1.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)// push parent's left node in queue

                 q.push(currNode->left);

             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?";

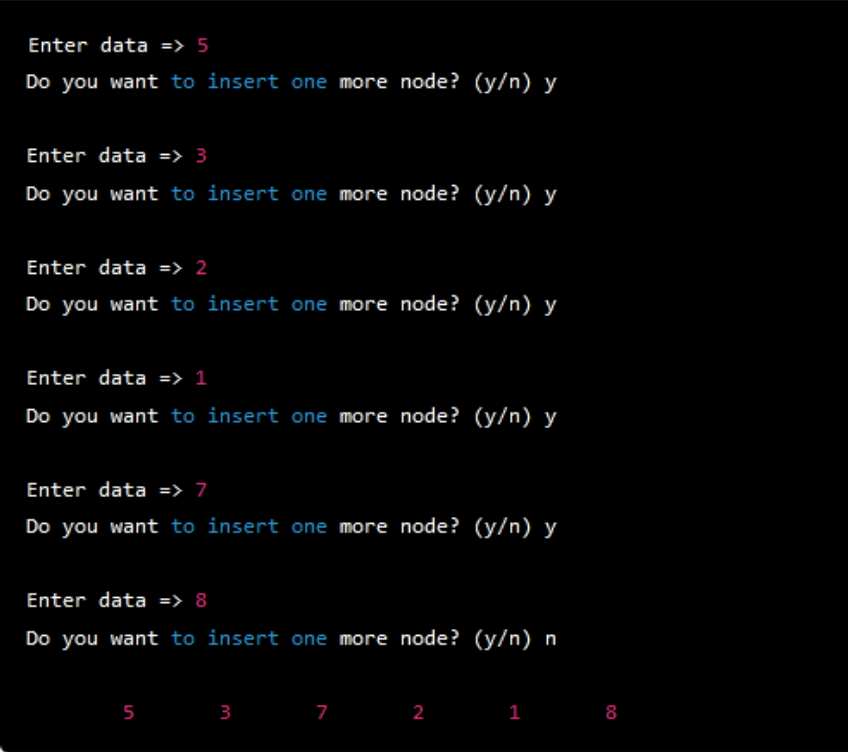
     cin>>ans;

    }while(ans=='y'||ans=='Y');

    bfs(root);

    return 0;

}



2.DFS

#include <iostream>

#include <vector>

#include <stack>

#include <omp.h>

using namespace std;

const int MAX = 100000;

vector<int> graph[MAX];

bool visited[MAX];

void dfs(int node) {

    stack<int> s;

    s.push(node);

    while (!s.empty()) {

        int curr\_node = s.top();

        s.pop();

        if (!visited[curr\_node]) {

            visited[curr\_node] = true;

            if (visited[curr\_node]) {

            cout << curr\_node << " ";

        }

            #pragma omp parallel for

            for (int i = 0; i < graph[curr\_node].size(); i++) {

                int adj\_node = graph[curr\_node][i];

                if (!visited[adj\_node]) {

                    s.push(adj\_node);

                }

            }

        }

    }

}

int main() {

    int n, m, start\_node;

    cout << "Enter No of Node,Edges,and start node:" ;

    cin >> n >> m >> start\_node;

         //n: node,m:edges

cout << "Enter Pair of edges:" ;

    for (int i = 0; i < m; i++) {

        int u, v;

        cin >> u >> v;

//u and v: Pair of edges

        graph[u].push\_back(v);

        graph[v].push\_back(u);

    }

    #pragma omp parallel for

    for (int i = 0; i < n; i++) {

        visited[i] = false;

    }

    dfs(start\_node);

    for (int i = 0; i < n; i++) {

        if (visited[i]) {

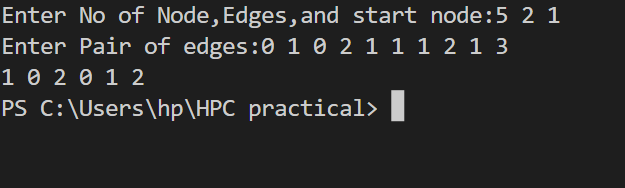
            cout << i << " ";

        }

    }

    return 0;

}



Bubble sort:

