//

// main.cpp

// Struct1

//

// Created by Zahra Khatami on 4/10/15.

// Copyright (c) 2015 Zahra Khatami. All rights reserved.

//plot results and compare them

//if you can plot them on something that shows moving

#include <iostream>

#include <string>

#include <sstream>

#include <vector>

#include <math.h>

#include <fstream>

#include <cmath>

#include <array>

#include <boost/timer/timer.hpp>

using std::vector;

using namespace std;

typedef basic\_ofstream<char> ofstream;

typedef unsigned int coord\_t;

//Parameters for computing force and well seperated cells for each node

struct Parameters{

int th=10,th\_A=40, it=1;

double time\_step=0.1;

double theta=3, G1=6.673\*pow(10.0,-11.0);

};

//Data for each node

struct Body {

int ID1,parent,Hilbert\_dis,num\_A;

double m1,v1[3],force[3]={};

unsigned int r1[3];

};

//Data for each cell

struct Cell {

int ID2, parent2,NumNodes=0,level,neighbors[6]={-1,-1,-1,-1,-1,-1},Ncell;

vector<int> child, scell, list\_cell1, list\_cell2, A;

double r2[3],rd[3],m2,boundary[6]={4444,0,4444,0,4444,0};

};

//Read input from txt file

void read\_Input(Body body[]){

int N; string temp,temp1;

fstream textfile;

string::size\_type sz;

textfile.open("ex1000.txt");

textfile>>temp;

N = atoi(temp.c\_str());

for (int i=0; i<N; ++i){

textfile>>temp>>temp1;

body[i].ID1=atoi(temp.c\_str());

body[i].m1=stof(temp1,&sz); body[i].num\_A=i;

for (int j=2; j<5; ++j){

textfile>>temp; body[i].r1[j-2]=stof(temp,&sz);}

for (int j=5; j<8; ++j){

textfile>>temp; body[i].v1[j-5]=stof(temp,&sz);}

body[i].parent=0;}}

//Detremining max and min in the boundary cells

void apply\_changes(vector<Cell> &cell, Body body[],int n){

//cout<<"\*"<<cell[n].A.size()<<"\*";

for(int j=0; j<cell[n].A.size(); ++j){

int i=cell[n].A[j];

if (body[i].r1[0]<cell[n].boundary[0]) cell[n].boundary[0]=body[i].r1[0];

if (body[i].r1[0]>cell[n].boundary[1]) cell[n].boundary[1]=body[i].r1[0];

if (body[i].r1[1]<cell[n].boundary[2]) cell[n].boundary[2]=body[i].r1[1];

if (body[i].r1[1]>cell[n].boundary[3]) cell[n].boundary[3]=body[i].r1[1];

if (body[i].r1[2]<cell[n].boundary[4]) cell[n].boundary[4]=body[i].r1[2];

if (body[i].r1[2]>cell[n].boundary[5]) cell[n].boundary[5]=body[i].r1[2];}}

//Hilbert distance

void Hilbert\_distance(coord\_t\* X, int b, int n){

coord\_t M = 1 << (b - 1), P, Q, t;

int i;

// Inverse undo

for (Q = M; Q > 1; Q >>= 1){

P = Q - 1;

for (i = 0; i < n; i++)

if ((X[i] & Q) != 0)

X[0] ^= P; // invert

else{

t = (X[0] ^ X[i]) & P;

X[0] ^= t;

X[i] ^= t;}}

// Gray encode

for (i = 1; i < n; i++)

X[i] ^= X[i - 1];

t = 0;

for (Q = M; Q > 1; Q >>= 1)

if ((X[n - 1] & Q)!=0)

t ^= Q - 1;

for (i = 0; i < n; i++)

X[i] ^= t;}

//Hilbert

void Hilbert(Body body[], int N){

for(int i=0; i<N; ++i){

coord\_t X[3]={body[i].r1[0],body[i].r1[1],body[i].r1[2]};

int m=10;

int n=m\*3; int Y[n];

