算法基础实验三报告

> PB19051035周佳豪

实验设备和环境

C++11

实验内容

Bellman-Ford算法

- 图的数据结构

  ```
  //弧的结构
  typedef struct Edge
  {
      int h, t; //该边的源结点与尾结点编号
      int w;    //弧的权值
  } Edge;

  //顶点的结构
  typedef struct Vertex
  {
      int id;     //顶点编号
      int degree; //顶点的出度
      int d;
      struct Vertex *pi;
      vector<Edge *> edges; //以该点为起始点的边的集合
  } Vertex;

  typedef struct
  {
      map<int, Vertex *> vertexs; //id and address
      int vexnum, edgenum;        //图的当前顶点数和弧数
      map<pair<int, int>, Edge *> edges;
  } Graph;
  ```

  主要包括边、顶点与图，图中记录了每条边和顶点的信息。

- 图的ADT，仅包含图的初始化，插入顶点和边，释放空间（删除顶点和边这个实验用不到）

  ```
  Graph *new_graph()
  {
      Graph *graph = new Graph();
      graph->vertexs.clear();
      graph->vexnum = graph->edgenum = 0;
      graph->edges.clear();
      return graph;
  }

  Vertex *createVertex(Graph *G, int id)
  {
      Vertex *vertex = new Vertex();
  ```

```
    vertex->id = id;
    vertex->degree = 0;
    vertex->pi = nullptr;
    vertex->edges.clear();
    G->vertexs[id] = vertex;
    G->vexnum++;
}
Edge *createEdge(Graph *G, int h_id, int t_id, int w)
{
    Edge *edge = new Edge();
    edge->h = h_id;
    edge->t = t_id;
    edge->w = w;
    pair<int, int> p(h_id, t_id);
    G->edges[p] = edge;
    G->edgenum++;
    Vertex *h_vertex = G->vertexs[h_id];
    h_vertex->degree++;
    h_vertex->edges.push_back(edge);
}

void addVertexsAndEdges(Graph *G, int id, vector<int> w)
{
    for (int i = 0; i < w.size(); i++)
    {
        if (w[i] != 0)
        {
            createEdge(G, id, i, w[i]);
        }
    }
}

void destroy(Graph *G)
{
    for (auto i = 0; i < G->vexnum; i++)
    {
        delete G->vertexs[i];
    }
    for (auto iter = G->edges.begin(); iter != G->edges.end(); iter++)
    {
        delete iter->second;
    }
    delete G;
}
```

- 对input文件的读，并创建相应的图

```
    inFile.open(inPath);
    resultFile.open(resultPath);
    timeFile.open(timePath);
    Graph *G = new_graph();
    int N = 27;
    for (int i = 0; i < N; i++)
    {
        createVertex(G, i);
    }
```

```
    row = 0;
    while (getline(inFile, buffer))
    {
        row++;
        vector<int> w;
        w.clear();
        stringstream line(buffer);
        string temp;
        while (getline(line, temp, ','))
        {
            w.push_back(stoi(temp));
        }
        addVertexsAndEdges(G, row - 1, w);
    }
```

- Bellman-ford算法

```
void initialize_single_source(Graph *G, int s)
{
    for (int i = 0; i < G->vexnum; i++)
    {
        auto v = G->vertexs[i];
        v->d = max_weight;
        v->pi = nullptr;
    }
    G->vertexs[s]->d = 0;
}
void relax(Vertex *u, Vertex *v, int w)
{
    if (v->d > u->d + w)
    {
        v->d = u->d + w;
        v->pi = u;
    }
}

bool bellman_ford(Graph *G, int s)
{
    initialize_single_source(G, s);
    for (int i = 1; i < G->vexnum; i++)
    {
        for (auto iter = G->edges.begin(); iter != G->edges.end(); iter++)
        {
            Vertex *u = G->vertexs[iter->first.first];
            Vertex *v = G->vertexs[iter->first.second];
            int w = iter->second->w;
            relax(u, v, w);
        }
    }
    for (auto iter = G->edges.begin(); iter != G->edges.end(); iter++)
    {
        Vertex *u = G->vertexs[iter->first.first];
        Vertex *v = G->vertexs[iter->first.second];
        int w = iter->second->w;
        if (v->d > u->d + w)
            return false;
    }
```

```
    return true;
}
```

Johnson算法

图的数据结构以及ADT、文件的读写以及图的创建与Bellman-Ford算法相同，这里不再赘述。

- Dijkstra算法

```
void dijkstra(Graph *G, int s)
{
    initialize_single_source(G, s);
    list<int> S;
    list<int> Q;
    list<int>::iterator q_iter;
    list<int>::iterator iter;
    for (int i = 0; i < G->vexnum - 1; i++)
        Q.push_back(i);
    iter = find(Q.begin(), Q.end(), s);
    if (iter != Q.end())
    {
        Q.erase(iter);
    }
    S.push_back(s);
    int start = 0;
    while (!Q.empty())
    {
        if (start == 0)
        {
            Vertex *u = G->vertexs[s];
            for (auto edge : u->edges)
            {
                Vertex *v = G->vertexs[edge->t];
                relax(u, v, edge->w);
            }
            start = 1;
        }
        else
        {
            Vertex *temp = G->vertexs[*Q.begin()];
            int min_temp = G->vertexs[*Q.begin()]->d;
            q_iter = Q.begin();
            for (iter = Q.begin(); iter != Q.end(); iter++)
            {
                if (min_temp > G->vertexs[*iter]->d)
                {
                    min_temp = G->vertexs[*iter]->d;
                    temp = G->vertexs[*iter];
                    q_iter = iter;
                }
            }
            if (q_iter != Q.end())
            {
                Q.erase(q_iter);
            }
            S.push_back(temp->id);
            for (auto edge : temp->edges)
```

```
            {
                Vertex *v = G->vertexs[edge->t];
                relax(temp, v, edge->w);
            }
        }
    }
}
```

其中采用list容器表示S与Q，未使用优先队列，采用遍历容器Q选择d最小的那个顶点，将其从Q中删去，再加入S。

- johnson算法

```
void johnson(Graph *G, map<pair<int, int>, int> &D)
{
    map<int, int> h;
    int N = G->vexnum;
    createVertex(G, N);
    for (int i = 0; i < N; i++)
    {
        createEdge(G, N, i, 0);
    }
    if (!bellman_ford(G, N))
        cout << "the input graph contains a negative-weight cycle" << endl;
    else
    {
        for (int i = 0; i < G->vexnum; i++)
        {
            Vertex *v = G->vertexs[i];
            h[v->id] = v->d;
        }
        for (auto iter = G->edges.begin(); iter != G->edges.end(); iter++)
        {
            int h_id = iter->first.first;
            int t_id = iter->first.second;
            G->edges[make_pair(h_id, t_id)]->w += (h[h_id] - h[t_id]);
        }
        for (int i = 0; i < N; i++)
        {
            Vertex *u = G->vertexs[i];
            dijkstra(G, u->id);

            for (int j = 0; j < N; j++)
            {
                D[make_pair(u->id, j)] = G->vertexs[j]->d + h[j] - h[u->id];
            }
        }
    }
}
```

实验结果与分析

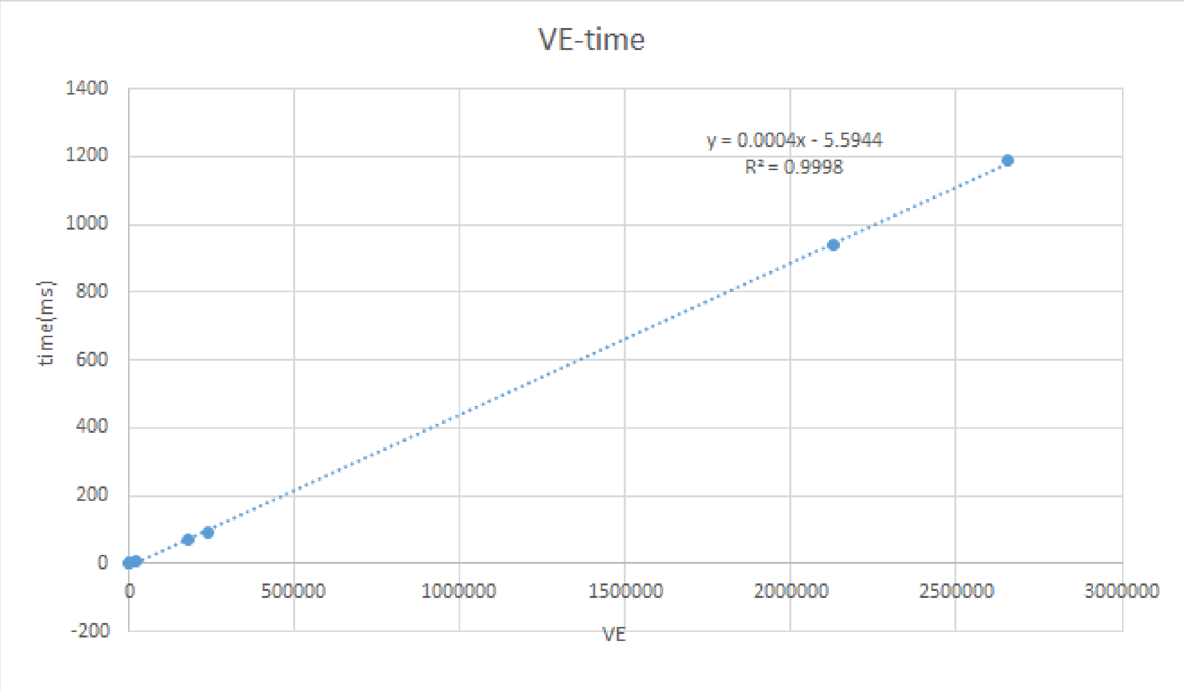
Bellman-Ford

运行10次程序得到相应的time，最后求得平均

总顶点数	每个顶点延伸出的边数	VE	time（ms）
27	3	2187	0.5756
27	2	1458	0.3837
81	3	19683	6.4559
81	3	19683	6.5184
243	4	236196	90.973
243	3	177147	69.4686
729	5	2657205	1190.36
729	4	2125764	937.792

由于VE=19683对应于两个time，取二者的平均值得6.48715ms

拟合结果为：



可见time和VE呈现很好的线性关系，验证了Bellman-Ford算法的总运行时间为(VE)

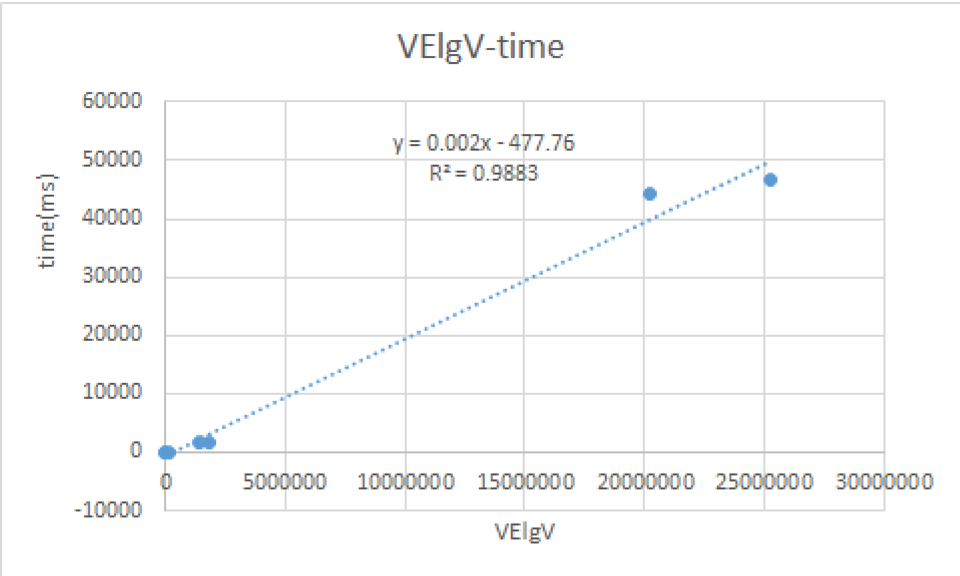
Johnson

运行十次程序得到相应的time，最后取平均

V	E	VElgV	time（ms）
27	81	10399	4.1108
27	54	6933	3.8212
81	243	124787	69.3819
81	243	124787	73.1004
243	972	1871809	1701.53
243	729	1403857	1635.78
729	3645	25269422	46688
729	2916	20215537	44298

由于VElgV=124787时，time有两个值，取二者的平均得71.24115

拟合得：



可见time与VElgV呈线性关系，验证了算法的运行时间为O(V^2 lgV + VE)