LABORATORY MANUAL

LABORATORY PRACTICE - I

BE-COMP

SEMESTER-I

TEACHING SCHEME	EXAMINATION SCHEME	
Lectures: 3 Hrs/Week	Theory:	100 Marks
Practical: 4 Hrs/Week	Practical:	50 Marks
	TW:	50 Marks

DEPARTMENT OF COMPUTER ENGINEERING

SGOI's College of Engineering,Belhe

2018-2019

Sr. No.	Name of Experiment
	Group A
1	a) Implement Parallel Reduction using Min, Max, Sum and Average operations. b) Write a CUDA program that, given an N-element vector, find-

GROUP A: ASSIGNMENTS

Experiment No: 01a

Aim: Implement parallel reduction using Min, Max, Sum and Average Operations.

- Minimum
- Maximum
- Sum
- Average

Objective: To study and implementation of directive based parallel programming model.

Outcome: Students will be understand the implementation of sequential program augmented with compiler directives to specify parallelism.

Pre-requisites:

64-bit Open source Linux or its derivative

Programming Languages: C/C++

Theory:

OpenMP:

[https://www.geeksforgeeks.org/openmp-introduction-with-installation-guide/]

[**https://www.geeksforgeeks.org/openmp-hello-world-program/]**

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.
OpenMP Core Syntax:

Most of the constructs in OpenMP are compiler directives:
#pragma omp construct [clause [clause]...]
Example

#pragma omp parallel num_threads(4)

Function prototypes and types in the file:

#include <omp.h>

Most OpenMP constructs apply to a “structured block”

Structured block:

a block of one or more statements surrounded by “{ }”, with one point of entry at the top and one point of exit at the bottom.

Following is the sample code which illustrates max operator usage in OpenMP :

```
#include <stdio.h>
#include <omp.h>                    ;
int main()
{
   double arr[10];
   omp_set_num_threads(4);     ;;
   double max_val=0.0;
   int i;
   for( i=0; i<10; i++)
      arr[i] = 2.0 + i;
   #pragma omp parallel for reduction(max : max_val)           ;;
   for( i=0;i<10; i++)
   {
      printf("thread id = %d and i = %d", omp_get_thread_num(), i);
      if(arr[i] > max_val)
      {
         max_val = arr[i];
      }
   }
   printf("\nmax_val = %f", max_val);
}
```

Following is the sample code which illustrates min operator usage in OpenMP :

```
#include <stdio.h>
#include <omp.h>
int main()
{
   double arr[10];
   omp_set_num_threads(4);
   double min_val=0.0;
   int i;
   for( i=0; i<10; i++)
      arr[i] = 2.0 + i;
```

```
            #pragma omp parallel for reduction(min : min_val)
    for( i=0;i<10; i++)
    {
       printf("thread id = %d and i = %d", omp_get_thread_num(), i);
       if(arr[i] < min_val)
       {
          min_val = arr[i];
       }
    }
    printf("\nmin_val = %f", min_val);
}
```

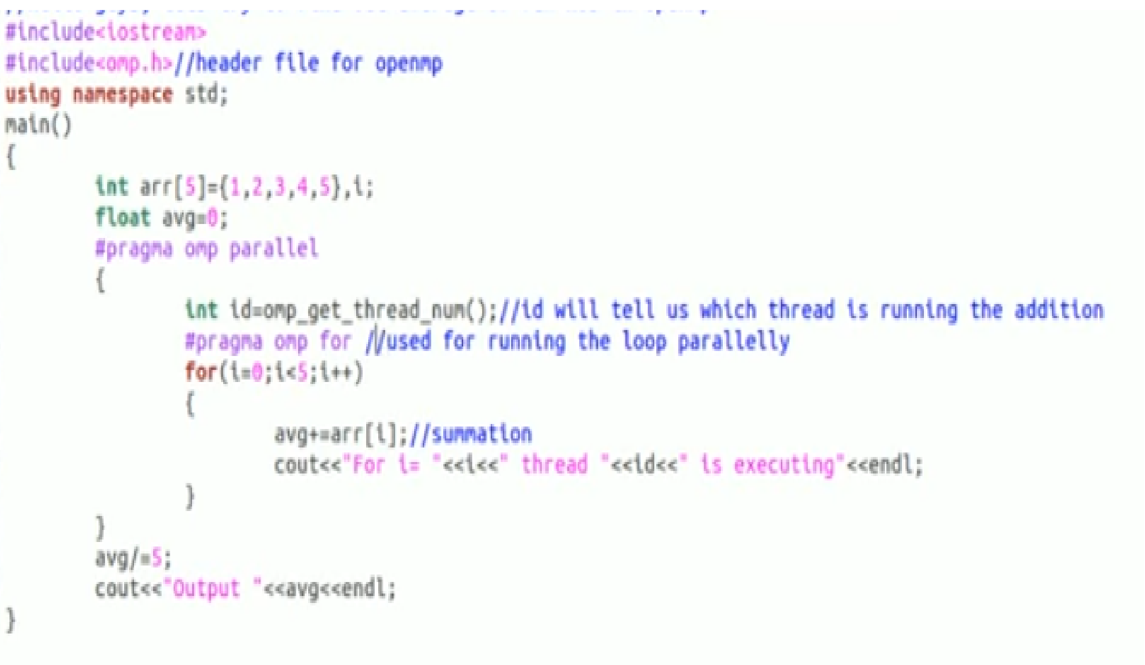
Following is the sample code which illustrates sum operation usage in OpenMP :

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
int   i, n;
float a[100], b[100], sum;
/* Some initializations */
n = 100;
for (i=0; i < n; i++)
  a[i] = b[i] = i * 1.0;
sum = 0.0;


#pragma omp parallel for reduction(+:sum)
  for (i=0; i < n; i++)
    sum = sum + (a[i] * b[i]);
printf("   Sum = %f\n",sum);
}
```

Following is the sample code which illustrates sum operation usage in OpenMP :



