**פרויקט גמר במערכות הפעלה**

**שאלה 1:**

בשאלה זו מימשנו מחלקה של גרף לא מכוון וללא משקלים על הצלעות. מימשנו את הגרף בעזרת מטריצה דו ממדית שמיוצגת על יד וקטור של וקטורים.

קובץ hpp:

#ifndef GRAPH\_HPP

#define GRAPH\_HPP

#include <iostream>

#include <vector>

class Graph{

private:

    int numOfVertex;

    std::vector<std::vector<bool>> neighborsMatrix;

public:

    Graph(int numOfVertex);

    Graph(int numOfVertex ,const std::vector<std::vector<bool>> neighborsMatrix);

    void addEdge(int src,int dest);

    void printGraph();

};

#endif

המימוש של קובץ ה-cpp נמצא בקוד.

**שאלה 2:**

בשאלה זו הוספנו למחלקה של השאלה הקודמת פונקציה נוספת שמטרתה למצוא מעגל אוילר בגרף.

על מנת למצוא מעגל אוילר הוספנו תחילה פונקציה שתדמה הרצת BFS ותבדוק אם הגרף קשיר או לא.

/\*\*

 \* @brief Checks if the graph is connected.

 \* @return true if the graph is connected, false otherwise.

 \* This function uses a fake breadth-first search (BFS) algorithm to traverse the graph and check

 \* we build this function beacuse graph must be connected to have Euler cycle.

 \*/

bool Graph::isConnected(){

    vector<int> distance;

    for(int i=0;i<numOfVertex;i++){

        distance.push\_back(1);

    }

    queue<int> bfs;

    bfs.push(0);

    while(!bfs.empty()){

        int v=bfs.front();

        bfs.pop();

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

            if(neighborsMatrix[v][i] && distance[i] == 1){

                distance[i] =0;

                bfs.push(i);

            }

        }

    }

    for(int i=0;i<numOfVertex;i++){

        if(distance[i]==1){

            return false;

        }

    }

    return true;

}

לאחר שסיימנו בדיקת קשירות חזרנו לפונקציה עצמה. וידאנו שכל הדרגות זוגיות ואז התחלנו לחפש את המסלול.

vector<int> Graph::findEuler(){

    int counter=0;

    vector<int> EulerCycle;

    if(!this->isConnected()){

        cout<<"There cant be Euler cycle in unconnected graph"<<endl;

        return EulerCycle;

    }

    for(int i=0;i<numOfVertex;i++){

        counter=0;

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

            if(neighborsMatrix[i][j]){

                counter++;

            }

        }

        if(counter%2==1){

            cout<<"This graph is not containing Euler cycle becase vtx "<<i<<" have odd degree"<<endl;

            return EulerCycle;

        }

    }

    vector<vector<bool>> copyMat = neighborsMatrix;

    stack<int> s;

    s.push(0);

    while(!s.empty()){

        int v= s.top();

        bool foundEdge = false;

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

            if(copyMat[v][i]){

                foundEdge=true;

                copyMat[v][i]=false;

                copyMat[i][v]=false;

                s.push(i);

                break;

            }

        }

        if(!foundEdge){

            EulerCycle.push\_back(v);

            s.pop();

        }

    }

    return EulerCycle;

}

**שאלה 3:**

בשאלה זו רצינו להוסיף אופציה של יצירת גרף רנדומלי ולאחר מכן למצוא עליו מעגל אוילר בעזרת הפונקציה מהסעיף הקודם.

בעבור זה קיבלנו מהמשתמש 3 ארגומנטים עם דגלים מתאימים שמסמנים לנו: מספר צלעות, מספר קודקודים ו- random seed . לאחר מכן בדקנו את תקינות הקלט, הדפסנו את הגרף ולבסוף הרצנו את הפונקציה מהסעיף הקודם.

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

    if( argc != 4){

    }

    int opt;

    int numOfVertex=-1;

    int numOfEdges=-1;

    int RandomSeed = -1;

    while ((opt = getopt(argc, argv, "v:e:r:")) != -1){

        if(opt == 'v'){

            numOfVertex=atoi(optarg);

        }

        else if(opt == 'e'){

            numOfEdges = atoi(optarg);;

        }else if(opt == 'r'){

            RandomSeed = atoi(optarg);;

        }

    }

    if(numOfEdges==-1 || numOfVertex==-1 || RandomSeed==-1){

        std::cerr<<"Usage: ./"<<argv[0]<<" -v numOfVertex (int) -e numOfEdge(int) -r Randomseed(int)"<<std::endl;

        exit(1);

    }

    unsigned long maxEdge = (numOfVertex -1)\*numOfVertex/2;

    if((unsigned long)numOfEdges>maxEdge){

        std::cerr<<"E"<<numOfEdges<<" maxEdge "<<maxEdge<<std::endl;

        std::cerr<<"ypu enter too many edges"<<std::endl;

        exit(1);

    }

    int edgesAdded=0;

    Graph g(numOfVertex);

    srand(RandomSeed);

    while(edgesAdded<numOfEdges){

        int src=rand()%numOfVertex;

        int dest =rand()%numOfVertex;

        try{

            g.addEdge(src,dest);

            edgesAdded++;

        }catch(const std::exception& e){

        }

    }

    g.printGraph();

    std::vector<int> resualt = g.findEuler();

    if(resualt.size()!=0){

        std::cout<<"The resualt is:"<<std::endl;

        for(size\_t i=0;i<resualt.size();i++){

            if(i!=resualt.size()-1){

                std::cout<<resualt[i]<<" -> ";

            }

            else{

                std::cout<<resualt[i]<<std::endl;

            }

        }

    }

}

**שאלה 4:**

בשאלה זו הרצנו על הקוד משאלה 3 code coverage, gprof, valgrind memcheck , valgrind call graph  
את התוצאות של כל אחד שמרנו בתיקייה מתאימה ובקבצים המתאימים.

