**TRAIN DOCK SIMULATION**

**SOURCE CODE:**

#include<iostream>

#include<algorithm>

#include<deque>

#include<map>

#include<cmath>

#include<cstdlib>

#include<queue>

#include<iomanip>

#include <fstream>

using namespace std;

float Sim\_time=0.0;

float poissontime=0.0;

int trainNum = 0;

int crewNum = 0;

float T = 0.0;

float MAX=99999.0;

int docknum=0;

int MAXHOG = 10;

fstream travelt;

bool filein = false;

class crew

{

public:

float crewremaintime;

int Number;

void setcrew(float crewremainT)

{

crewremaintime = crewremainT;

}

void setcrewnum(int num)

{

Number = num;

}

int getcrewnum()

{

return Number;

}

float calremaintime()

{

return (6.0 + ((float)rand() / (RAND\_MAX + 1)) \* 5);

}

float calreplacetime()

{

if (filein == false)

{

return (2.5 + (float)rand() / (RAND\_MAX + 1));

}

else

{

float temp=0.0;

travelt >> temp;

return temp;

}

}

void printcrewtime()

{

if (!crewremaintime == 0)

{

cout << " crew " << Number << " with" << crewremaintime << "h before hogout" << endl;

}

else

{

cout << " Cannot enter dock(SERVER HOGGED)" << endl;

}

}

};

class train

{

private:

float arrivaltime;

float entertime;

float unloadtime;

float departuretime;

float hoggedtime;

//permit to hog or replace at most 100 times in a train

float hoggedbox[10];

float replacetime;

float replacebox[10];

int Number;

bool hoggedmanytimes = false;

public:

crew c;

void settrain(int Num, float arrivalT, float unloadT,crew newc)

{

Number = Num;

arrivaltime = arrivalT;

unloadtime = unloadT;

c = newc;

hoggedtime = MAX;

replacetime = MAX;

departuretime = MAX;

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

{

hoggedbox[i] = MAX;

replacebox[i] = MAX;

}

}

int gettrainnum()

{

return Number;

}

//about arrival

float calarrival()

{

return -logf(1.0f - (float)rand() / (RAND\_MAX + 1)) \* poissontime;//poisson process

}

float getarrivaltime()

{

return arrivaltime;

}

void setarrivaltime(float t)

{

arrivaltime = t;

}

void printarrival()

{

cout << fixed << setprecision(2) << "Time " << arrivaltime << ":train " << Number << " arrival for " << unloadtime << "h of unloading" << endl;

}

//about enter dock

float calunload()

{

return (3.5 + (float)rand() / (RAND\_MAX + 1));

}

void setentertime(float time)

{

entertime = time;

}

void printenterdock()

{

cout << fixed << setprecision(2) << "Time " << entertime << ":train " << Number << " entering dock for " << unloadtime << "h of unloading" << endl;

}

//departure

void caldeparture(float lastdeparture)

{

float temp = lastdeparture - arrivaltime;

if (c.crewremaintime-temp >= unloadtime)

{

departuretime = lastdeparture + unloadtime;

if (temp >= 0)

{

c.crewremaintime = c.crewremaintime - temp;

}

}

else

{

if (c.crewremaintime-temp >= 0)

{

unloadtime = unloadtime - (c.crewremaintime-temp);

hoggedtime = arrivaltime + c.crewremaintime;

replacetime = hoggedtime + c.calreplacetime();

departuretime = replacetime + unloadtime;

c.crewremaintime = hoggedtime - lastdeparture;

}

else

{

hoggedtime = arrivaltime + c.crewremaintime;

replacetime = hoggedtime + c.calreplacetime();

if (hoggedtime<lastdeparture&&replacetime>lastdeparture)

{

c.crewremaintime = 0;

departuretime = replacetime + unloadtime;

}

else if (replacetime<lastdeparture)

{

if (lastdeparture - hoggedtime + unloadtime <= 12.0)

{

departuretime = lastdeparture + unloadtime;

c.crewremaintime = 12.0 - (lastdeparture-hoggedtime);

}

else

{//MANY hogged

int i = 1;

hoggedmanytimes = true;

hoggedbox[0] = hoggedtime + 12.0;

replacebox[0] = hoggedbox[0] + c.calreplacetime();

while (lastdeparture - hoggedbox[i - 1] + unloadtime > 12.0)

