Train Unloading Dock

Wenbo Zhao 10/17/2017

First, I want to say, I am sorry for not finishing this task as required. As I over estimate my speed of processing the Txt files input part. Basically, I finish the first input method and can run the program on Xcode perfectly. But I meet some troubles when the data is read from txt files. The time is limited and I carelessly messed up the codes when adding txt read method so I had to go back to the original version. Also, I have no time to make a Makefile.

The source codes are as follows:

main.cpp:

//

//  main.cpp

//  Train Unloading Dock

//

//  Created by Zhao Wenbo on 2017/10/10.

//  Copyright © 2017年 Zhao Wenbo. All rights reserved.

//

#include <iostream>

#include <deque>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <math.h>

#include "train.hpp"

#include "GenerateRandom.hpp"

using namespace std;

float average\_train\_arrive=0.0;

float total\_time =0.0;

deque<train> waitlist;

int countnum=0; //trains have arrived,start from 0

int train\_index[5]={-1,-1,-1,-1,-1}; // the index of the train of the following event

float current\_time = 0;

float event\_list[5] ={-1,-1,-1,-1,-1};//the following event time

bool event\_has\_passed[5]={false,false,false,false,false}; // a mark that represents if the event in the event\_list has happended or not

int event\_type=-1;

GenerateRandom gr;

int continueunload=0;//a mark that will continue unload after hogged out

int hogged\_out\_trains[10]={0,0,0,0,0,0,0,0,0,0};

/\*last required output\*/

int serverd\_trains=0;

float max\_in\_system\_time=0;

float total\_in\_system\_time=0.0;

float total\_dock\_busy\_time= 0.0;

float total\_dock\_idle\_time = 0.0;

float total\_dock\_hogged\_out\_time=0.0;

float area =0; // used to calculate average num of trains in queue;

int max\_num\_train\_in\_queue =0;

void init(){

    train t;

    t.set(countnum, 0 + gr.genTNextarrivetime(average\_train\_arrive), gr.genUnloadtime(),gr.genRemainingworktime(), gr.genCNextarrivetime());

    t.setEnter\_dock\_time(t.getArrivetime());

    t.state= NOTARRIVE;

    waitlist.push\_back(t);

    event\_list[0]=t.getArrivetime();

    event\_list[3]=t.getEnterdocktime();

    train\_index[0]=countnum;

    train\_index[3]=countnum;

}

/\*select the min hogged start time from the waitlist\*/

float select\_min\_hogged\_start\_time(){

    float temp\_time = 999999;

    train\_index[1]=-1;

    for(int i=0;i<waitlist.size();i++){

        /\*to avoid to get in a endless loop of the same value and select the min value, we use these four limitations \*/

        if(waitlist[i].getHoggedstarttime()>0 & waitlist[i].getHoggedstarttime()<=temp\_time & waitlist[i].getHoggedstarttime()>=current\_time

           & waitlist[i].state!=HOGGEDOUT){

            temp\_time=waitlist[i].getHoggedstarttime();

            train\_index[1] = waitlist[i].getIndex();

        }

    }

    return temp\_time;

}

/\*select the min hogged end time(min crew arrive time) from the waitlist\*/

float select\_min\_crew\_arrive\_time(){

    float temp\_time = 999999;

    train\_index[2]=-1;

    for(int i=0;i<waitlist.size();i++){

        if(waitlist[i].getHoggedendtime()>0 &waitlist[i].getHoggedendtime()<=temp\_time & waitlist[i].getHoggedendtime()>=current\_time

           & waitlist[i].state==HOGGEDOUT){

            temp\_time=waitlist[i].getHoggedendtime();

            train\_index[2] = waitlist[i].getIndex();

        }

    }

    return temp\_time;

}

/\*find the position of the train in the waitlist by inputing the train number(index)\*/

int findtrainbyIndex(int index){

    int a=-1;

    for(int i=0;i<waitlist.size();i++){

        if(index==waitlist[i].getIndex()){

            a=i;

        }

    }

    return a;

}

/\*decide on the next event happens time and event times under the help of event\_list\*/

void timing(){

    event\_list[1] = select\_min\_hogged\_start\_time();

    event\_list[2] = select\_min\_crew\_arrive\_time();

    float temp\_time=event\_list[0];

    event\_type = 0;

    /\*choose the next event\*/

    for(int i=1;i<5;i++){

        if(event\_list[i]>0 & event\_list[i] >= current\_time & event\_list[i] < temp\_time & event\_has\_passed[i]==false){

            temp\_time = event\_list[i];

            event\_type= i;

