// -----------------------------------------

// CS 6596 - Project 2

// Name: Unmesh Suryawanshi

// Net id: qd6395

// -----------------------------------------

// Declaration of the header files

#include<stdlib.h>

#include<iostream>

#include<math.h>

#include<fstream>

using namespace std;

#define first\_call 1

#define end\_call 2

// Declaration of structure

template<class d\_type> class elt

{

public:

d\_type val;

elt<d\_type> \*forward;

};

template<class d\_type> class seq

{

public:

elt<d\_type> \*first;

seq()

{

first = NULL;

}

//login to calculate the length of the list

int l\_len()

{

elt<d\_type> \*node=first;

int t\_elt=0;

while(node!=NULL)

{

node = node->forward;

t\_elt++;

}

return t\_elt;

}

// logic to check the black data in LL

bool check\_empty()

{

return(first == NULL);

}

// add the data in LL

void add\_data(d\_type tmp1)

{

elt<d\_type> \*tmp = new elt<d\_type>;

tmp->val = tmp1;

tmp->forward = NULL;

if(first == NULL)

{

first = tmp;

}

else

{

elt<d\_type> \*node=first;

while(node->forward!=NULL)

{

node = node->forward;

}

node->forward = tmp;

}

}

// to initiate with the 1st val

d\_type& find\_first\_val()

{

return first->val;

}

d\_type& find\_val\_at\_index(int idx)

{

elt<d\_type> \*node=first;

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

{

if(node == NULL)

return first->val;

node = node->forward;

}

return node->val;

}

void del(d\_type &tmp1)

{

elt<d\_type> \*node=first;

if(first->val == tmp1)

{

first = NULL;

}

while(node->forward!=NULL)

{

if(node->forward->val == tmp1)

{

node->forward = node->forward->forward;

}

node = node->forward;

}

}

void del\_first()

{

elt<d\_type> \*first\_elt\_new;

if(first == NULL)

return;

first\_elt\_new = first->forward;

delete first;

first = first\_elt\_new;

}

void del()

{

first = NULL;

}

d\_type& del\_end()

{

elt<d\_type> \*node=first,\*tmp;

if(node==NULL)

return node->val;

if(node->forward == NULL)

{

first = NULL;

return node->val;

}

while(node->forward->forward != NULL)

{

node = node->forward;

}

tmp = node->forward;

node->forward = NULL;

return tmp->val;

}

};

class fixed\_data

{

public:

static double cell\_r;

static int ch\_no, pl\_exp, cell\_no, total\_cl, cluster\_cell\_no, path\_mat[9][9];

static void set\_vals()

{

//Declaration of all the data

cell\_r = 1000, ch\_no = 15, pl\_exp = 4, cell\_no = 9, total\_cl = 3, cluster\_cell\_no = 3;

path\_mat[0][0] = 0; path\_mat[0][1] = 2000; path\_mat[0][2] = 2000; path\_mat[0][3] = 4000;

path\_mat[0][4] = 6000; path\_mat[0][5] = 4000; path\_mat[0][6] = 4000; path\_mat[0][7] = 6000; path\_mat[0][8] = 6000;

path\_mat[1][0] = 2000; path\_mat[1][1] = 0; path\_mat[1][2] = 2000; path\_mat[1][3] = 2000;

path\_mat[1][4] = 4000; path\_mat[1][5] = 2000; path\_mat[1][6] = 4000; path\_mat[1][7] = 4000; path\_mat[1][8] = 6000;

path\_mat[2][0] = 2000; path\_mat[2][1] = 2000; path\_mat[2][2] = 0; path\_mat[2][3] = 4000;

path\_mat[2][4] = 4000; path\_mat[2][5] = 2000; path\_mat[2][6] = 2000; path\_mat[2][7] = 4000; path\_mat[2][8] = 4000;

path\_mat[3][0] = 4000; path\_mat[3][1] = 2000; path\_mat[3][2] = 4000; path\_mat[3][3] = 0;

path\_mat[3][4] = 2000; path\_mat[3][5] = 2000; path\_mat[3][6] = 4000; path\_mat[3][7] = 4000; path\_mat[3][8] = 6000;