{

//AGAIN hogged

hoggedbox[i] = hoggedbox[i-1] + 12.0;

replacebox[i] = hoggedbox[i] + c.calreplacetime();

i++;

}

if (replacebox[i - 1] < lastdeparture)

{

departuretime = lastdeparture + unloadtime;

c.crewremaintime = 12.0-(lastdeparture-hoggedbox[i-1]);

}

else if (replacebox[i - 1] > lastdeparture && hoggedbox[i - 1] < lastdeparture)

{

departuretime = replacebox[i - 1] + unloadtime;

c.crewremaintime = 0;

}

else if (hoggedbox[i - 1] > lastdeparture)

{

departuretime = hoggedbox[i - 1] - lastdeparture + replacebox[i - 1];

c.crewremaintime = hoggedbox[i-1] - lastdeparture;

}

}

}

}

}

}

void printdeparture()

{

cout << fixed << setprecision(2) << "Time " << departuretime << ":train " << Number << " departing " << endl;

}

float getdeparture()

{

return departuretime;

}

//hogged

float gethogged()

{

if ( hoggedmanytimes == false)

{

return hoggedtime;

}

else

{

float min=hoggedtime;

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

{

if (min>hoggedbox[i])

{

min = hoggedbox[i];

}

}

return min;

}

}

void printhogged()

{

cout << fixed << setprecision(2) << "Time " << hoggedtime << ":train " << Number << "crew " << c.getcrewnum() << " hogged out during survice (SERVER HOGGED)" << endl;

}

void sethogged(float t)

{

if (hoggedmanytimes == false)

{

hoggedtime = t;

}

else

{

float min = hoggedtime;

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

{

if (min>hoggedbox[i])

{

min = hoggedbox[i];

}

}

if (min == hoggedtime)

{

hoggedtime = t;

}

else

{

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

{

if (min == hoggedbox[i])

{

hoggedbox[i] = t;

}

}

}

}

}

//replace

float getreplace()

{

if (hoggedmanytimes == false)

{

return replacetime;

}

else

{

float min = replacetime;

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

{

if (min>replacebox[i])

{

min = replacebox[i];

}

}

return min;

}

}

void setreplace(float t)

{

if (hoggedmanytimes == false)

{

replacetime = t;

}

else

{

float min = replacetime;

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

{

if (min>replacebox[i])

{

min = replacebox[i];

}

}

if (min == replacetime)

{

replacetime = t;

}

else

{

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

{

if (min == replacebox[i])

{

replacebox[i] = t;

}

}

}

}

}

void printreplace()

{

cout << fixed << setprecision(2) << "Time " << replacetime << ":train " << Number << " replacement crew " << c.getcrewnum() << " arrives (SERVER UNHOGGED)" << endl;

}

};

class event

{

private:

int trainnum;

int eventtype;//arrival(0) enterdock(1) hoggedout(2) replace(3) departure(4)

float takeplacetime;

public:

event(int tnum, int type, float time)

{

trainnum = tnum;

eventtype = type;

takeplacetime = time;

}

int gettrainnum()

{

return trainnum;

}

int geteventtype()

{

return eventtype;

}

float geteventtime()

{

return takeplacetime;

}

};

queue<train> arrivalqueue;

deque<train> dockqueue;

queue<event> eventqueue;

void setArrival(float poisson\_t,float simtime)

{

Sim\_time=simtime;

poissontime = poisson\_t;

float time = 0.0;

while (time< Sim\_time+10000)

{

train t;

crew c;

c.setcrew(c.calremaintime());

t.settrain(trainNum, time, t.calunload(), c);

time = time + t.calarrival();

trainNum++;

arrivalqueue.push(t);

}

}

//find min

float findmin(float nextarrival)

{

int i = dockqueue.size();

float min = nextarrival;

//find min in nextarrival,hogged out,replace,departure

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

{

if (dockqueue[j].gethogged() < min)

{

min = dockqueue[j].gethogged();

}

if (dockqueue[j].getreplace() < min)

{

min = dockqueue[j].getreplace();

}

if (dockqueue[j].getdeparture() < min)

{

min = dockqueue[j].getdeparture();

}

}

if (min == nextarrival)

{

docknum = i - 1;

return min;

}

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

{

if (dockqueue[j].gethogged() == min)

{

docknum = j;

return min;

}

if (dockqueue[j].getreplace() == min)

{

docknum = j;

return min;

}

if (dockqueue[j].getdeparture() == min)

