基本实现

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）、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。

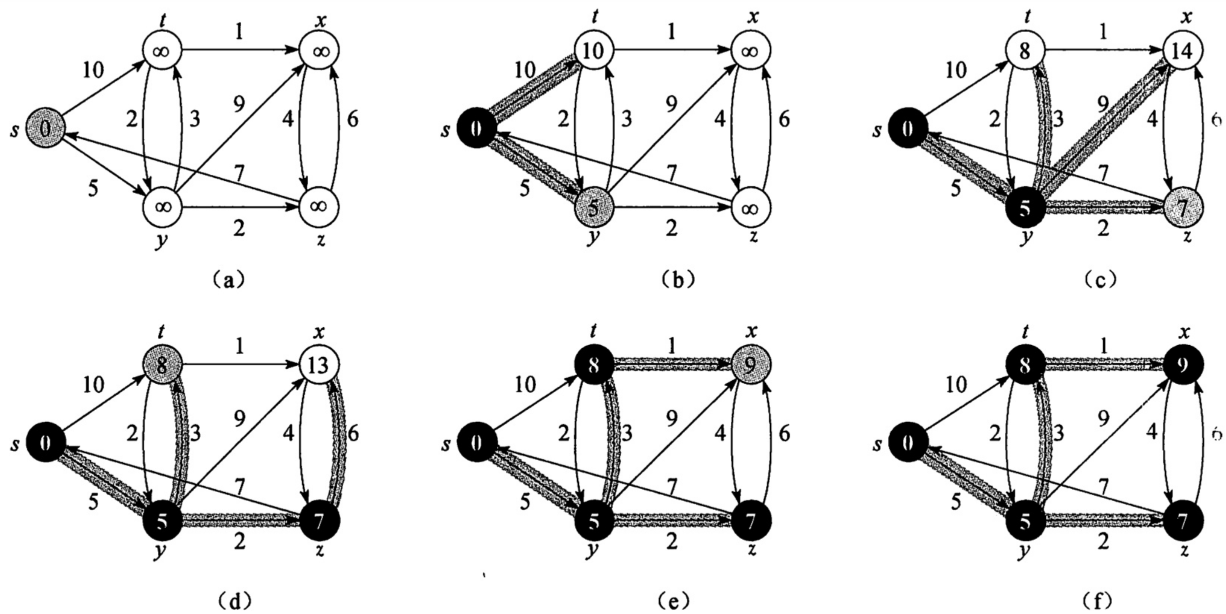
```
struct Vertex{
    int groupNum_;    // there is no need to add nodeNum for it is just the index of array
    int distance_;
    int found;        // 1 already found shortest path, otherwise 0
    int parent;
    struct EdgeVertex *next;

    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;
    }
};
```

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。



2. 基本算法实现

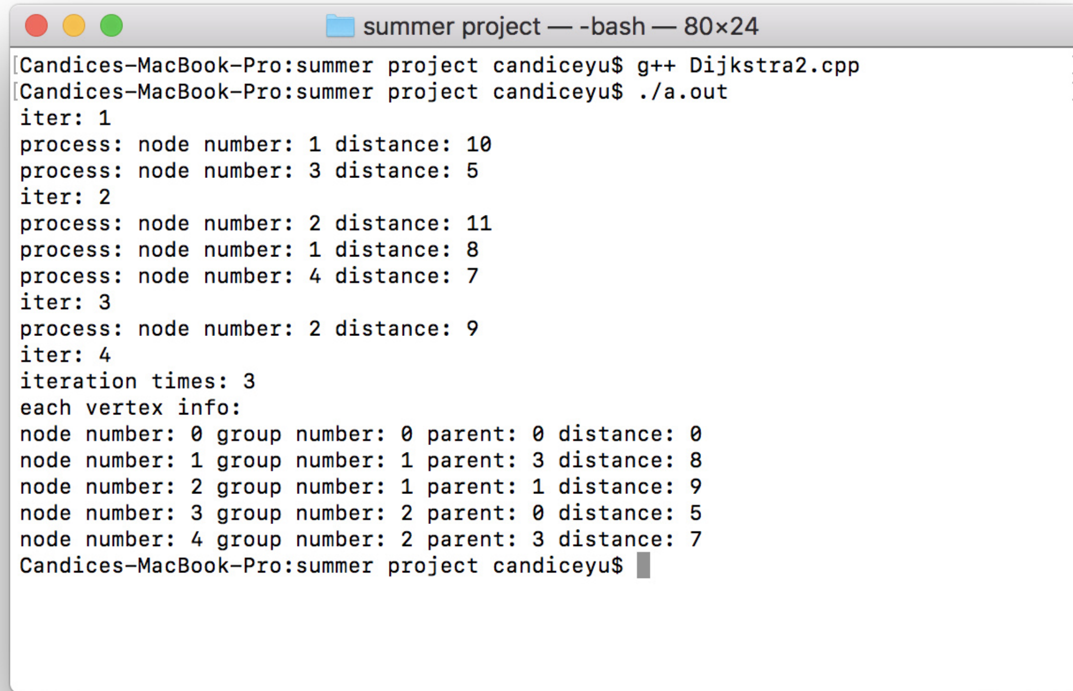
每次都按照节点的顺序，更新其边所对应的尾节点的最短路径，这样一次作为一个迭代。

每次都把过程信息先记录下来，最后再一起更新所有点的最短路径。

```
do{
        iter ++;
        cout << "iter: " << iter << endl;
        flag = false;
        memset(update, -1, sizeof(update));
        for(i=0; i<nodeNum; i++){
            Relax(graph, i, process, processNum, update);
        }

        for(i=0; i<nodeNum; i++){
            if(update[i][0] != -1){
                flag = true;
                // update info
                graph->vertices[i].distance_ = update[i][0];
                graph->vertices[i].parent = update[i][1];
            }
        }
    }while(flag);
```