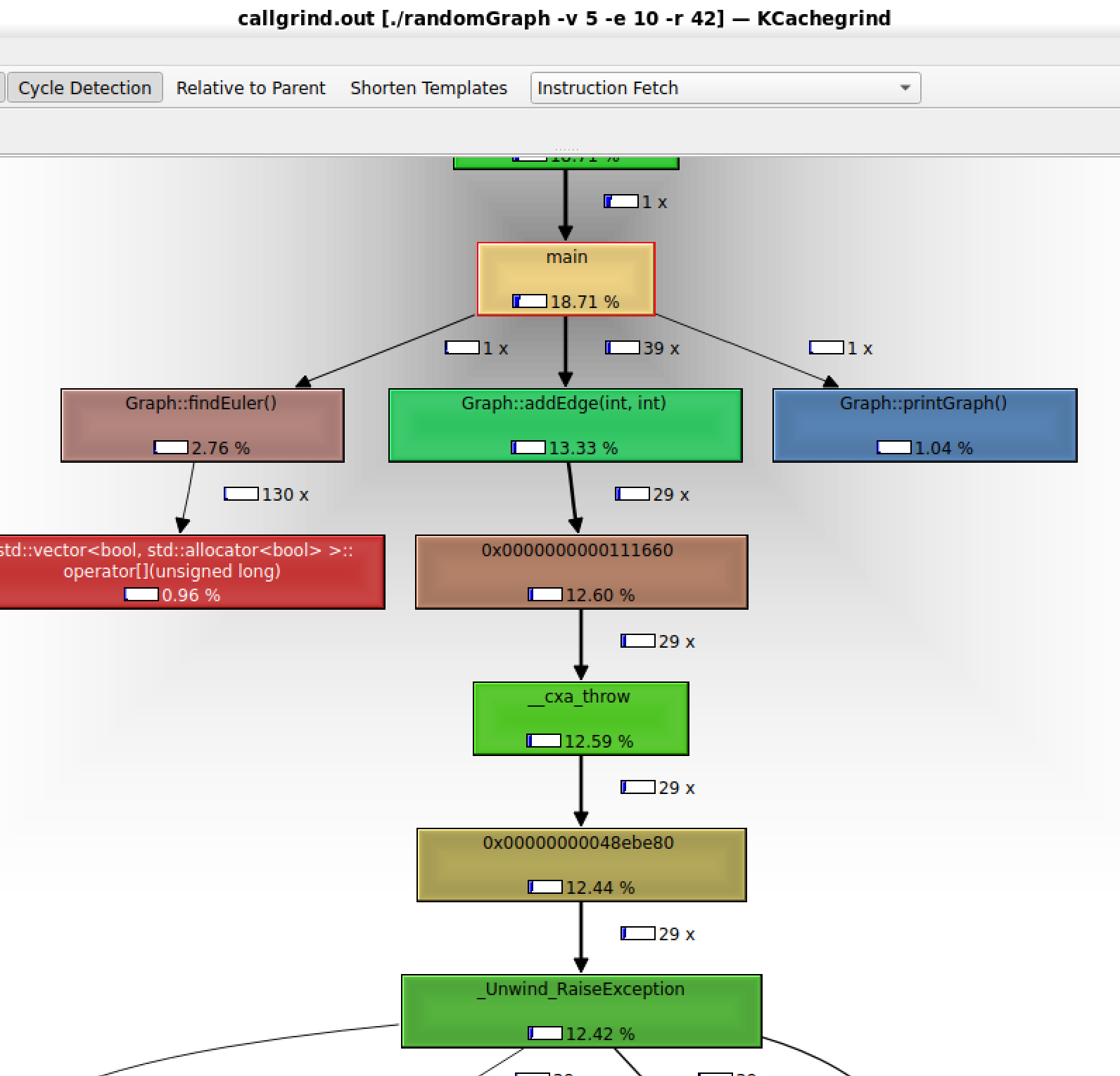
בנוגע ל-coverage את הקובץ של randomgraph . כיסינו במלואו את הקובץ של graph.cpp לא כיסינו בו הרבה שגיאות מכיוון שאנחנו מוודאים תקינות ב-randomgraph ושולחים שם רק בקשות תקינות.

בעזרת ה-profiling ניתן לראות בשורה 265 כי "בילינו" 96.2% מהזמן בפונקציית findEuler. בשורה 400 ניתן לראות כי 1.5% מהזמן היינו בבנאי של המחלקה graph ובשורה 473 ניתן לראות כי "בילינו" 1.1% מהזמן בפונקצייה addEdge על מנת לבנות את הגרף האקראי.

