	به نام خدا	

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه‌های کامپیوتری

پروژه‌ی برنامه‌نویسی ۳

نام و نام‌خانوادگی	**نرگس غلامی - ریحانه احمدپور**
شماره دانشجویی	**۸۱۰۱۹۸۴۴۷ - ۸۱۰۱۹۸۴۹۴**
تاریخ ارسال گزارش	**۱۴۰۱/۳/۱۰ - شنبه ۱۰ خرداد**

فهرست گزارش سؤالات:

بخش ۰- توضیح پروژه

در این پروژه به پیاده سازی دو الگوریتم مسیریابی LSRP و DVRP در شبکه پرداختیم و آن‌ها را مقایسه کردیم.

بخش ۱- کلاس commandHandler و توضیحات مربوط به آن:

این کلاس، کلاسی است که ورودی‌های کاربر را مدیریت می‌نماید و وظایف را تقسیم کرده و به هر شخص می‌سپارد. نمای کلی این کلاس را در عکس زیر مشاهده می‌کنید:

```
32  class CommandHandler
33  {
34  public:
35      CommandHandler(Network* nw) : network(nw) {}
36      void getQuery(std::string query);
37
38  private:
39      std::vector<std::string> splitLine(std::string &choosenService, char splitor);
40
41      void diagnoseQuery();
42      void topology();
43      void show();
44      void lsrp();
45      void dvrp();
46      void modify();
47      void remove();
48      bool topologyIsValid();
49      bool showIsValid();
50      bool lsrpIsValid();
51      bool dvrpIsValid();
52      bool modifyIsValid();
53      bool removeIsValid();
54
55      std::vector<std::string> wordsOfLine;
56      std::vector<std::vector<int>> route;
57      std::vector<std::string> modifyInfo;
58
59      Network* network;
60  };
```

این کلاس یک constructor دارد که در آن کلاس network را ست می‌کند. این کلاس همان توپولوژی شبکه و شش کامند گفته شده را پیاده می‌کند و در صورت نیاز ورودی را بررسی می‌کند(تابع‌های isValid) در صورت درستی تابع‌های کامند را صدا می‌زند در غیر اون exception می‌فرستد که به کاربر اطلاع داده می‌شود و دوباره منتظر ورودی می‌ماند. در ادامه مفصل به توضیح کلاس network خواهیم پرداخت. یک متد getQuery هم داریم که دستورات(درخواست‌ها) را از کاربر دریافت می‌کند و آن را به diagnoseQuery تحویل می‌دهد که بررسی شود.

این کلاس بعد از تشخیص دستور ورودی و بررسی exception ها (که در بخش ۵ به توضیح آن می‌پردازیم) توابع مربوطه کلاس network را صدا می‌زند و نتورک بقیه‌ی کار را انجام می‌دهد.

به توضیح چند متد شاخص این کلاس می‌پردازیم.

متد getQuery:

```
void CommandHandler::getQuery(string query)
{
    wordsOfLine = splitLine(query, S_SPLITOR);

    if (wordsOfLine.empty())
        throw BadRequestError();

    diagnoseQuery();
}
```

در این تابع string کامد از یوزر دریافت شده و split می‌شود و در wordOfLine ریخته می‌شود. سپس تابع dignoseQuery برای شناسایی نوع درخواست با توجه به ۶ درخواست موجوده، فراخوانی می‌شود.

متد dignoseQuery:

```
131  void CommandHandler::diagnoseQuery()
132  {
133      if (wordsOfLine[QUERY_TYPE] == TOPOLOGY)
134          topologyIsValid() ? topology() : throw NotFoundError() ;
135      else if (wordsOfLine[QUERY_TYPE] == SHOW)
136          showIsValid() ? show() : throw NotFoundError() ;
137      else if (wordsOfLine[QUERY_TYPE] == LSRP)
138          lsrpIsValid() ? lsrp() : throw NotFoundError() ;
139      else if (wordsOfLine[QUERY_TYPE] == DVRP)
140          dvrpIsValid() ? dvrp() : throw NotFoundError() ;
141      else if (wordsOfLine[QUERY_TYPE] == MODIFY)
142          modifyIsValid() ? modify() : throw NotFoundError() ;
143      else if (wordsOfLine[QUERY_TYPE] == REMOVE)
144          removeIsValid() ? remove() : throw NotFoundError() ;
145      else
146          throw BadRequestError();
147  }
```

در این تابع بررسی می‌کند که دستور ورودی کاربر کدام یک از دستورات topology، show, LSRP, DVRP, modify, remove است. در صورتی که توپولوژی valid باشد دستور مربوطه صدا زده می‌شود تا کلاس network بتواند دستورات را اجرایی کند و در غیر این صورت خطا داده می‌شود.