```
#include<iostream>
#include<omp.h>//header file for openmp
using namespace std;
main()
{
        int arr[5]={1,2,3,4,5},i;
        float avg=0;
        #pragma omp parallel
        {
                int id=omp_get_thread_num();//id will tell us which thread is running the addition
                #pragma omp for //used for running the loop parallelly
                for(i=0;i<5;i++)
                {
                        avg+=arr[i];//summation
                        cout<<"For i= "<<i<<" thread "<<id<<" is executing"<<endl;
                }
        }
        avg/=5;
        cout<<"Output "<<avg<<endl;
}
```

Conclusion: We have implemented parallel reduction using Min, Max, Sum and Average Operations.

Experiment No: 01 b

Aim: To write a CUDA program that, given an N-element vector, find.

- Minimum element in vector
- Maximum element in vector
- Arithmetic mean of the vector
- Standard deviation of the values in the vector

Objective: 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 be understand the implementation of operations on vector, generate o/p as two computed max with respect to time.

Pre-requisites:

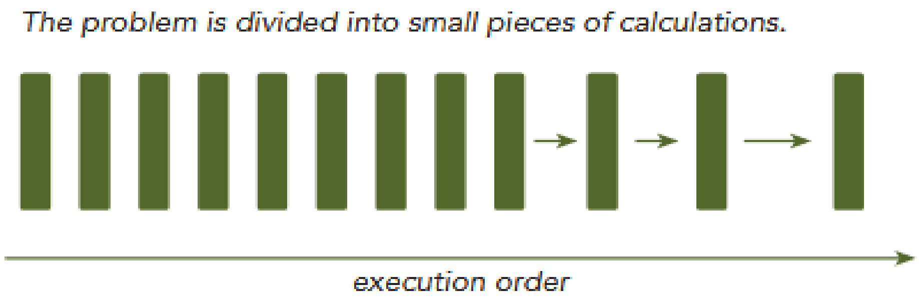
64-bit Open source Linux or its derivative

Programming Languages: C/C++,CUDA

Theory:

Sequential Programming:

When solving a problem with a computer program, it is natural to divide the problem into a discrete series of calculations; each calculation performs a specified task, as shown in following Figure .Such a pro-gram is called a sequential program.



Parallel Programming:

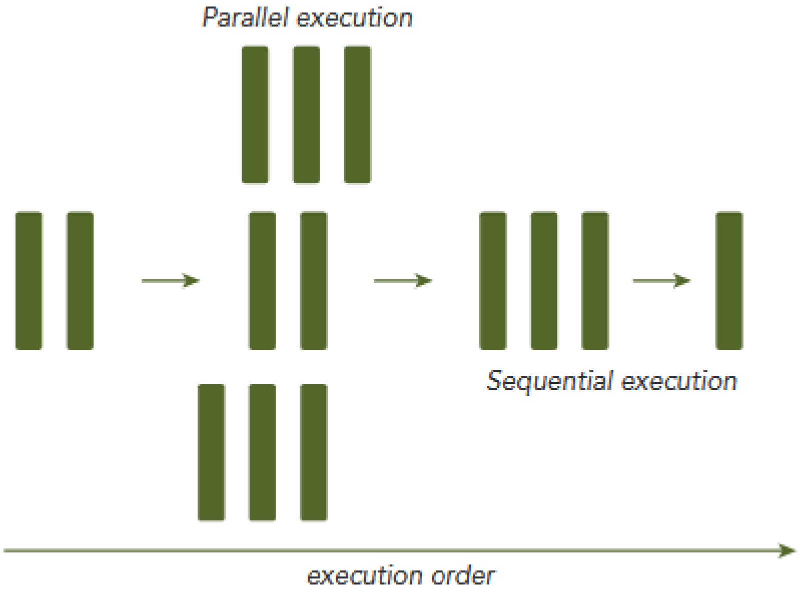
There are two fundamental types of parallelism in applications:

- 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.

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.

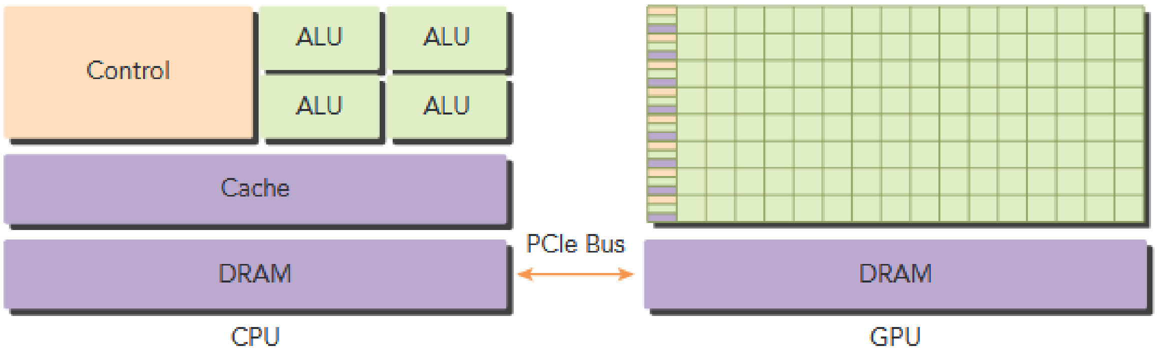
CUDA Architecture:

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 exe-

cuted on the GPU. Launch the kernel function with the following code:

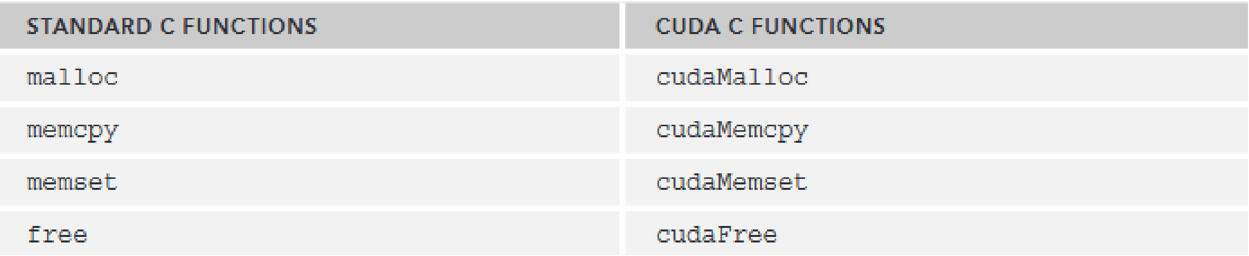
```
helloFromGPU <<<1,10>>>();
```

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

Table lists the standard C functions and their corresponding CUDA C functions for memory operations. Host and Device Memory Functions are follows.

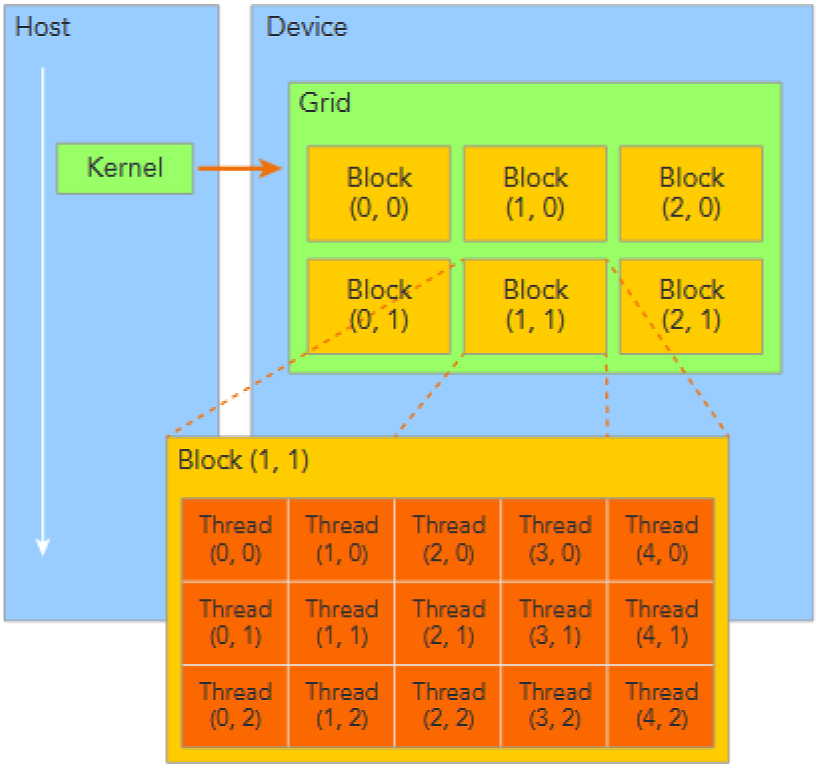


STANDARD C FUNCTIONS	CUDA C FUNCTIONS
`malloc`	`cudaMalloc`
`memcpy`	`cudaMemcpy`
`memset`	`cudaMemset`
`free`	`cudaFree`

Organizing Threads:

When a kernel function is launched from the host side, execution is moved to a device where a large number of threads are generated and each thread executes the statements specified by the

kernel function. The two-level thread hierarchy decomposed into blocks of threads and grids of blocks, as shown in following figure:



All threads spawned by a single kernel launch are collectively called a grid . All threads in a grid

share the same global memory space. A grid is made up of many thread blocks. A thread block is a group of threads that can cooperate with each other using:

- Block-local synchronization
- Block-local shared memory

Threads from different blocks cannot cooperate.

Threads rely on the following two unique coordinates to distinguish themselves from each other:

- blockIdx (block index within a grid)
- threadIdx (thread index within a block)

These variables appear as built-in, pre-initialized variables that can be accessed within kernel functions. When a kernel function is executed, the coordinate variables blockIdx and threadIdx are assigned to each thread by the CUDA runtime. Based on the coordinates, you can assign portions of data to different threads. It is a structure containing three unsigned integers, and the 1st, 2nd, and 3rd components are accessible through the fields x, y, and z respectively.

blockIdx.x
blockIdx.y
blockIdx.z

threadIdx.x
threadIdx.y
threadIdx.z
CUDA organizes grids and blocks in three dimensions. The dimensions of a grid and a block are specified by the following two built-in variables:

➢blockDim (block dimension, measured in threads)

➢gridDim (grid dimension, measured in blocks)

These variables are of type dim3, that is used to specify dimensions. When defining a variable of type dim3, any component left unspecified is initialized to 1.Each component in a variable of type dim3 is accessible through its x,y,, and z fields, respectively, as shown in the following example:

blockDim.x
blockDim.y
blockDim.

CUDA program for calculating Min for given N-element vector

```
 #include <cuda.h>
#include <stdio.h>
#include <time.h>

#define SIZE 100

__global__ void min(int *a , int *c)// kernel function definition

{
int i = threadIdx.x;       // initialize i to thread ID

*c = a[55];

        if(a[i] < *c)
                {
                    *c = a[i];
                    }

    }

    int main()
    {
    int i;
    srand(time(NULL));  //makes use of the computer's internal clock to 
control the choice of the seed

    int a[SIZE];
    int c;

    int *dev_a, *dev_c;       //GPU / device parameters

    cudaMalloc((void **) &dev_a, SIZE*sizeof(int));   //assign memory to 
parameters on GPU from CUDA runtime API

    cudaMalloc((void **) &dev_c, SIZE*sizeof(int));



    for( i = 0 ; i < SIZE ; i++)
    {
      a[i] = rand();    // input the numbers
    }
    for( i = 0 ; i < SIZE ; i++)
    {
      printf("%d", a[i]);     // input the numbers
    }

    cudaMemcpy(dev_a , a, SIZE*sizeof(int),cudaMemcpyHostToDevice);
      //copy the array from CPU to GPU
    min<<<1,SIZE>>>(dev_a,dev_c);
            // call kernel function <<<number of blocks, number of threads
    cudaMemcpy(&c, dev_c, SIZE*sizeof(int),cudaMemcpyDeviceToHost);
      // copy the result back from GPU to CPU

    printf("\nmin =  %d ",c);

    cudaFree(dev_a);          // Free the allocated memory
    cudaFree(dev_c);
    printf("");

    return 0;
    }
```

CUDA program for calculating Max for given N-element vector

```
#include <cuda.h>
    #include <stdio.h>
    #include <time.h>

    #define SIZE 1000

    __global__ void max(int *a , int *c)  // kernel function definition
    {
    int i = threadIdx.x;      // initialize i to thread ID

    *c = a[0];

            if(a[i] > *c)
                    {
                    *c = a[i];
                    }

    }

    int main()
    {
    int i;
    srand(time(NULL));  //makes use of the computer's internal clock to
control the choice of the seed

    int a[SIZE];
    int c;

    int *dev_a, *dev_c;   //GPU / device parameters

    cudaMalloc((void **) &dev_a, SIZE*sizeof(int));      //assign memory
to parameters on GPU
    cudaMalloc((void **) &dev_c, SIZE*sizeof(int));

    for( i = 0 ; i < SIZE ; i++)
    {
    a[i] = i; // rand()% 1000 + 1;      // input the numbers
    }

    cudaMemcpy(dev_a , a, SIZE*sizeof(int),cudaMemcpyHostToDevice);

  //copy the array from CPU to GPU

    max<<<1,SIZE>>>(dev_a,dev_c);   // call kernel function <<<number of
blocks, number of threads

    cudaMemcpy(&c, dev_c, SIZE*sizeof(int),cudaMemcpyDeviceToHost);