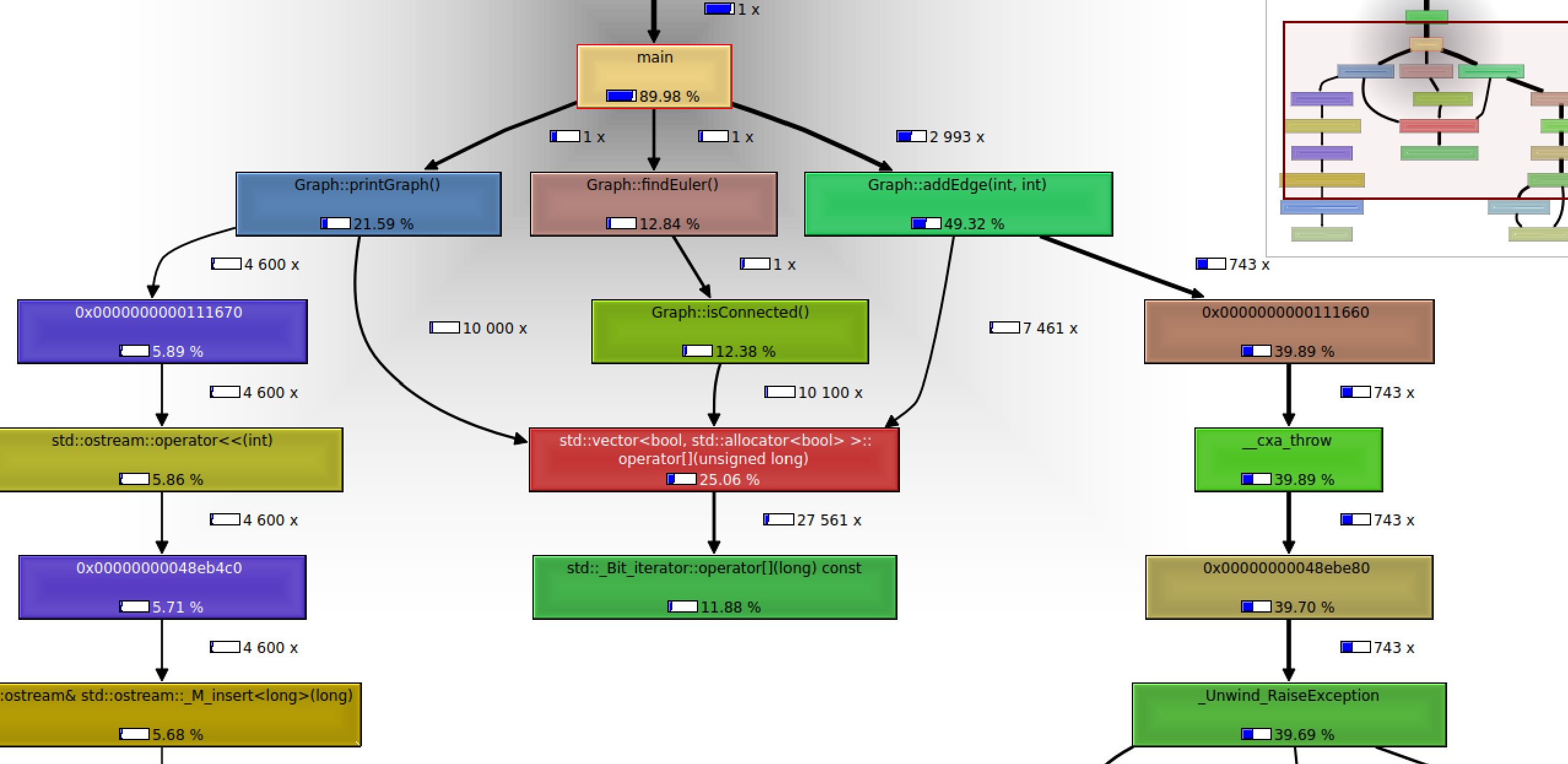
בנוגע ל-valgrind memcheck , ניתן לראות כי בכל 3 הקבצים המצורפים כתוב   
in use at exit: 0 bytes in 0 blocks , All heap blocks were freed -- no leaks are possible , ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0).  
ומשלוש השורות האלה אנחנו יכולים לראות כי אין בעיות בנוגע לזיכרון.

בנוגע ל-valgrind call graph הוספנו בתיקייה 2 תמונות שניתן לראות דרכן את הזמן שהמחשב העביר בכל פונקציה וגם בתוך כל פונקציה איך זה יתחלק. התמונות מצורפות גם פה:

הרצה מספר 1:



הרצה מספר 2:



**שאלה 6:**

בשלב זה הוספנו שרת שמקבל מהלקוח בקשות לבניית גרף מסוים והרצת מעגל אוילר עליו.

תהליך קליטת הגרף בלקוח והשליחה לשרת:

for(;;){

        cout<<"enter number of vertices if you want to exit enter 0"<<endl;

        cin>>n;

        if(n==0){

            close(sock);

            cout << "Connection closed." << endl;

            break;

        }

        if(n<0){

            cout<<"n must be positive number"<<endl;

            continue;

        }

        int arr[n\*n+1];//n\*n for the neighbor matrix and extra place for the size

        arr[0]=n;

        for(int i=1;i<n\*n+1;i++){

            arr[i]=0;

        }

        cout<<"now you will need to enter the neighbor matrix if you want to stop enter 0 for both src and dest"<<endl;

        for(;;){

            cout<<"enter src dest"<<endl;

            cin>>src>>dest;

            if(dest==src && dest==0){

                break;