{

docknum = j;

return min;

}

}

}

void setEvent()

{

while (1)

{

train trainnow;

if (dockqueue.empty())

{

if (arrivalqueue.empty())

{

Sim\_time = T;

break;

}

T = arrivalqueue.front().getarrivaltime();

if (T > Sim\_time)

{

break;

}

dockqueue.push\_back(arrivalqueue.front());

dockqueue.front().c.setcrewnum(crewNum++);

dockqueue.front().printarrival();

dockqueue.front().c.printcrewtime();

event e(dockqueue.front().gettrainnum(), 0, T);

eventqueue.push(e);

dockqueue.front().setentertime(arrivalqueue.front().getarrivaltime());

dockqueue.front().printenterdock();

dockqueue.front().c.printcrewtime();

event e2(dockqueue.front().gettrainnum(), 1, T);

eventqueue.push(e2);

dockqueue.front().caldeparture(dockqueue.front().getarrivaltime());

arrivalqueue.pop();

}

else if (!dockqueue.empty())

{

trainnow = dockqueue.front();

//next event

float temp = 0.0;

if (!arrivalqueue.empty())

{

temp = findmin(arrivalqueue.front().getarrivaltime());

}

else

{

temp = findmin(MAX);

}

if (temp == trainnow.getdeparture())

{

T = dockqueue.front().getdeparture();

if (T > Sim\_time)

{

break;

}

dockqueue.front().printdeparture();

event e(dockqueue.front().gettrainnum(), 4, T);

eventqueue.push(e);

dockqueue.pop\_front();

if (!dockqueue.empty())

{

dockqueue.front().setentertime(trainnow.getdeparture());

dockqueue.front().printenterdock();

dockqueue.front().c.printcrewtime();

event e3(dockqueue.front().gettrainnum(), 1, T);

eventqueue.push(e3);

}

}

else if(temp == dockqueue[docknum].gethogged())

{

T = dockqueue[docknum].gethogged();

if (T > Sim\_time)

{

break;

}

dockqueue[docknum].printhogged();

dockqueue[docknum].sethogged(MAX);

event e(dockqueue[docknum].gettrainnum(), 2, T);

eventqueue.push(e);

}

else if (temp == dockqueue[docknum].getreplace())

{

T = dockqueue[docknum].getreplace();

if (T > Sim\_time)

{

break;

}

dockqueue[docknum].c.setcrewnum(crewNum++);

dockqueue[docknum].printreplace();

dockqueue[docknum].setreplace(MAX);

event e(dockqueue[docknum].gettrainnum(), 3, T);

eventqueue.push(e);

}

else if (temp == arrivalqueue.front().getarrivaltime())

{

T = arrivalqueue.front().getarrivaltime();

if (T > Sim\_time)

{

break;

}

arrivalqueue.front().c.setcrewnum(crewNum++);

arrivalqueue.front().printarrival();

arrivalqueue.front().c.printcrewtime();

event e(arrivalqueue.front().gettrainnum(), 0, T);

eventqueue.push(e);

arrivalqueue.front().caldeparture(dockqueue[docknum].getdeparture());

dockqueue.push\_back(arrivalqueue.front());

arrivalqueue.pop();

}

}

}

cout << "Time " << Sim\_time << ": simulation ended" << endl;

}

void statistics()

{

int trainserved = 0;

//average time

float \*\*everytrain;

everytrain = new float \*[trainNum];

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

{

everytrain[i] = new float [5];

}

float insystemtime=0.0;

//maximum time

float maxtime = 0.0;

//idlebusy

event lastevent = eventqueue.front();

bool busy = false;

float busytime=0.0;

float idletime=0.0;

float temp=0.0;

//hogged out percentage

float hoggedouttime = 0.0;

//average time-in-queue

float timeinqueue = 0.0;

//maximum number of trains in queue

int \*numberinqueue;

numberinqueue = new int[trainNum];

//idle time per train

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

{

numberinqueue[i] = 0;

}

int maxnumber=0;

float idletimetrain=0.0;

//histogram

int \*hoggedtimes;

hoggedtimes = new int[trainNum];

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

{

hoggedtimes[i] = 0;

}

while (!eventqueue.empty())

{

everytrain[eventqueue.front().gettrainnum()][eventqueue.front().geteventtype()] = eventqueue.front().geteventtime();

float temp = eventqueue.front().geteventtime()-lastevent.geteventtime();

switch (eventqueue.front().geteventtype())