متد topologyIsValid:

```
bool CommandHandler::topologyIsValid()
{
    vector<string> connection;
    int count = wordsOfLine.size();
    vector<vector<int>> link(CN_NODE_COUNT+1, vector<int>(CN_NODE_COUNT+1, -1));

    for (int i = 1; i < count; i++)
    {
        connection = splitLine(wordsOfLine[i], D_SPLITOR);
        if (connection[FIRST_NODE] == connection[SECOND_NODE])
            throw IDsAlike();
        int src = stoi(connection[FIRST_NODE]) -1;
        int des = stoi(connection[SECOND_NODE]) -1;
        int cost = stoi(connection[COST]);
        link[src][des] = cost;
        link[des][src] = cost;
    }
    for (int i = 0; i < CN_NODE_COUNT; i++)
        link[i][i] = 0;

    route = link;
    return true;
}
```

پس از وارد شدن دستور topology توسط کاربر و وارد کردن مبدا، مقصد و هزینه هر ارتباط، این متد صدا زده می‌شود و در یک حلقه بررسی می‌شود که در هر ارتباط اگر نود مبدا و مقصد با یکدیگر برابر بود خطای مناسب را throw نماید و منتظر دستور جدید کاربر بماند و در غیر این به ست کردن مقدار هزینه در گراف نتورک بپردازد. هزینه‌ی رفتن نود از خود به خود برابر صفر است که این نیز در نهایت ست می‌شود و چون توپولوژی valid است مقدار true بازگردانده می‌شود.

تابع lsrp و dvrp:

```
void CommandHandler::lsrp()
{
    if (wordsOfLine.size() == NO_ARG)
        for (int i = 0; i < CN_NODE_COUNT; i++)
        {
            network->runLSRP(i);
            cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
        }
    else
        network->runLSRP(stoi(wordsOfLine[NO_ARG])-1);
}

void CommandHandler::dvrp()
{
    if (wordsOfLine.size() == NO_ARG)
        for (int i = 0; i < CN_NODE_COUNT; i++)
        {
            network->runDVRP(i);
            cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
        }
    else
        network->runDVRP(stoi(wordsOfLine[NO_ARG])-1);

}
```

وقتی کلاس تشخیص می‌دهد که دستور ورودی کاربر اجرا کردن الگوریتم lsrp یا dvrp است، به متدهای مربوطه مراجعه می‌کنند. در این دو دستور که ساختار یکسانی دارند اگر آرگومانی وارد نشده باشد، به این معناست که خروجی این الگوریتم باید به ازای تمامی نودها چاپ شوند، پس یک حلقه زده می‌شود و به ازای تمامی i ها متد runDVRP نتورک صدا زده می‌شود. اگر هم آرگومان داده شده بود، تنها الگوریتم را روی آن نود مربوطه اجرا می‌کنیم.

بخش ۲-کلاس network و توضیحات مربوط به آن:

این کلاس نمایانگر شبکه و توابع مربوطه به آن است. خود شبکه در طراحی ما یک وکتور دو بعدی است که ردیف آن مبدا و ستون آن مقصد است و مقدار وکتور در این ردیف و ستون نمایانگر هزینه است.

دو متغییر showLength , sizes به دلیل این است که بتوانیم خروجی را مرکزی کنیم و اعداد میان ستون‌ها و نه در طرفین قرار گیرند.

```
15  class Network
16  {
17  public:
18      Network();
19      void implementCN(std::vector <std::vector<int>> network);
20      void showTopology();
21      void modifyCost(int src, int dst, int cost);
22      void runLSRP(int src);
23      void runDVRP(int src);
24      void removeConnection(int src, int dst);
25      int  minDistance(int distance[], bool visited[]);
26
27  private:
28      void updateCNSize();
29
30      void printSolutionLSRP(int distance[]);
31      void printSolution(int src, int distance[], int parent[]);
32      std::string printPath(int parent[], int j);
33      void specialPrinter(int limit, std::string context);
34
35      std::vector<std::vector<int>> sizes;
36      std::vector <std::vector<int>> network;
37
38      int showLength=-1; // for show table, max digit count of n
39  };
```

در این بخش به توضیح منطق دستورات مختلف می‌پردازیم.

