EXP3 算法基础实验报告

廖佳怡 PB19151776

1. 实验设备和环境

编译环境：Windows10

编程语言：C++

机器内存：16.0GB

时钟主频：2.30GHz

2. 实验内容及要求

- 实验3.1 Bellman-Ford算法
    - 内容

      实现求单源最短路径的Bellman-Ford算法。有向稀疏图的顶点数 N 的取值分别为: 27、81、243、729，每个顶点作为起点引出的边的条数取值分别为：log5N、log7N（取上整），其邻接矩阵存放在输入文件中。输入规模总共有4*2=8个，统计算法所需运行时间，画出时间曲线，分析程序性能。
    - 要求

      input/
      - 输入图是有向稀疏图，顶点数N分别为27, 81, 243, 729，每个顶点延伸出的边数分别为$\log_5 N$、$\log_7 N$（取上整）。信息以邻接矩阵的格式存放在input11.txt，input12.txt，...，input42.txt中。相邻数据之间用逗号分割。
      - 边权值的范围是[-100, 1000]的非0整数，0代表没有边相连。图中**不存在负环**。

      ex1/output/
      - result.txt：输出**以0号节点为起点**，到所有其他节点的最短路径，包含结点序列及路径长，不同规模写到不同的txt文件中。因此共有8个txt文件，文件名称为result11.txt, result12.txt,……, result42.txt；**文件格式：**每行存一结点对的最短路径，格式为：i,j,w;i,u1,u2,...,j 。即每行先输出节点对序号和路径总长度，用分号分隔后输出完整路径（此处i恒为0）。若有从0号节点出发不可达到的节点，则不需要输出在文件中。
      - time.txt：运行时间效率的数据，不同规模的时间都写到同一个文件。
- 实验3.2 Johnson算法
    - 内容

      实现求所有点对最短路径的Johnson算法。输入数据与实验3.1相同。图的输入规模总共有4*2=8个，统计算法所需运行时间，画出时间曲线，分析程序性能。
    - 要求

- input/
  - **输入与上个实验相同。**输入图是有向稀疏图，顶点数N分别为27, 81, 243, 729，每个顶点延伸出的边数分别为$\log_5 N$、$\log_7 N$（取上整）。信息以邻接矩阵的格式存放在input11.txt，input12.txt，...，input42.txt中。相邻数据之间用逗号分割。
  - 边权值的范围是[-100, 1000]的非0整数，0代表没有边相连。图中**不存在负环**。
- ex2/output/
  - result.txt：输出对应规模图中**所有可达点对之间**的最短路径，包含结点序列及路径长，不同规模写到不同的txt文件中。因此共有8个txt文件，文件名称为result11.txt, result12.txt,……, result42.txt；**文件格式：**输出格式为图的最短路径矩阵。具体来说，输出一个矩阵，第i行第j列的数据表示从节点i到节点j的最短路径总权值，X代表不可达。不需要输出路径信息。
  - time.txt：运行时间效率的数据，不同规模的时间都写到同一个文件。

3. 方法和步骤

Exp1

- 方法

  Bellman-Ford算法求单源最短路径，边权可以为负值，能检测负环。

  算法对图的每条边进行$|V|-1$次处理，该循环对每条边进行一次松弛操作。若循环之后还能松弛则存在负环，否则不存在负环。

- 步骤

  BellmanFord求单源最短路径

```
bool BellmanFord(int N)
{
    for(int i=0;i<=N-1;i++)
        pi[i]=-1;
    for(int i=1;i<=N-1;i++)
    {
        for(auto edge:edge_set)
        {
            if(d[edge->v]>d[edge->u]+edge->weight)
            {
                d[edge->v]=d[edge->u]+edge->weight;
                pi[edge->v]=edge->u;
            }
        }
    }
    for(auto edge:edge_set)
    {
        if(d[edge->v]>d[edge->u]+edge->weight)
        {
            return false;
        }
    }
    return true;
}
```

  ShortestPATH递归求出最短路径

```
int ShortestPATH(int path_k)
{
    int node=path[path_k];
```

```
    if(node==0)
    {
        return path_k;
    }
    if(pi[node]==-1)
    {
        return 0;
    }
    else
    {
        path_k--;
        path[path_k]=pi[node];
        return ShortestPATH(path_k);
    }
}
```

Exp2

- 方法

  Johnson算法求所有结点对之间的最短路径。

  先通过Bellman-Ford算法求出扩展了s结点的图的关于s的单源最短距离h，再由h重新赋予权重。新权重为 `new_w(u,v)=w(u,v)+h(u)-h(v)`，该权重赋予机制有保持**最短路径不变**和**非负**两大性质。

