研究生算法课课堂笔记

上课日期：12 月 21 日　　第(1)节课

组长：王咪咪　　　　组员：刘文利

内容概要：

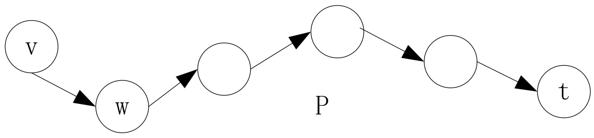
本节课主要内容是动态规划中求单源最短路径的 Bellman-Ford 算法。

1.图中的最短路径

1.1 递推方程

$$OPT(i,v)=\begin{cases}\infty & if\ \ i=0\\ \min\left\{OPT(i-1,v),\ \min\limits_{(v,w)\in E}\left\{OPT(i-1,w)+c_{vw}\right\}\right\} & otherwise\end{cases}$$

- 如果路径 P 至多用 i-1 条边，那么 $OPT(i,v)=OPT(i-1,v)$；
- 如果路径 P 用 i 条边，并且第一条边是（v,w），那么 $OPT(i,v)=OPT(i-1,w)+c_{vw}$；如图所示：



问题 1：

$OPT(i-1,v)$ 与 $\min\limits_{(v,w)\in E}\left\{OPT(i-1,w)+c_{vw}\right\}$ 是否可能有重复？比如 $OPT(i-1,w)+c_{vw}$ 中若 w 到 t 的边数≤i-2，则 v 到 t 的边数就≤i-1，这样的边可能出现吗？若出现有什么影响？

本节课未给出答案。

1.2 算法的描述

```
SHORTEST-PATHS (V, E, c, t)
-----------------------------------------------
FOREACH node v ∈ V            //初始化
   M[0, v] ← ∞.               //其他点到汇点没有路径，初始化为∞
M[0, t] ← 0.                  //汇点到汇点本身初始化为 0
FOR i = 1 TO n – 1            //i 是循环次数不是顶点序号，是为了扫描所有的边。1→n-1
   FOREACH node v ∈ V         求没有负环时的最短路径，一般写成 1→n 这样可进行负环
      M[i, v] ← M[i – 1, v].  检测，距离值变小表明有负环
      FOREACH edge (v, w) ∈ E
         M[i, v] ← min { M[i, v], M[i – 1, w] + c_vw }.
-----------------------------------------------
```

输出最短路径：

- 方法一：维护一个后继数组 $successor(i,v)$。$OPT(i,v) = OPT(i-1,v)$ 时，路径不更新；$OPT(i,v) = OPT(i-1,w) + c_{vw}$ 时，对 v 进行了更新，所以更新 $successor(i,v) = w$。
- 方法二：只用 M 表，利用 $M[i,v] = M[i-1,w] + c_{vw}$ 向下推，计算出最短路径。

1.3 复杂度分析

- 时间复杂度：看似内层循环是对每个顶点先遍历了一次，后又对每个顶点的边遍历了一次，内层循环时间复杂度为 $\Theta(nm)$，总时间复杂度是 $\Theta(mn^2)$。但是内层循环合在一起是将所有出边遍历了一次，所以内层循环复杂度为 $\Theta(m)$，总时间复杂度为 $\Theta(mn)$
- 空间复杂度：此算法利用二维动规，空间复杂度为 $\Theta(mn)$，例如 Edit-Distance 问题空间复杂度为 $\Theta(n^2)$，LCS 问题空间复杂度为 $\Theta(mn)$（此处 m，n 都是字符串长度）。

1.4 优化

- 空间优化：利用滚动数组的方法进行空间优化。一般问题如 LCS 问题利用滚动数组压缩后无法输出具体答案，但最短路径问题利用滚动数组压缩到一维后，可输出具体路径。
- 性能优化：如图一，这里维护入边表，不需要维护出边表。上一轮中若 w→t 没有更新，则该轮不要考虑 w 的入边。

2.Bellman-Ford 算法

2.1 算法的描述

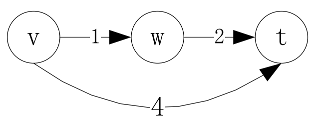
```
BELLMAN-FORD (V, E, c, t)
--------------------------------------------------------------
FOREACH node v ∈ V                                   //初始化
    d(v) ← ∞.                                        //其他点到汇点没有路径，初始化为∞
    successor(v) ← null.                             //开始没有后继结点
d(t) ← 0.                                            //汇点本身初始化为 0
FOR i = 1 TO n – 1
    FOREACH node w ∈ V
        IF (d(w) was updated in previous iteration)  //W 被更新过
            FOREACH edge (v, w) ∈ E                  1 pass
                IF ( d(v) > d(w) + c_vw)             //判断能否以 w 为跳转得到 v→t 最短路径
                    d(v) ← d(w) + c_vw.              //若以 w 为跳转得到 v→t 最短路径，更新 d(v)
                    successor(v) ← w.                //赋值 v 的后继结点为 w
    IF no d(w) value changed in iteration i, STOP.
--------------------------------------------------------------
```

问题 2：

为什么在最短路径的状态转移方程中可进行空间压缩，并能输出答案？

本节课未给出答案。

问题 3：



第 i 次循环，v→t 最短路径所用的边数可能大于 i，这时求出的最短路径不会比 v→t 只用 i 条边的距离更差，还可能会更优，为什么？

本节课未给出答案。

2.2 代码

如果要实现一个最朴素的 Bellman-Ford 算法，处理边的时候是比较简单的，不用维护出边表和入边表。PPT 上的优化是每一个顶点只有在发生距离改变的时候，才进行相应入边的处理。这里实现一个简单的版本，把所有的边放到一个大数组里，不用把这些边按照出边或者入边归到它的顶点下。