      // copy the result back from GPU to CPU

    printf("\nmax =  %d ",c);

    cudaFree(dev_a);          // Free the allocated memory
    cudaFree(dev_c);
    printf("");

    return 0;
    }
```

CUDA program for calculating standard deviation for given N-element vector

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>

// CUDA kernel. Each thread takes care of one element of c
__global__ void vecAdd(int *a, int *c, int n)
{
    // Get our global thread ID
    int id = blockIdx.x*blockDim.x+threadIdx.x;
   // c[id]=0;
    // Make sure we do not go out of bounds
    if (id < n)
        *c+= a[id];
   // printf("\n%d", c[id]);
}

int main( int argc, char* argv[] )
{
    // Size of vectors
    // int n = 100000;
      int n=5;
 const int size = n * sizeof(int);
    // Host input vectors
    int *h_a;
   // double *h_b;
    //Host output vector
    int *h_c;

    // Device input vectors
    int *d_a;
    //double *d_b;
    //Device output vector
    int *d_c;
    int dev=0;
    // Size, in bytes, of each vector
    size_t bytes = n*sizeof(double);

    // Allocate memory for each vector on host
    //h_a = (int*)malloc(bytes);
    //h_b = (double*)malloc(bytes);
    h_c = (int*)malloc(bytes);

    // Allocate memory for each vector on GPU
    cudaMalloc(&d_a, bytes);
   // cudaMalloc(&d_b, bytes);
    cudaMalloc(&d_c, bytes);

    int i;
    printf("Input array");
    // Initialize vectors on host
    /*for( i = 0; i < n; i++ ) {
        // h_a[i] = sin(i)*sin(i);
        //printf("\n",i);
      h_a[i]=i;
      //printf("\n%d", h_a[i]);
      //h_b[i]=i;
        //h_b[i] = cos(i)*cos(i);
    }*/
   int a[]= {0, 1, 2, 3, 4};

   cudaMalloc(&h_a, size);

    // Copy host vectors to device
    cudaMemcpy( h_a, a, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy( d_c, &dev, sizeof(int), cudaMemcpyHostToDevice);
//    cudaMemcpy( d_b, h_b, bytes, cudaMemcpyHostToDevice);

    int blockSize, gridSize;

    // Number of threads in each thread block
    blockSize = 2;

    // Number of thread blocks in grid
    gridSize = (int)ceil((float)n/blockSize);

    // Execute the kernel
    vecAdd<<<gridSize, blockSize>>>(d_a,d_c,n);
      int result;
    // Copy array back to host
    cudaMemcpy( &result,d_c, sizeof(int), cudaMemcpyDeviceToHost );

    // Sum up vector c and print result divided by n, this should equal 1
within error
    double sum = 0;
    //for(i=0; i<n; i++)
      //  sum += h_c[i];

    printf("final result: %f\n",result );

   // vecdev<<<gridSize, blockSize>>>(d_a,d_c, n);

    // Release device memory
    cudaFree(d_a);
    //cudaFree(d_b);
    cudaFree(d_c);

    // Release host memory
    free(h_a);
    //free(h_b);
    free(h_c);

    return 0;
}
```

CUDA program for calculating arithmetic mean for given N-element vector

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// CUDA kernel. Each thread takes care of one element of c
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
    // Get our global thread ID
    int id = blockIdx.x*blockDim.x+threadIdx.x;             // get global
index

    // Make sure we do not go out of bounds
    if (id < n)
        c[id] = a[id] + b[id];
}

int main( )
{

    //int n = 100000;
      int n=5;                // Size of vectors


    double *h_a;        // Host input vector
    double *h_b;        // Host input vector


    double *h_c;        //Host output vector


    double *d_a;        // Device input vector
    double *d_b;        // Device input vector

    double *d_c;         //Device output vector


    size_t bytes = n*sizeof(double);             // Size, in bytes, of
each vector

    // Allocate memory for each vector on host
    h_a = (double*)malloc(bytes);
    h_b = (double*)malloc(bytes);
    h_c = (double*)malloc(bytes);

    // Allocate memory for each vector on GPU
    cudaMalloc(&d_a, bytes);
    cudaMalloc(&d_b, bytes);
    cudaMalloc(&d_c, bytes);

    int i;

    // Initialize vectors on host
    for( i = 0; i < n; i++ ) {

      h_a[i]=i;
      h_b[i]=i;

    }

    // Copy host vectors to device
    cudaMemcpy( d_a, h_a, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy( d_b, h_b, bytes, cudaMemcpyHostToDevice);

    int blockSize, gridSize;
    // Number of threads in each thread block
    blockSize = 1024;

    // Number of thread blocks in grid
    gridSize = (int)ceil((float)n/blockSize);

    // Execute the kernel
    vecAdd<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);

    // Copy array back to host
    cudaMemcpy( h_c, d_c, bytes, cudaMemcpyDeviceToHost );

    // Sum up vector c and print result divided by n, this should equal 1
within error
    double sum = 0;
    for(i=0; i<n; i++)
        sum += h_c[i];
    printf("Average mean of 2 vectors: %f\n", sum/n);

    // Release device memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    // Release host memory
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

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

Experiment No: 2

Title: **Vector and Matrix Operations-**
Design parallel algorithm to
1. Add two large vectors

2. Multiply Vector and Matrix
3. Multiply two N × N arrays using n2 processors

Aim: Implement *nxn* matrix parallel addition, multiplication using CUDA, use shared memory.

Prerequisites:

- Concept of matrix addition, multiplication.
- Basics of CUDA programming

Objectives:

Student should be able to learn parallel programming, CUDA architecture and CUDA processing flow

Theory:

A straightforward matrix multiplication example that illustrates the basic features of memory and thread management in CUDA programs

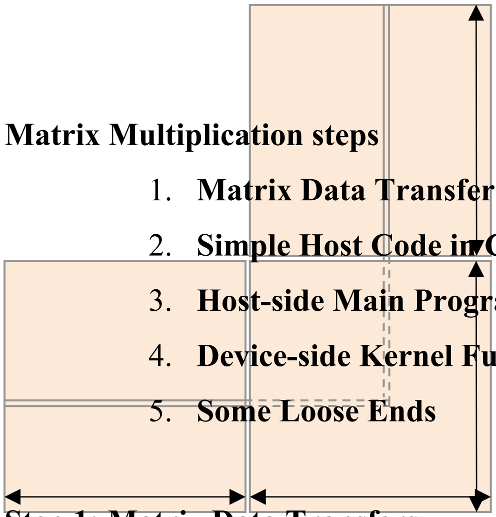
- Leave shared memory usage until later
- Local, register usage
- Thread ID usage

Memory data transfer API between host and device

- *P = M * N of size WIDTH x WIDTH*

Without tiling:

- One thread handles one element of P
- M and N are loaded WIDTH times from global memory



Matrix Multiplication steps

1. **Matrix Data Transfers**
2. **Simple Host Code in C**
3. **Host-side Main Program Code**
4. **Device-side Kernel Function**
5. **Some Loose Ends**

Step 1: Matrix Data Transfers

```
    // Allocate the device memory where we will copy M to
```

```
Matrix Md;
Md.width  = WIDTH;
Md.height = WIDTH;
Md.pitch  = WIDTH;
int size = WIDTH * WIDTH * sizeof(float);
cudaMalloc((void**)&Md.elements, size);
// Copy M from the host to the device
cudaMemcpy(Md.elements, M.elements, size, cudaMemcpyHostToDevice);
// Read M from the device to the host into P
cudaMemcpy(P.elements, Md.elements, size, cudaMemcpyDeviceToHost);
...
// Free device memory
cudaFree(Md.elements);
```

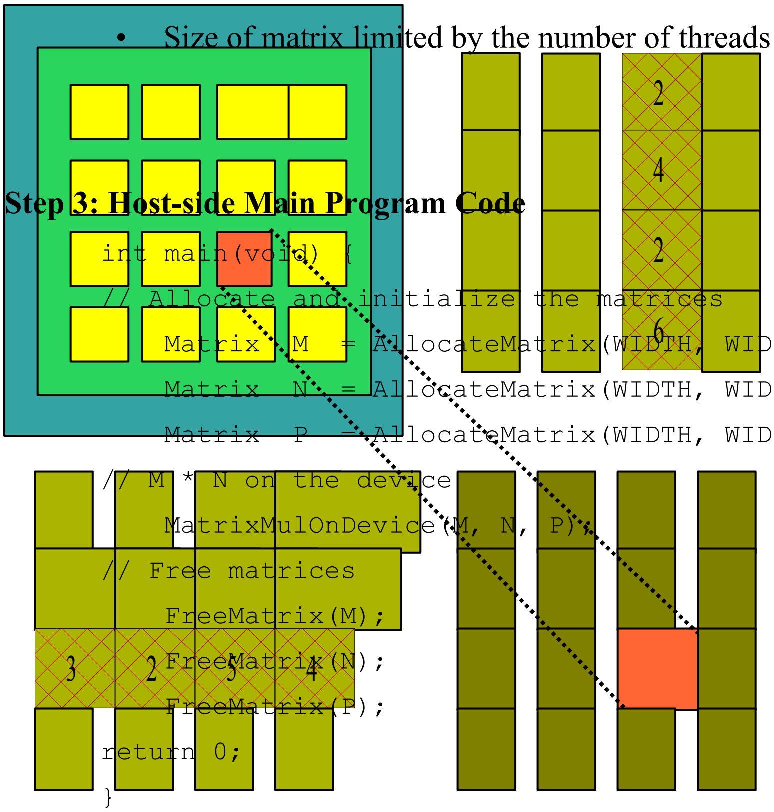
Step 2: Simple Host Code in C

```
// Matrix multiplication on the (CPU) host in double precision
// for simplicity, we will assume that all dimensions are equal
void MatrixMulOnHost(const Matrix M, const Matrix N, Matrix P)
{
    for (int i = 0; i < M.height; ++i)
        for (int j = 0; j < N.width; ++j) {
            double sum = 0;
            for (int k = 0; k < M.width; ++k) {
                double a = M.elements[i * M.width + k];
                double b = N.elements[k * N.width + j];
                sum += a * b;
            }
            P.elements[i * N.width + j] = sum;
        }
```