            }

            if(dest<0 || dest>=n || src<0 || src>=n || src==dest){

                cout<<"illegal arguments src,dest should different numbers be between 0 to n-1"<<endl;

                continue;

            }

            arr[dest\*n+src+1]=1;//dest\*n + src to change from matrix to arr +1 is because of the size

            arr[src\*n+dest+1]=1;//src\*n + dest to change from matrix to arr +1 is because of the size

        }

        send(sock, &arr,(n\*n + 1)\*sizeof(int), 0);

    }

תהליך הקבלה בשרת:

        int byteRecv = recv(new\_socket,buffer,BUFFERSIZE,0);

        if(byteRecv==0){

            cout << "Client disconnected." << endl;

            close(new\_socket);

            break;

        }

        int n=buffer[0];

        vector<vector<bool>> neighborsMatrix(n, vector<bool>(n, false));

        for(int i=0;i<n;i++){

            for(int j=0;j<n;j++){

                if(buffer[i\*n+j+1]==1){

                    neighborsMatrix[i][j]=true;

                }

            }

        }

        Graph g(n,neighborsMatrix);

        std::vector<int> resualt = g.findEuler();

        if(resualt.size()!=0){

            std::cout<<"The resualt is:"<<std::endl;

            for(size\_t i=0;i<resualt.size();i++){

                if(i!=resualt.size()-1){

                    std::cout<<resualt[i]<<" -> ";

                }

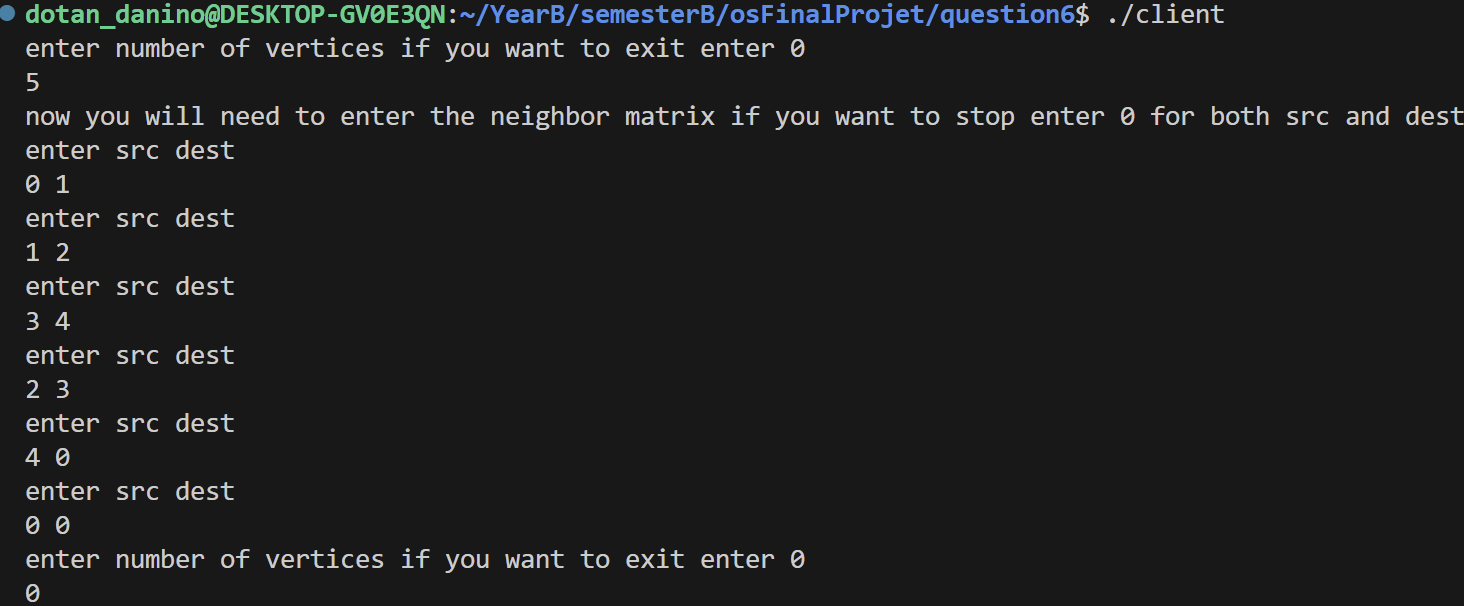
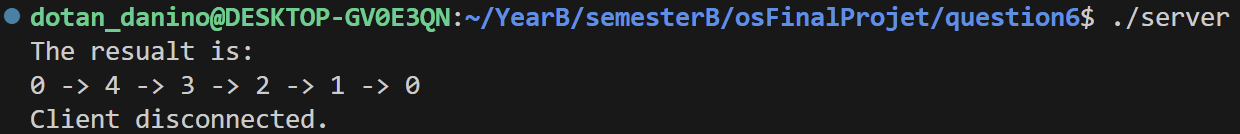
                else{

                    std::cout<<resualt[i]<<std::endl;

                }

            }

דוגמת הרצה:



**שלב 7:**

בשלב זה הוספנו לשרת אופציה להריץ 4 אלגוריתמים חדשים שהם:

1. מציאת משקל של עץ פורש מינימלי
2. מציאת רכיבי קשירות חזקים
3. מציאת מסלולים זרים בקודקודים
4. מציאת זרימה מקסימלית בין הקודקוד הראשון לאחרון

על מנת לממש את האלגוריתמים שינינו את מימוש הגרף כך שיהיה עם משקלים/ קיבולות לצלעות ובנוסף הוא יהיה מכוון.