دستور topology:

```
void Network::implementCN(std::vector <std::vector<int>> network_)
{
    network = network_;
}
```

بعد از این که دستور topology وارد می‌شود commandHandler شبکه مربوطه را ساخته و سپس تابع implementCN نتورک را فراخوانی می کند.

در این تابع تنها شبکه مورد نظر ست می شود.

دستور show:

```
void Network::showTopology()
{
    updateCNSize();
    printf("%*su|v%*s|", (showLength-3)/2, "", (showLength - 3 - (showLength-3)/2), "");
    for (int i=0 ; i < sizes.size() ; i++)
    {
        int digit = int(to_string(i+1).size());
        int first = (showLength-digit)/2;
        int sec = showLength-digit-first;
        printf("%*s%s%*s", (first), "", to_string(i+1).c_str(), (sec), "");
    }
    printf("\n");
    for(int i=0 ; i < (showLength * (NODE_NUM+1)) ; i++)
        printf("-");
    printf("\n");
    for (int j = 0; j < NODE_NUM; j++)
    {
        int digit = int(to_string(j+1).size());
        int first = (showLength-digit)/2;
        int sec = showLength-digit-first;
        printf("%*s%s%*s|", (first), "", to_string(j+1).c_str(), (sec), "");
        for (int i=0 ; i < NODE_NUM ; i++)
        {
            int first = (showLength-sizes[j][i])/2;
            int sec = showLength-sizes[j][i]-first;
            printf("%*s%s%*s", (first), "", to_string(network[j][i]).c_str(), (sec), "");
        }
        printf("\n");
    }
}
```

پس از این که کامندهندلر تشخیص داد دستور show می‌باشد تابع showTopology در نتورک صدا زده می‌شود و این تابع شبکه را چاپ می‌کند.

علت طولانی بودن این تابع سعی در زیبا چاپ شدن آن است:).

دستور lsrp و dvrp:

```
void runLSRP(int src);
void runDVRP(int src);
```

اگر دستورات ورودی کاربر اجرا کردن lsrp یا dvrp بود این توابع در نتورک صدا زده می‌شوند و روی آن‌ها الگوریتم اجرا میشود. منطق این دو الگوریتم در بخش ۳ و بخش ۴ به صورت مفصل توضیح داده می‌شود.

دستور modify :

```
void Network::modifyCost(int src, int dst, int cost)
{
    network[src][dst] = cost;
    network[dst][src] = cost;
}
```

در این بخش با عوض کردن مقدار هزینه modify انجام می‌شود.

دستور remove:

```
void Network::removeConnection(int src, int dst)
{
    network[src][dst] = -1;
    network[dst][src] = -1;
}
```

در شبکه ما ۱- به معنای نبود ارتباط بین دو نود است. پس در نتیجه وقتی می‌خواهیم نودی را حذف کنیم کافی است هزینه‌ی آن را برابر با ۱- بگذاریم.

بخش ۳- توضیح الگوریتم lsrp

```
void Network::runLSRP(int src)
{
    int distance[NODE_NUM];
    bool visited[NODE_NUM];
    int parent[NODE_NUM] = {-1};
    for (int i = 0 ; i < NODE_NUM ; i++)
        distance[i] = MAX_VAL, visited[i] = 0;
    distance[src] = 0;

    for (int count = 0 ; count < NODE_NUM - 1 ; count++)
    {
        int u = minDistance(distance, visited);
        visited[u] = true;
        for (int v = 0; v < NODE_NUM; v++)
            if (!visited[v] && network[u][v] != -1 && distance[u] != MAX_VAL
                && distance[u] + network[u][v] < distance[v])
                {
                    distance[v] = distance[u] + network[u][v];
                    parent[v] = u;
                }
        cout << "Iter " << count << ":" << endl;
        printSolutionLSRP(distance);
    }
    printSolution(distance, parent);
}
```

به توضیح منطق lsrp می پردازیم:

۱- ابتدا راس مبدا را انتخاب می‌کنیم.