Multiply Using One Thread Block

- One Block of threads compute matrix P
  - Each thread computes one element of P
- Each thread
  - Loads a row of matrix M
  - Loads a column of matrix N
  - Perform one multiply and addition for each pair of M and N elements
  - Compute to off-chip memory access ratio close to 1:1 (not very high)

- Size of matrix limited by the number of threads allowed in a thread block



Step 3: Host-side Main Program Code

```
int main(void) {
// Allocate and initialize the matrices
    Matrix  M  = AllocateMatrix(WIDTH, WIDTH, 1);
    Matrix  N  = AllocateMatrix(WIDTH, WIDTH, 1);
    Matrix  P  = AllocateMatrix(WIDTH, WIDTH, 0);
// M * N on the device
    MatrixMulOnDevice(M, N, P);
// Free matrices
    FreeMatrix(M);
    FreeMatrix(N);
    FreeMatrix(P);
return 0;
}
```

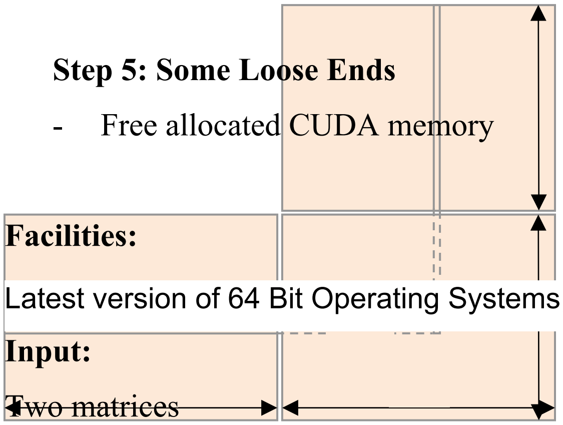
Host-side code

```
// Matrix multiplication on the device
void MatrixMulOnDevice(const Matrix M, const Matrix N, Matrix P)
{
    // Load M and N to the device
    Matrix Md = AllocateDeviceMatrix(M);
    CopyToDeviceMatrix(Md, M);
    Matrix Nd = AllocateDeviceMatrix(N);
    CopyToDeviceMatrix(Nd, N);
    // Allocate P on the device
    Matrix Pd = AllocateDeviceMatrix(P);
    CopyToDeviceMatrix(Pd, P); // Clear memory
    // Setup the execution configuration
    dim3 dimBlock(WIDTH, WIDTH);
    dim3 dimGrid(1, 1);
    // Launch the device computation threads!
    MatrixMulKernel<<<dimGrid, dimBlock>>>(Md, Nd, Pd);
```

```
        // Read P from the device
        CopyFromDeviceMatrix(P, Pd);
        // Free device matrices
        FreeDeviceMatrix(Md);
        FreeDeviceMatrix(Nd);
        FreeDeviceMatrix(Pd);
    }
```

Step 4: Device-side Kernel Function

```
    // Matrix multiplication kernel – thread specification
    __global__ void MatrixMulKernel(Matrix M, Matrix N, Matrix P)
    {
        // 2D Thread ID
        int tx = threadIdx.x;
        int ty = threadIdx.y;
        // Pvalue is used to store the element of the matrix
        // that is computed by the thread
        float Pvalue = 0;
    for (int k = 0; k < M.width; ++k)
        {
             float Melement = M.elements[ty * M.pitch + k];
             float Nelement = Nd.elements[k * N.pitch + tx];
             Pvalue += Melement * Nelement;
        }
        // Write the matrix to device memory;
        // each thread writes one element
        P.elements[ty * P.pitch + tx] = Pvalue;
    }
```



Step 5: Some Loose Ends

- Free allocated CUDA memory

Facilities:

Latest version of 64 Bit Operating Systems, CUDA enabled NVIDIA Graphics card

Input:

Two matrices

Output:

[Type here]

Multiplication of two matrix

Software Engg.:

Mathematical Model:

Conclusion:

We learned parallel programming with the help of CUDA architecture.

Questions:

1. What is CUDA?
2. Explain Processing flow of CUDA programming.
3. Explain advantages and limitations of CUDA.
4. Make the comparison between GPU and CPU.
5. Explain various alternatives to CUDA.
6. Explain CUDA hardware architecture in detail.

Assignment No: 3

Title: For Bubble Sort and Merger Sort, based on existing sequential algorithms, design and implement parallel algorithm utilizing all resources available.

Aim: Understand Parallel Sorting Algorithms like Bubble sort and Merge Sort.

Prerequisites:

Student should know basic concepts of Bubble sort and Merge Sort.

Objective: Study of Parallel Sorting Algorithms like Bubble sort and Merge Sort

Theory:

i) **What is Sorting?**

Sorting is a process of arranging elements in a group in a particular order, i.e., ascending order, descending order, alphabetic order, etc.

Characteristics of Sorting are:

- Arrange elements of a list into certain order
- Make data become easier to access
- Speed up other operations such as searching and merging. Many sorting algorithms with different time and space complexities

ii) **What is Parallel Sorting?**

A sequential sorting algorithm may not be efficient enough when we have to sort a huge volume of data. Therefore, parallel algorithms are used in sorting.

Design methodology:

- Based on an existing sequential sort algorithm
  - Try to utilize all resources available
  - Possible to turn a poor sequential algorithm into a reasonable parallel algorithm (bubble sort and parallel bubble sort)
- Completely new approach
  - New algorithm from scratch
  - Harder to develop
  - Sometimes yield better solution

Bubble Sort

The idea of bubble sort is to compare two adjacent elements. If they are not in the right order,switch them. Do this comparing and switching (if necessary) until the end of the array is reached. Repeat this process from the beginning of the array n times.

- One of the straight-forward sorting methods
  - Cycles through the list
  - Compares consecutive elements and swaps them if necessary
  - Stops when no more out of order pair
- Slow & inefficient
- Average performance is $O(n^2)$

Bubble Sort Example

Here we want to sort an array containing [8, 5, 1]. The following figure shows how we can sortthis array using bubble sort. The elements in consideration are shown in **bold.**

8, 5, 1	Switch 8 and 5
5, **8, 1**	Switch 8 and 1
5, 1, 8	Reached end start again.
5, 1, 8	Switch 5 and 1
1, **5, 8**	No Switch for 5 and 8
1, 5, 8	Reached end start again.
1, 5, 8	No switch for 1, 5
1, **5, 8**	No switch for 5, 8
1, 5, 8	Reached end.

But do not start again since this is the nth iteration of same process

Parallel Bubble Sort

- Implemented as a pipeline.
- Let local_size = n / no_proc. We divide the array in no_proc parts, and each process executes the bubble sort on its part, including comparing the last element with the first one belonging to the next thread.
- Implement with the loop (instead of j<i)
for (j=0; j<n-1; j++)
- For every iteration of i, each thread needs to wait until the previous thread has finished that iteration before starting.
- We'll coordinate using a barrier.

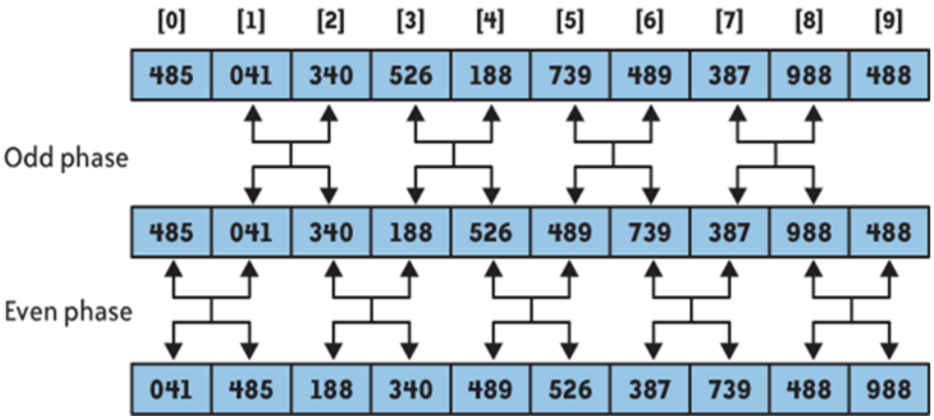
Algorithm for Parallel Bubble Sort

```
For $k = 0$ to $n-2$
If $k$ is even then
    for $i = 0$ to $(n/2)-1$ do in parallel
        If $A[2i] > A[2i+1]$ then
            Exchange $A[2i] \leftrightarrow A[2i+1]$
Else
    for $i = 0$ to $(n/2)-2$ do in parallel
        If $A[2i+1] > A[2i+2]$ then
            Exchange $A[2i+1] \leftrightarrow A[2i+2]$
```

10. Next *k*

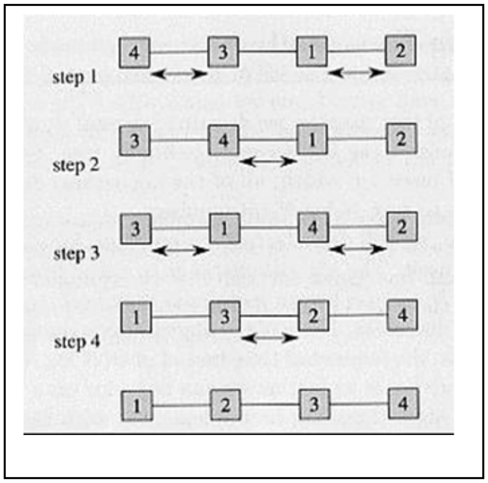
Parallel Bubble Sort Example 1

- Compare all pairs in the list in parallel
- Alternate between odd and even phases
- Shared flag, **sorted**, initialized to true at beginning of each iteration (2 phases), if any processor perform swap, **sorted** = false



Parallel Bubble Sort Example 2

- How many steps does it take to sort the following sequence from least to greatest using the Parallel Bubble Sort? How does the sequence look like after 2 cycles?
- Ex: 4,3,1,2



Merge Sort

- Collects sorted list onto one processor
- Merges elements as they come together
- Simple tree structure
- Parallelism is limited when near the root

Theory:

To sort *A*[*p* .. *r*]:

1. **Divide Step**

If a given array *A* has zero or one element, simply return; it is already sorted. Otherwise, split*A*[*p* .. *r*] into two subarrays*A*[*p* .. *q*] and *A*[*q* + 1 .. *r*], each containing about half of the elements of *A*[*p* .. *r*]. That is, *q* is the halfway point of *A*[*p* .. *r*].