        }

    }

    current\_time= temp\_time;

}

/\*

 \* no matter what event happens, we must change the train state at once

 \*/

/\*when one train arrives, we need to calculate init the next train and put it in the waitlist(queue)\*/

void arrive(){

    event\_has\_passed[0]=true;

    if(findtrainbyIndex(train\_index[0])!=-1){

        waitlist.back().setHogged\_start\_time(event\_list[0]);

        waitlist.back().state= WAITING;

        printf("train %d arrived at %f.\n",train\_index[0],event\_list[0]);

    }else{

       // exit(0);

    }

    /\*generate next arrival train\*/

    float temp\_time =gr.genTNextarrivetime(average\_train\_arrive);

    countnum++;

    train t;

    t.set(countnum, temp\_time +event\_list[0], gr.genUnloadtime(),gr.genRemainingworktime(), gr.genCNextarrivetime());

    t.state= NOTARRIVE;

    waitlist.push\_back(t);

    event\_list[0]=t.getArrivetime();

    event\_has\_passed[0]=false;

    train\_index[0]=countnum;

}

/\* when one train hog out, we need to set the new crew arrive time and if the train was unloading before hog out , we need to set a new departure time\*/

void hog\_out(){

    if(findtrainbyIndex(train\_index[1])!=-1){

        if(waitlist[findtrainbyIndex(train\_index[1])].state==UNLOADING ){

            waitlist[findtrainbyIndex(train\_index[1])].setDepature\_time(waitlist[findtrainbyIndex(train\_index[1])].getDeparturetime()+waitlist[findtrainbyIndex(train\_index[1])].getCrewarrivetime());

            event\_list[4]=waitlist[findtrainbyIndex(train\_index[1])].getDeparturetime();

            event\_has\_passed[4]=false;

            train\_index[4]=train\_index[1];

            continueunload=1; //a mark that will continue unload after hogged out

            total\_dock\_hogged\_out\_time = total\_dock\_hogged\_out\_time + waitlist[findtrainbyIndex(train\_index[1])].getCrewarrivetime();

        }

        waitlist[findtrainbyIndex(train\_index[1])].setHogged\_end\_time();

        waitlist[findtrainbyIndex(train\_index[1])].state = HOGGEDOUT;

        printf("train %d hogged out at %f.\n",train\_index[1],event\_list[1]);

        waitlist[findtrainbyIndex(train\_index[1])].Hogged\_num\_add();

    }else{

       // exit(0);

    }

}

/\*when a new crew arrive, we need to set up the next hog out time of this train\*/

void crew\_arrive(){

    waitlist[findtrainbyIndex(train\_index[2])].setCrew(12,gr.genCNextarrivetime());

    waitlist[findtrainbyIndex(train\_index[2])].setHogged\_start\_time(event\_list[2]);

    if(continueunload==1 & train\_index[2]==train\_index[3]){

        waitlist[findtrainbyIndex(train\_index[2])].state=UNLOADING;

        continueunload=0;

    }else{

         waitlist[findtrainbyIndex(train\_index[2])].state=WAITING;

    }

    printf("train %d new crews arrive at %f.\n",train\_index[2],event\_list[2]);

}

/\* when a train enters dock, we will calculate and set the departure time first, but if the train hog out when unloading, the departure time will calculate again\*/

void enter\_dock(){

    event\_has\_passed[3]=true;

    waitlist[findtrainbyIndex(train\_index[3])].setDepature\_time(event\_list[3]+waitlist[findtrainbyIndex(train\_index[3])].getUnloadlast());

    event\_has\_passed[4]=false;

    train\_index[4]=train\_index[3];

    waitlist[findtrainbyIndex(train\_index[3])].state=UNLOADING;

    event\_list[4]=waitlist[findtrainbyIndex(train\_index[3])].getDeparturetime();

    printf("train %d enter dock at %f.\n",train\_index[3],event\_list[3]);

    total\_dock\_busy\_time =total\_dock\_busy\_time+waitlist[findtrainbyIndex(train\_index[3])].getUnloadlast();

