گزارش پروژه سوم درس شبکه‌های کامپیوتری

دکتر یزدانی

بهار ۱۴۰۱

پرنیان فاضل ۸۱۰۱۹۸۵۱۶ - پریا خوش‌تاب ۸۱۰۱۹۸۳۸۷

پیاده‌سازی الگوریتم‌های مسیریابی در شبکه

ما در این پروژه کلاس‌های زیر را پیاده‌سازی کرده‌ایم:

- Routing
- LinkState
- DistanceVector

کلاس Routing:

این کلاس وظیفه هندل کردن دستورات ورودی، ساختن و تغییر دادن توپولوژی (گراف) شبکه را بر عهده دارد.

فیلدها:

- numberOfNodes: تعداد نودهای گراف یا همان تعداد روترهای موجود در شبکه می‌باشد.
- edges: یک آرایه دوبعدی از ساختار Edge (شامل نود مبدا و مقصد) می‌باشد که برای ذخیره کردن یال‌های گراف یا همان لینک‌های شبکه استفاده می‌شود.
- cost: یک آرایه دوبعدی از float می‌باشد که برای ذخیره کردن هزینه‌ی لینک‌های میان روترها استفاده می‌شود.

```
381  private:
382      int numberOfNodes;
383      Edges edges;
384      float cost[MAXN][MAXN];
385  };
```

متدها:

- initialize: در این متد آرایه cost مقداردهی اولیه می‌شود. در صورتی که مبدا و مقصد یکسان باشد، هزینه لینک با صفر و در غیر این صورت با 1- مقدار دهی می‌شود.

```
236      void initialize() {
237          for (int i = 0; i < MAXN; i++) {
238              for (int j = 0; j < MAXN; j++) {
239                  if (i == j)
240                      cost[i][j] = 0;
241                  else
242                      cost[i][j] = -1;
243              }
244          }
245      }
```

- parse: این متد یک رشته و یک delimiter را به عنوان ورودی دریافت می‌کند و سپس یک وکتور از رشته تجزیه شده را ریترن می‌کند.

```
247      vector<string> parse(const string& input, char delim) {
248          vector<string> parsed;
249          stringstream ssInp(input);
250          string str;
251          while (getline(ssInp, str, delim))
252              parsed.push_back(str);
253          return parsed;
254      }
```

- setTopology: این متد توپولوژی شبکه را با ذخیره کردن کردن لینک‌ها و هزینه آن‌ها تشکیل می‌دهد.

```
255
256     void setTopology(const vector<string>& args) {
257         numberOfNodes = INFMIN;
258         for (int i = 0; i < args.size(); i++) {
259             vector<string> edgeTpg = parse(args[i], '-');
260             edges.push_back({ stoi(edgeTpg[0]), stoi(edgeTpg[1]) });
261             edges.push_back({ stoi(edgeTpg[1]), stoi(edgeTpg[0]) });
262             int u = stoi(edgeTpg[0]);
263             int v = stoi(edgeTpg[1]);
264             int maximum = max(u, v);
265             numberOfNodes = max(maximum, numberOfNodes);
266             cost[u][v] = stof(edgeTpg[2]);
267             cost[v][u] = stof(edgeTpg[2]);
268         }
269     }
```

- showTopology: این متد هندلر دستور show می‌باشد که ارتباط بین نودهای شبکه به همراه هزینه بین آن‌ها را در قالب یک جدول چاپ می‌کند.

```
271     void showTopology() {
272         cout << " u|v  ";
273         for (int i = 1; i <= numberOfNodes; i++) {
274             cout << std::setw(3) << i;
275         }
276         cout << endl;
277 
278         for (int i = 0; i<numberOfNodes; i++) {
279             cout << std::setw(3) << "------";
280         }
281         cout << endl;
282         for (int i = 1; i <= numberOfNodes; i++) {
283             cout << std::setw(3) << i << " | ";
284             for (int j = 1; j <= numberOfNodes; j++) {
285                 cout << std::setw(3) << cost[i][j];
286             }
287             cout << endl;
288         }
289     }
```