path\_mat[4][0] = 6000; path\_mat[4][1] = 4000; path\_mat[4][2] = 4000; path\_mat[4][3] = 2000;

path\_mat[4][4] = 0; path\_mat[4][5] = 2000; path\_mat[4][6] = 4000; path\_mat[4][7] = 2000; path\_mat[4][8] = 4000;

path\_mat[5][0] = 4000; path\_mat[5][1] = 2000; path\_mat[5][2] = 2000; path\_mat[5][3] = 2000;

path\_mat[5][4] = 2000; path\_mat[5][5] = 0; path\_mat[5][6] = 2000; path\_mat[5][7] = 2000; path\_mat[5][8] = 4000;

path\_mat[6][0] = 4000; path\_mat[6][1] = 4000; path\_mat[6][2] = 2000; path\_mat[6][3] = 4000;

path\_mat[6][4] = 4000; path\_mat[6][5] = 2000; path\_mat[6][6] = 0; path\_mat[6][7] = 2000; path\_mat[6][8] = 2000;

path\_mat[7][0] = 6000; path\_mat[7][1] = 4000; path\_mat[7][2] = 4000; path\_mat[7][3] = 4000;

path\_mat[7][4] = 2000; path\_mat[7][5] = 2000; path\_mat[7][6] = 2000; path\_mat[7][7] = 0; path\_mat[7][8] = 2000;

path\_mat[8][0] = 6000; path\_mat[8][1] = 6000; path\_mat[8][2] = 4000; path\_mat[8][3] = 6000;

path\_mat[8][4] = 4000; path\_mat[8][5] = 4000; path\_mat[8][6] = 2000; path\_mat[8][7] = 2000; path\_mat[8][8] = 0;

}

};

int fixed\_data::ch\_no;

int fixed\_data::total\_cl;

int fixed\_data::cell\_no;

int fixed\_data::cluster\_cell\_no;

int fixed\_data::pl\_exp;

double fixed\_data::cell\_r;

int fixed\_data::path\_mat[9][9];

// Logic to checking the dropped calls cause due to c0 channel interference.

class track\_calls

{

public:

seq<int> interferer;

double stored\_SIR;

int selected\_ch;

bool current\_val;

track\_calls()

{

current\_val = false;

}

};

//Logic to check cell channel strategy

class cell\_ch

{

public:

int chnl\_no;

bool current\_val;

int \*cl\_list;

cell\_ch(int num)

{

chnl\_no=num;

current\_val = false;

cl\_list = new int[9];

for(int i=0; i<9; i++)

cl\_list[i]=0;

}

// Logic to verify - which cell uses all channels

bool chck\_ch\_cluster(int cluster\_no)

{

switch(cluster\_no)

{

case 0:

return (cl\_list[0]|cl\_list[1]|cl\_list[2])>0?true:false;

case 1:

return (cl\_list[3]|cl\_list[4]|cl\_list[5])>0?true:false;

case 2:

return (cl\_list[6]|cl\_list[7]|cl\_list[8])>0?true:false;

}

return false;

}

};

// Logic to check the call details

class call\_stored

{

public:

double stored\_SIR;

int number, cell, duration, time\_req, decision, channel, first\_time, end\_time;

seq<track\_calls> check\_list;

bool operator==(call\_stored &tmp1)

{

return (( number!=tmp1.number) || (time\_req!=tmp1.time\_req) || (first\_time!=tmp1.first\_time)

||(end\_time!=tmp1.end\_time) || (cell!=tmp1.cell) || (duration!=tmp1.duration)

||(stored\_SIR!=tmp1.stored\_SIR) ||(decision!=tmp1.decision) || (channel!=tmp1.channel))?false:true;

}

// Logic to display all the info. about the call no. start and end time, duration, cell, channel

void first\_op\_call();

void call\_end()

{

cout<<"Number = "<<number<<" StartTime = "<<first\_time<<" EndTime = "<<end\_time<<" Cell = "<<(cell+1);

cout<<" Duration = "<<duration<<" Channel = "<<(channel+1);

}

};

class total\_call\_stored

{

public:

seq<call\_stored> call\_list;

// Logic to make a arrangement of call events as per the time

void add\_call\_stored(call\_stored x)