Hilbert\_distance(X,m, 3); int sum=0;

int j=n-1, M1=m;

while(j>=0){

M1=M1-1;

for(int k=0; k<3; ++k){

Y[j]=(X[k]>>M1 & 1);

j=j-1; }}

for(int i=0; i<n; ++i)

sum=sum+Y[i]\*pow(2,i);

body[i].Hilbert\_dis=sum;}}

//Sorting

void merge(vector<int> &A,Body body[],int, int , int ,int);

void sort\_merge(vector<int> &A,Body body[], int low,int high, int N){

int mid;

if(low<high){

mid=(low+high)/2;

sort\_merge(A,body,low,mid,N);

sort\_merge(A,body,mid+1,high,N);

merge(A,body,low,mid,high,N);

}

}

void merge(vector<int> &A,Body body[],int low, int mid, int high, int N){

int h,i,j,k,B[N];

i=low; h=low; j=mid+1;

while((h<=mid) && (j<=high)){

if((body[A[h]].Hilbert\_dis<body[A[j]].Hilbert\_dis)){

B[i]=body[A[h]].ID1;

body[A[h]].num\_A=i;

h++; }

else{

B[i]=body[A[j]].ID1;

body[A[j]].num\_A=i;

j++;}

i=i+1;}

if(h>mid)

for(k=j;k<=high; ++k){

B[i]=body[A[k]].ID1;

body[A[k]].num\_A=i;

i=i+1;}

else

for(k=h; k<=mid; ++k){

B[i]=body[A[k]].ID1;

body[A[k]].num\_A=i;

i=i+1;}

for(k=low; k<=high; ++k)

A[k]=B[k];}

//Finding out if the node is in that subcubic

bool InCube(Body body[], vector<Cell> &cell, int node, int p){

for (int i=0; i<3; ++i)

if(body[node].r1[i]<cell[p].boundary[2\*i] || body[node].r1[i]>cell[p].boundary[2\*i+1])

return false;

return true;}

//Determining the boundry of the each subcube

void det\_boundary\_subcube(vector<Cell> &cell,int n){

double a1,a2,a3,b1,b2,b3,c1,c2,c3; Cell cell1[8];

a1=cell[n].boundary[0]; a2=(cell[n].boundary[0]+cell[n].boundary[1])/2; a3=cell[n].boundary[1];

b1=cell[n].boundary[2]; b2=(cell[n].boundary[2]+cell[n].boundary[3])/2; b3=cell[n].boundary[3];

c1=cell[n].boundary[4]; c2=(cell[n].boundary[4]+cell[n].boundary[5])/2; c3=cell[n].boundary[5];

double A1[]={a1,a2,b1,b2,c1,c2},A2[]={a1,a2,b1,b2,c2,c3},A3[]={a1,a2,b2,b3,c1,c2}, A4[]={a1,a2,b2,b3,c2,c3}, A5[]={a2,a3,b1,b2,c1,c2}, A6[]={a2,a3,b1,b2,c2,c3}, A7[]={a2,a3,b2,b3,c1,c2}, A8[]={a2,a3,b2,b3,c2,c3};

copy(begin(A1),end(A1),begin(cell1[0].boundary)); cell.push\_back(cell1[0]);

copy(begin(A2),end(A2),begin(cell1[1].boundary)); cell.push\_back(cell1[1]);

copy(begin(A3),end(A3),begin(cell1[2].boundary)); cell.push\_back(cell1[2]);

copy(begin(A4),end(A4),begin(cell1[3].boundary)); cell.push\_back(cell1[3]);

copy(begin(A5),end(A5),begin(cell1[4].boundary)); cell.push\_back(cell1[4]);

copy(begin(A6),end(A6),begin(cell1[5].boundary)); cell.push\_back(cell1[5]);

copy(begin(A7),end(A7),begin(cell1[6].boundary)); cell.push\_back(cell1[6]);