- modifyTopology: این متد هندلر دستور modify می‌باشد که با استفاده از آن می‌توان هزینه مسیریابی بین دو گره در شبکه را تغییر داد.

```
291     void modifyTopology(string topologies_str) {
292         vector<string> topology_str = parse(topologies_str, '-');
293         int source = stoi(topology_str[0]);
294         int dest = stoi(topology_str[1]);
295         float weight = stof(topology_str[2]);
296         if (dest == source) {
297             cout << "ERROR: Source and destination cannot be the same!" << endl;
298             return;
299         }
300         numberOfNodes = max(numberOfNodes, max(source, dest));
301         if (cost[source][dest] == -1) {
302             cost[source][dest] = cost[dest][source] = weight;
303             cost[source][source] = cost[dest][dest] = 0;
304             edges.push_back({ source, dest });
305             edges.push_back({ dest, source });
306         }
307         else {
308             cost[source][dest] = cost[dest][source] = weight;
309         }
310     }
```

- removeEdge: این متد یک یال را از آرایه یال‌ها (edges) پاک می‌کند.

```
312     void removeEdge(int src, int dest) {
313         for (int i = 0; i < edges.size(); i++) {
314             if (edges[i].u == src && edges[i].v == dest) {
315                 edges.erase(std::remove(edges.begin(), edges.end(), edges[i]), edges.end());
316             }
317         }
318     }
```

- removeTopology: این متد هندلر مربوط به دستور remove می‌باشد که با استفاده از آن می‌توان مسیر میان دو گره در شبکه را حذف کرد.

```
320     void removeTopology(string removeCommand) {
321         vector<string> parsedCommand = parse(removeCommand, '-');
322         int source = stoi(parsedCommand[0]);
323         int dest = stoi(parsedCommand[1]);
324 
325         if (dest == source) {
326             cout << "ERROR: Source and destination cannot be the same!" << endl;
327             return;
328         }
329 
330         if (cost[source][dest] == -1) {
331             cout << "ERROR: this topology does NOT exists!" << endl;
332             return;
333         }
334 
335         cost[source][dest] = cost[dest][source] = -1;
336         removeEdge(source, dest);
337         removeEdge(dest, source);
338     }
```

- handleCommand: این متد دستورات را از رابط خط فرمان دریافت میکند و بسته به نوع دستور توابع مربوطه آن را فراخوانی می‌کند.

```
340     void handleCommand() {
341         string input;
342         while (getline(cin, input)) {
343             vector<string> parsedCommand = parse(input, ' ');
344             string command = parsedCommand[0];
345             vector<string> arguments(parsedCommand.begin() + 1, parsedCommand.end());
346 
347             if (command == TOPOLOGY_CMD) {
348                 initialize();
349                 setTopology(arguments);
350             }
351 
352             else if (command == SHOW_CMD) {
353                 showTopology();
354             }
355 
356             else if (command == LSRP_CMD) {
357                 LinkState* ls = new LinkState();
358                 if (arguments.size() == 1) {
359                     ls->lsrp(stoi(arguments[0]), numberOfNodes, cost, edges);
360                 }
361                 else {
362                     ls->lsrp(-1, numberOfNodes, cost, edges);
363                 }
364             }
365             else if (command == DVRP_CMD) {
366                 DistanceVector* dv = new DistanceVector();
367                 if (arguments.size() == 1)
368                     dv->dvrp(stoi(arguments[0]), numberOfNodes, edges, cost);
369                 else
370                     dv->dvrp(-1, numberOfNodes, edges, cost);
371             }
372             else if (command == MODIFY_CMD) {
373                 modifyTopology(arguments[0]);
374             }
375             else if (command == REMOVE_CMD) {
376                 removeTopology(arguments[0]);
377             }
378         }
379     }
```

کلاس LinkState:

در نوع مسیریابی LinkState هر روتر اطلاعات همسایگی خود را با هر روتر دیگری در اینترنت به اشتراک می گذارد. در این الگوریتم، هر روتر در شبکه توپولوژی شبکه را درک می کند و سپس جدول مسیریابی را به این توپولوژی وابسته می کند. برای این روش از الگوریتم دایکسترا استفاده میکنیم.