}

/\* when a train depart, we will first pop the train and then set the next train's enter\_dock time according to the state\*/

void depart(){

    event\_has\_passed[4]=true;

    switch (waitlist[findtrainbyIndex(train\_index[4])].getHoggedoutnum()) {

        case 0:

            hogged\_out\_trains[0]++;

            break;

        case 1:

            hogged\_out\_trains[1]++;

            break;

        case 2:

            hogged\_out\_trains[2]++;

            break;

        case 3:

            hogged\_out\_trains[3]++;

            break;

        case 4:

            hogged\_out\_trains[4]++;

            break;

        case 5:

            hogged\_out\_trains[5]++;

            break;

        case 6:

            hogged\_out\_trains[6]++;

            break;

        case 7:

            hogged\_out\_trains[7]++;

            break;

        case 8:

            hogged\_out\_trains[7]++;

            break;

        case 9:

            hogged\_out\_trains[7]++;

            break;

        default:

            break;

    }

    waitlist.pop\_front();

    if(waitlist.front().state==HOGGEDOUT){

        waitlist.front().setEnter\_dock\_time(waitlist.front().getHoggedendtime());

        total\_dock\_idle\_time = total\_dock\_idle\_time+waitlist.front().getHoggedendtime()- current\_time;

    }else if(waitlist.front().state==NOTARRIVE){

        waitlist.front().setEnter\_dock\_time(waitlist.front().getArrivetime());

        total\_dock\_idle\_time = total\_dock\_idle\_time+waitlist.front().getArrivetime()- current\_time;

    }else{

          waitlist.front().setEnter\_dock\_time(event\_list[4]);

    }

    event\_has\_passed[3]=false;

    event\_list[3]=waitlist.front().getEnterdocktime();

    train\_index[3]=waitlist.front().getIndex();

    printf("train %d depart at %f.\n",train\_index[4],event\_list[4]);

    serverd\_trains++;

    float temp\_time=waitlist[findtrainbyIndex(train\_index[4])].getDeparturetime()-waitlist[findtrainbyIndex(train\_index[4])].getArrivetime();

    total\_in\_system\_time = total\_in\_system\_time + temp\_time;

    if(temp\_time>=max\_in\_system\_time){

        max\_in\_system\_time=temp\_time;

    }

}

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

    average\_train\_arrive = atoi(argv[1]);

    total\_time = atoi(argv[2]);

    srand((unsigned)time(NULL));

    init();

    while(waitlist[0].getArrivetime()<=total\_time){

        float last\_current\_time= current\_time;

        timing();

        area= area+ (current\_time - last\_current\_time)\*waitlist.size();

        switch(event\_type){

            case 0:

                arrive();

                break;

            case 1:

                hog\_out();

                break;

            case 2:

                crew\_arrive();

                break;

            case 3:

                enter\_dock();

                break;

            case 4:

                depart();

                break;

        }

        if(waitlist.size()> max\_num\_train\_in\_queue){

            max\_num\_train\_in\_queue= (int)waitlist.size();

        }

    }

    printf("total number of train served:%d\n",serverd\_trains-1);

    printf("everage time-in-system over trains:%f\n",total\_in\_system\_time/serverd\_trains);

    printf("max time-in-system over trains:%f\n",max\_in\_system\_time);

    printf("dock busy percentage: %f\n",total\_dock\_busy\_time/total\_time);

    printf("dock idle percentage:%f\n",total\_dock\_idle\_time/total\_time);

    printf("dock hogged out percentage:%f\n",total\_dock\_hogged\_out\_time/total\_time);

    printf("everage number of trains in queue: %f\n",area/total\_time);

    printf("max number of trains in queue:%d\n",max\_num\_train\_in\_queue);

    for(int i=0;i<10;i++){

        printf("number of trains hodded out %d times:%d\n",i,hogged\_out\_trains[i]);

    }

    return 0;

}

train.hpp:

//

//  train.hpp

//  Train Unloading Dock

//

//  Created by Zhao Wenbo on 2017/10/10.

//  Copyright © 2017年 Zhao Wenbo. All rights reserved.

//

#ifndef train\_hpp

#define train\_hpp

#include <stdio.h>

enum State{

    NOTARRIVE, WAITING, HOGGEDOUT,UNLOADING, LEAVED

};

class train{

private:

    int Index; //the index of arriving trains

    int Hogged\_out\_num=0; // how many times has this train hogged out

    float Unload\_last;

    float Remaining\_work\_time; //as there is only two variables related to crew, so to simplify we don't create Crew.class

    float Crew\_arrive\_time;

    /\*time will use for events: arrive, hogged out, new crew arrive(hogged out end), enter dock, departure\*/

    float Arrive\_time=-1;

    float Hogged\_start\_time=-1;// equals to the arrive\_time plus remaining work time

    float Hogged\_end\_time=-1;

    float Enter\_dock\_time=-1;

    float Departure\_time=-1;

public:

    State state = NOTARRIVE;

    void set(int index, float arrive\_time,float unload\_last, float remaining\_work\_time, float crew\_arrive\_time);

    void setCrew(float remaining\_work\_time,float crew\_arrive\_time);

    void setHogged\_start\_time(float t);

    void setHogged\_end\_time();

    void setEnter\_dock\_time(float enter\_dock\_time);

    void setDepature\_time(float departure\_time);

    void Hogged\_num\_add();

    int getIndex();

    int getHoggedoutnum();

    float getUnloadlast();

    float getArrivetime();

    float getHoggedstarttime();

    float getHoggedendtime();

    float getCrewarrivetime();

    float getEnterdocktime();

    float getDeparturetime();

};

#endif /\* train\_hpp \*/

train.cpp:

//

//  train.cpp

//  Train Unloading Dock

//

//  Created by Zhao Wenbo on 2017/10/10.

//  Copyright © 2017年 Zhao Wenbo. All rights reserved.

//

#include "train.hpp"

void train::set(int index, float arrive\_time,float unload\_last, float remaining\_work\_time, float crew\_arrive\_time){

    Index=index;

    Arrive\_time= arrive\_time;

    Unload\_last= unload\_last;

    Remaining\_work\_time=remaining\_work\_time;

    Crew\_arrive\_time= crew\_arrive\_time;

}

void train::setCrew(float remaining\_work\_time,float crew\_arrive\_time){

    Remaining\_work\_time=remaining\_work\_time;

    Crew\_arrive\_time=crew\_arrive\_time;

}

void train::setHogged\_start\_time(float t){

    Hogged\_start\_time = t + Remaining\_work\_time;

}

void train::setHogged\_end\_time(){

    Hogged\_end\_time= Hogged\_start\_time + Crew\_arrive\_time;

}

void train::setEnter\_dock\_time(float enter\_dock\_time){

    Enter\_dock\_time=enter\_dock\_time;

}

void train::setDepature\_time(float departure\_time){

    Departure\_time=departure\_time;

}

void train::Hogged\_num\_add(){

    Hogged\_out\_num ++;

}

int train::getIndex(){

    return Index;

}

int train::getHoggedoutnum(){

     return Hogged\_out\_num;

}

float train::getArrivetime(){

     return Arrive\_time;

}

float train::getHoggedstarttime(){

     return Hogged\_start\_time;

}

float train::getHoggedendtime(){

    return Hogged\_end\_time;

}

float train::getCrewarrivetime(){

     return Crew\_arrive\_time;

}

float train::getEnterdocktime(){

     return Enter\_dock\_time;

}

float train::getDeparturetime(){

     return Departure\_time;

}

float train::getUnloadlast(){

    return Unload\_last;

}

GenerateRandom.hpp:

//

//  GenerateRandom.hpp

//  Train Unloading Dock

//

//  Created by Zhao Wenbo on 2017/10/10.

//  Copyright © 2017年 Zhao Wenbo. All rights reserved.

//

#ifndef GenerateRandom\_hpp

#define GenerateRandom\_hpp

#include <stdio.h>

class GenerateRandom{

public:

    //correspond to poisson process

    float genTNextarrivetime(float time);

    //random uniformly

    float genfloat(float min, float max);

    float genRemainingworktime();

    float genCNextarrivetime();

    float genUnloadtime();

};

#endif /\* GenerateRandom\_hpp \*/

GenerateRandom.cpp:

//

//  GenerateRandom.cpp

//  Train Unloading Dock

//

//  Created by Zhao Wenbo on 2017/10/10.

//  Copyright © 2017年 Zhao Wenbo. All rights reserved.

//

#include "GenerateRandom.hpp"

#include <time.h>

#include <stdlib.h>

#include <math.h>

float GenerateRandom:: genTNextarrivetime(float time){

    float temp=rand() / float(RAND\_MAX);

    //printf ("temp:%f",temp);

    float next= -log(temp)\*time;

    //printf("next:%f\n",next);

    return next;

}

float GenerateRandom::genfloat(float min, float max){

    float a = rand() / float(RAND\_MAX);

    min++;

    float b=(float)((rand() % (int)(max- min + 1))+ min);

    b=b-1;

    return a+b;

}

float GenerateRandom::genRemainingworktime(){

    return genfloat(6.0, 11.0);

}

float GenerateRandom::genCNextarrivetime(){

    return genfloat(2.5, 3.5);

}

float GenerateRandom::genUnloadtime(){

    return genfloat(3.5, 4.5);

}

Running result: