初步实现

1. 图结构

每个节点有组号、距离（一开始为10000，没找到则依旧为10000）、是否找到的标志位（1找到，0未找到）、其最短路径的父节点编号、以及指向以该点为起始点的边的终节点的指针。

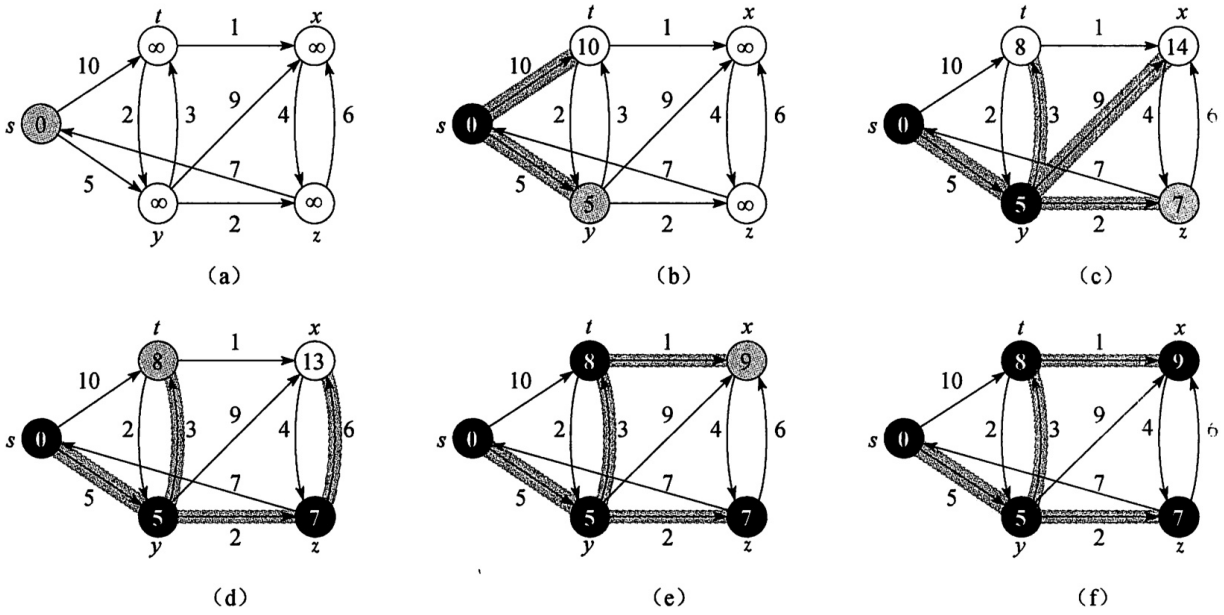
中间代表边的节点的结构：边的权重以及终节点的编号。

为了丢失信息之后的部分信息还原，增加了一个数据结构：边的权重、以及以该节点为终节点对应的起始节点的编号。

```
struct Vertex{
    int groupNum_;    // there is no need to add nodeNum for it is just the
index of array
    int distance_;
    int found;        // 1 already found shortest path, otherwise 0
    int parent;
    struct EdgeVertex *next;

    Vertex(){}
    Vertex(int groupNum, int distance):
        groupNum_(groupNum), distance_(distance){
        found = 0;
        parent = -1;
        next = NULL;
    }
};

struct EdgeVertex{
    // it is the start point of the edge
    int weight_;
    int nodeNum_;    // the index of the vertices
    struct EdgeVertex *next;

    EdgeVertex(){}
    EdgeVertex(int weight, int nodeNum):
        weight_(weight), nodeNum_(nodeNum){
        next = NULL;
    }
};

struct InverseEdge{
    // it is the end point of the edge, used to update missing info
```

```
    int weight_;
    int nodeNum_;    // the index of the vertices, if there is no inverse edge
for this point, then nodeNum_ is -1
    struct InverseEdge *next;

    InverseEdge(){}
    InverseEdge(int weight, int nodeNum):
        weight_(weight), nodeNum_(nodeNum){
        next = NULL;
    }
};
```