فیلدها:

- dist: یک آرایه 2 بعدی از float می‌باشد که برای ذخیره کردن کمترین هزینه از نود مبدا به نود مقصد می‌باشد.
- parent: یک آرایه از int می‌باشد که برای ذخیره کردن والد هر نود استفاده می‌شود.

```
35  private:
36      float dist[MAXN];
37      int parent[MAXN];
```

متدها:

- lsrp: این متد هندلری برای دستور lsrp می‌باشد. در این متد بر اساس اینکه آرگومان به دستور lsrp داده شده یا نه تصمیم میگیرد که یک بار تابع دایکسترا(که بعدا توضیح داده خواهد شد) را با source داده شده صدا بزند یا در صورتی که بدون آرگومان داده شود، دایکسترا را روی همه گره‌ها صدا کند.

```
99  void lsrp(int src, int numberOfNodes, float cost[MAXN][MAXN], Edges edges) {
100     set <int> vertices;
101     for (auto edge : edges)
102         vertices.insert(edge.u);
103
104
105     if (src != -1) {
106         auto start = std::chrono::steady_clock::now();
107         dijkstra(src, numberOfNodes, cost);
108         auto end = std::chrono::steady_clock::now();
109         std::cout << "Convergence time: " << std::chrono::duration <double, std::milli>(end - start).count() << " ms" << std::endl;
110     }
111     else {
112         for (auto v : vertices) {
113             dijkstra(v, numberOfNodes, cost);
114         }
115     }
116 }
```

- dijkstra: در این تابع الگوریتم دایکسترا را پیاده‌سازی کرده‌ایم. کد این الگوریتم صورت زیر می‌باشد. پیچیدگی زمانی این کد برابر $O(E\ logV)$ است. توضیحات بیشتر این الگوریتم در صفحات 246 تا 248 کتاب مرجع گفته شده است.

```
51      void dijkstra(int src, int numberOfNodes, float cost[MAXN][MAXN]) {
52          for (int i = 0; i < MAXN; i++) {
53              dist[i] = INFMAX;
54              parent[i] = -1;
55          }
56
57          dist[src] = 0;
58
59          bool visited[MAXN] = { 0 };
60          int iterNum = 1;
61          for (int i = 1; i <= numberOfNodes; i++) {
62              if (i == src)
63                  continue;
64              int nearest = getMinNodeInDijkstra(visited, numberOfNodes);
65              visited[nearest] = 1;
66              for (int adj = 1; adj <= numberOfNodes; adj++) {
67                  float newDistance = dist[nearest] + cost[nearest][adj];
68                  if (!visited[adj] && dist[nearest] != INFMAX && cost[nearest][adj] != -1 && newDistance < dist[adj])
69                      dist[adj] = newDistance;
70                      parent[adj] = nearest;
71                  }
72              }
73              printIterations(iterNum, numberOfNodes);
74              iterNum++;
75          }
76          print(numberOfNodes, src);
77      }
```

- getMinNodeInDijkstra: تابع دایکسترا از این متد استفاده میکند. در واقع در هر iteration باید گره با کمترین هزینه را پیدا کنیم که این تابع مسئولیت انجام این کار را دارد.

```
40      int getMinNodeInDijkstra(bool visited[], int numberOfNodes) {
41          int node = 1, tempMin = INFMAX;
42          for (int i = 1; i <= numberOfNodes; i++) {
43              if ((!visited[i]) && tempMin >= dist[i]) {
44                  node = i;
45                  tempMin = dist[i];
46              }
47          }
48          return node;
49      }
```

- printIterations: در هر iteration تابع دایکسترا، نتایج آن با استفاده از این تابع چاپ می‌شود.

```
79      void printIterations(int iterNum, int numberOfNodes) {
80          cout << "Iter: " << iterNum << endl;
81          cout << "Dest |";
82          for (int n = 1; n <= numberOfNodes; n++) {
83              cout << std::setw(3) << n << " |";
84          }
85          cout << endl;
86          cout << "Cost |";
87          for (int n = 1; n <= numberOfNodes; n++) {
88              if (dist[n] == INFMAX)
89                  cout << std::setw(3) << "-1" << " |";
90              else
91                  cout << std::setw(3) << dist[n] << " |";
92          }
93          cout << endl;
94          for (int i = 0; i < numberOfNodes; i++) cout << "--------";
95          cout << endl;
96      }
```

- print: با استفاده از این متد جدول مربوط به خروجی اعمال دایسکترا چاپ می شود.

```
127     void print(int numberOfNodes, int src) {
128         cout << "Path: [s] -> [d] | \tMin-Cost \t|\t Shortest Path";
129         for (int i = 1; i < numberOfNodes + 1; i++) {
130             if (i == src)
131                 continue;
132 
133             cout << endl;
134             string path = "";
135             printPath(i, path);
136             path.pop_back();
137             path.pop_back();
138             cout << "\t" << src << " -> " << i << " \t\t " << dist[i] << " \t\t\t " << path;
139         }
140         cout << endl;
141         for (int i = 0; i < numberOfNodes; i++) cout << "--------------------";
142         cout << endl;
143     }
```