בנוסף מימשנו את כל האלגוריתמים בעזרת שימוש בתבניות העיצוב strategy, factory. בשביל להשתמש בתבניות העיצוב יצרנו מחלקה כללית של graph algorithm ולכל אלגוריתם יצרנו מחלקה בפני עצמו שתירש מהמחלקה הכללית על מנת לממש את strategy בנוסף יצרנו קובץ חדש של algofactory שבתוכו יש פונקציה שמחזירה לנו uniquePtr לאלגוריתם שבחרנו.

std::unique\_ptr<GraphAlgorithm> createAlgorithm(AlgorithmType algoType) {

    switch(algoType) {

        case AlgorithmType::MaxFlow:

            return std::make\_unique<maxFlowAlgo>();

        case AlgorithmType::MST:

            return std::make\_unique<MSTAlgo>();

        case AlgorithmType::PathCover:

            return std::make\_unique<PathCoverAlgo>();

        case AlgorithmType::SCC:

            return std::make\_unique<SCCAlgo>();

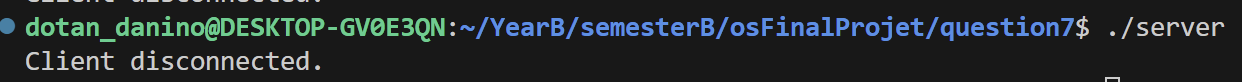
        default:

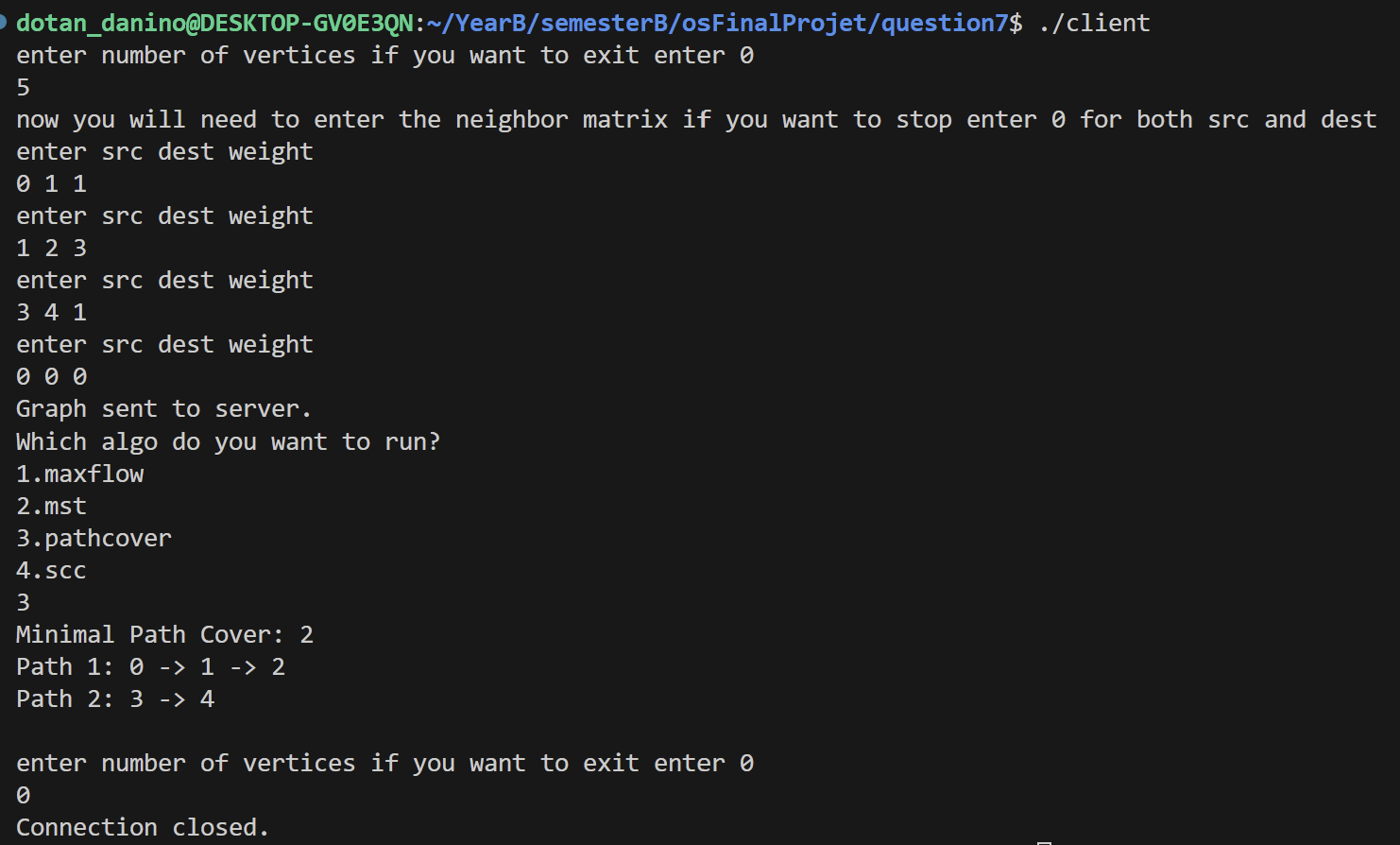
            return nullptr; // there is no such algorithm

    }

}

דוגמת הרצה:





**שאלה 8:**

בשלב זה רצינו לשנות את השרת כך שיעבוד בצורה של LF שכל פעם יש thread אחד מוביל ויש עוד threads שמחכים שהוא יסיים כדי לקחת את ההובלה.