鉴于现在WebGraph还没有尝试成功，所以就自己设置的以上图结构，利用《算法导论》书上的下图，手动初始化图结构来运行一下算法。



Boost 中有数据结构 graph，windows下还没有安装成功，依旧在尝试中。

2. Dijkstra 算法实现

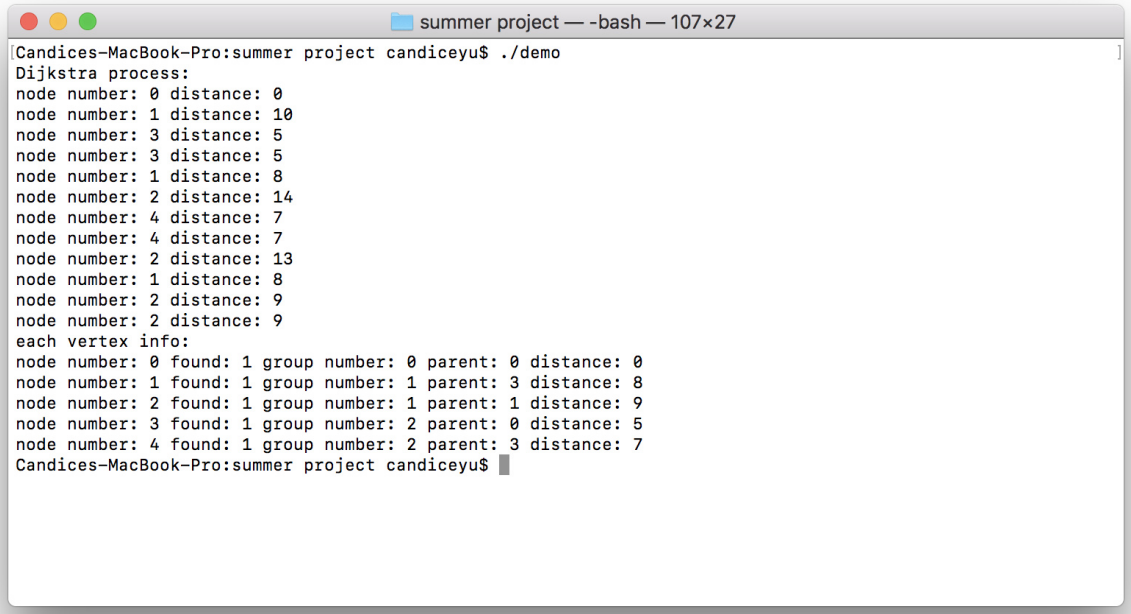
首先找到当前具有最短距离的那个点，若所有的点都不可达到，则返回负数，否则返回其编号。

标志其已找到最短路径，记录过程并松弛该节点作为起始节点的边的终节点的距离。

```
while(undo--){
        cur_ = Shortest(graph, nodeNum, 0);
        if(cur_ >= 0){
            graph->vertices[cur_].found = 1;
            // record process
            process[processNum][0] = cur_;
```

```
            process[processNum++][1] = graph->vertices[cur_].distance_;
            next_ = graph->vertices[cur_].next;

            // relax distance
            while(next_ != NULL){
                next_index = next_->nodeNum_;
                weight = next_->weight_;
                Relax(graph, cur_, next_index, weight, process, processNum);
                next_ = next_->next;
            }
        }
    }
```