- printPath: این متد در متد print صدا زده می‌شود و برای چاپ کردن کوتاه ترین مسیر پیدا شده در جدول این الگوریتم استفاده می‌شود.

```
118     void printPath(int v, string& path) {
119         if (parent[v] == -1) {
120             path += to_string(v) + "->";
121             return;
122         }
123         printPath(parent[v], path);
124         path += to_string(v) + "->";
125     }
```

کلاس DistanceVector:

در این نوع مسیریابی distante vector، فرآیند مسیریابی را با فرض هزینه هر پیوند یک واحد ساده می کند. بنابراین، کارایی انتقال را می توان با تعداد لینک ها برای رسیدن به مقصد اندازه گیری کرد. در مسیریابی برداری فاصله ای، هزینه بر اساس تعداد hop است.

فیلدها:

- dist: یک آرایه 2 بعدی از float می‌باشد که برای ذخیره کردن کمترین هزینه از نود مبدا به نود مقصد می‌باشد.
- parent: یک آرایه از int می‌باشد که برای ذخیره کردن والد هر نود استفاده می‌شود.

- nextHop: اولین نودی که در مسیر با کوتاه‌ترین هزینه از مبدا به آن می‌رویم، در این متغیر ذخیره می‌شود.
- nextHopFound: یک متغیر bool می‌باشد که زمانی که نود nextHop پیدا شود، آن را true می‌کنیم.

```
150   private:
151       float dist[MAXN];
152       int parent[MAXN];
153       int nextHop;
154       bool nextHopFound;
```

متدها:

- print: با استفاده از این متد جدول مربوط به خروجی اعمال bellman ford چاپ می شود.

```
161   void print(int src, int numberOfNodes)
162   {
163       cout << "Dest   \tNext Hop  Dist    \tShortest Path" << endl;
164       for (int i = 0; i < numberOfNodes; i++) cout << "--------------------";
165       cout << endl;
166       for (int i = 1; i <= numberOfNodes; i++) {
167           string path = "";
168           nextHop = src;
169           printPath(i, path);
170           path.pop_back();
171           path.pop_back();
172           cout << i << "\t\t" << nextHop << "\t\t" << dist[i] << "\t\t[" << path << "]" << endl;
173       }
174       cout << endl;
175   }
```

- printPath: این متد در متد print صدا زده می‌شود و برای چاپ کردن کوتاه ترین مسیر پیدا شده در جدول این الگوریتم استفاده می‌شود.

```
177     void printPath(int v, string& path) {
178         if (parent[v] == -1) {
179             nextHopFound = true;
180             path += to_string(v) + "->";
181             return;
182         }
183         printPath(parent[v], path);
184         if (nextHopFound) {
185             nextHop = v;
186             nextHopFound = false;
187         }
188         path += to_string(v) + "->";
189     }
```

- bellmanFord: در این تابع الگوریتم bellman ford را پیاده‌سازی کرده‌ایم. کد این الگوریتم صورت زیر می‌باشد. پیچیدگی زمانی این کد برابر $O(V.E)$ است. توضیحات بیشتر این الگوریتم در صفحات 234 تا 237 کتاب مرجع گفته شده است.

```
192     void bellmanFord(int src, int numberOfNodes, vector<Edge> edges, float cost[MAXN][MAXN])
193     {
194         for (int i = 0; i < MAXN; i++) {
195             dist[i] = INFMAX;
196             parent[i] = -1;
197         }
198         dist[src] = 0;
199 
200         for (int i = 1; i <= numberOfNodes - 1; i++)
201         {
202             for (int j = 0; j < edges.size(); j++)
203             {
204                 int u = edges[j].u;
205                 int v = edges[j].v;
206                 int weight = cost[u][v];
207                 if (dist[u] != INFMAX && (dist[u] + weight < dist[v])) {
208                     dist[v] = dist[u] + weight;
209                     parent[v] = u;
210                 }
211             }
212         }
213         print(src, numberOfNodes);
214     }
```

