
- A marketing manager at a company needs to analyze a customer with a given profile, who will buy a new computer.

In both of the above examples, a model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data and yes or no for marketing data.

What is prediction?

Following are the examples of cases where the data analysis task is Prediction −

Suppose the marketing manager needs to predict how much a given customer will spend during a sale at his company. In this example we are bothered to predict a numeric value. Therefore the

data analysis task is an example of numeric prediction. In this case, a model or a predictor will be constructed that predicts a continuous-valued-function or ordered value.

Decision Tree:

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

The decision tree algorithm forms a node, where the most important attribute is placed at the root node. For evaluation we start at the root node and work our way down the tree by following the corresponding node that meets our condition or "decision". This process continues until a leaf node is reached, which contains the prediction or the outcome of the decision tree.

DecisionTreeClassifier(): This is the classifier function for DecisionTree. It is the main function for implementing the algorithms. Some important parameters are:

- **criterion:** It defines the function to measure the quality of a split. Sklearn supports “gini” criteria for Gini Index & “entropy” for Information Gain. By default, it takes “gini” value.
- **max_depth:** The max_depth parameter denotes maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By default, it takes “None” value.
- **min_samples_split:** This tells above the minimum no. of samples reqd. to split an internal node. If an integer value is taken then consider min_samples_split as the minimum no. If float, then it shows percentage. By default, it takes “2” value.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node. If an integer value is taken then consider min_samples_leaf as the minimum no. If float, then it shows percentage. By default, it takes “1” value.

Methods

decision_path(X[, check_input])	Return the decision path in the tree
fit(X, y[, sample_weight, check_input, …])	The fit method of this class is called to train the algorithm on the training data, which is passed as parameter to the fit method
predict	To make predictions, the predict method of the DecisionTreeClassifier class is used.
train_test_split	Used to randomly split the data into training and testing sets

KNN:

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real world datasets do not follow mathematical theoretical assumptions. Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier. Costly testing phase means time and memory. In the worst case, KNN needs more time to scan all data points and scanning all data points will require more memory for storing training data.

How to implement k-Nearest Neighbors in Python

1. **Handle** Data: Open the dataset from CSV and split into test/train datasets.
2. **Similarity**: Calculate the distance between two data instances.
3. **Neighbors**: Locate k most similar data instances.
4. **Response**: Generate a response from a set of data instances.
5. **Accuracy**: Summarize the accuracy of predictions.
6. **Main**: Tie it all together

Parameters:

n_neighbors : int, optional (default = 5)

Number of neighbors to use by default for kneighbors queries.

weights : str or callable, optional (default = 'uniform')

uniform' : uniform weights. All points in each neighborhood are weighted equally.

p : integer, optional (default = 2)

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : string or callable, default 'minkowski'

The distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric.

metric_params : dict, optional (default = None)

Additional keyword arguments for the metric function.

n_jobs : int or None, optional (default=None)

The number of parallel jobs to run for neighbors search. `None` means 1 unless in a joblib.parallel_backend context. `-1` means using all processors.

Conclusion:

Hence, we have studied classification based on KNN and Decision tree.

Assignment No. 05

Aim: Twitter Data Analysis: **Use** Twitter data for sentiment analysis. The dataset is 3MB in size and has 31,962 tweets. **Identify the tweets which are hate tweets and which are not. S**ample Test data set available here https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/

Prerequisites: Fundamentals of Python Programming Languages

Objective: To learn the concept of natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and classification.

Theory:

- Python regular expression Library: Regular expressions are used to identify whether a pattern exists in a given sequence of characters (string) or not. They help in manipulating textual data, which is often a pre-requisite for data science projects that involve text mining. You must have come across some application of regular expressions: they are used at the server side to validate the format of email addresses or password during registration, used for parsing text data files to find, replace or delete certain string, etc.
- Python Tweepy library: This library provides access to the entire twitter RESTful API methods. Each method can accept various parameters and return responses.
- Python TextBlob library: TextBlob is a Python (2 and 3) library for processing textual data. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and more.
- Authentication: create OAuthHandler object : Tweepy supports oauth authentication. Authentication is handled by the tweepy AuthHandler class.
- Utility Function in Python : This function provides sentiments analysis of tweets.

Facilities: Linux Operating Systems, Python editor.

Input:

Structured Dataset : Twitter Dataset

File: Twitter.csv

Output:

1. Sentiment analysis of twitter dataset.

2. Categorization of tweets as positive and negative tweets..

Conclusion: Hence, we have studied sentiment analysis of Twitter dataset to classify the tweets from dataset.

Questions:

1. What is Sentiment analysis?
2. Which API is required to handle authentication?
3. What is syntax of utility function?
4. What is function of Text Blob library?
5. What is Re library in python?