{

call\_list.add\_data(x);

}

void add\_call\_stored\_per\_time(call\_stored x)

{

if(call\_list.check\_empty())

{

call\_list.add\_data(x);

return;

}

elt<call\_stored> \*up;

up= call\_list.first;

call\_stored call\_picker;

while(up!= NULL)

{

call\_picker = up->val;

if(call\_picker.time\_req > x.time\_req)

{

elt<call\_stored> \*tmp = new elt<call\_stored>;

tmp->val = x;

tmp->forward = up;

elt<call\_stored> \*node = call\_list.first;

while(node->forward!= up)

{

node = node->forward;

}

node->forward = tmp;

return;

}

up= up->forward;

}

call\_list.add\_data(x);

return;

}

void del\_call\_stored(call\_stored x)

{

call\_list.del(x);

}

void del\_first\_call\_stored()

{

call\_list.del\_first();

}

};

class dptr

{

public:

// Declaration of the variables and functions

static cell\_ch \*\*channel\_cells;

static total\_call\_stored all\_calls;

static int t\_calls, t\_rj\_calls, t\_ac\_calls;

static double t\_ac\_SIR;

static void starts();

static void ip\_fp(char \*path);

static bool cl\_connect(call\_stored &e);

static void operate\_stored\_call();

static int hunt\_ch(call\_stored &e);

};

cell\_ch \*\*dptr::channel\_cells;

total\_call\_stored dptr::all\_calls;

int dptr::t\_calls;

int dptr::t\_rj\_calls;

int dptr::t\_ac\_calls;

double dptr::t\_ac\_SIR;

void call\_stored::first\_op\_call()

{

if (stored\_SIR >= 22) // condition for the connected call

{

// Display the details for the accepted call

cout<<"Number = "<<number<<" "<<"Time = "<<first\_time<<" "<<"Cell = "<<cell+1<<" "<<"Duration = "<<duration<<" "<<"Accepted"<<" ""Channel = "<<channel+1<<" "<<"SIR = "<<stored\_SIR<<endl;

if (check\_list.check\_empty())

{

cout<<" Interferers: None";

}

else

{

cout<<" Interferers: ";

while(check\_list.find\_first\_val().interferer.l\_len()>0)

{

int interferer\_cell = check\_list.find\_first\_val().interferer.del\_end();

int interferer\_path = fixed\_data::path\_mat[cell][interferer\_cell];

cout<<(interferer\_cell+1)<<"/"<<interferer\_path<<" ";

}

}

}

else

{

// Display the details of the rejected call

cout<<"Number = "<<number<<" "<<"Time = "<<time\_req<<" "<<"Cell = "<<cell+1<<" "<<"Duration = "<<duration<<" "<<"Rejected"<<endl;

cout<<"Reasons:";

while(check\_list.l\_len()>0)

{

track\_calls store\_temp = check\_list.find\_first\_val();

if(store\_temp.current\_val == true)

{

cout<<(store\_temp.selected\_ch+1)<<"/In Use ";

}

else

{

cout<<(store\_temp.selected\_ch+1)<<"/Low SIR = "<<store\_temp.stored\_SIR<<" dB ";

}

check\_list.del\_first();

}

}

cout<<endl<<endl;

}

void dptr::starts()

{

// At start, all values are set to zero

t\_calls = 0, t\_rj\_calls = 0, t\_ac\_calls = 0, t\_ac\_SIR = 0.0;

channel\_cells = new cell\_ch\*[15];

for (int i=0; i<fixed\_data::ch\_no; i++)

{

channel\_cells[i] = new cell\_ch(i);

}

}

bool dptr::cl\_connect(call\_stored &y)

{

int channel = hunt\_ch(y);

int first\_cell = y.cell;

if(channel >= 0)

{

channel\_cells[channel]->current\_val=true;

channel\_cells[channel]->chnl\_no = channel;

y.channel = channel;

channel\_cells[channel]->cl\_list[first\_cell]=1;

return true;

}

else

{

return false;

}

}

void dptr::operate\_stored\_call()

{

// Logic to select the next call. Also to look whether we can allocate the channel if its free, if not then drop the call

while(all\_calls.call\_list.check\_empty() == false)