۲- آرایه visited که از نوع بولین است و اندازه آن برابر با تعداد رئوس گراف است معین می‌کند که تا الان چند راس بررسی شده است. در شروع، مقادیر این آرایه false است و با پیشرفت الگوریتم، رئوسی که کوتاه‌ترین مسیر به آن‌ها یافت شده‌است در آرایه true می‌شوند. یک آرایه distance نیز داریم که فاصله از نود مبدا را مشخص می‌کند.

۳- distance راس مبدا برابر با صفر قرار داده می‌شود.

۴- ابتدا نودی که کمترین فاصله را از مبدا دارد انتخاب می‌کنیم و visited آن را true می‌کنیم. برای تمامی همسایگان این نود بررسی میکنیم که اگر فاصله‌ی این نود تا نود همسایه کمتر از هزینه‌ی قبلی آن باشد آن را آپدیت نماید.

۵- به اندازه‌ی یک واحد کمتر از تعداد نودها مرحله‌ی ۴ را تکرار می‌کنیم.

بدین صورت کمترین فاصله از نود مبدا مشخص می‌شود.

در هر بار اتمام مرحله ۴ iteration مربوطه به آن چاپ می‌شود. تابع مربوطه به چاپ iteraitionها:

بعد از اتمام تمام iteraitionها حالت اخر چاپ می‌شود.در این قسمت باید shortest path نیز چاپ شود. در نتیجه در منطق الگوریتم یک آرایه تحت عنوان parent قرار داده شده که هر وقت مسیر کوتاه مشخص شد پرنت نود مورد نظر نیز ست شود. بدین صورت می‌توان shortest path را چاپ نمود. تابع مربوط به چاپ حالت نهایی:

بخش ۴ -توضیحات مربوط به الگوریتم dvtp:

```
void Network::runDVRP(int src)
{
    vector<pair<pair<int,int>,int>> edges;
    int dis[NODE_NUM];

    for (int i = 0 ; i < NODE_NUM ; i++)
        dis[NODE_NUM] = {MAX_VAL};

    for(int i = 0 ; i < NODE_NUM ; i++)
        for (int j = 0 ; j < NODE_NUM ; j++)
            if (i != j and network[i][j] !=-1)
                edges.push_back(make_pair(make_pair(i,j), network[i][j]));

    dis[src] = 0;
    int edgCount = edges.size();
    int parent[NODE_NUM] = {-1};

    for(int i=1;i<NODE_NUM;i++)
        for(int j=0;j<edgCount;j++)
        {
            int src (edges[j].first.first);
            int dest (edges[j].first.second);
            int wt (edges[j].second);
            if(dis[src] != inf and  dis[src] + wt < dis[dest])
            {
                dis[dest] = dis[src] + wt;
                parent[dest] = src;
            }
        }
    printSolution(dis, parent);
}
```

برای ران کردن این الگوریتم نیاز داریم که تمام یال‌ها را داشته باشیم. بنابر این قبل از شروع منطق اصلی تمام یال‌ها را در آرایه‌ی edges میریزیم. آرایه edges یک آرایه از pair ها است که عضو اول آن خود یک pair است که مبدا و مقصد را در دل خود دارد و عضو دوم آن هزینه یال می‌باشد.

حال به توضیح منطق برنامه می‌پردازیم.

منطق این برنامه مشابه الگوریتم بخش قبل است منتها اساس این منطق بر اساس ریلکستیشن یال‌ها می‌باشد.

۱- ابتدا راس مبدا را انتخاب می‌کنیم.

۲- یک آرایه distance تعریف می‌کنیم که فاصله از نود مبدا را مشخص می‌کند.

۳- distance راس مبدا برابر با صفر قرار داده می‌شود.

۴- به اندازه یکی کمتر از تعداد رئوس مرحله ۵ را انجام می‌دهیم.

۵- به ازای هر یال مرحله ۶ را انجام می‌دهیم.

۶- هر یال یک مبدا و یک مقصد دارد. بررسی می‌شود که آیا این یال باعث ایجاد هزینه‌ی کمتر برای رسیدن به مقصد می‌شود یا خیر. اگر باعث این موضوع بشود هزینه یا همان آرایه dis آپدیت می‌شود و همچنین parent نود نیز ست می‌شود.