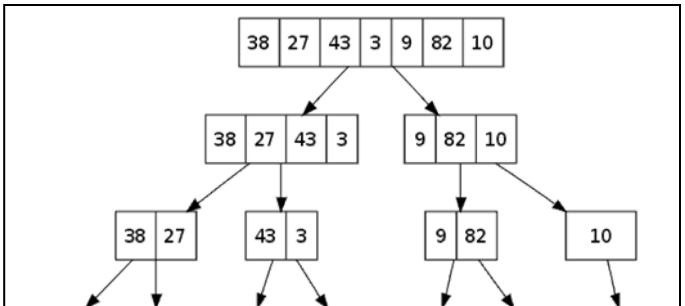
2. **Conquer Step**

Conquer by recursively sorting the two subarrays*A*[*p* .. *q*] and *A*[*q* + 1 .. *r*].

3. **Combine Step**

Combine the elements back in *A*[*p* .. *r*] by merging the two sorted subarrays*A*[*p* .. *q*] and *A*[*q* + 1 .. *r*] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (*A*, *p*, *q*, *r*).

Example:

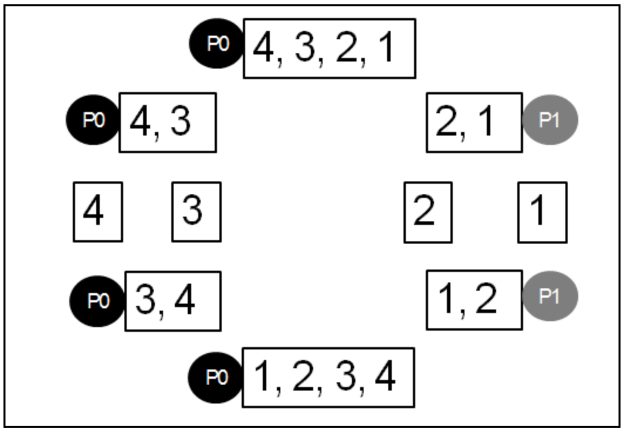


Parallel Merge Sort

- Parallelize processing of sub-problems
- Max parallelization achived with one processor per node (at each layer/height)

Parallel Merge Sort Example

- Perform Merge Sort on the following list of elements. Given 2 processors, P0 & P1, which processor is reponsible for which comparison?
- 4,3,2,1



Algorithm for Parallel Merge Sort

```
Procedure parallelMergeSort
Begin
Create processors Pi where i = 1 to n
if i > 0 then recieve size and parent from the root
recieve the list, size and parent from the root
endif
midvalue= listsize/2
if both children is present in the tree then
send midvalue, first child
send listsize-mid,second child
send list, midvalue, first child
send list from midvalue, listsize-midvalue, second child
call mergelist(list,0,midvalue,list, midvalue+1,listsize,temp,0,listsize)
store temp in another array list2
else
call parallelMergeSort(list,0,listsize)
endif
if i >0 then
send list, listsize,parent
endif
end
```

INPUT:

1. Array of integer numbers.

OUTPUT:

1. Sorted array of numbers

FAQ

1. What is sorting?
2. What is parallel sort?
3. How to sort the element using Bubble Sort?
4. How to sort the element using Parallel Bubble Sort?

5. How to sort the element using Parallel Merge Sort?
6. How to sort the element using Merge Sort?
7. What is searching?
8. Different types of searching methods.
9. Time complexities of sorting and searching methods.
10. How to calculate time complexity?
11. What are space complexity of all sorting and searching methods?
12. Explain what is best, worst and average case for each method of searching and sorting.

ALGORITHM ANALYSIS

1. Time Complexity Of parallel Merge Sort and parallel Bubble sort in best case is(when all data is already in sorted form):O(n)
2. Time Complexity Of parallel Merge Sort and parallel Bubble sort in worst case is: O(n logn)
3. Time Complexity Of parallel Merge Sort and parallel Bubble sort in average case is: O(n logn)

APPLICATIONS

1. Representing Linear data structure & Sequential data organization : structure & files
2. For Sorting sequential data structure

CONCLUSION

Thus, we have studied Parallel Bubble and Parallel Merge sort implementation.

Experiment No: 4

Aim: Design and implement parallel algorithm utilizing all resources available. For

- Binary Search for Sorted Array
- 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)

Objective: To study and implementation of searching techniques.

Outcome: Students will be understand the implementation of Binary search and BFS, DFS

Pre-requisites:

64-bit Open source Linux or its derivative

Programming Languages: C++/JAVA/PYTHON/R

Theory:

Binary Search:

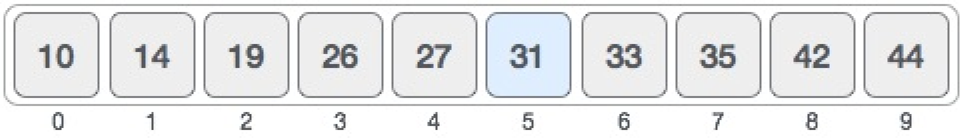
In computer science, binary search, also known as half-interval search,logarithmic search,or binary chop,is a search algorithm that finds the position of a target value within a sorted array. Binary search compares the target value to the middle element of the array. If they are not equal, the half in which the target cannot lie is eliminated and the search continues on the remaining half, again taking the middle element to compare to the target value, and repeating this until the target value is found. If the search ends with the remaining half being empty, the target is not in the array. Even though the idea is simple, implementing binary search correctly requires attention to some subtleties about its exit conditions and midpoint calculation.

Binary search runs in logarithmic time in the worst case, making *O*(log *n*) comparisons, where *n* is the number of elements in the array, the *O* is Big O notation, and log is the logarithm. Binary search takes constant (*O*(1)) space, meaning that the

space taken by the algorithm is the same for any number of elements in the array.Binary search is faster than linear search except for small arrays, but the array must be sorted first. Although specialized data structures designed for fast searching, such as hash tables, can be searched more efficiently, binary search applies to a wider range of problems.

How Binary Search Works?

For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.

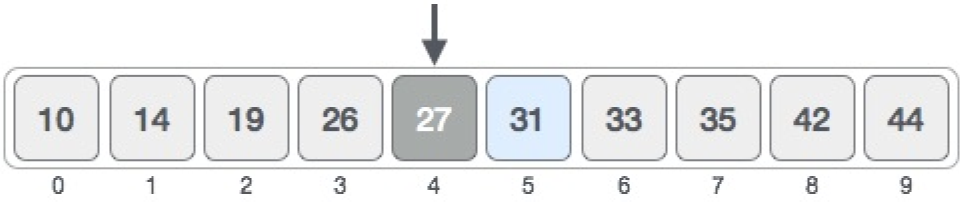


10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

First, we shall determine half of the array by using this formula −

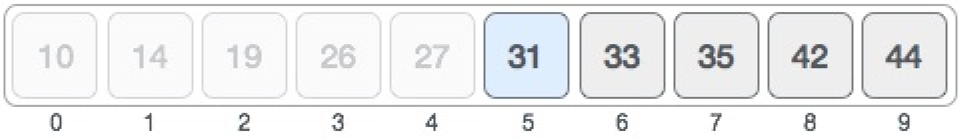
```
mid = low + (high - low) / 2
```

Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.



10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

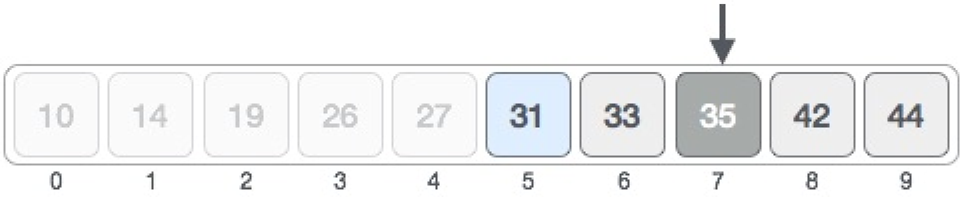
Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.



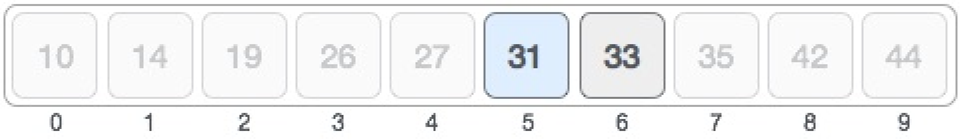
We change our low to mid + 1 and find the new mid value again.

```
low = mid + 1
mid = low + (high - low) / 2
```

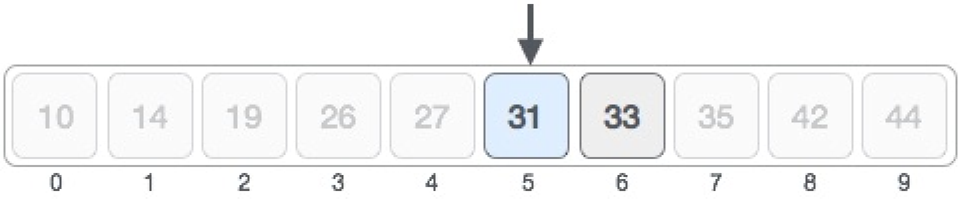
Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



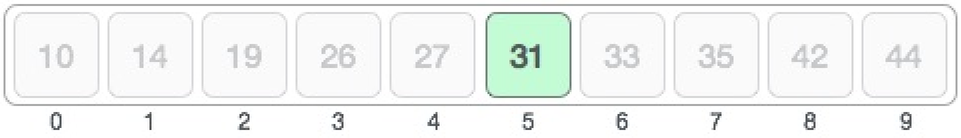
The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.