{

call\_stored call\_picker = all\_calls.call\_list.find\_first\_val();

if (call\_picker.decision == first\_call) // new call

{

t\_calls++;

cout<<"New Call:";

if(cl\_connect(call\_picker))

{

t\_ac\_calls++;

t\_ac\_SIR+=call\_picker.stored\_SIR;

call\_picker.time\_req = call\_picker.first\_time+call\_picker.duration;

call\_picker.decision = end\_call;

all\_calls.add\_call\_stored\_per\_time(call\_picker);

}

else // rejected call

{

t\_rj\_calls++;

}

call\_picker.first\_op\_call();

}

else if (call\_picker.decision == end\_call) // disconnect call

{

call\_picker.end\_time = call\_picker.time\_req;

cout<<"Disconnect:";

call\_picker.call\_end();

cout<<endl<<endl;

channel\_cells[call\_picker.channel]->current\_val=false;

channel\_cells[call\_picker.channel]->cl\_list[call\_picker.cell]=0;

}

all\_calls.del\_first\_call\_stored();

}

}

// Logic for Hunting

// In this fixed channel allocation,

// Channel 1,2,3,4,5 are allocated to cell 1, channel 2,3,4,5,1 are allocated to cell 6, channel 3,4,5,1,2 are allocated to cell 9

// Channel 6,7,8,9,10 are allocated to cell 2, channel 7,8,9,10,6 are allocated to cell 5, channel 8,9,10,6,7 are allocated to cell 8

// Channel 11,12,13,14,15 are allocated to cell 3, channel 12,13,14,15,11 are allocated to cell 4, channel 13,14,15,11,12 are allocated to cell 7

int dptr::hunt\_ch(call\_stored &y)

{

int cell\_grp[2],\*asgn\_chs,asgn\_ch\_no;

int cluster = (y.cell)/fixed\_data::cluster\_cell\_no;

switch(y.cell)

{

// Group 1 contains cell 1,6,9 and they are sharing the same channels 1,2,3,4,5 in a different sequence

case 0:

cell\_grp[0] = 5, cell\_grp[1] = 8;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 0, asgn\_chs[1] = 1, asgn\_chs[2] = 2, asgn\_chs[3] = 3, asgn\_chs[4] = 4;

break;

case 5:

cell\_grp[0] = 0, cell\_grp[1] = 8;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 1, asgn\_chs[1] = 2, asgn\_chs[2] = 3, asgn\_chs[3] = 4, asgn\_chs[4] = 0;

break;

case 8:

cell\_grp[0] = 0, cell\_grp[1] = 5;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 2, asgn\_chs[1] = 3, asgn\_chs[2] = 4, asgn\_chs[3] = 0, asgn\_chs[4] = 1;

break;

// Group 2 contains cell 2,5,8 and they are sharing the same channels 6,7,8,9,10 in a different sequence

case 1:

cell\_grp[0] = 4, cell\_grp[1] = 7;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 5, asgn\_chs[1] = 6, asgn\_chs[2] = 7, asgn\_chs[3] = 8, asgn\_chs[4] = 9;

break;

case 4:

cell\_grp[0] = 1, cell\_grp[1] = 7;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 6, asgn\_chs[1] = 7, asgn\_chs[2] = 8, asgn\_chs[3] = 9, asgn\_chs[4] = 5;

break;

case 7:

cell\_grp[0] = 1, cell\_grp[1] = 4;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 7, asgn\_chs[1] = 8, asgn\_chs[2] = 9, asgn\_chs[3] = 5, asgn\_chs[4] = 6;

break;

// Group 3 contains cell 3,4,7 and they are sharing the same channels 11,12,13,14,15 in a different sequence

case 2:

cell\_grp[0] = 3, cell\_grp[1] = 6;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 10, asgn\_chs[1] = 11, asgn\_chs[2] = 12, asgn\_chs[3] = 13, asgn\_chs[4] = 14;

break;

case 3:

cell\_grp[0] = 2, cell\_grp[1] = 6;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 11, asgn\_chs[1] = 12, asgn\_chs[2] = 13, asgn\_chs[3] = 14, asgn\_chs[4] = 11;

break;

case 6:

cell\_grp[0] = 2, cell\_grp[1] = 3;

asgn\_ch\_no = 5;

asgn\_chs = new int[5];

asgn\_chs[0] = 12, asgn\_chs[1] = 13, asgn\_chs[2] = 14, asgn\_chs[3] = 10, asgn\_chs[4] = 11;

break;