copy(begin(A8),end(A8),begin(cell1[7].boundary)); cell.push\_back(cell1[7]);}

//creating octree and determining its parameters

template<class type> class octree {

public:

octree() {};

void root(vector<Cell> &cell, Body body[], int n);

void strc(Parameters params,vector<Cell> &cell, Body body[], vector<int> &cell\_arrange,int n);

void neighbor(vector<Cell> &cell, Body body[], vector<int> &cell\_arrange);

void cell\_list(Parameters params,vector<Cell> &cell, Body body[], vector<int> &cell\_arrange);

void traverse\_tree(Parameters params,vector<Cell> &cell, Body body[], int root,int node, int root2);

void compute\_force(Parameters params,vector<Cell> &cell,Body body[],int parent,int node);

void new\_tree(Parameters params,vector<Cell> &cell,Body body[],vector<int> &cell\_arrange,int parent,int node);

void insert\_node(Parameters params,vector<Cell> &cell,Body body[],vector<int> &cell\_arrange,int parent, int node);

};

//Computing Center of mass and position of cell

template <class type> void octree<type>::root(vector<Cell> &cell, Body body[], int n){

vector<float> CM(3,0);

for (int j=0; j<cell[n].A.size(); ++j){

cell[n].m2=cell[n].m2+body[cell[n].A[j]].m1;

for (int i=0; i<3; ++i)

CM[i]=CM[i]+body[cell[n].A[j]].r1[i] \* body[cell[n].A[j]].m1;}

for (int i=0; i<3;++i){

cell[n].r2[i]=CM[i]/cell[n].m2;

cell[n].rd[i]=cell[n].boundary[2\*i+1]-cell[n].boundary[2\*i];}}

//Arranging cells from root to leaf in Octree

template <class type> void octree<type>::strc(Parameters params,vector<Cell> &cell, Body body[], vector<int> &cell\_arrange,int n){

int i,j,p; float a1[8],b1[8],c1[8]; octree<float> tree;

p=(cell[n].Ncell)\*8; Cell cell1;

for(i=1; i<9; ++i){

cout<<"\n---\n"<<i+p<<"\n";

cell[i+p].level=cell[n].level+1; cell[i+p].ID2=i+p;

cell[i+p].parent2=cell[n].ID2; cell[i+p].NumNodes=(int)((float)((cell[n].NumNodes)/8));

int start=(int)ceil((i-1)\*((float)(cell[n].A.size())/8)), en=(int)ceil((float)((i)\*(cell[n].A.size()))/8);

//cout<<"^"<<start<<"^"<<en<<"^"<<cell[i+p].NumNodes<<"^"<<cell[n].A.size()<<"\n";

a1[i]=0, b1[i]=0, c1[i]=0;

if(start<en){

for(int k=start; k<en; ++k){

//cout<<":"<<cell[n].A[k]<<":";

cell[i+p].A.push\_back(cell[n].A[k]);}

if(cell[i+p].A.size()>=1){

/\* for(j=0; j<cell[i+p].A.size(); ++j)

cout<<"~"<<cell[i+p].A[j]<<"~";\*/

apply\_changes(cell,body,i+p);

//cout<<"{"<<cell[i+p].boundary[0]<<","<<cell[i+p].boundary[1]<<","<<cell[i+p].boundary[2]<<","<<cell[i+p].boundary[3]<<","<<cell[i+p].boundary[4]<<","<<cell[i+p].boundary[5]<<"}\n";

a1[i]=cell[i+p].boundary[1]-cell[i+p].boundary[0], b1[i]=cell[i+p].boundary[3]-cell[i+p].boundary[2], c1[i]=cell[i+p].boundary[5]-cell[i+p].boundary[4];

