# 基础算法与数据结构（九） 最短路径问题

## 最短路径问题

首先定义最短路径是什么。最短路径就是在有权图中，从顶点s到顶点t的路径中权重最小的那条路。

同样，由于可能出现一点到另一点相同权重的路径，所以最短路径并不是唯一的。

本章节讨论有向图的最短路径问题，无向图可以转换为有向图进行讨论。

## 加权有向图的数据结构

```
struct DirectedEdge{
      int from;
      int to;
      int weight;
      //getter setter省略
}
```

利用这样的数据结构存储边，可以利用邻接表来进行图的存储，具体如下列代码：

```
int V;
int E;
vector<DirectedEgde> adj[V];

void addEdge(DirectedEgde e){
  adj[e.from()].add(e);
  E++;
}
```

存储S->V最短路径的一个数据结构，需要有两个元素组成，
一个是edgeTo用来记录本点V到本点父节点A(A->V)的这条路径，也就是S->V最短路径中的最后一条边。

另一个就是distTo用来记录从S->V最短路径的权重。
约定edgeTo[S]的值为null，distTo[S]的值为0，初始定义其余distTo的值为无穷大。

## 边的松弛操作

distTo数组在一开始的时候除源节点外均为无穷大。所以定义一个边的松弛操作。

简单来说，就是判断S->W的最短路径是不是需要从第三点V经由可以使路径根更短，如果是，那么更新S->W最短路径的数据。

边的松弛代码：

```
void relax(DirectedEdge e){
  int v =  e.from();
  int w =  e.to();
  //直接到W的距离比经由V更长
  if(distTo[w] > distTo[v] + e.weight()){
      distTo[w] = distTo[v] + e.weight();
      edgeTo[w] = e;
  }
}
```

对于一个边可以进行边的松弛操作，如果对于一个图，那么可以进行顶点的松弛工作，就是遍历一个顶点所有的边，分别进行边的松弛操作，那么就可以逐渐找到到达每个顶点的最短路径算法。

```
void relax(EdgeWeighteDiagraph G,int v){
  for(DirectedEdge e : G.adj(v)){
      int w = e.to();
      if(distTo[w] > distTo[v] + e.weight()){
      distTo[w] = distTo[v] + e.weight();
      edgeTo[w] = e;
  }
}
```

## 通用算法

将distTo[s]初始化为0，其他均为无穷大，那么只要放送G中的任意边直到不存在有效边（还没有遍历到的边）为止，这时候distTo中的值即为S到任意点的最短距离。

因为这个算法不规定遍历的顺序，所有效率较低。

## Dijkstra算法

(巧记 D + ijk(循环遍历要用的三个字母) + stra )

这个算法可以解决 边权重非负的加权有向图的单起点最短路径问题。

### 语言描述

将distTo数组按照之前约定初始化，然后将distTo最小的非树顶点（还没有走到的点中距离起点最近的点）放送并加入树中（连接到最短路径中）。直到所有可以到达的顶点均已经计算出非无穷大的最短路径或者所有无法到达的顶点均为无穷大。

### 数据结构

需要在distTo和edgeTo的基础上增加一个优先队列p，用来保存需要放松的顶点以及快速确定下一个要被放松的顶点。

### 复杂度

空间O(V)，时间O(ElogV)

### 代码描述

```
//此处或许无法通过编译，优先队列的第一个int为顶点号，第二个为这个顶点的distTo值，根据第二个排序优先。change含义为改变已有值，insert为插入
struct Dijkstra{
  vector<DirectedEged> edgeTo;
  vector<int> distTo;

  priority_queue<pair<int,int>> pq;

  void DijkstraRun(EdegWeightedDigraph G,int s){
      for(int v = 0;v<G.v();v++){
          if(v == s) distTo.push_back(0);
          else distTo.push_back(INT32_MAX);
```

```
            edgeTo.push_back(new DirectedEdge());
        }

        pq.insert(s);
        while(!pq.empty()){
            int v = pq.front();
            pq.pop();
            relax(G,v);
        }

        void relax(EdgeWeighteDiagraph G,int v){
            for(DirectedEdge e : G.adj(v)){
                int w = e.to();
                if(distTo[w] > distTo[v] + e.weight()){
                    distTo[w] = distTo[v] + e.weight();
                    edgeTo[w] = e;
                    //如果含有这个顶点w了，那么就更新从起点到这个点的w在优先队列中的优先度。
                    if(pq.contains(w))  pq.change(w,distTo[w]);
                    else pq.insert(w,distTo[w]);
                }
            }
      }
}
```

