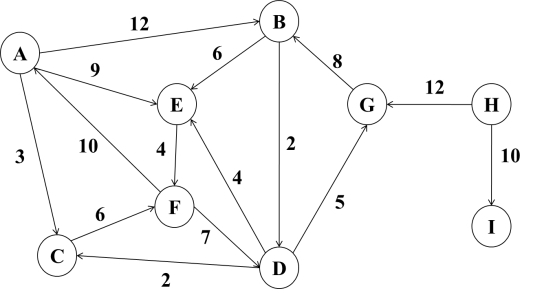
**姓名: 张景耀 学号: 3200602123 班级: 计算机2005**

# Lab4 图实验

**本次实验测试数据采用样例图见下:**

****

**注: 所有数据可在样例图上进行验证**

**建图代码(Graph.cpp文件):**

#include<bits/stdc++.h>

using namespace std;

struct Edge

{

    char idx;

    int weight;

    Edge\* next;

};

struct Vertex{

    char idx;

    Edge\* head;

};

unordered\_map<char, Vertex\*> graph;

unordered\_map<char, bool> vis;

void init()

{

    vis.clear();

    auto A = new Vertex({'A', NULL});

    auto B = new Vertex({'B', NULL});

    auto C = new Vertex({'C', NULL});

    auto D = new Vertex({'D', NULL});

    auto E = new Vertex({'E', NULL});

    auto F = new Vertex({'F', NULL});

    auto G = new Vertex({'G', NULL});

    auto H = new Vertex({'H', NULL});

    auto I = new Vertex({'I', NULL});

    A->head = new Edge({'B', 12, new Edge({'E', 9, new Edge({'C', 3, NULL})})});

    B->head = new Edge({'E', 6, new Edge({'D', 2, NULL})});

    C->head = new Edge({'F', 6, NULL});

    D->head = new Edge({'C', 2, new Edge({'E', 4, new Edge({'G', 5, NULL})})});

    E->head = new Edge({'F', 4, NULL});

    F->head = new Edge({'A', 10, new Edge({'D', 7, NULL})});

    G->head = new Edge({'B', 8, NULL});

    H->head = new Edge({'G', 12, new Edge({'I', 10, NULL})});

    graph['A'] = A;

    graph['B'] = B;

    graph['C'] = C;

    graph['D'] = D;

    graph['E'] = E;

    graph['F'] = F;

    graph['G'] = G;

    graph['H'] = H;

    graph['I'] = I;

}

### 实验内容1. 基于图的深度优先搜索策略写一个算法，判别以邻接表方式存储的有向图中是否存在由顶点vi到顶点vj的路径（i != j)

#### 源代码

#include "Graph.cpp"

char ed;

bool dfs(char st)

{

    vis[st] = true;

    if(vis[ed])

        return true;

    for(auto edge = graph[st]->head; edge; edge = edge->next)

    {

        if(vis[edge->idx])

            continue;

        if(dfs(edge->idx))

            return true;

    }

    return false;

}

int main(int argc, char const \*argv[])

{

    init();

    char st;

    cout << "Please Input start and end idx: \n";

    cin >> st >> ed;

    if(st == ed)

    {

        cout << "Self-Loop!" << endl;

        return 0;

    }

    if(dfs(st))

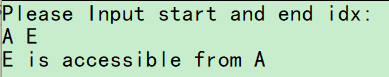
        cout << ed << " is accessible from " << st << endl;

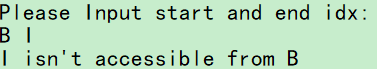
    else

        cout << ed << " isn't accessible from " << st << endl;

}

#### 测试结果





### 实验内容2. 基于图的广度优先搜索策略写一个算法，判别以邻接表方式存储的有向图中是否存在由顶点vi到顶点vj的路径（i != j):

#### 源代码

#include "Graph.cpp"

char ed;

bool bfs(char st)

{

    queue<char> q;

    q.push(st);

    while(q.size())

    {

        auto t = q.front();

        q.pop();

        vis[t] = true;

        if(vis[ed])

            return true;

        for(auto edge = graph[t]->head; edge; edge = edge->next)

        {

            if(vis[edge->idx])

                continue;

            q.push(edge->idx);

        }

    }

    return false;

}

int main(int argc, char const \*argv[])

{

    init();

    char st;

    cout << "Please Input start and end idx: \n";

    cin >> st >> ed;

    if(st == ed)

    {

        cout << "Self-Loop!" << endl;

        return 0;

    }

    if(bfs(st))

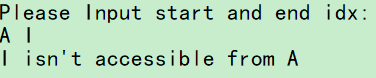
        cout << ed << " is accessible from " << st << endl;

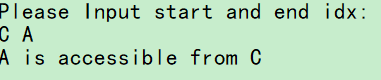
    else

        cout << ed << " isn't accessible from " << st << endl;

}

#### 测试结果





### 实验内容3. 以邻接表为存储结构实现从源点到其余各顶点的最短路径的Dijkstra算法，要求输出最短路径及其长度:

#### 源代码

#include "Graph.cpp"

char st;

int n;

unordered\_map<char, long long> dist;

unordered\_map<char, vector<char>> path;

unordered\_map<char, char> pre;

void printPath(char ed)

{

    if(dist[ed] == INT\_MAX)

    {

        cout << ed << " isn't accessible from " << st << "\n\n";

        return;

    }

    printf("The Shortest Path(from %c to %c) is :\n", st, ed);

    stack<char> stk;

    auto t = ed;

    while(t != '$')

        stk.push(t), t = pre[t];

    cout << "    ";

    while(stk.size())

    {

        cout << stk.top() << " ", stk.pop();

        if(stk.size())

            printf("-> ");

    }

    puts("");

    printf("The Val of this Path is : %d\n", dist[ed]);

    puts("");

}

void Dijkstra()

{

    for(int i = 0; i < n; i++)

        dist['A' + i] = INT\_MAX, pre['A' + i] = '$';

    dist[st] = 0;

    for(int i = 0; i < n; i++)

    {

        char t = '$';

        for(const auto& p : graph)

            if(!vis[p.first] and (t == '$' or dist[p.first] < dist[t]))

                t = p.first;

        vis[t] = true;

        for(auto edge = graph[t]->head; edge; edge = edge->next)

            if(!vis[edge->idx] and dist[edge->idx] > dist[t] + edge->weight)

                dist[edge->idx] = dist[t] + edge->weight, pre[edge->idx] = t;

    }

    for(int i = 0; i < n; i++)

    {

        char ed = 'A' + i;

        printPath(ed);

    }

}

int main(int argc, char const \*argv[])

{

    init();

    n = graph.size();

    printf("Please Input the index of Source vertex\n");

    cin >> st;

    Dijkstra();

    return 0;

}

#### 测试结果