Hence, we calculate the mid again. This time it is 5.



We compare the value stored at location 5 with our target value. We find that it is a match.



We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Binary Search Code:

```
    #include<iostream>

    #include<stdlib.h>

    #include<omp.h> using
    namespace std;

    int binary(int *, int, int, int);

    int binary(int *a, int low, int high, int key){
    int mid; mid=(low+high)/2;

    int low1,low2,high1,high2,mid1,mid2,found=0,loc=-1;
    #pragma omp parallel sections { #pragma omp section {
    low1=low; high1=mid;

    while(low1<=high1) {

    if(!(key>=a[low1] && key<=a[high1])) { low1=low1+high1; continue; }
    cout<<"here1";

    mid1=(low1+high1)/2;

    if(key==a[mid1]) { found=1; loc=mid1; low1=high1+1; } else
    if(key>a[mid1]) { low1=mid1+1; } else if(key<a[mid1])
    high1=mid1-1;}
      }


     #pragma omp section {
     low2=mid+1; high2=high;
     while(low2<=high2) {

     if(!(key>=a[low2] && key<=a[high2])) { low2=low2+high2; continue; }
cout<<"here2";

mid2=(low2+high2)/2;

if(key==a[mid2]) { found=1; loc=mid2;
low2=high2+1; } else if(key>a[mid2]) {
low2=mid2+1; } else if(key<a[mid2])
high2=mid2-1;

} } }

return loc;

}
```

```
int main(){

int *a,i,n,key,loc=-1;

cout<<"\n enter total no of elements=>";
cin>>n; a=new int[n]; cout<<"\n enter
elements=>"; for(i=0;i<n;i++) { cin>>a[i]; }
cout<<"\n enter key to find=>"; cin>>key;
loc=binary(a,0,n-1,key);

if(loc==-1) cout<<"\n Key not found.";

else cout<<"\n Key found at
position=>"<<loc+1; return 0;
}
```

Breadth-First Search :

Graph traversals

Graph traversal means visiting every vertex and edge exactly once in a well-defined order. While using certain graph algorithms, you must ensure that each vertex of the graph is visited exactly once. The order in which the vertices are visited are important and may depend upon the algorithm or question that you are solving.

During a traversal, it is important that you track which vertices have been visited. The most common way of tracking vertices is to mark them.

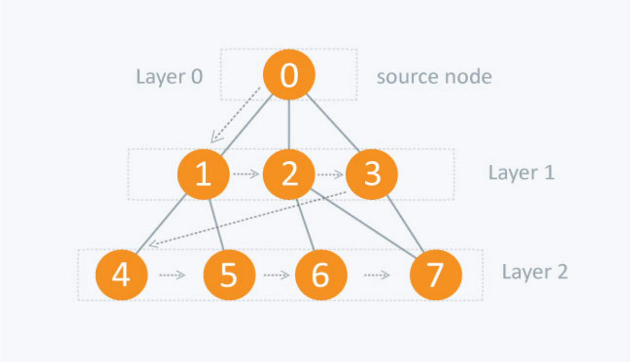
Breadth First Search (BFS)

There are many ways to traverse graphs. BFS is the most commonly used approach.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer



Consider the following diagram.

The distance between the nodes in layer 1 is comparitively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

Traversing child nodes

A graph can contain cycles, which may bring you to the same node again while traversing the graph. To avoid processing of same node again, use a boolean array which marks the node after it is processed. While visiting the nodes in the layer of a graph, store them in a manner such that you can traverse the corresponding child nodes in a similar order.

In the earlier diagram, start traversing from 0 and visit its child nodes 1, 2, and 3. Store them in the order in which they are visited. This will allow you to visit the child nodes of 1 first (i.e. 4 and 5), then of 2 (i.e. 6 and 7), and then of 3 (i.e. 7) etc.

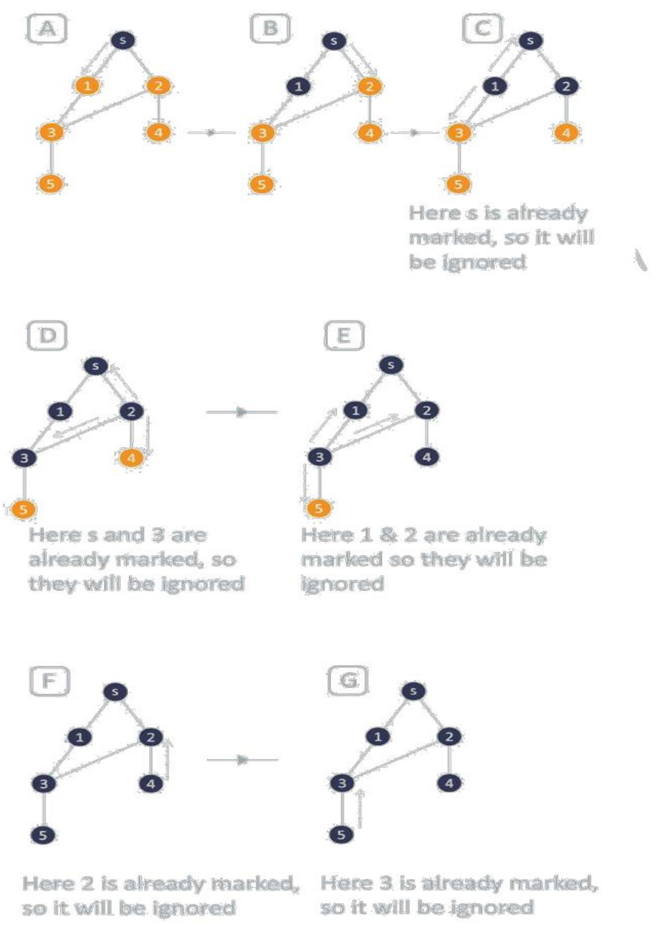
To make this process easy, use a queue to store the node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neigbors of the node will be visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.

Pseudo code for BFS:

```
 FS (G, s) //Where G is the graph and s is the
source node let Q be queue.
Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are
marked.
marks as visited.
while ( Q is not
empty)
   //Removing that vertex from queue,whose neighbour will be
   visited now v = Q.dequeue( )
```

[Type here]

```
//processing      all      the
neighbours   of   v   for   all
neighbours w of v in Graph G
   if w is not visited
        Q.enqueue( w ) //Stores w in Q to further visit its
        neighbour mark w as visited.
```


A
B
C
s
1
2
3
4
5
Here s is already marked, so it will be ignored
D
E
Here s and 3 are already marked, so they will be ignored
Here 1 & 2 are already marked so they will be ignored
F
G
Here 2 is already marked, so it will be ignored
Here 3 is already marked, so it will be ignored

Traversing process

Code for BFS:

```
#include<iostream>
#include<stdlib.h>
#include<queue> using
namespace std;
class node{ public: node *left, *right; int data;};

class Breadthfs{ public: node *insert(node *, int); void bfs(node *); };
node *insert(node *root, int data){ / /inserts a node in tree if(!root)
{


      root=new node; root-
      >left=NULL; root-
      >right=NULL; root-
      >data=data; return root;

  }

  queue<node *> q; q.push(root);
  while(!q.empty()) {

  node *temp=q.front(); q.pop();
  if(temp->left==NULL) {
```

```
  temp->left=new node; temp->left-
  >left=NULL; temp->left->right=NULL ;
  temp->left->data=data; return root;

  }

  else { q.push(temp->left); }
  if(temp->right==NULL) { temp-
  >right=new node; temp->right-
  >left=NULL; temp->right-
  >right=NULL; temp->right-
  >data=data;
  return root; }

else { q.push(temp->right); }


} }


void bfs(node *head){

queue<node*> q; q.push(head); int
qSize; while (!q.empty()) {

qSize = q.size();

#pragma omp parallel for / /creates
parallel threads for (int i = 0; i <
qSize; i++) { node* currNode; #pragma
omp critical {

currNode = q.front();

 q.pop(); cout<<"\t"<<currNode->data; } / /prints parent
node #pragma omp critical {

if(currNode->left) q.push(currNode->left); / /push parent's left node in
queue if(currNode->right) q.push(currNode->right); } / /push parent's
right node in queue } }

}

int main(){

node *root=NULL; int data; char
ans; do { cout<<"\n enter
data=>"; cin>>data;
root=insert(root,data);

cout<<"do you want insert one more node?";
cin>>ans; }while(ans=='y'||ans=='Y');

bfs(root);
return 0;
}
```

Conclusion: We have implemented Binary searching and BFS .

GROUP B: ASSIGNMENTS

Heuristic Search

- **Aim:**

Use heuristic search techniques to implement best first search and A* algorithm.