- dvrp: این متد هندلری برای دستور dvrp می‌باشد. در این متد بر اساس اینکه آرگومان به دستور dvrp داده شده یا نه تصمیم میگیرد که یک بار تابع bellmanFord را با source داده شده صدا بزند یا در صورتی که بدون آرگومان داده شود، bellmanFord را روی همه گره‌ها صدا کند.

```cpp
    void dvrp(int src, int numberOfNodes, vector<Edge> edges, float cost[MAXN][MAXN]) {
        set <int> vertices;
        for (auto edge : edges)
            vertices.insert(edge.u);

        if (src != -1) {
            auto start = std::chrono::steady_clock::now();
            bellmanFord(src, numberOfNodes, edges, cost);
            auto end = std::chrono::steady_clock::now();
            std::cout << "Convergence time: " << std::chrono::duration <double, std::milli>(end - start).count() << " ms" << std::endl;
        }
        else {
            for (auto v : vertices) {
                bellmanFord(v, numberOfNodes, edges, cost);
            }
        }
    }
};
```

ارزیابی و مقایسه نتایج

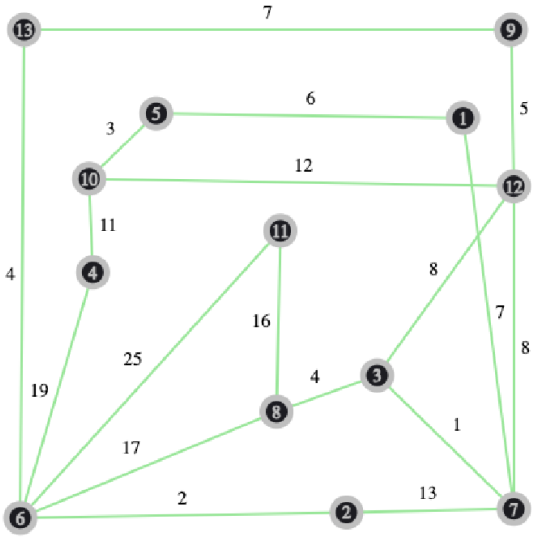
خروجی توپولوژی داده شده در صورت پروژه به عنوان نمونه:

```
topology 1-2-19 2-4-3 3-1-9
show
 u|v    1  2  3  4
------------------------
  1 |   0 19  9 -1
  2 |  19  0 -1  3
  3 |   9 -1  0 -1
  4 |  -1  3 -1  0
lsrp 1
Iter: 1
Dest |  1 |  2 |  3 |  4 |
Cost |  0 | 19 |  9 | -1 |
--------------------------------
Iter: 2
Dest |  1 |  2 |  3 |  4 |
Cost |  0 | 19 |  9 | -1 |
--------------------------------
Iter: 3
Dest |  1 |  2 |  3 |  4 |
Cost |  0 | 19 |  9 | 22 |
--------------------------------
Path: [s] -> [d] |      Min-Cost        |        Shortest Path
        1 -> 2          19                      1->2
        1 -> 3          9                       1->3
        1 -> 4          22                      1->2->4
-------------------------------------------------------------------------------
dvrp 1
Dest            Next Hop        Dist            Shortest Path
-------------------------------------------------------------------------------
1               1               0               [1]
2               2               19              [1->2]
3               3               9               [1->3]
4               2               22              [1->2->4]
```

```
modify 1-3-4
lsrp 1
Iter: 1
Dest |  1 |  2 |  3 |  4 |
Cost |  0 | 19 |  4 | -1 |
--------------------------------
Iter: 2
Dest |  1 |  2 |  3 |  4 |
Cost |  0 | 19 |  4 | -1 |
--------------------------------
Iter: 3
Dest |  1 |  2 |  3 |  4 |
Cost |  0 | 19 |  4 | 22 |
--------------------------------
Path: [s] -> [d] |      Min-Cost        |        Shortest Path
       1 -> 2          19                      1->2
       1 -> 3          4                       1->3
       1 -> 4          22                      1->2->4
-------------------------------------------------------------------------------
dvrp 1
Dest            Next Hop        Dist            Shortest Path
-------------------------------------------------------------------------------
1               1               0               [1]
2               2               19              [1->2]
3               3               4               [1->3]
4               2               22              [1->2->4]

show
 u|v    1  2  3  4
------------------------
  1 |   0 19  4 -1
  2 |  19  0 -1  3
  3 |   4 -1  0 -1
  4 |  -1  3 -1  0
remove 1-3
show
 u|v    1  2  3  4
------------------------
  1 |   0 19 -1 -1
  2 |  19  0 -1  3
  3 |  -1 -1  0 -1
  4 |  -1  3 -1  0
```

خروجی توپولوژی زیر که در صورت پروژه گفته شده بود همراه با دستورات خواسته شده در فایل output.txt قرار دارد.