در نهایت پس از اجرا شدن کامل الگوریتم بالا می‌توان هزینه را از مبدا به تمامی مقصدها با استفاده از تابع printSolution چاپ نمود.

بخش ۵- تابع main و توضیح نحوه‌ی مدیریت exception:

```
int main()
{
    Network* network = new Network();
    CommandHandler commandHandler = CommandHandler(network);
    string choosenService;
    while(getline(cin, choosenService))
    {
        try
        {
            commandHandler.getQuery(choosenService);
        }
        catch(...){}
    }
    return 0;
}
```

این تابع ورود برنامه‌ی ماست. ابتدا یک کلاس commandHandler و یک کلاس Network می‌سازد و سپس در یک حلقه شروع به دریافت دستور از کاربر می‌کند. اگر در طی برنامه کاربر دستوری وارد کند که خطا داشته باشد ارور مربوطه throw می‌شود و سپس توسط catch گرفته شده و ارور چاپ می‌شود. انواع این ارور ها در کتابخانه exception.h قرار داده شده است.

چند نمونه از ارورها:

```
class ModifySameID: public exception
{
public:
    ModifySameID()
    {
        cout<< "Can't Modify Same IDs\n";
    }
};

class ShowEmptyTable: public exception
{
public:
    ShowEmptyTable()
    {
        cout<< "No Topology Created\n";
    }
};

class IDsAlike: public exception
{
public:
    IDsAlike()
    {
        cout<< "Source and Destination node IDs are alike\n";
    }
};
```

مثلا در صورتی که نود مبدا و مقصد یکی باشند ارور IDsAlike داده می‌شود و پیام آن چاپ می‌شود.

بخش ۶- خروجی‌ها

بخش ۶.۱- خروجی show


```
ryhn@ryhnap:~/CNca/3$ ./Routing.out < entry.in
 u|v |  1   2   3   4   5   6   7   8   9  10  11  12  13
----------------------------------------------------------
  1 |  0  -1  -1  -1   6  -1   7  -1  -1  -1  -1  -1  -1
  2 | -1   0  -1  -1  -1   2  13  -1  -1  -1  -1  -1  -1
  3 | -1  -1   0  -1  -1  -1   1   4  -1  -1  -1   8  -1
  4 | -1  -1  -1   0  -1  19  -1  -1  -1  11  -1  -1  -1
  5 |  6  -1  -1  -1   0  -1  -1  -1  -1   3  -1  -1  -1
  6 | -1   2  -1  19  -1   0  -1  17  -1  -1  25  -1   4
  7 |  7  13   1  -1  -1  -1   0  -1  -1  -1  -1   8  -1
  8 | -1  -1   4  -1  -1  17  -1   0  -1  -1  16  -1  -1
  9 | -1  -1  -1  -1  -1  -1  -1  -1   0  -1  -1   5   7
 10 | -1  -1  -1  11   3  -1  -1  -1  -1   0  -1  12  -1
 11 | -1  -1  -1  -1  -1  25  -1  16  -1  -1   0  -1  -1
 12 | -1  -1   8  -1  -1  -1   8  -1   5  12  -1   0  -1
 13 | -1  -1  -1  -1  -1   4  -1  -1   7  -1  -1  -1   0
```

همان‌طور که مشاهده می‌شود در فایل ورودی که entry.in نام دارد ورودی topology را که یال ها و هزینه‌های شبکه باشند را وارد می‌کنیم و سپس دستور show را وارد می‌کنیم و مطابق انتظار جدول هزینه‌ها و یال‌ها را می‌بینیم.

