گزارش پروژه سوم درس شبکه‌های کامپیوتری (Routing Protocols)

کیمیا محمدطاهری (۸۱۰۱۹۸۵۳۵)
دانشور امراللهی (۸۱۰۱۹۷۶۸۵)

مقدمه

در این تمرین کامپیوتری، به پیاده‌سازی الگوریتم‌های Distance Vector و Link State برای مسیریابی در شبکه‌های کامپیوتری می‌پردازیم.

ساختار کلی برنامه

ساختار کد این پروژه به شکل زیر است:

```
├── build
│   ├── Graph.o
│   ├── Main.o
│   └── Network.o
├── Documentation
│   └── CN-CA3.pdf
├── include
│   ├── Graph.hpp
│   └── Network.hpp
├── input.txt
├── main.out
├── Makefile
├── README.md
└── src
    ├── Graph.cpp
    ├── main.cpp
    └── Network.cpp
```

برای پیاده‌سازی الگوریتم، از دو کلاس Graph و Network استفاده شده است.
کلاس Network شامل یک گراف است و با دریافت دستورات، متدهای متناسب از این کلاس Graph را صدا می‌زند.

```
class Network
{
private:
    Graph* graph;
public:
    void run();
};
```

کلاس Graph، گراف راس‌ها را نگهداری می‌کند و مسئول اجرای منطق الگوریتم‌ها می‌باشد.

```
class Graph
{
private:
    set<int> nodes;
    map<pair<int, int>, int> weight;
public:
    void add_edge(int v, int u, int w);
    void show();
    void delete_edge(int v, int u);
    void modify_edge(int v, int u, int cost);
    Graph(string topology);
    void link_state(int source);
    void distance_vector(int source);
};
```

الگوریتم‌های مسیریابی

دو نوع الگوریتم در این پروژه پیاده سازی شده است:

- الگوریتم Distance Vector

این الگوریتم در متد `distance_vector` کلاس `Graph` پیاده سازی شده است. پیاده سازی آن به شکل زیر است:

```
vector<bool> mark(n + 1, false);
vector<int> dist(n + 1, INF);
vector<int> par(n + 1, -1);
dist[source] = 0;
while (1)
{
    bool updated = false;
    for (map<pair<int, int>, int> :: iterator it = weight.begin(); it != 
weight.end(); it++)
    {
        int v = it->first.first, u = it->first.second, w = it->second;
        if (dist[v] + w < dist[u])
        {
            dist[u] = dist[v] + w;
            par[u] = v;
```

```
            updated = true;
        }
    }
    if (!updated)
        break;
}
```

این الگوریتم کاملا مشابه Bellman-Ford عمل می‌کند. آنقدر روی یال‌ها عمل relaxation انجام می‌دهد (آپدیت کردن فاصله‌ها) تا موقعی که تغییر جدیدی در فاصله رخ ندهد.
عمل چک کردن شرط $dist_v + w < dist_u$ معادل همان چک کردن distance vector table همسایه‌ها است. در اینجا u همسایه v است و w وزن یال v به u است.

- الگوریتم Link State

این الگوریتم در متد `link_state` کلاس `Graph` پیاده سازی شده است. پیاده سازی آن به شکل زیر است:

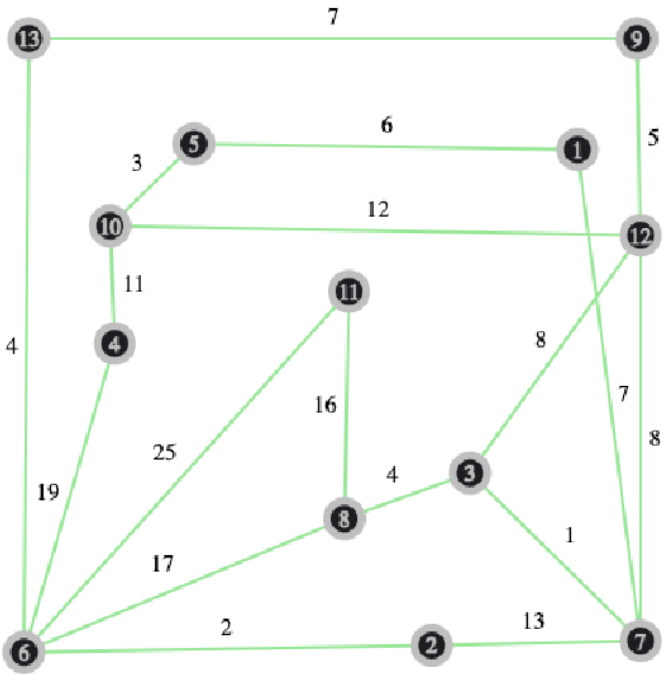
```
mark[source] = true;
dist[source] = 0;
int sz = 1;
while (sz < n)
{
    int mn = INF, v;
    for (auto node: nodes)
    {
        if (mark[node])
            continue;
        if (dist[node] < mn)
        {
            mn = dist[node];
            v = node;
        }
    }


//Picked up vertex (v) with min distance out of set
    mark[v] = true; //bring new vertex into set
    sz += 1;
```

```
    for (auto node: nodes) //update distances
    {
        if (mark[node])
            continue;
        if (weight.find(make_pair(v, node)) == weight.end())
            continue;
        if (dist[v] + weight[make_pair(v, node)] < dist[node])
        {
            dist[node] = dist[v] + weight[make_pair(v, node)];
            par[node] = v;
        }
    }
}
```