- **Software Requirements:**

NetBeans IDE

- **Program Execution:**

Using BFS

```
run:
Enter No of nodes : 5

Enter the name of node 1 : A
Enter the name of node 2 : B
Enter the name of node 3 : C
Enter the name of node 4 : D
Enter the name of node 5 : E

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : B
Enter distance between nodes A and B : 3

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : C
Enter distance between nodes A and C : 1

Do you want to add any adjacent node to node A : n

Do you want to add any adjacent node to node B : y

Enter the name of adjacent node of B : D
Enter distance between nodes B and D : 3

Do you want to add any adjacent node to node B : y

Enter the name of adjacent node of B : E
Enter distance between nodes B and E : 2

Do you want to add any adjacent node to node B : n

Do you want to add any adjacent node to node C : n

Do you want to add any adjacent node to node D : n

Do you want to add any adjacent node to node E : n

A : (B,3), (C,1)
B : (D,3), (E,2)
C :
```

```
D :
E :

Priority queue contents :

A
C         B
B
B

E         D
D
D

Goal node 'D'  found

Path :
A, B, D
BUILD SUCCESSFUL (total time: 1 minute 8 seconds)
```

Using A* algorithm

```
run:
Enter No of nodes : 4

Enter the name of node 1 : A
Enter the heuristic value of node A : 6
Enter the name of node 2 : B
Enter the heuristic value of node B : 4

Enter the name of node 3 : C
Enter the heuristic value of node C : 3
Enter the name of node 4 : D
Enter the heuristic value of node D : 1

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : B
Enter distance between nodes A and B : 1

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : C
Enter distance between nodes A and C : 3

Do you want to add any adjacent node to node A : n

Do you want to add any adjacent node to node B : y
```

```
Enter the name of adjacent node of B : D
Enter distance between nodes B and D : 2

Do you want to add any adjacent node to node B : n

Do you want to add any adjacent node to node C : y

Enter the name of adjacent node of C : D
Enter distance between nodes C and D : 5

Do you want to add any adjacent node to node C : n

Do you want to add any adjacent node to node D : n

A (hx = 6) : (B,1), (C,3)
B (hx = 4) : (D,2)
C (hx = 3) : (D,5)
D (hx = 1) :



Fx of node A = 6
Open List : A
Closed List : Empty
Open List : Empty
Closed List : A
Fx of node B = 5
Fx of node C = 6
Open List : B    C
Open List : C
Closed List : A  B
Fx of node D = 4
Open List : D    C
Open List : C
Closed List : A  B          D

Path :
A, B, D

Goal node 'D'  found
BUILD SUCCESSFUL (total time: 1 minute 37 seconds)
```

Eight Puzzle Problem

- **Aim:**

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

- **Software Requirements:**

NetBeans IDE

- **Program Execution:**

```
run:

 Enter start Board :
Enter one tile as '-' ie. Blank tile

Enter the value of tile [0][0] : -
Enter the value of tile [0][1] : a
Enter the value of tile [0][2] : c
Enter the value of tile [1][0] : h
Enter the value of tile [1][1] : b
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 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)
```

Medical Expert System

- **Aim:**

Implement expert system for medical diagnosis of 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 .

ChatBot

- **Aim:**

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

- **Software Requirements:**

NetBeans, program-ab library.

- **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.

- **Program Execution:**
  Execute a ChatBot Jar file using following command
  Java –jar dist/ChatBot.jar (Due to execution of jar file, aimlif folder updated automatically)
  Aimlif contains csv(comma separated files) which are the filed used for early execution

Human : Hello

Robot : Hi nice to see 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

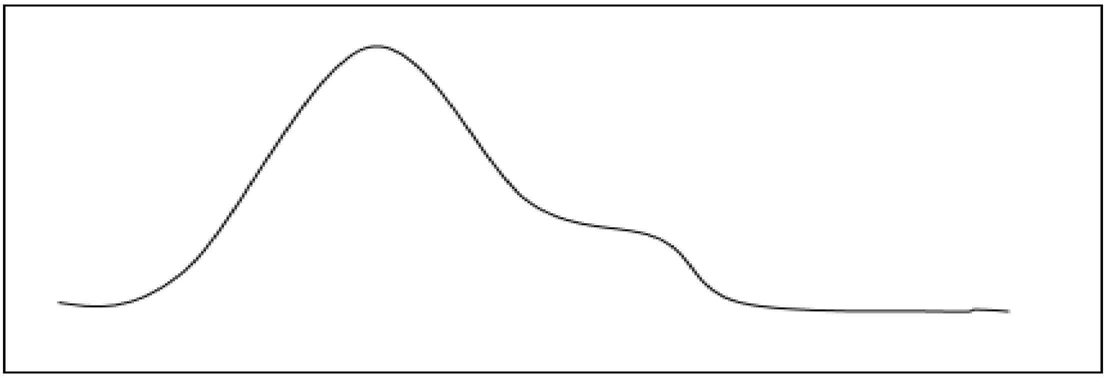
Robot : Goodbye ! It was nice to talk to you

Hill Climbing Algorithm

AIM : To solve 8 queens problem using Hill Climbing Algorithm.

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.

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 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
```

Goal Stack Planning

Title : Implement goal stack planning for the following configurations from the blocks world.

Aim: To implement goal stack planning for given the blocks world configurations.

Requirement: Java Editor (Eclipse/Net beans)

Theory:

Planning is process of determining various actions that often lead to a solution.

Planning is useful for non-decomposable problems where subgoals often interact.

Goal Stack Planning (in short GSP) is the one of the simplest planning algorithm that is designed to handle problems having compound goals. And it utilizes STRIP as a formal language for specifying and manipulating the world with which it is working.

This approach uses a Stack for plan generation. The stack can contain Sub-goal and actions described using predicates. The Sub-goals can be solved one by one in any order.

Algorithm:

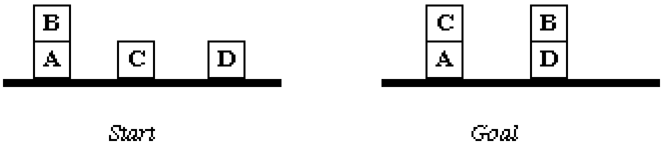
```
Push the Goal state in to the Stack
Push the individual Predicates of the Goal State into the Stack
Loop till the Stack is empty
   Pop an element E from the stack
   IF E is a Predicate
        IF E is True then
             Do Nothing
        ELSE
            Push the relevant action into the Stack
            Push the individual predicates of the Precondition of the action into the Stack
    Else IF E is an Action
        Apply the action to the current State.
        Add the action 'a' to the plan
```

The Goal Stack Planning Algorithms works will the stack. It starts by pushing the unsatisfied goals into the stack. Then it pushes the individual subgoals into the stack and its pops an element out of the stack. When popping an element out of the stack the element could be either a predicate describing a situation about our world or it could be an action that can be applied to our world under consideration. So based on the kind of element we are popping out from the stack a decision has to be made. If it is a Predicate. Then compares it with the description of the current world, if it is satisfied or is already present in our current situation then there is nothing to do because already its true. On the contrary if the Predicate is not true then we have to select and push relevant action satisfying the predicate to the Stack.

So after pushing the relevant action into the stack its precondition should also has to be pushed into the stack. In order to apply an operation its precondition has to be satisfied. In other words the present situation of the world should be suitable enough to apply an operation. For that, the preconditions are pushed into the stack once after an action is pushed.

Input:

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



No of Blocks : 4

Initial stage : (on b a)^(ontable c)^(ontable a)^(ontable d)^(clear b)^(clear c)^(clear d)^(AE)

Final stage : (on c a)^(on b d)^(ontable a)^(ontable d)^(clear c)^(clear b)^(AE)

Output:

Set of actions to be taken:

1. (unstack b d)
2. (stack b d)

3. (pick c)
4. (stack c a)

Eight puzzle problem

Aim: Implement A star algorithm for eight puzzle problem.

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 node s 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.

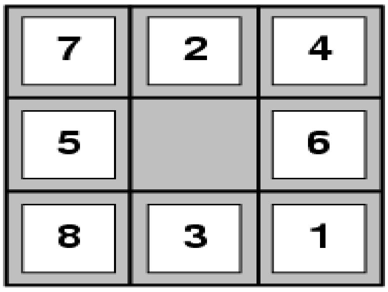
5. Otherwise, expand n generating its children and directing pointers from each child node to n.

   - 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.

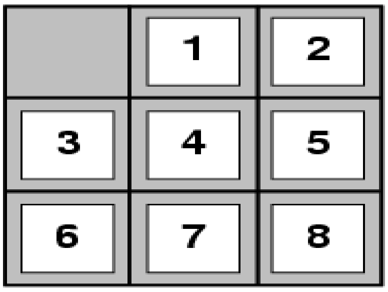
Example of calculation of heuristic values for 8-puzzle problem:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ =no. of squares from desired location of each tile



7	2	4
5		6
8	3	1

Start State

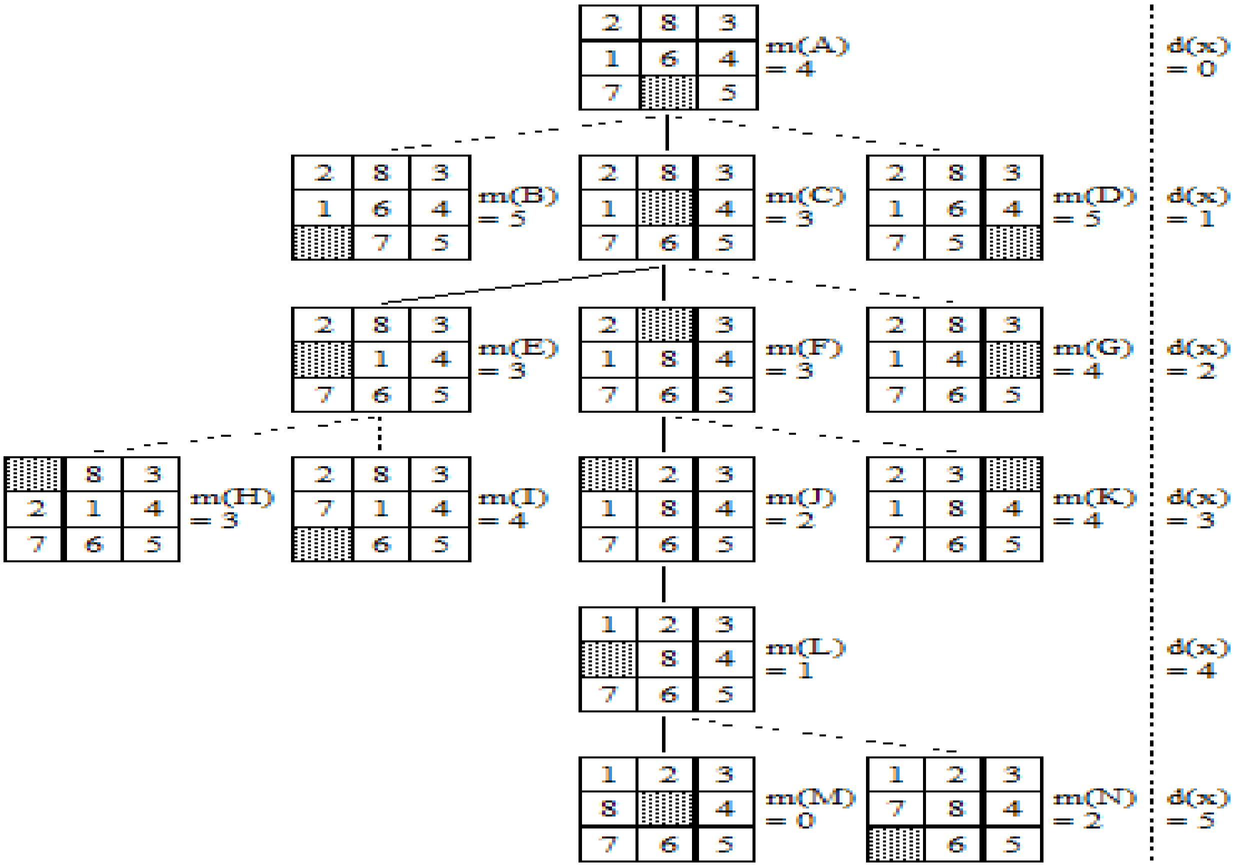


	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = 8$
- $h_2(S) = 3+1+2+2+2+3+3+2 = 18$

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.

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

n-queens problem

Aim: Implement n-queens problem using Backtracking.

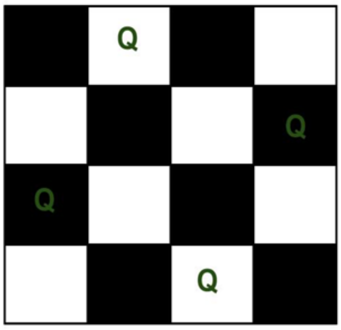
Objective:

Student will learn:

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

Theory:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



ıinary matrix which has 1s for the blocks where queens are

ıe output matrix for above 4 queen solution.

{ 0, 0, 1, 0}

Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints.while there are untried conflagrations