با توجه به اینکه پرینت کردن زمان زیادی میگرفت، برای دقت بیشتر در محاسبه، پرینت کردن ها را کامنت

کردیم و سپس زمان همگرایی را محاسبه کردیم که به صورت زیر است:

```
topology 1-5-6 1-7-7 2-7-13 2-6-2 3-7-1 3-8-4 4-6-19 4-10-11 5-10-3 5-1-6 6-8-17 6-11-25 6-13-4 7-12-8 8-11-16 9-12-5 9-13-7 10-12-12
show
 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 -1 -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 -1 -1 -1 -1 -1  8 -1  5 12 -1  0 -1
 13 |  -1 -1 -1 -1 -1 -1  4 -1 -1  7 -1 -1 -1  0
lsrp
Convergence time of LSRP: 0.0081 ms
Convergence time of LSRP: 0.0062 ms
Convergence time of LSRP: 0.0056 ms
Convergence time of LSRP: 0.0056 ms
Convergence time of LSRP: 0.0048 ms
Convergence time of LSRP: 0.0049 ms
Convergence time of LSRP: 0.005 ms
Convergence time of LSRP: 0.005 ms
Convergence time of LSRP: 0.0051 ms
Convergence time of LSRP: 0.0051 ms
Convergence time of LSRP: 0.005 ms
Convergence time of LSRP: 0.0051 ms
Convergence time of LSRP: 0.0047 ms
dvrp
Convergence time of DVRP: 0.5324 ms
Convergence time of DVRP: 0.3924 ms
Convergence time of DVRP: 0.3527 ms
Convergence time of DVRP: 0.3942 ms
Convergence time of DVRP: 0.4659 ms
Convergence time of DVRP: 0.3171 ms
Convergence time of DVRP: 0.3166 ms
Convergence time of DVRP: 0.3197 ms
Convergence time of DVRP: 0.3062 ms
Convergence time of DVRP: 0.3009 ms
Convergence time of DVRP: 0.3018 ms
Convergence time of DVRP: 0.3051 ms
Convergence time of DVRP: 0.3216 ms
```

```
remove 4-10
show
 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 -1 -1
  4 |  -1 -1 -1  0 -1 19 -1 -1 -1 -1 -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 -1  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 -1 -1 -1 -1  8 -1  5 12 -1  0 -1
 13 |  -1 -1 -1 -1 -1  4 -1 -1  7 -1 -1 -1  0
lsrp
Convergence time of LSRP: 0.0079 ms
Convergence time of LSRP: 0.0074 ms
Convergence time of LSRP: 0.0057 ms
Convergence time of LSRP: 0.0067 ms
Convergence time of LSRP: 0.0053 ms
Convergence time of LSRP: 0.0055 ms
Convergence time of LSRP: 0.0052 ms
Convergence time of LSRP: 0.005 ms
Convergence time of LSRP: 0.0061 ms
Convergence time of LSRP: 0.0066 ms
Convergence time of LSRP: 0.0056 ms
Convergence time of LSRP: 0.0069 ms
Convergence time of LSRP: 0.0071 ms
dvrp
Convergence time of DVRP: 0.3678 ms
Convergence time of DVRP: 0.3782 ms
Convergence time of DVRP: 0.3596 ms
Convergence time of DVRP: 0.3308 ms
Convergence time of DVRP: 0.3018 ms
Convergence time of DVRP: 0.289 ms
Convergence time of DVRP: 0.2882 ms
Convergence time of DVRP: 0.4113 ms
Convergence time of DVRP: 0.2967 ms
Convergence time of DVRP: 0.2893 ms
Convergence time of DVRP: 0.2961 ms
Convergence time of DVRP: 0.2961 ms
Convergence time of DVRP: 0.2973 ms
```

همانطور که انتظار داشتیم با توجه به اینکه پیچیدگی زمانی LSRP که با دایکسترا است ($O(E\,logV)$) کمتر از DVRP است($O(V.E)$) که با Bellman-ford است، با وجود یکسان بودن خروجی(که در فایل output.txt قرار دارد)، زمان همگرایی LSRP بسیار کمتر از DVRP شده است و درست است.