نتایج

برای بررسی عملکرد دو الگوریتم از گراف زیر استفاده کردیم:



فایل تست ورودی بر اساس این گراف به صورت زیر خواهد بود:

```
topology 1-5-6 1-7-7 2-6-2 2-7-13 3-7-1 3-8-4 3-12-8 4-6-19
```

```
4-10-11 5-10-3 6-8-17 6-11-25 6-13-4 7-12-8 8-11-16 9-12-5
9-13-7 10-12-12
```

خروجی الگوریتم distance vector به ازای راس ۱ :

```
Dest     NextHop       Dist         Shortest Path
-------------------------------------------------------
2          7           20          1 -> 7 -> 2
3          7           8           1 -> 7 -> 3
4          5           20          1 -> 5 -> 10 -> 4
5          5           6           1 -> 5
6          7           22          1 -> 7 -> 2 -> 6
7          7           7           1 -> 7
8          7           12          1 -> 7 -> 3 -> 8
9          7           20          1 -> 7 -> 12 -> 9
10         5           9           1 -> 5 -> 10
11         7           28          1 -> 7 -> 3 -> 8 -> 11
12         7           15          1 -> 7 -> 12
13         7           26          1 -> 7 -> 2 -> 6 -> 13
```

خروجی الگوریتم link state به ازای راس ۱ :

```
   |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|  -1|  -1|  -1|  -1|
-----------------------------------------------------------------------
   |Iter 2:
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 3:
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 4:
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 5:
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 6:
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 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|  -1|
-----------------------------------------------------------------------
   |Iter 8:
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|  -1|
-----------------------------------------------------------------------
   |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|  -1|
-----------------------------------------------------------------------
   |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|  27|
-----------------------------------------------------------------------
   |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|
-----------------------------------------------------------------------
   |Iter 12:
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|
-----------------------------------------------------------------------

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
```

مقایسه زمان‌های اجرا

- پیش از حذف یال ۱۰-۴:

Source	Distance Vector	Link State
1	64.785	183.288
2	37.181	178.447
3	39.974	129.033
4	49.553	126.016
5	45.614	124.373
6	40.473	122.045
7	35.008	120.723
8	44.802	134.012
9	37.941	124.309
10	51.398	131.036
11	49.815	127.694
12	38.057	119.668
13	41.372	123.039

- پس از حذف یال ۱۰-۴:

Source	Distance Vector	Link State
1	25.679	126.274
2	23.975	123.144
3	23.37	120.816
4	29.918	125.318
5	30.593	126.701
6	25.863	121.049
7	22.062	119.814
8	27.333	123.464
9	23.189	125.835
10	32.699	153.534
11	30.459	148.407
12	24.413	139.213
13	25.333	134.61

اعداد نوشته‌شده در جدول‌ها به واحد میکروثانیه هستند.

پیچیدگی زمانی الگوریتم دایکسترا (Link State) برابر $O(V^3)$ می‌باشد. البته پیاده‌سازی‌های مختلفی از آن وجود دارد.
همچنین پیچیدگی زمانی الگوریتم بلمن-فورد (Distance Vector) برابر $O(V.E)$ می‌باشد. در این مورد هم بسته به پیاده‌سازی، ممکن است ورژن‌های مختلفی وجود داشته باشد.

به طور کلی $E = O(V^2)$. اما در مثال تست‌شده، تعداد یال‌ها (E) از اردر مربع تعداد راس‌ها (V) نیست. بنابراین انتظار می‌رود الگوریتم Link State کندتر عمل کند که اعداد نیز همین موضوع را تایید می‌کنند.