بخش ۶.۲- خروجی lsrp

دستور بدون آرگومان به دلیل طولانی شدن در تست قرار نمی‌دهیم.

```
Iter 0:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | -1 | -1 | -1 | 6  | -1 | 7  | -1 | -1 | -1 | -1 | -1 | -1 |
-----------------------------------------------------------------------
Iter 1:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | -1 | -1 | -1 | 6  | -1 | 7  | -1 | -1 | 9  | -1 | -1 | -1 |
-----------------------------------------------------------------------
Iter 2:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | -1 | 6  | -1 | 7  | -1 | -1 | 9  | -1 | 15 | -1 |
-----------------------------------------------------------------------
Iter 3:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | -1 | 6  | -1 | 7  | 12 | -1 | 9  | -1 | 15 | -1 |
-----------------------------------------------------------------------
Iter 4:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | -1 | 7  | 12 | -1 | 9  | -1 | 15 | -1 |
-----------------------------------------------------------------------
Iter 5:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | 29 | 7  | 12 | -1 | 9  | 28 | 15 | -1 |
-----------------------------------------------------------------------
Iter 6:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | 29 | 7  | 12 | 20 | 9  | 28 | 15 | -1 |
-----------------------------------------------------------------------
Iter 7:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | 29 | 7  | 12 | 20 | 9  | 28 | 15 | 27 |
-----------------------------------------------------------------------
Iter 8:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | 29 | 7  | 12 | 20 | 9  | 28 | 15 | 27 |
-----------------------------------------------------------------------
Iter 9:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | 22 | 7  | 12 | 20 | 9  | 28 | 15 | 27 |
-----------------------------------------------------------------------
Iter 10:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | 22 | 7  | 12 | 20 | 9  | 28 | 15 | 26 |
-----------------------------------------------------------------------
Iter 11:
Dest  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
Cost  | 0  | 20 | 8  | 20 | 6  | 22 | 7  | 12 | 20 | 9  | 28 | 15 | 26 |
-----------------------------------------------------------------------
```

```
entry.in
1 topology 1-5-6 1-7-7 2-6-2 2-7-13 3-7-1 3-8-4 3-12-8 4-10-11 4-6-19 5-10-3 6-13-4 6
2 show
3 lsrp 1
```

Path: [s] -> [d]	Min-Cost	Shortest Path
[1] -> [2]	20	1->7->2
[1] -> [3]	8	1->7->3
[1] -> [4]	20	1->5->10->4
[1] -> [5]	6	1->5
[1] -> [6]	22	1->7->2->6
[1] -> [7]	7	1->7
[1] -> [8]	12	1->7->3->8
[1] -> [9]	20	1->7->12->9
[1] -> [10]	9	1->5->10
[1] -> [11]	28	1->7->3->8->11
[1] -> [12]	15	1->7->12
[1] -> [13]	26	1->7->2->6->13

بخش ۶.۳- خروجی dvrp

دستور بدون آرگومان به دلیل طولانی شدن در تست قرار نمی‌دهیم.

Path: [s] -> [d]	Min-Cost	Shortest Path
[1] -> [2]	20	1->7->2
[1] -> [3]	8	1->7->3
[1] -> [4]	20	1->5->10->4
[1] -> [5]	6	1->5
[1] -> [6]	22	1->7->2->6
[1] -> [7]	7	1->7
[1] -> [8]	12	1->7->3->8
[1] -> [9]	20	1->7->12->9
[1] -> [10]	9	1->5->10
[1] -> [11]	28	1->7->3->8->11
[1] -> [12]	15	1->7->12
[1] -> [13]	26	1->7->2->6->13

entry.in

```
1  topology 1-5-6 1-7-7 2-6-2 2-7-13 3-7-1 3-8-4 3-12-8 4-10-11 4-6-19 5-10-3 6-13-4 6
2  show
3  dvrp 1
```

بخش ۶.۴- خروجی تغییر هزینه مسیریابی

u|v	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	-1	-1	-1	6	-1	7	30	-1	-1	-1	-1	-1
2	-1	0	-1	-1	-1	2	13	-1	-1	-1	-1	-1	-1
3	-1	-1	0	-1	-1	-1	1	4	-1	-1	-1	8	-1
4	-1	-1	-1	0	-1	19	-1	-1	-1	11	-1	-1	-1
5	6	-1	-1	-1	0	-1	-1	-1	-1	3	-1	-1	-1
6	-1	2	-1	19	-1	0	-1	17	-1	-1	25	-1	4
7	7	13	1	-1	-1	-1	0	-1	-1	-1	-1	8	-1
8	30	-1	4	-1	-1	17	-1	0	-1	-1	16	-1	-1
9	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	5	7
10	-1	-1	-1	11	3	-1	-1	-1	-1	0	-1	12	-1
11	-1	-1	-1	-1	-1	25	-1	16	-1	-1	0	-1	-1
12	-1	-1	8	-1	-1	-1	8	-1	5	12	-1	0	-1
13	-1	-1	-1	-1	-1	4	-1	-1	7	-1	-1	-1	0

entry.in

```
1  topology 1-5-6 1-7-7 2-6-2 2-7-13 3-7-1 3-8-4 3-12-8 4-10-11 4-6-19 5-10-3 6-13-4
2  show
3  dvrp 1
4  modify 1-8-30
5  show
```

دستور modify را وارد می‌کنیم و می‌بینیم بین گره ۱و۸ یال مربوطه اضافه شده است.