{

case 0://arrival

{

if (busy==false)

{

busy = true;

idletime = temp + idletime;

}

else

{

busytime = temp + busytime;

}

maxnumber++;

numberinqueue[eventqueue.front().gettrainnum()] = maxnumber;

break;

}

case 1://enterdock

{

if (busy == false)

{

busy = true;

}

timeinqueue = eventqueue.front().geteventtime() - everytrain[eventqueue.front().gettrainnum()][0] + timeinqueue;

idletimetrain = eventqueue.front().geteventtime() - everytrain[eventqueue.front().gettrainnum()][0] + idletimetrain;

break;

}

case 2://hogged out

{

busytime = temp + busytime;

hoggedtimes[eventqueue.front().gettrainnum()]++;

break;

}

case 3://replace

{

busytime = temp + busytime;

hoggedouttime = eventqueue.front().geteventtime() - everytrain[eventqueue.front().gettrainnum()][2] + hoggedouttime;

if (everytrain[eventqueue.front().gettrainnum()][1] == NULL){}

else if (everytrain[eventqueue.front().gettrainnum()][1] < everytrain[eventqueue.front().gettrainnum()][2])

{

idletimetrain = idletimetrain + eventqueue.front().geteventtime() - everytrain[eventqueue.front().gettrainnum()][2];

}

else if (everytrain[eventqueue.front().gettrainnum()][1] > everytrain[eventqueue.front().gettrainnum()][2])

{

idletimetrain = idletimetrain + eventqueue.front().geteventtime() - everytrain[eventqueue.front().gettrainnum()][1];

}

break;

}

case 4://departure

{

busytime = temp + busytime;

busy = false;

trainserved++;

maxnumber--;

break;

}

}

lastevent = eventqueue.front();

eventqueue.pop();

}

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

cout << "Total number of trains served: " << trainserved << endl;

//calculate averrage/max time

maxtime = everytrain[0][4] - everytrain[0][0];

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

{

insystemtime = (everytrain[i][4] - everytrain[i][0]) + insystemtime;

if (everytrain[i][4] - everytrain[i][0] > maxtime)

{

maxtime = everytrain[i][4] - everytrain[i][0];

}

}

cout << "Average time-in-system per train: " << insystemtime/trainserved<<"h"<<endl;

cout << "Maximum time-in-system per train: " << maxtime <<"h"<< endl;

//idlebusy

cout << "Dock idle percentage: " << (idletime / Sim\_time)\*100 << "%" << endl;

cout << "Dock busy percentage: " << (busytime / Sim\_time)\*100 << "%" << endl;

cout << "Dock hogged-out percentage: " <<(hoggedouttime/Sim\_time)\*100<<"%"<< endl;

cout << "Average time-in-queue over trains: " << timeinqueue / trainserved << "h" << endl;

//max in queue

int max = numberinqueue[0];

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

{

if (max < numberinqueue[i])

{

max = numberinqueue[i];

}

}

cout << "Maximum number of trains in queue: " << max << endl;

// cout << "Average idle time per train: " << idletimetrain / trainserved << "h" << endl;

//histogram

int \*histogram;

histogram = new int[MAXHOG];

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

{

histogram[i] = 0;

}

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

{

int temp = hoggedtimes[i];

histogram[temp]++;

}

cout << "Histogram of hogout count per train:" << endl;

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

{

if (histogram[i] != 0)

{

cout<<"["<<i<<"]: " << histogram[i] << endl;

}

}

}

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

{

srand((unsigned)time(NULL));

if (argc == 3)

{

setArrival(atoi(argv[1]), atoi(argv[2]));

setEvent();

statistics();

}

if (argc == 4)

{

filein = true;

//arrival

fstream schedule;

schedule.open(argv[2],ios::in);

float fileunload = 0.0;

float fileremain = 0.0;

float time = 0.0;

while (!schedule.eof())

{

schedule >> time;

schedule >> fileunload;

schedule >> fileremain;

train t;

crew c;

c.setcrew(fileremain);

t.settrain(trainNum, time, fileunload, c);

trainNum++;

arrivalqueue.push(t);

Sim\_time = time+1000;

}

schedule.close();

travelt.open(argv[3], ios::in);

setEvent();

statistics();

travelt.close();

}

}

**OUTPUT**

**When input “Project21.exe -s schedule.txt traveltimes.txt”**





**When input “10 72000”**