// cout<<"=="<<a1[i]<<","<<b1[i]<<","<<c1[i]<<"==";

if(a1[i] <= params.th\_A && b1[i] <= params.th\_A && c1[i] <= params.th\_A){

if(cell[i+p].A.size()==1){

cell[n].child.push\_back(body[cell[i+p].A[0]].ID1);

cout<<","<<body[cell[i+p].A[0]].ID1<<",";

cell[i+p].child.push\_back(body[cell[i+p].A[0]].ID1);

body[cell[i+p].A[0]].parent=n;}

else{

cell[i+p].Ncell=(int)cell\_arrange.size();

cell\_arrange.push\_back(cell[i+p].ID2);

cell[n].scell.push\_back(cell[i+p].ID2);

cout<<"-"<<cell[i+p].ID2<<"-";

for(j=0; j<cell[i+p].A.size(); ++j){

body[body[cell[i+p].A[j]].ID1].parent=i+p;

cell[i+p].child.push\_back(body[cell[i+p].A[j]].ID1);

cout<<","<<body[cell[i+p].A[j]].ID1<<",";

}}}

else{

cell[i+p].Ncell=(int)cell\_arrange.size();

cell\_arrange.push\_back(cell[i+p].ID2);

cout<<"-"<<cell[i+p].ID2<<"-";

cell[n].scell.push\_back(cell[i+p].ID2);

cell[i+p].parent2=n;

for(j=1; j<9; ++j)

cell.push\_back(cell1);}

}}}

for(i=1; i<9; ++i)

if(cell[i+p].A.size()>=1)

if(a1[i] > params.th\_A || b1[i] > params.th\_A || c1[i] > params.th\_A){

tree.root(cell,body,i+p);

tree.strc(params,cell,body,cell\_arrange,i+p);}}

//Modifying octree when position of nodes are changed (1)

template <class type> void octree<type>::insert\_node(Parameters params,vector<Cell> &cell,Body body[],vector<int> &cell\_arrange,int parent, int node){

octree<float> tree; float a1[2],b1[2],c1[2];

a1[0]=cell[parent].boundary[1]-cell[parent].boundary[0], b1[0]=cell[parent].boundary[3]-cell[parent].boundary[2], c1[0]=cell[parent].boundary[5]-cell[parent].boundary[4];

if(cell[parent].boundary[0]>body[node].r1[0]) cell[parent].boundary[0]=body[node].r1[0];

if(cell[parent].boundary[1]<body[node].r1[0]) cell[parent].boundary[1]=body[node].r1[0];

if(cell[parent].boundary[2]>body[node].r1[1]) cell[parent].boundary[2]=body[node].r1[1];

if(cell[parent].boundary[3]<body[node].r1[1]) cell[parent].boundary[3]=body[node].r1[1];

if(cell[parent].boundary[4]>body[node].r1[2]) cell[parent].boundary[4]=body[node].r1[2];

if(cell[parent].boundary[5]<body[node].r1[2]) cell[parent].boundary[5]=body[node].r1[2];

a1[1]=cell[parent].boundary[1]-cell[parent].boundary[0], b1[1]=cell[parent].boundary[3]-cell[parent].boundary[2], c1[1]=cell[parent].boundary[5]-cell[parent].boundary[4];

cell[parent].NumNodes=cell[parent].NumNodes+1;

cell[parent].A.push\_back(node);

body[node].parent=parent;

if(a1[1] == a1[0] && b1[1] == b1[0] && c1[1] == c1[0]){

cell[parent].child.push\_back(node); body[node].parent=parent; cell[parent].A.push\_back(node);}

if((a1[1]!=a1[0] && a1[1]>a1[0]) || (b1[1]!=b1[0] && b1[1]>b1[0]) || (c1[1]!=c1[0] && c1[1]>c1[0]))

tree.strc(params,cell,body,cell\_arrange,parent);

if(a1[1] >= params.th\_A || b1[1] >= params.th\_A || c1[1] >= params.th\_A){

int p=cell[parent].Ncell;

for(int i=1; i<9; ++i){

if(InCube(body,cell,node,i+p)==true)