### 适用题目

1. 是否存在一条s到v的路径，如果有找出。
2. 给定两点之间的最短路径： 从优先队列取出t之后即可停止继续搜索。
3. 任意顶点对之间的最短路径：遍历所有顶点，对每个顶点做dijkstra算法。

## 拓扑排序的扩展 – 无环加权有向图中的最短路径算法

这个算法基于有向图中无环的基础（不适用于无向图），通过拓扑排序的推进顺序更新所有顶点的最短路径值。

证明思想为因为某一条边v->w的松弛只会进行一次，在v被放松的时候，在松弛结束前distTo[w] <= distTo[v]+e.weight这个不等式成立，如果按照拓扑排序顺序进行计算，因为v被放松后不会再处理和v有关的边了，所以distTo[v]的值是不会变化的，而distTo[w]只会不断减小，所以当s可达的顶点全部走一遍之后就可以等到最短路径了。

这个算法的时间复杂度为O(E+V)

算法代码实现从略，只需要在最开始加一步拓扑排序，得到排序后的结果，将上面D算法中的优先队列换成这个排序结果进行松弛顶点即可。

## Bellman-Ford算法 – 解决含有负权重边的图的最短路径

含有负权重环的图无法计算最短路径。

这个算法可以用来找负权重环。

实现一个队列来更新下一步需要更新distTo值的顶点，根据这个队列来进行更新操作。

时间复杂度为O(EV)，空间为O(V)

图中可以找到负权重环的条件：将所有边放送V轮之后，当且仅当队列非空时有向图存在从起点可以到达的负权重环。

因为如果含有负权重环的话，更新了W之后，如果在循环之后队列中又出现W，W会重复出现，那么就会发生V轮（边数轮）之后，队列中非空。

上代码：(java代码)

```
public class BellmanFord{
  private int[] distTo;      //起点到某顶点的最短路径
  private DirectedEdge[] edgeTo; //起点到某顶点的最后一条边
  private boolean[] onQ;     //是否在队列中
  private Queue<Integer> queue;  //预定被放松的顶点
  private int cost;          //relax调用次数
  private Iterable<DirectedEdge> cycle;   //edgeTo中是否含有负权重环

  public BellmanFord(EdgeWeightedDigraph G,int s){
      distTo = new int[G.V()];  //G.V() 边的条数
      edgeTo = new DirectedEdge[G.v()];
      onQ = new boolean[G.V()];
      queue = new Queue<Integer>();
      for(int v = 0 ;v<G.V();v++){
          distTo[v] = INT32_MAX;
```

```
        }
        distTo[s] = 0;
        queue.enqueue(s);
        onQ[s] = true;
        while(!queue.empty() && !hasNegativeCycle()){
            int v = queue.dequque();
            onQ[v] = false;
            relax(G,v);
        }
    }

    private void relax(EdgeWeightedDigraph G,int v){
        for(DirectedEdge e : G.adj(v)){
            int w = e.to();
            if(distTo[w] > distTo[v] + e.weight()){
                distTo[w] = distTo[v] + e.weight();
                edgeTo[w] = e;
                //如果更新的w不在队列中，那么加入队列，因为和w有关的边需要更改最短路径。
                if(!onQ[w] {
                    queue.enqueue(w);
                    onQ[w] = true;
                }
            }
            //对于顶点E，经过V轮后算法还未能结束，那么检测是否含有环,有环就结束。
            if(cost++%G.V() == 0){
                findNegativeCycle();
                if (hasNegativeCycle()) return;
            }
        }
    }

    private void findNegativeCycle(){
        int V = edgeTo.length;
        EdgeWeightDigraph spt;
        spt = new EdgeWeightedDigraph(V);
        //根据现在的V画出现在有的图
        for(int v = 0;v<V;v++){
            if(edgeTo[v] != null) spt.addEdge(edgeTo[v]);
        }
```

```
        //cf根据构建出来的图中找出环
        EdgeWeightedCycle cf;
        cf = new EdgeWeightedCycle(spt);
        cycle = cf.cycle();
    }

    private boolean hasNegativeCycle(){
            return cycle != null;
    }
}
```

利用dfs找出环的算法可以参见 EdgeWeightedDirectedCycle.java

套汇问题就是一个典型的找负权重环问题。