מעבר לכך הוספנו ללקוח אופציה לבקש הגרלה של גרף רנדומלי עם מספר צלעות, קודקודים שהוא בוחר. בנוסף עליו להכניס דרגה מקסימלית ומינמלית ומספר random seed. כשהגרף מגיע לשרת הוא מפעיל עליו את 4 האלגוריתמים הפעם ומחזיר שרשור של התשובות.

יצירת גרף רנדומלי:

if(n == 0) {

            cout << "Random graph selected." << endl;

            int numOfVertex = buffer[1];

            int numOfEdges = buffer[2];

            int RandomSeed = buffer[3];

            int minWeight = buffer[4];

            int maxWeight = buffer[5];

            if (minWeight > maxWeight) {

                string e = "Error: minWeight cannot be greater than maxWeight";

                send(fd, e.c\_str(), e.size(), 0);

                continue;

            }

            cout << "Creating random graph with " << numOfVertex << " vertices, " << numOfEdges << " edges, and random seed " << RandomSeed << endl;

            Graph g = createRandomGraph(numOfVertex, numOfEdges, RandomSeed,minWeight, maxWeight);

            std::string result = algo1->execute(g);

            result += "\n" + algo2->execute(g);

            result += "\n" + algo3->execute(g);

            result += "\n" + algo4->execute(g);

            cout << "Sending results to client..." << endl;

            send(fd, result.c\_str(), result.size(), 0);

            continue;

        }

Graph createRandomGraph(int numOfVertex, int numOfEdges, int RandomSeed,int minWeight, int maxWeight) {

    unsigned long maxEdge = (numOfVertex -1)\*numOfVertex;

    if((unsigned long)numOfEdges>maxEdge){

        std::cerr<<"E"<<numOfEdges<<" maxEdge "<<maxEdge<<std::endl;

        std::cerr<<"ypu enter too many edges"<<std::endl;

        exit(1);

    }

    int edgesAdded=0;

    Graph g(numOfVertex);

    srand(RandomSeed);

    while(edgesAdded<numOfEdges){

        int src=rand()%numOfVertex;

        int dest =rand()%numOfVertex;

        int weight = rand() % (maxWeight - minWeight + 1) + minWeight;

        try{

            g.addDirectedEdge(src,dest,weight);

            edgesAdded++;