#include<iostream>

#include<stdlib.h>

#include<omp.h>

using namespace std;

void bubble(int \*, int);

void swap(int &, int &);

void bubble(int \*a, int n)

{

    for(  int i = 0;  i < n;  i++ )

     {

     int first = i % 2;

     #pragma omp parallel for shared(a,first)

     for(  int j = first;  j < n-1;  j += 2  )

       {

         if(  a[ j ]  >  a[ j+1 ]  )

          {

                 swap(  a[ j ],  a[ j+1 ]  );

          }

           }

     }

}

void swap(int &a, int &b)

{

    int test;

    test=a;

    a=b;

    b=test;

}

int main()

{

    int \*a,n;

    cout<<"\n enter total no of elements=>";

    cin>>n;

    a=new int[n];

    cout<<"\n enter elements=>";

    for(int i=0;i<n;i++)

    {

     cin>>a[i];

    }

    bubble(a,n);

    cout<<"\n sorted array is=>";

    for(int i=0;i<n;i++)

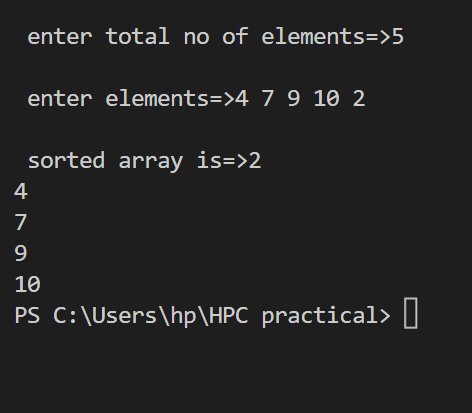
    {

     cout<<a[i]<<endl;

    }

return 0;

}



Merge sort:

#include<iostream>

#include<stdlib.h>

#include<omp.h>

using namespace std;

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

void mergesort(int a[],int i,int j)

{

    int mid;

    if(i<j)

    {

        mid=(i+j)/2;

        #pragma omp parallel sections

        {

            #pragma omp section

            {

                mergesort(a,i,mid);

            }

            #pragma omp section

            {

                mergesort(a,mid+1,j);

            }

        }

        merge(a,i,mid,mid+1,j);

    }

}

void merge(int a[],int i1,int j1,int i2,int j2)

{

    int temp[1000];

    int i,j,k;

    i=i1;

    j=i2;

    k=0;

    while(i<=j1 && j<=j2)

    {

        if(a[i]<a[j])

        {

            temp[k++]=a[i++];

        }

        else

        {

            temp[k++]=a[j++];

    }

    }

    while(i<=j1)

    {

        temp[k++]=a[i++];

    }

    while(j<=j2)

    {

        temp[k++]=a[j++];

    }

    for(i=i1,j=0;i<=j2;i++,j++)

    {

        a[i]=temp[j];

    }

}

int main()

{

    int \*a,n,i;

    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];

    }

   //    start=.......

//#pragma omp…..

    mergesort(a, 0, n-1);

//          stop…….

    cout<<"\n sorted array is=>";

    for(i=0;i<n;i++)

    {

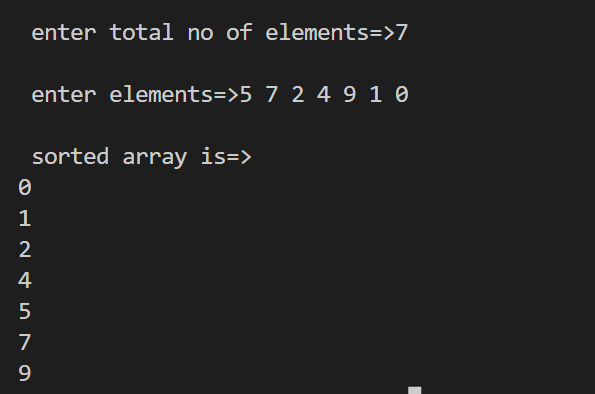
        cout<<"\n"<<a[i];

    }

    // Cout<<Stop-Start

    return 0;

}



Min and Max

#include <iostream>

//#include <vector>

#include <omp.h>

#include <climits>

using namespace std;

void min\_reduction(int arr[], int n) {

  int min\_value = INT\_MAX;