```
{
  generate the next configuration
  if queens don't attack in this configuration then
  {
    print this configuration;
  }
}
```

Backtracking Algorithm

Backtracking is finding the solution of a problem whereby the solution depends on the previous steps taken.

In backtracking, we first take a step and then we see if this step taken is correct or not i.e., whether it will give a correct answer or not. And if it doesn't, then we just come back and change our first step. In general, this is accomplished by recursion. Thus, in backtracking, we first start with a partial sub-solution of the problem (which may or may not lead us to the solution) and then check if we can proceed further with this sub-solution or not. If not, then we just come back and change it.

Thus, the general steps of backtracking are:

- start with a sub-solution
- check if this sub-solution will lead to the solution or not
- If not, then come back and change the sub-solution and continue again

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

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.

3) If all rows have been tried and nothing worked, return false to trigger backtracking.

Conclusion: **N-queens problem is implemented using backtracking.**

Assignment No. **Goal Stack planing Algorithm.**

Aim: Implement Goal Stack planing Algorithm.

Objective:

Student will learn:

i) The Basic Concepts of Goal Stack Planing.

ii) General Algorithm for Goal Stack Planing.

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.

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

When two subgoals G1 and G2 are given, a noninterleaved 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 Planing Algorithm.**

Experiment

Title :

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

Aim :

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.

Prerequisites:

Fundamentals of R -Programming Languages

Objectives :

To learn the concept of how to display summary statistics for each feature Available in the dataset

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 ∑ 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.

Division by N-1

Divide the sum of the squared differences by one less than the number of data points. 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 a 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,Varience value and Percentiles of probabilities also Display Specific use quantile.

3. Dispaly 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?

Experiment No:

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

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.

Fundamentals of R -Programming Languages

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

I.Bayes Theorem:

Bayes' Theorem is a way of finding a probability when we know certain other probabilities.

The 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)***P(Smoke)**
=*1% x 90%***10%**
=9%

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

II. 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

Input:

Structured Dataset : PimaIndiansDiabetes 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:

6. What is Bayes Theorem?
7. What is confusion matrix?
8. Which function is used to split the dataset in R?
9. What are steps of Naïve Bayes algorithm?
10. What is conditional probability?

Experiment No:

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. Sample Test data set available here https://www.capitalbikeshare.com/trip-history-data

Aim: Implement a classification algorithm that is Recursive Partitioning and Regression Trees. 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 predicted and actual values.
5. Plot the tree and label it.

Prerequisites: Fundamentals of R -Programming Languages

Objective: To learn the concept of rpart package and algorithm.

Theory: I . *Recursive partitioning,* or "*classification and regression trees,*" is a recursive partitioning method which is a fundamental tool in data mining. It is a prediction method often used with dichotomous outcomes that avoids the assumptions of linearity. This technique creates prediction rules by repeatedly dividing the sample into subgroups, with each subdivision being formed by separating the sample on the value of one of the predictor variables. The end result is a set of branching questions that forms a treelike structure in which each final branch provides a yes/no prediction of the outcome. rpart is for modeling decision trees.

This section briefly describes CART modeling, conditional inference trees, and random forests.

CART Modeling via rpart

Classification and regression trees (as described by Brieman, Freidman, Olshen, and Stone) can be generated through the rpart package. The general steps are provided below followed by two examples.

1. Grow the Tree

To grow a tree, use

rpart(*formula*, **data=**, **method=,control=)** where

formula	is in the format *outcome ~ predictor1+predictor2+predictor3*+ect.
data=	specifies the data frame
method=	**"class"** for a classification tree **"anova"** for a regression tree
	optional parameters for controlling tree growth. For example, control=rpart.control(minsplit=30, cp=0.001)

control= requires that the minimum number of observations in a node be 30 before attempting a split and that a split must decrease the overall lack of fit by a factor of 0.001 (cost complexity factor) before being attempted.

2. Examine the results

The following functions help us to examine the results.

printcp(*fit*)	display cp table
plotcp(*fit*)	plot cross-validation results
rsq.rpart(*fit*)	plot approximate R-squared and relative error for different splits (2 plots). labels are only appropriate for the "anova" method.
print(*fit*)	print results
summary(*fit*)	detailed results including surrogate splits
plot(*fit*)	plot decision tree
text(*fit*)	label the decision tree plot
post(*fit*, file=)	create postscript plot of decision tree

In trees created by **rpart( )**, move to the **LEFT** branch when the stated condition is true (see the graphs below).

E:g:

The rpart function builds a model of recursive partitioning and regression trees. The following code snippet shows how to use the rpart function to construct a decision tree.

```
fit <- rpart(Play- Outlook+ Temperature +Humidity+ Wind,
method="class11,data=play_decision,control=rpart.control(minsplit=l),
parms=list(split=•information•))
```

The rpart function has four parameters. The first parameter, Play - Out look + Temperature+ Hurnidi ty + Wind, is the model indicating that attribute *Play* can be predicted based on attributes *Outlook, Temperature, Humidity,* and *Wind.* The second parameter, *method,* is set to "class," telling R it is building a classification tree. The third parameter, *data,* specifies the dataframe containing those attributes mentioned in the formula. The fourth parameter, *control,* is optional and controls the tree growth. In the preceding example, control=rpart. control (minsplit=l) requires that each node have at least one observation before attempting a split. The rminsplit= 1 makes sense for the small dataset, but for larger datasets rminsplit could be set to 10% of the dataset size to combat overfitting. Besides minsp 1 it, other parameters are available to control the construction of the decision tree. For example, rpart. control (rnaxdepth=10, cp=O. 001) limits the depth of the tree to no more than 10, and a split must decrease the overall lack of fit by a factor of 0.001 before being attempted. The last parameter (parms) specifies the purity measure being used for the splits. The value of sp 1 it can be either information (for using the information gain) or g ini (for using the Gini index).

For the tripadvisory dataset the following code snippet for rpart is used:
fit<-rpart(train$Member.type~.,data=train, method="class")

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

Input:

Structured Dataset : capitalbikeshare Dataset

File: capitalbikeshare-tripdata.csv

Output:

1. Splitted dataset according to Split ratio.
2. Summary of every node in the constructed decision tree
3. To generate predictions from a fitted *rpart* object

Conclusion: Hence, we have studied classification based on rpart package and its usage.

Questions:

1. What are decision tree?
2. What is rpart?
3. What are applications of rpart?
4. Advantages of decision trees?

Experiment No:

Title: 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.

Aim: Implement the sentiment analysis of twitter data. Implement the following operations:

1. Sentiment analysis of twitter dataset.
2. Classify the tweets as positive and negative tweets from dataset.

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: I. 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.

II.Python Tweepy library: This library provides access to the entire twitter RESTful API methods. Each method can accept various parameters and return responses.

III. 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.

IV. Authentication: create OAuthHandler object : Tweepy supports oauth authentication. Authentication is handled by the tweepy AuthHandler class.

V.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?