tree.insert\_node(params,cell, body, cell\_arrange, i+p, node);}}}

//Modifying octree when position of nodes are changed (2)

template <class type> void octree<type>::new\_tree(Parameters params,vector<Cell> &cell,Body body[],vector<int> &cell\_arrange,int parent,int node){

int k=-1; octree<float> tree;

if (parent!=0){

for(int j=0; j<6; ++j)

if(cell[parent].neighbors[j]!=-1 && k==-1)

if(InCube(body,cell,node,cell[parent].neighbors[j])==true){

k=1; int temp=cell[parent].neighbors[j];

tree.insert\_node(params,cell,body,cell\_arrange,temp,node);}

if(k==-1 && cell[parent].parent2!=0)

tree.new\_tree(params,cell, body,cell\_arrange,cell[parent].parent2,node);}}

//Determining neighbors of each cell (left,right,up,down,back,front)

template <class type> void octree<type>::neighbor(vector<Cell> &cell, Body body[], vector<int> &cell\_arrange){

for (int i=1; i<cell\_arrange.size(); ++i){

int temp=cell\_arrange[i];

for(int j=1; j<cell\_arrange.size(); ++j)

if(cell[cell\_arrange[j]].level==cell[temp].level && cell\_arrange[j]!=temp){

if(cell[cell\_arrange[j]].boundary[1]==cell[temp].boundary[0])

cell[temp].neighbors[0]=cell[cell\_arrange[j]].ID2;

else if(cell[cell\_arrange[j]].boundary[0]==cell[temp].boundary[1])

cell[temp].neighbors[1]=cell[cell\_arrange[j]].ID2;

else if(cell[cell\_arrange[j]].boundary[4]==cell[temp].boundary[5])

cell[temp].neighbors[2]=cell[cell\_arrange[j]].ID2;

else if(cell[cell\_arrange[j]].boundary[5]==cell[temp].boundary[4])

cell[temp].neighbors[3]=cell[cell\_arrange[j]].ID2;

else if(cell[cell\_arrange[j]].boundary[3]==cell[temp].boundary[2])

cell[temp].neighbors[4]=cell[cell\_arrange[j]].ID2;

else if(cell[cell\_arrange[j]].boundary[2]==cell[temp].boundary[3])

cell[temp].neighbors[5]=cell[cell\_arrange[j]].ID2;}}}

//creating interaction list for each cell (1)

template <class type> void octree<type>::cell\_list(Parameters params,vector<Cell> &cell, Body body[], vector<int> &cell\_arrange){

int i,temp; octree<float> tree;

for (i=0; i<cell\_arrange.size(); ++i){

tree.traverse\_tree(params,cell,body,cell\_arrange[i],temp=cell[cell\_arrange[i]].child[0],0);}}

//creating interaction list for each cell (2)

template <class type> void octree<type>::traverse\_tree(Parameters params,vector<Cell> &cell, Body body[],int root,int node, int root2){

int i;

octree<float> tree;

float D=sqrt(pow((body[node].r1[0]-cell[root2].r2[0]),2.0)+pow((body[node].r1[1]-cell[root2].r2[1]),2.0)+pow((body[node].r1[2]-cell[root2].r2[2]),2.0));

float r=sqrt(pow(cell[root2].rd[0],2.0)+pow(cell[root2].rd[1],2.0)+pow(cell[root2].rd[2],2.0));

if (D<(r/params.theta)){

if(cell[root2].child.size()>=1)

for(i=0; i<cell[root2].child.size(); ++i)

cell[root].list\_cell1.push\_back(cell[root2].child[i]);

if(cell[root2].scell.size()>=1)

for(i=0; i<cell[root2].scell.size(); ++i)

tree.traverse\_tree(params,cell,body,root,node,cell[root2].scell[i]);}

else

cell[root].list\_cell2.push\_back(cell[root2].ID2);}

//computing force, new position and new velocity

template <class type> void octree<type>::compute\_force(Parameters params,vector<Cell> &cell,Body body[],int parent,int node){