بخش ۶.۵- خروجی حذف ارتباط بین دو گره

u|v	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	-1	-1	-1	6	-1	7	-1	-1	-1	-1	-1	-1
2	-1	0	-1	-1	-1	2	13	-1	-1	-1	-1	-1	-1
3	-1	-1	0	-1	-1	-1	1	4	-1	-1	-1	8	-1
4	-1	-1	-1	0	-1	19	-1	-1	-1	11	-1	-1	-1
5	6	-1	-1	-1	0	-1	-1	-1	-1	3	-1	-1	-1
6	-1	2	-1	19	-1	0	-1	17	-1	-1	25	-1	4
7	7	13	1	-1	-1	-1	0	-1	-1	-1	-1	8	-1
8	-1	-1	4	-1	-1	17	-1	0	-1	-1	16	-1	-1
9	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	5	7
10	-1	-1	-1	11	3	-1	-1	-1	-1	0	-1	12	-1
11	-1	-1	-1	-1	-1	25	-1	16	-1	-1	0	-1	-1
12	-1	-1	8	-1	-1	-1	8	-1	5	12	-1	0	-1
13	-1	-1	-1	-1	-1	4	-1	-1	7	-1	-1	-1	0

entry.in

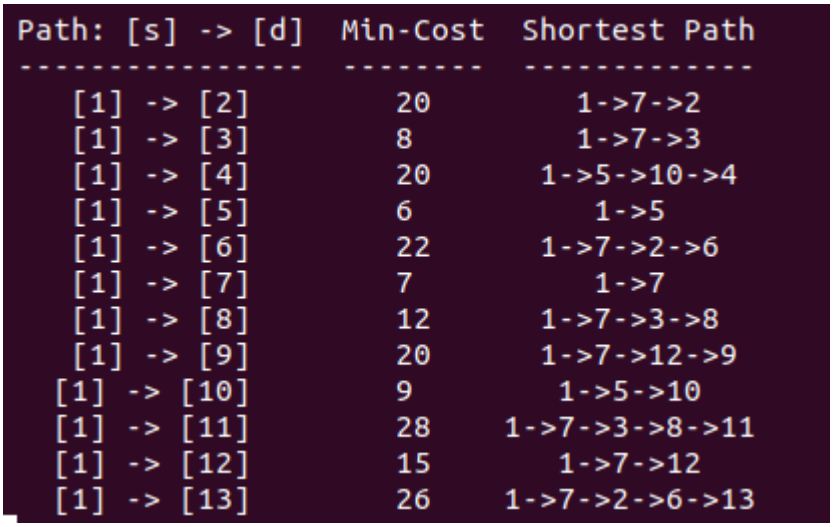
```
topology 1-5-6 1-7-7 2-6-2 2-7-13 3-7-1 3-8-4 3-12-8 4-10-11 4-6-19 5-10-3 6-13-4
show
dvrp 1
modify 1-8-30
show
remove 1-8
show
```

دستور remove را وارد می‌کنیم و می‌بینیم یال بین گره ۱و۸ مربوطه پاک شده است.