        }catch(const std::exception& e){

            //nothing to do here just wait for another graph

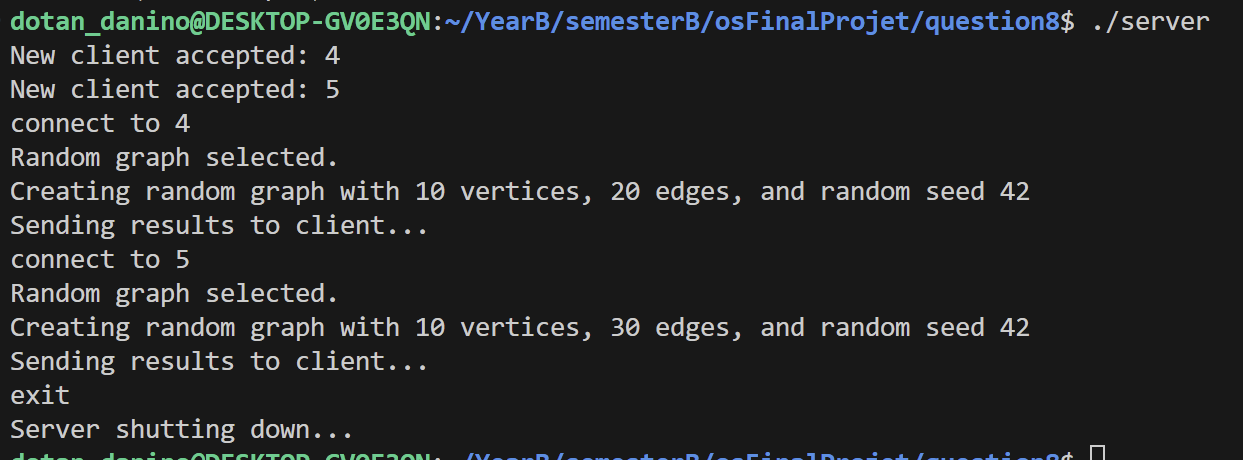
        }

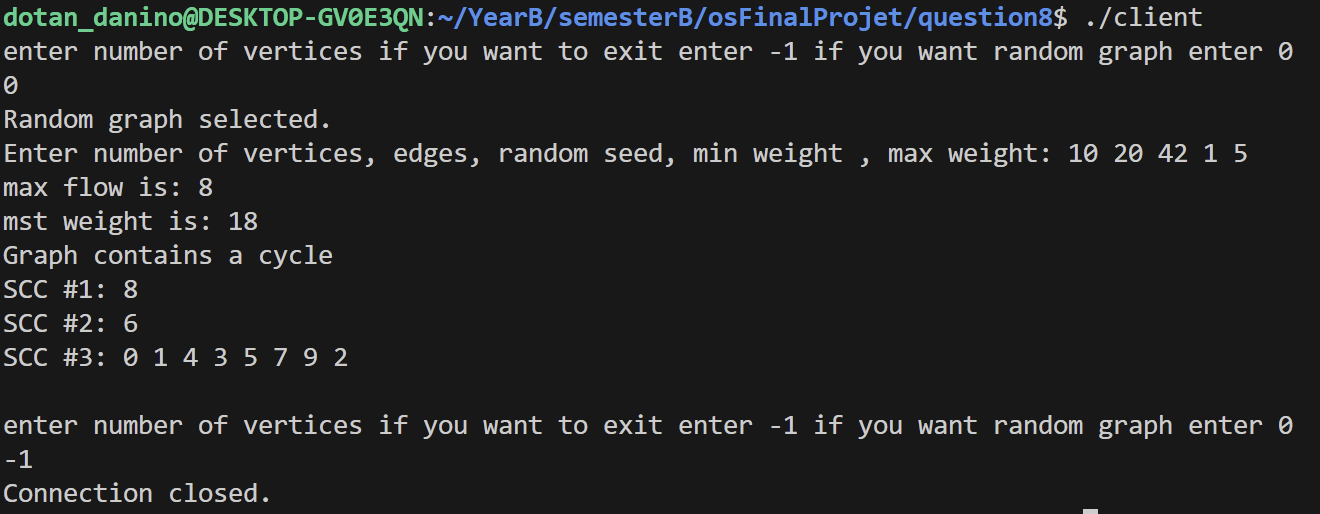
    }

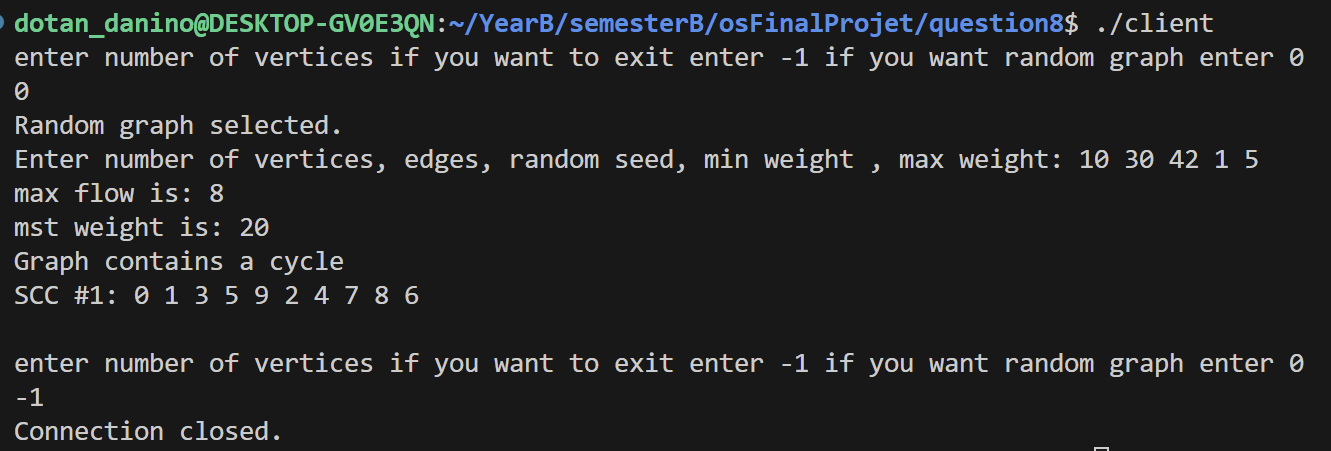
    return g;

}

דוגמת הרצה:



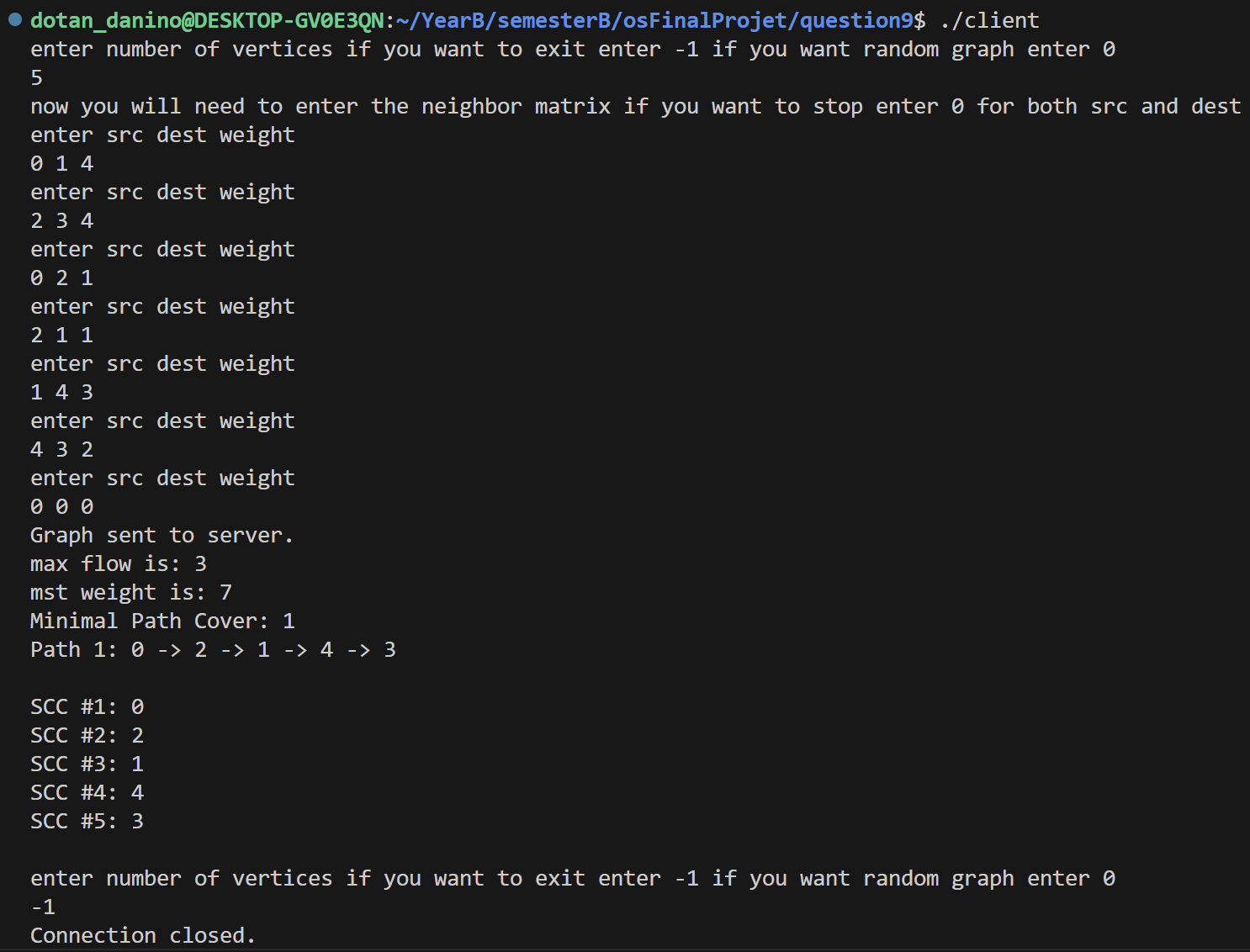


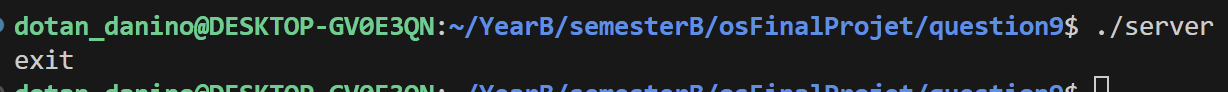


**שלב 9:**

בשלב זה המשכנו עם מנגנון של תהליכים אך הפעם במקום לפתוח תהליך חדש לכל לקוח יש לנו תהליך עבור כל אלגוריתם ולכל תהליך יש תור משלו. כמובן שכל תור מוגן במיוטקס על מנת למנוע מצב של גישה בו זמנית. ברגע שתהליך סיים עם fd הוא מעביר בעזרת struct ייעודי את התהליך ה-fd והגרף לתור של התהליך הבא. יש תהליך נוסף חמישי שאחראי על שליחת הנתונים ללקוח.

דוגמת הרצה:





**שאלה 10:**

בשלב זה הרצנו valgrind על השרת והלקוח הרצנו , memcheck, helgrind , callgraphc.

בהתייחס ל-memcheck ניתן לראות כי אין מידע עדיין נגיש/ בשימוש אין מידע שלא שחררנו ואין שגיאות.

ב-hellgrind ניתן לראות שאין לנו שגיאות.

ב- callgraph ניתן לראות את אחוזי השימוש במשאבים לפי פונקציות.

**שאלה 11:**

בלקוח כל שורות הקוד שלא כיסינו הינן שורת קוד הקשורות לכישלון ביצירת ה-socket וההתחברות דברים שלא תלויים בנו ואין ביכולתנו לגרום לקריסה בשורות אלו.

בלקוח יש לנו את השורות שמעידות על שגיאה ביצירת ה-socket ובנוסף לכך יש לנו את האופציות לשגיאה ב- condition variable שקראנו לאחד מהם בלי שבפועל הוא אמור היה לקבל התראה וגם את זה אין לנו יכולת לדמות אך זה מקרה קצה שעלינו לכסות.

ביצירת האלגוריתמים algoFactory לא כיסינו את המקרה בו התקבל פרמטר שלא מתאים לאף אחד מהאלגוריתמים מכיוון שבשרת שלנו אנחנו מוודאים את התקינות לפני כל הקריאות.

באלגוריתמים של MST,maxflow כיסינו את כל שורות הקוד ללא יוצא מן הכלל.

באלגוריתם של scc יש לנו שורה אחת שלא כיסינו והיא הסוגר של הפונקציה getTranspose אחרי ה-return.

באלגוריתם של pathCover יש שורה אחת שמסומנת כאילו לא כוסתה והיא ההצהרה על פונקציית ה-lambda

במחלקה graph יש חלק גדול מהקוד שלא מכוסה לנו.

מדובר בפונקצייה addEdge שלא בשימוש אצלנו כי אנחנו מוסיפים רק צלעות מכוונות, בפונקצייה findEuler שאנחנו לא משתמשים בה יותר בשלב זה ובפונקצייה isConnected שהינה פונקציית עזר שבשימוש בפונקצייה findEuler.