  再由dijkstra求出更新新权重后图中的最短路径。

- 步骤（关键代码解析）

  扩展s结点

```
for(int i=0;i<N[T];i++)
{
    Edge* new_edge=new Edge;
    new_edge->u=N[T];
    new_edge->v=i;
    new_edge->weight=0;
    edge_set.insert(new_edge);
    h[i]=0;
}
h[N[T]]=0;
```

  对s作Bellman-Ford求最短路长度h

```
bool BellmanFord(int N)
{
    for(int i=1;i<=N-1;i++)
    {
        for(auto edge:edge_set)
        {
            if(h[edge->v]>h[edge->u]+edge->weight)
            {
                h[edge->v]=h[edge->u]+edge->weight;
            }
        }
    }
    for(auto edge:edge_set)
    {
```

```
        if(h[edge->v]>h[edge->u]+edge->weight)
        {
            return false;
        }
    }
    return true;
}
```

重新赋予权值

```
for(auto edge:edge_set)
{
    NodeEdge* node_edge=new NodeEdge;
    node_edge->adj=edge->v;
    node_edge->weight=edge->weight+h[edge->u]-h[edge->v];
    node[edge->u].adj_set.insert(node_edge);
}
```

dijsktra求单源最短路

```
void Dijkstra(int k,int N)
{
    priority_queue<pii,vector<pii>,greater<pii>> q;
    bool done[800];
    for(int i=0;i<N;i++)
    {
        dis[i]=INF;
        done[i]=false;
    }
    dis[k]=0;
    q.push(make_pair(dis[k],k));
    while(!q.empty())
    {
        auto item=q.top();
        q.pop();
        int u=item.second;
        if(done[u])continue;
        done[u]=true;
        for(auto edge:node[u].adj_set)
        {
            int v=edge->adj;
            if(done[v]==false&&dis[v]>dis[u]+edge->weight)
            {
                dis[v]=dis[u]+edge->weight;
                q.push(make_pair(dis[v],v));
            }
        }
    }
}
```

复原最短路径长度

```
int D=dis[j]+h[j]-h[i];
```

4. 结果和分析

Exp1

- 结果

  顶点0的单源最短路如result文件中所示，e.g. `result11.txt`如下：

```
0,1,726;0,22,10,21,20,1
0,2,295;0,22,10,2
0,4,415;0,22,10,2,4
0,5,719;0,22,10,2,12,15,5
0,6,1578;0,22,23,19,6
0,7,1351;0,22,10,17,9,7
0,9,1267;0,22,10,17,9
0,10,169;0,22,10
0,12,763;0,22,10,2,12
0,13,572;0,13
0,14,679;0,22,23,19,18,14
0,15,800;0,22,10,2,12,15
0,16,989;0,22,10,21,16
0,17,618;0,22,10,17
0,18,653;0,22,23,19,18
0,19,655;0,22,23,19
0,20,769;0,22,10,21,20
0,21,235;0,22,10,21
0,22,29;0,22
0,23,575;0,22,23
0,24,787;0,13,26,24
0,25,1163;0,22,10,21,20,1,25
0,26,768;0,13,26
```

- 时间
  - 理论时间复杂度分析：

    初始化时间为$O(V)$。

    第一个for循环，外层循环$(V-1)$次，内层循环E次，每次松弛操作时间为$O(1)$，故时间为$O(VE)$。

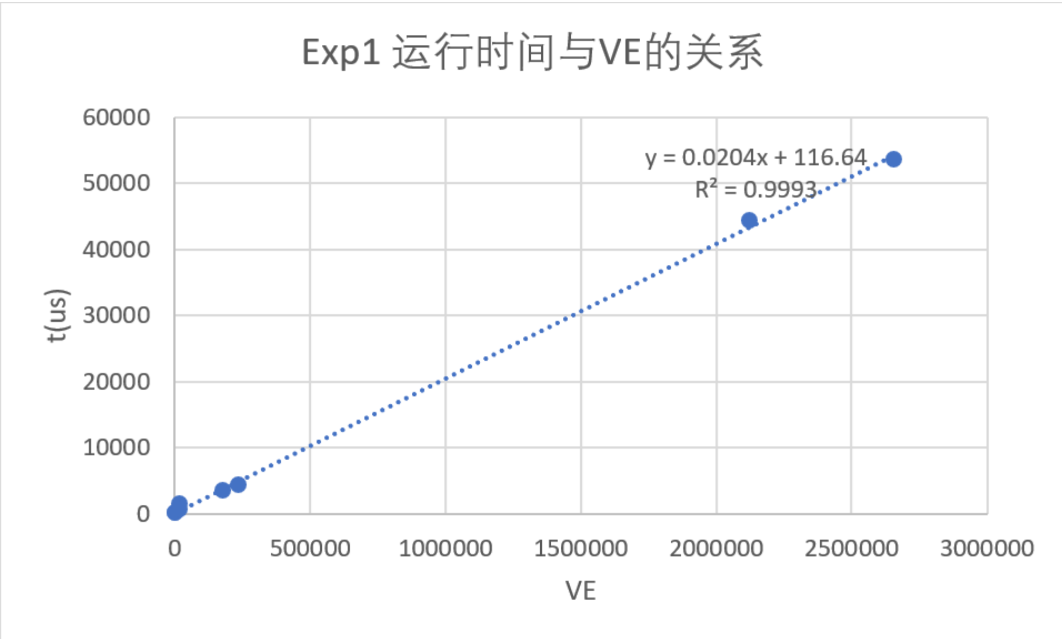
    第二个for循环E次，每次操作时间为$O(1)$故时间为$O(E)$。

    总时间为$O(VE)$。

- 实际运行时间：

```
42us
25.7us
1387.8us
537.1us
4330.7us
3430.9us
53634.9us
44401.6us
```

如下图分析知运行时间与VE呈线性关系，拟合优度$R^2 = 0.9993$。



Exp2

- 结果

  所有顶点对的最短距离如result文件中所示，e.g. `result11.txt`如下：

```
0 726 295 X 415 719 1578 1351 X 1267 169 X 763 572 679 800 989 6
377 0 227 X 347 651 1308 931 X 847 546 X 695 949 409 732 521 445
530 709 0 X 120 424 1574 1400 X 1316 342 X 468 875 562 505 990 7
119 253 139 0 259 -31 1119 528 X 861 288 X 13 691 220 50 535 698
410 589 356 X 0 780 1639 1412 X 1328 230 X 824 755 442 861 1050
491 879 170 X 290 0 1422 976 X 892 512 X 638 1045 523 675 566 96
841 571 798 X 918 1014 0 1070 X 986 1005 X 1266 1413 873 1303 66
1094 717 765 X 885 1189 742 0 X 1564 1107 X 1233 1640 1126 1270
1023 801 807 X 927 1231 826 84 0 906 681 X 1275 1206 1055 1312 1
117 801 412 X 532 758 826 84 X 0 286 X 880 689 149 868 261 735 1
601 557 126 X 246 550 1532 1182 X 1098 0 X 594 1001 633 631 820
729 1000 1024 X 1144 1370 429 922 X 838 898 0 1492 1301 761 1529
106 832 126 X 246 -44 1106 932 X 848 275 X 0 678 207 37 522 724
307 1033 602 X 722 480 1693 1306 X 1222 476 X 1070 0 339 1107 89
-32 694 263 X 383 687 1546 1319 X 1235 137 X 731 540 0 768 957 5
69 795 89 X 209 -81 1069 895 X 811 238 X 557 641 170 0 485 687 1
443 902 471 X 591 895 1152 410 X 326 345 X 939 1015 475 976 0 79
368 1094 663 X 783 1009 1475 733 X 649 537 X 1131 940 400 1168 9
-6 720 289 X 409 635 1380 1345 X 1261 163 X 757 566 26 794 983 6
-8 718 287 X 407 633 923 1343 X 1259 161 X 755 564 24 792 981 61
334 -43 184 X 304 608 1265 888 X 804 503 X 652 906 366 689 478 4
535 491 718 X 838 721 1466 1164 X 1080 704 X 1186 1107 567 1223
618 697 266 X 386 690 1549 1322 X 1238 140 X 734 665 650 771 960
72 798 367 X 487 258 1003 1234 X 1150 241 X 835 644 104 872 824
92 818 387 X 507 265 1478 1241 X 1157 261 X 855 664 124 892 831
-60 666 235 X 355 581 871 494 X 410 109 X 703 512 -28 740 84 558
```

- 时间
  - 理论时间复杂度分析：

    添加s结点时经历V次循环，每次操作时间$O(1)$，总时间为$O(V)$。

    Bellman-Ford计算h的运行时间为$O(VE)$。

    更新权重时经历E次循环，每次操作时间为$O(1)$，总时间为$O(E)$。

    最后一个循环V次，每次$Dijkstra$算法运行时间为$O((V+E)lgV)$（二叉堆实现最小优先队列），然后对每个端点计算最短路的总时间为$O(V)$，该循环总时间为$O(V(V+E)lgV)$。

    总时间为$O(V(V+E)lgV)$。
  - 实际运行时间：

```
1108.3us
751us
7154.8us
7616.3us
75361.1us
59893.2us
913274us
734793us
```

由下图知运行时间与V(V+E)lgV呈线性关系，拟合优度为$R^2=0.9995$。