  #pragma omp parallel for reduction(min: min\_value)

  for (int i = 0; i < n; i++) {

  if (arr[i] < min\_value) {

    min\_value = arr[i];

  }

  }

  cout << "Minimum value: " << min\_value << endl;

}

void max\_reduction(int arr[], int n) {

  int max\_value = INT\_MIN;

  #pragma omp parallel for reduction(max: max\_value)

  for (int i = 0; i < n; i++) {

  if (arr[i] > max\_value) {

    max\_value = arr[i];

  }

  }

  cout << "Maximum value: " << max\_value << endl;

}

void sum\_reduction(int arr[], int n) {

  int sum = 0;

   #pragma omp parallel for reduction(+: sum)

   for (int i = 0; i < n; i++) {

  sum += arr[i];

  }

  cout << "Sum: " << sum << endl;

}

void average\_reduction(int arr[], int n) {

  int sum = 0;

  #pragma omp parallel for reduction(+: sum)

  for (int i = 0; i < n; i++) {

  sum += arr[i];

  }

  cout << "Average: " << (double)sum / (n) << endl;

}

int main() {

    int \*arr,n;

    cout<<"\n enter total no of elements=>";

    cin>>n;

    arr=new int[n];

    cout<<"\n enter elements=>";

    for(int i=0;i<n;i++)

    {

     cin>>arr[i];

    }

//   int arr[] = {5, 2, 9, 1, 7, 6, 8, 3, 4};

//   int n = size(arr);

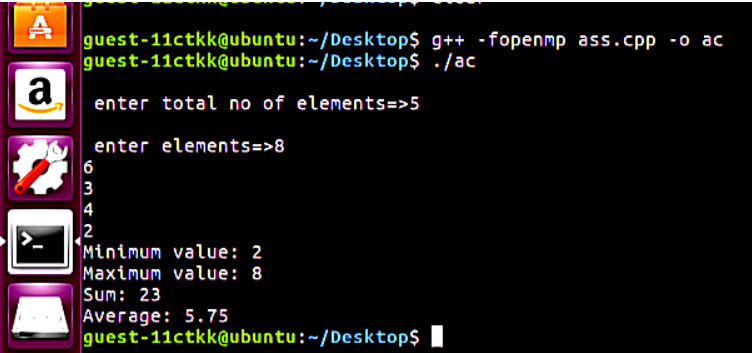
  min\_reduction(arr, n);

  max\_reduction(arr, n);

  sum\_reduction(arr, n);

  average\_reduction(arr, n);

}



Accuracy

import tensorflow as tf

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(32, (3,3), activation='relu', input\_shape=(28, 28, 1)),

tf.keras.layers.MaxPooling2D((2, 2)),tf.keras.layers.Flatten(),

tf.keras.layers.Dense(10, activation='softmax')

])

mnist = tf.keras.datasets.mnist

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

def train(model, x\_train, y\_train, rank, size):

# Split the data across the nodes

    n =len(x\_train)

    chunk\_size = n

    start = rank \* chunk\_size

    end = (rank + 1) \* chunk\_size

    if rank == size - 1:

        end = n

    x\_train\_chunk = x\_train[start:end]

    y\_train\_chunk = y\_train[start:end]

# Compile the model

    model.compile(optimizer='adam',

        loss='sparse\_categorical\_crossentropy',

        metrics=['accuracy'])

#Train the model

    model.fit(x\_train\_chunk, y\_train\_chunk, epochs=1, batch\_size=32)

#. Compute the accuracy on the training data

    train\_loss, train\_acc = model.evaluate(x\_train\_chunk, y\_train\_chunk, verbose=2)

# Reduce the accuracy across all nodes

    train\_acc = comm.allreduce(train\_acc, op=MPI.SUM)

    return train\_acc / size

epochs = 5

for epoch in range(epochs):

# Train the model

    train\_acc = train(model, x\_train, y\_train, rank, size)

# Compute the accuracy on the test data

    test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

    test\_acc = comm.allreduce(test\_acc, op=MPI.SUM)

    if rank ==0:

        print(f"Epoch {epoch + 1}: Train accuracy = {train\_acc:.4f}, Test accuracy = {test\_acc / size:.4f}")