vector<float> a(3,0); double t=params.time\_step;

double G=params.G1; //////Get it from input

if(cell[parent].list\_cell1.size()!=0)

for(int j=0; j<3; ++j)

for (int i=0; i<cell[parent].list\_cell1.size(); ++i)

body[node].force[j]=body[node].force[j]+ (body[cell[parent].list\_cell1[i]].r1[j]-body[node].r1[j]) \*G\* body[cell[parent].list\_cell1[i]].m1 \* body[node].m1 /pow((1+(pow((body[cell[parent].list\_cell1[i]].r1[0]-body[node].r1[0]),2.0)+pow((body[cell[parent].list\_cell1[i]].r1[1]-body[node].r1[1]),2.0)+pow((body[cell[parent].list\_cell1[i]].r1[2]-body[node].r1[2]),2.0))),1.5);

if(cell[parent].list\_cell2.size()!=0)

for(int j=0; j<3; ++j)

for (int i=0; i<cell[parent].list\_cell2.size()!=0; ++i)

body[node].force[j]=body[node].force[j]+ (cell[cell[parent].list\_cell2[i]].r2[j]-body[node].r1[j]) \*G\* cell[cell[parent].list\_cell2[i]].m2 \* body[node].m1 / pow((1+(pow((cell[cell[parent].list\_cell2[i]].r2[0]-body[node].r1[0]),2.0)+pow((cell[cell[parent].list\_cell2[i]].r2[1]-body[node].r1[1]),2.0)+pow((cell[cell[parent].list\_cell2[i]].r2[2]-body[node].r1[2]),2.0))),1.5);

for(int j=0; j<3; ++j){

a[j]=body[node].force[j]/body[node].m1;

body[node].r1[j]=body[node].r1[j]+body[node].v1[j]\*t+0.5\*a[j]\*t\*t;

body[node].v1[j]=body[node].v1[j]+a[j]\*t;}}

int main(int argc, const char \* argv[]){

int i;

vector<int> cell\_arrange; octree<float> tree;

vector<int> m, list\_cell1, list\_cell2;

int N; string temp; fstream textfile;

textfile.open("ex1000.txt");

textfile>>temp;

N = atoi(temp.c\_str());

Parameters params;

Body \*body=new Body[N];

vector<Cell> cell;

Cell cell1,cell2;

read\_Input(body);

cell\_arrange.push\_back(0);

cell1.NumNodes=N; cell1.ID2=0; cell1.level=0;

cell1.Ncell=0;

for(i=0; i<N; ++i)

cell1.A.push\_back(i);

cell.push\_back(cell1);

Hilbert(body,N);

sort\_merge(cell[0].A,body,0,N,N);

apply\_changes(cell,body,0);

for(int j=1; j<9; ++j)

cell.push\_back(cell2);

tree.root(cell,body,0);

tree.strc(params,cell,body,cell\_arrange,0);

/\*tree.neighbor(cell,body,cell\_arrange);

boost::timer::cpu\_timer timer;

tree.cell\_list(params,cell,body,cell\_arrange);

for(int it=0 ; it<params.it; it++)

for (i=0; i<N; ++i){

tree.compute\_force(params,cell,body,body[i].parent,i);

if((InCube(body,cell,i,body[i].parent)==false)){

cell[body[i].parent].NumNodes=cell[body[i].parent].NumNodes-1;

if(InCube(body,cell,i,0)==true)

tree.new\_tree(params,cell,body,cell\_arrange,body[i].parent,i);}}

boost::timer::cpu\_times elapsed = timer.elapsed();

std::cout << " CPU TIME: " << (elapsed.user + elapsed.system) / 1e9 << " seconds"<< " WALLCLOCK TIME: " << elapsed.wall / 1e9 << " seconds"<< std::endl;\*/

return 0;

}