بخش ۷- مقایسه الگوریتم‌ها

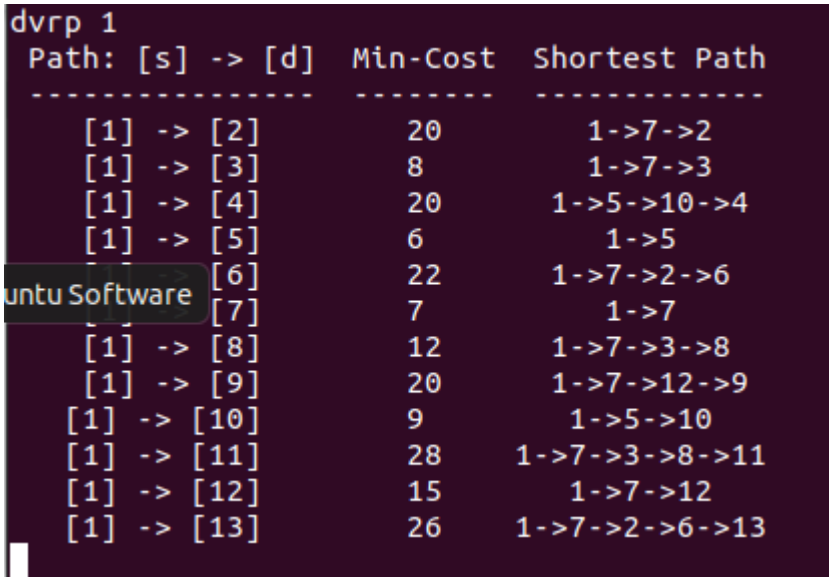
هر دو الگوریتم پاسخ‌های یکسانی می‌دهند.

اجرای الگوریتم lsrp روی نود 1:



Path: [s] -> [d]	Min-Cost	Shortest Path
[1] -> [2]	20	1->7->2
[1] -> [3]	8	1->7->3
[1] -> [4]	20	1->5->10->4
[1] -> [5]	6	1->5
[1] -> [6]	22	1->7->2->6
[1] -> [7]	7	1->7
[1] -> [8]	12	1->7->3->8
[1] -> [9]	20	1->7->12->9
[1] -> [10]	9	1->5->10
[1] -> [11]	28	1->7->3->8->11
[1] -> [12]	15	1->7->12
[1] -> [13]	26	1->7->2->6->13

اجرای الگوریتم dvrp روی نود 1:



dvrp 1

Path: [s] -> [d]	Min-Cost	Shortest Path
[1] -> [2]	20	1->7->2
[1] -> [3]	8	1->7->3
[1] -> [4]	20	1->5->10->4
[1] -> [5]	6	1->5
[1] -> [6]	22	1->7->2->6
[1] -> [7]	7	1->7
[1] -> [8]	12	1->7->3->8
[1] -> [9]	20	1->7->12->9
[1] -> [10]	9	1->5->10
[1] -> [11]	28	1->7->3->8->11
[1] -> [12]	15	1->7->12
[1] -> [13]	26	1->7->2->6->13

برای مقایسه‌ی الگوریتم‌ها با یکدیگر زمان اجرای هر کدام را محاسبه می‌کنیم. با استفاده از تکه کد زیر:

```
auto start = high_resolution_clock::now();
for (int i = 0; i < CN_NODE_COUNT; i++)
{
    cout << "lsrp for node "<< i+1<< ":" << endl;
    network->runLSRP(i);
}
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
cout << "Time taken by LSRP function: "<< duration.count() << " microseconds" << endl;
```

ابتدا زمان شروع در start ذخیره می‌شود. سپس حلقه‌ی اجرای الگوریتم انجام می‌شود. سپس بعد از اتمام اجرای حلقه زمان پایان در stop ذخیره می‌شود. stop - start برابر با مقدار زمانی است که اجرای الگوریتم برای پیدا کردن کوتاه‌ترین مسیر طول کشیده است.

برای پیدا کردن مقدار زمان دقیقی که خود الگوریتم طول می‌کشد به صورت موقت پرینت‌ها را پاک می‌نماییم.

```
lsrp
Time taken by LSRP function: 174 microseconds
dvrp
Time taken by DVRP function: 488 microseconds
remove 4-10
lsrp
Time taken by LSRP function: 182 microseconds
dvrp
Time taken by DVRP function: 472 microseconds
```

دو زمان اول قبل از حذف ارتباط بین دو نود 4 و 10 می‌باشد. مشاهده می‌شود که الگوریتم lsrp به مراتب سریع‌تر از الگوریتم dvrp است.

همین‌طور بهترین کامند بعد از make کردن برای مشاهده تک تک نکات گفته شده در زیر نمایش داده شده:

```
ryhn@ryhnap:~/CNca/3$ ./Routing.out < entry.in > response.txt
ryhn@ryhnap:~/CNca/3$ ./Routing.out < entry.in > response.txt
```