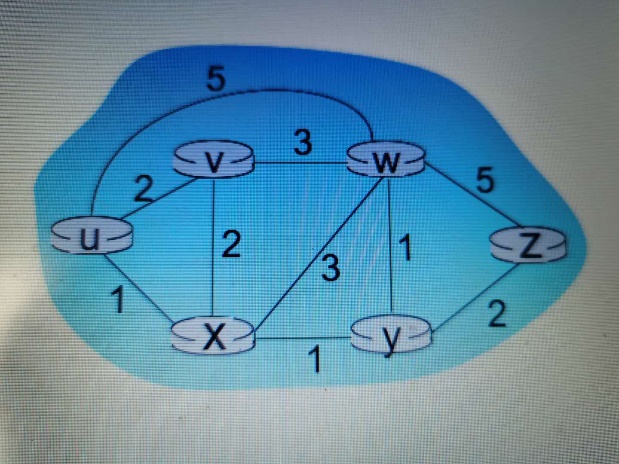
**1. 路由器相互之间的连接与链路权值如下图，计算从X路由器出发到其他路由器的最短通路（要求给出详细计算过程），X路由器为根的最短通路树。给出X路由器的路由表。**



第一步：将数据初始化，令N集合为已经求得的点（初始为‘x’），U集合为点集S-N，C(x,y)为两个点x,y的路径，如果不能连通就记为∞（“无穷”）。D(x)记为x到点‘x’的距离，Dway(x)记为x到点‘x’的所走路径。

第二步：选取U中点min(D(p))的点记为p，然后将p压入N中,重新计算D(x)和Dway(x)

重复第二步直到U为空集。

下面给出此题解答，图说明：**比如x到w 路径为xyw 代价为2。**

c++代码见末尾附件。



**求解步骤：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Step | N’ | u | v | w | y | z |
| 0 | x | 1 | 2 | 3 | 1 | ∞ |
| 1 | xu |  | 2 | 3 | 1 | ∞ |
| 2 | xuy |  | 2 | 2 |  | 3 |
| 3 | xuyv |  |  | 2 |  | 3 |
| 4 | xuyvw |  |  |  |  | 3 |
| 5 | xuyvwz |  |  |  |  |  |



**路由表**

|  |  |  |
| --- | --- | --- |
| 目的地（x出发） | 下一跳 | Cost |
| Ru | Ru | 1 |
| Ry | Ry | 1 |
| Rv | Rv | 2 |
| Rw | Ry | 2 |
| Rz | Ry | 3 |

**简化**

|  |  |  |
| --- | --- | --- |
| 目的地（x出发） | 下一跳 | Cost |
| Ru | Ru | 1 |
| Rv | Rv | 2 |
| \* | Ry | … |

**2.讨论Dijsktra's算法的复杂性以及存在的问题?**

**复杂性：**在第一次迭代中，我们需要搜索所有的n个节点以确定出不在M中且具有最低开销的节点在第二次迭代时，我们需要检查n-1个节点以确定最低开销。第三次对n-2个节点迭代，依次类推。总之，我们在所有迭代中需要搜寻的节点总数为n(n+1)/2,因此我们说前面实现的链路状态算法在最差情况下复杂性为O(n^2)。

**存在问题：**

1.算法时间复杂度较高，当路由器过多，寻找路径的算法复杂度过高。

2.可能产生“振荡”。算法存在隐患，可能导致拥塞。无全局平衡功能。（因为cost为动态的）

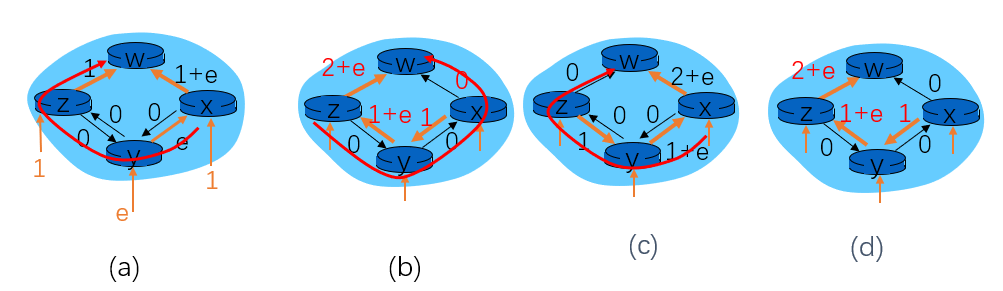
具体如下：

初始：图(a)，其链路费用相当于承载的流量

第一次：根据LS算法计算后，节点x、y都确定顺时针方向到w的路径费用最低，为1。产生图(b)费用

第二次：根据LS算法计算后，节点x、y、z都确定逆时针方向到w的路径费用最低，为0。产生图(c)费用

第三次：根据LS算法计算后，节点x、y、z都确定顺时针方向到w的路径费用最低，为0。产生图(d)费用



**3. 网络拥塞发生的条件与拥塞控制触发的条件有和不同？**

**网络拥塞**发生的条件是“路由器无法处理当前任务”

1. 多条流入线路有分组到达，并需要同一输出线路，此时，如果路由器没有足够的内存来存放所有这些分组，那么有的分组就会丢失。
2. 路由器处理器的缘故，以至于难以完成必要的处理工作，如缓冲区排队、更新路由表等。

**拥塞控制**是一个**全局性的过程**，防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。它是一种面临网络拥塞时遏制发送方的一些机制。

1. 端到端拥塞控制：通过对网络行为的观察（如分组丢失与时延）来推断之，如TCP采用端到端的方法解决拥塞，因为IP层不断反馈有关网络拥塞的信息。当触发“网络拥塞迹象”（如超时或3次冗余等）即触发拥塞控制
2. 网络辅助的拥塞控制：路由器向发送方提供关于网络中拥塞状态的显式反馈信息。这种反馈可以简单地使用一个bit来指示链路的拥塞情况。当得到“网络拥塞”的信息，即触发拥塞控制。

第一题附录代码：

#include <iostream>

#include <set>

#include <map>

#include <string>

using namespace std;

const unsigned MAXCNT = 0x00ffffff;

set<char> N;    //目前有的点

set<char> U;    //pArr去除N的点

map<string, unsigned> C;             //C(x1x2,value)

map<char, unsigned> D;               //D(v) x到v目前最短路径

map<char, string> Dway;

const char pArr[] = {'x', 'u', 'v', 'w', 'y', 'z'};

void setC(char c1, char c2, unsigned value);            //设置C(x1,x2)的值

unsigned getC(char c1, char c2);                       //得到C(x1,x2)的值

void setD(char c, unsigned value);            //设置D(y)的值

unsigned getD(char c);                       //得到D(y)的值

void setDway(char c, string value);            //设置Dway(y)的值

string getDway(char c);                       //得到Dway(y)的值

char getMinc();     //从U中拿出到N最小的点 放入N中

void init();        //初始化

void reCal(char c);        //重新计算D(c)

int main() {

    init();

    while (!U.empty()) {

        getMinc();

    }

    for (auto i:N) {

        cout <<"x--"<< i <<endl<< "value: " << getD(i) <<"  ||  "<< "way: " << getDway(i) << endl;

        cout<<"---------------------------"<<endl;

    }

    return 0;

}

void reCal(char c) {

    unsigned minn = getD(c);

    char ind = NULL;

    for (auto i:N) {

        unsigned tmp = getD(i) + getC(i, c);

        if (tmp < minn) {

            minn = tmp;

            ind = i;

        }

    }

    if (ind != NULL) {

        string str1 = getDway(ind);

        str1.push\_back(c);

        setDway(c, str1);

        setD(c, minn);

    }

}

char getMinc() {

    char x = NULL;

    unsigned minc = MAXCNT;

    for (auto i:U) {

        for (auto j:N) {

            unsigned tmp = getC(i, j);

            if (minc > tmp) {

                minc = tmp;

                x = i;

            }

        }

    }

    reCal(x);

    N.insert(x);

    U.erase(x);

    return x;

}

string getDway(char c) {

    return (Dway.find(c))->second;

}

void setDway(char c, string value) {

    auto iter = Dway.find(c);

    if (iter != Dway.end()) {

        iter->second = value;

    } else {

        Dway.insert(pair<char, string>(c, value));

    }

}

unsigned getC(char c1, char c2) {

    string tmp;

    tmp.push\_back(c1);

    tmp.push\_back(c2);

    return (C.find(tmp))->second;

}

void setC(char c1, char c2, unsigned value) {

    string tmp;

    tmp.push\_back(c1);

    tmp.push\_back(c2);

    C.insert(pair<string, unsigned>(tmp, value));

    tmp.clear();

    tmp.push\_back(c2);

    tmp.push\_back(c1);

    C.insert(pair<string, unsigned>(tmp, value));

}

unsigned getD(char c) {

    return (D.find(c))->second;

}

void setD(char c, unsigned value) {

    auto iter = D.find(c);

    if (iter != D.end()) {

        iter->second = value;

    } else {

        D.insert(pair<char, unsigned>(c, value));

    }

}

void init() {

    N.insert('x');

    for (auto i:pArr) {

        U.insert(i);

        setD(i, MAXCNT);

        string tmp = "";

        tmp.push\_back(i);

        tmp.push\_back(i);

        C.insert(pair<string, unsigned>(tmp, 0));

    }

    setD('x', 0);

    U.erase('x');

    setC('x', 'u', 1);

    setC('x', 'v', 2);

    setC('x', 'w', 3);

    setC('x', 'y', 1);

    setC('x', 'z', MAXCNT);

    setC('u', 'v', 2);

    setC('u', 'w', 5);

    setC('u', 'y', MAXCNT);

    setC('u', 'z', MAXCNT);

    setC('v', 'w', 3);

    setC('v', 'y', MAXCNT);

    setC('v', 'z', MAXCNT);

    setC('w', 'y', 1);

    setC('w', 'z', 5);

    setC('y', 'z', 2);

    for (auto i:pArr) {

        string tmp = "";

        tmp.push\_back(i);

        setDway(i, tmp);

    }

}