运行结果：




```
Candices-MacBook-Pro:summer project candiceyu$ g++ Dijkstra2.cpp
Candices-MacBook-Pro:summer project candiceyu$ ./a.out
iter: 1
process: node number: 1 distance: 10
process: node number: 3 distance: 5
iter: 2
process: node number: 2 distance: 11
process: node number: 1 distance: 8
process: node number: 4 distance: 7
iter: 3
process: node number: 2 distance: 9
iter: 4
iteration times: 3
each vertex info:
node number: 0 group number: 0 parent: 0 distance: 0
node number: 1 group number: 1 parent: 3 distance: 8
node number: 2 group number: 1 parent: 1 distance: 9
node number: 3 group number: 2 parent: 0 distance: 5
node number: 4 group number: 2 parent: 3 distance: 7
Candices-MacBook-Pro:summer project candiceyu$
```

3. 缺失信息的算法实现

输入参数：丢失信息的组号、丢失信息的时间（迭代次数）

方法和2中是一样的，唯一的区别在于当迭代次数是丢失信息的时间点时，将组号符合条件的节点信息初始化。

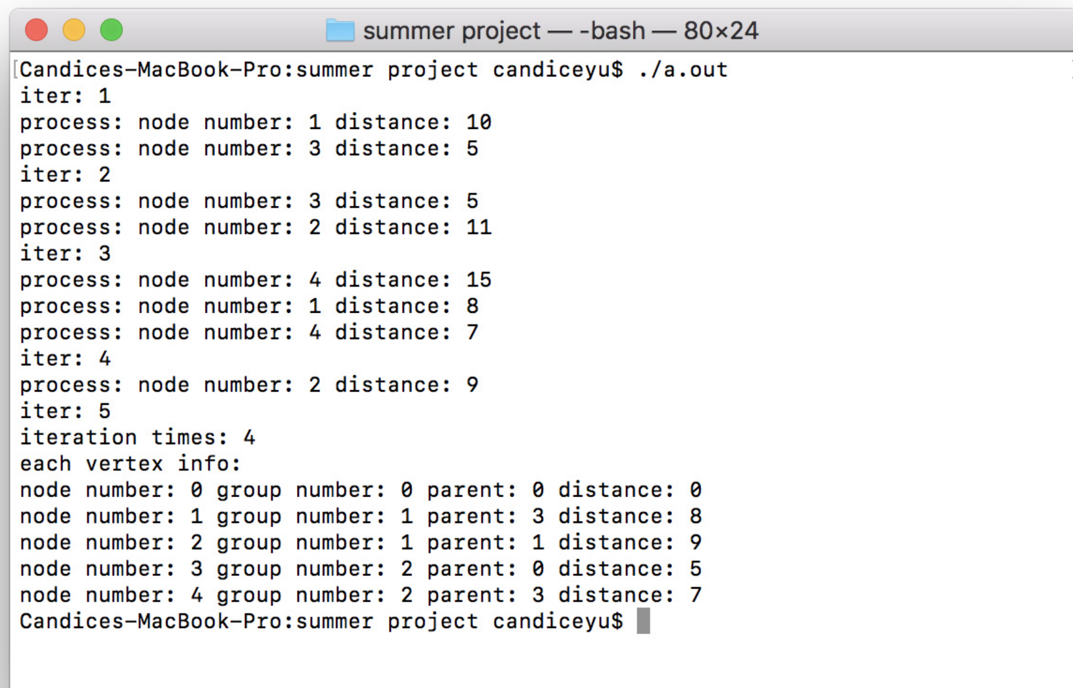
```
do{
        iter ++;
        cout << "iter: " << iter << endl;
        // missing info
        if(iter == loss){
            for(i=0; i<nodeNum; i++){
                if(graph->vertices[i].groupNum_ == missGroup){
                    graph->vertices[i].distance_ = 10000;
                    graph->vertices[i].parent = -1;
                }
            }
        }

        flag = false;
        memset(update, -1, sizeof(update));
        for(i=0; i<nodeNum; i++){
```

```
            Relax(graph, i, process, processNum, update);
        }

        for(i=0; i<nodeNum; i++){
            if(update[i][0] != -1){
                flag = true;
                // update info
                graph->vertices[i].distance_ = update[i][0];
                graph->vertices[i].parent = update[i][1];
            }
        }
    }while(flag);
```

运行结果（组号为2， 迭代次数为2时）：



上图表明，缺失信息后仍可以得到和原先相同的最短路径，但是会增加迭代次数。

4. 后续工作

1. 利用 [Laboratory for Web Algorithm]() 上所给的图（十多万节点 + 百万边信息），从而找到需要增加的运行周期的规律。
2. 可视化缺失信息迭代过程。