运行结果：



3. 缺失信息的 Dijkstra 算法实现

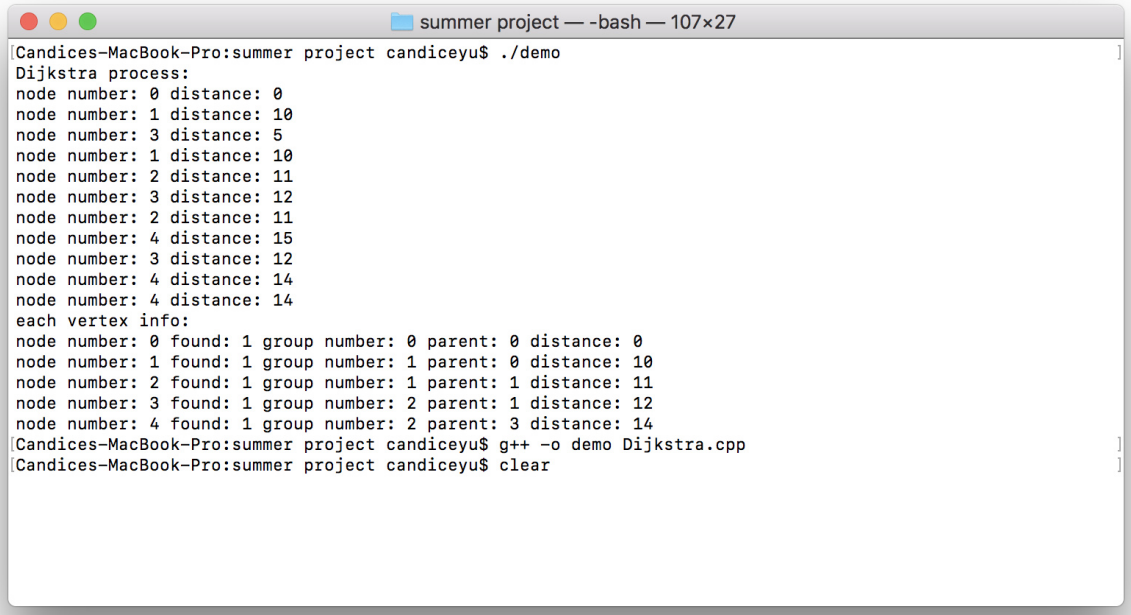
输入信息：丢失信息的组号、丢失信息的时间（迭代次数）

大致方法和上面的是一样的，增加了丢失信息的处理、还有当当前最短路径的节点是缺失信息的节点，并且丢失信息后没有其他节点对该节点进行更新，那么会利用连接该节点的起始节点去更新该节点，从而找到其最短路径。

```
while(undo--){
        // judge whether should miss group information
        if(nodeNum-undo == iter && flag){
```

```
            for(i=0; i<nodeNum; i++){
                if(graph->vertices[i].groupNum_ == missGroup){
                    // if it has already been found the shortest path, then
undo++
                    if(graph->vertices[i].found == 1)
                        undo ++;
                    // miss information
                    graph->vertices[i].distance_ = 10000;
                    graph->vertices[i].parent = -1;
                    graph->vertices[i].found = 0;
                }
            }
            flag = false;
        }

        cur_ = Shortest(graph, nodeNum, 0);
       // if it is in the missing group and still with distance 10000, then we
need to use
       // found vertices to update this vertex
        while(cur_ >= 0 && graph->vertices[cur_].distance_ == 10000 &&
            graph->vertices[cur_].groupNum_ == missGroup){
                if(first_){
                    cur_ = Shortest(graph, nodeNum, cur_+1);
                }
                UpdateMissing(graph, cur_, process, processNum);
                first_ ++;
        }

        if(cur_ >= 0){
            graph->vertices[cur_].found = 1;
            // record process
            process[processNum][0] = cur_;
            process[processNum++][1] = graph->vertices[cur_].distance_;
            next_ = graph->vertices[cur_].next;

            // relax distance
            while(next_ != NULL){
                next_index = next_->nodeNum_;
                weight = next_->weight_;
                Relax(graph, cur_, next_index, weight, process, processNum);
                next_ = next_->next;
            }
        }
    }
```

运行结果：



4. 当前问题

1. 迭代次数是指完成一次操作？还是完成一次当前节点的边所连接节点的距离更新?
2. 重新更新一个节点的丢失信息算1次迭代？还是更新所有需要顶点的丢失信息算1次迭代?
3. WebGraph 必须要用 java 写么？WebGraph++ 用 windows暂时失败了😅
4. WebGraph 中的图都很大，到时候可视化不会太大了吗？还是只是实现的时候用大图，到时候可视化截取其中图的一部分展示?