}

for(int j=0;j<asgn\_ch\_no;j++)

{

int i = asgn\_chs[j];

track\_calls store\_temp;

store\_temp.selected\_ch = i;

if (dptr::channel\_cells[i]->chck\_ch\_cluster(cluster) == true)

{

store\_temp.current\_val = true;

y.check\_list.add\_data(store\_temp);

continue;

}

if(dptr::channel\_cells[i]->current\_val == false)

{

y.stored\_SIR = 35; // shows that channel is free

y.check\_list.del();

return i;

}

// Logic to find the co channels interference between the same channels which are used different clusters

for(int k=0; k<2; k++)

{

if (dptr::channel\_cells[i]->cl\_list[cell\_grp[k]] == 1)

{

store\_temp.interferer.add\_data(int(cell\_grp[k]));

}

}

double SIR, stored\_SIR, dnr=0;

double nmr = pow(fixed\_data::cell\_r, -fixed\_data::pl\_exp);

for (int m=0; m<store\_temp.interferer.l\_len(); m++)

{

int number\_cell = store\_temp.interferer.find\_val\_at\_index(m);

int first\_cell = y.cell;

double path = fixed\_data::path\_mat[first\_cell][number\_cell];

dnr += pow(path, -fixed\_data::pl\_exp);

}

SIR = nmr/dnr; // Logic to calculate SIR

stored\_SIR = 10\*(log10(SIR));

store\_temp.stored\_SIR = stored\_SIR;

if (stored\_SIR >= 22) // If this condition satisfies, this channel is alloted

{

y.check\_list.del();

y.check\_list.add\_data(store\_temp);

y.stored\_SIR = stored\_SIR;

return i;

}

y.check\_list.add\_data(store\_temp); // If value of SIR is low then insert data to the list

}

delete[] asgn\_chs;

return -1;

}

void dptr::ip\_fp(char \*path)

{

// Logic to open and close the file

ifstream fp;

call\_stored newcall\_stored;

fp.open(path);

if(!fp)

{

cerr<<" File not present..! ";

exit(1);

}

char str[80];

fp.getline(str,79);

while(!fp.eof())

{

fp>>newcall\_stored.number>>newcall\_stored.time\_req>>newcall\_stored.cell>>newcall\_stored.duration;

newcall\_stored.first\_time = newcall\_stored.time\_req;

newcall\_stored.cell--;

newcall\_stored.decision = first\_call;

all\_calls.add\_call\_stored(newcall\_stored);

}

fp.close();

}

// Declaration of the main function

int main()

{

double SIR\_avg = 0.0, GOS = 0.0;

fixed\_data::set\_vals();

dptr::starts();

//input file (low and high)

dptr::ip\_fp("C:/Users/Anonymous/Desktop/Project\_2/input-highf16.txt");

//dptr::ip\_fp("C:/Users/Anonymous/Desktop/Project\_2/input-lowf16.txt");

dptr::operate\_stored\_call();

// logic to calculate GOS and average SIR

GOS = ((double)dptr::t\_rj\_calls/(double)dptr::t\_calls)\*100.0;

SIR\_avg = (dptr::t\_ac\_SIR/(double)(dptr::t\_ac\_calls));

// Display the final result

cout<<"-------------------------------------";

cout<<endl<<"| Final Result |";

cout<<endl<<"-------------------------------------";

cout<<endl<<" Total Incoming calls : "<<dptr::t\_calls<<endl<<" Total Accepted calls : "<<dptr::t\_ac\_calls<<endl;

cout<<" Total Rejected calls : "<<dptr::t\_rj\_calls<<endl<<" GOS : "<<GOS<<" % ";

cout<<endl<<" Average SIR : "<<SIR\_avg<<" dB ";

cout<<endl<<"-------------------------------------"<<endl;

return 0;

}

// End of the program