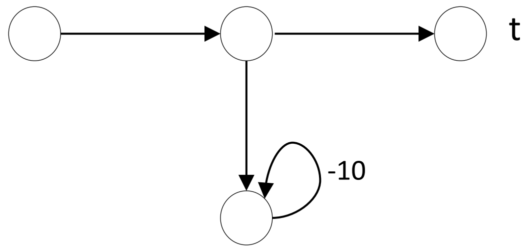
朴素的 Bellman-Ford 算法如下：

```
struct edge{
    int u, v;     //u到v的边
    double c;     //u到v的距离
};
double d[N]; //每个顶点当前求出的最短距离
bool valid[N];    //表示数组d里的值是否合法，到汇点是否可达，用来处理无穷，也可以用
double中的INF
edge e[M];        //所有的边的大数组,下标从0开始
int s[N];         //记录每个顶点最短路径的后继顶点

//可以定义add_edge处理输入，将边加到数组后面即可
void add_edge(int u, int v, double c){
}
    //初始化
    memset(valid, 0, sizeof(valid));
    d[t] = 0;
    valid[t] = true;

    for (int i = 0; i < n; i++){
        bool changed = false; //(1)性能优化
        for (int j = 0; j < m; j++){   //每一轮循环考察大数组中所有的边
            int u = e[j].u, v = e[j].v;
            double c = e[j].c;
            if (valid[v]){        //如果v是合法的，到汇点可达，最短距离已经求出，
如果v到t不可达，u不能通过v跳转
                double dist = c + d[v];        //u通过v做跳转
                if (!valid[u] || d[u] > dist){      //u到汇点不可达或可达距离比
dist大，要更新d[u]
                    d[u] = dist;
                    s[u] = v;     //u的下一个顶点是v
                    valid[u] = true;
                    changed = true;   //(1)性能优化
                }
            }
        }
        if (!changed)//(1)性能优化。边都循环了一遍,如果changed没有变化，说明以后也
不会变了
            return ;
    }
```

2.3 Bellman-Ford 算法用于负环检测

对于上机作业最后一道题负环检测，要加一个超级汇点 t，然后把所有顶点

到汇点 t 连一条边，距离都为 0。这么做的目的是什么？Bellman-Ford 算法是基于单汇点的，它只能检测有没有可达汇点的负环，如果图中有负环但和汇点没有任何关系，是检测不到的。比如下面这个图：



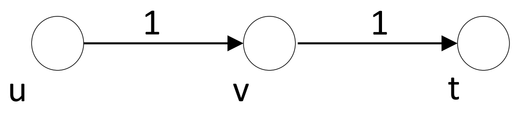
这个负环不可能到达 t，所以是检测不到的。它对于计算最短路径也是没有影响的。将所有顶点连到汇点，如果原图中任何一个地方有负环，这个负环一定是可达汇点的，负环上的点到汇点的最短距离在循环过程中会不断地变小。

问题 4：

有没有可能原图中没有负环，新图中出现负环？也就是新加一个 t 之后，t 成为负环上的顶点？这是不可能的。因为 t 没有出边，只有入边。

问题 5：

假设有一轮迭代过程中，内层循环所有边都循环完一次之后，d[v]没有被更新，是不是以后也不会被更新了？答案是否定的。一个反例如下图：



由于我们没有规定边的顺序，开始 d[u]和 d[v]都是∞，如果先 relax 边 u->v，由于 d[v]= ∞，所以 d[u]= ∞，然后 relax 边 v->t，d[v]= 1，下次再 relax 边 u->v，d[u]=2。所以，如果在所有边的一轮迭代中，只是某一个点或者某些点的距离没有变化，并不是这些点的最短距离已经求出，因为它所关联的其他点的最短距离有可能发生了变化，这些变化会一步一步传递到它的最短距离。但是如果在一轮迭代中，所有顶点的最短距离都没有发生变化，那么循环就可以结束了，因为这个循环所有输入数据都是不变的，包括它的拓扑结构，每个顶点连到哪些顶点，以及边的权重，如果一个顶点连接到的所有顶点的距离都没有变，那么这个顶点的最短距离也是不会变的，依赖于这个顶点的其他顶点的最短距离也都不会变。

所以一轮循环中所有顶点的距离都没有发生变化，就到达了 fix-point 的状态。

2.4 Bellman-Ford 算法性能优化

优化方法是，加入一个布尔变量 changed。具体见代码中标记(1)性能优化的地方。如果反复循环，始终没有在循环中返回，一直走到外层循环结束，说明存在负环。因为如果没有负环，i 从 0 到 n-2，已经循环了 n-1 次，所有点的最短距离都已经确定，最后一次循环 i=n-1 第 n 次循环，所有点的最短距离都不应该变化了，即 changed 不应该变为 true，应该在循环内返回。所以检测负环在最后的大循环外加 return false 即可。

PPT 上的优化是:只有当某一个顶点的最短距离发生变化后，在下一轮循环才需要松弛这个顶点的所有入边。这一点在我们的代码中没有体现，我们遍历了所有边，但是加上 changed 变量提前返回，就足以让性能有一个质的飞跃。它的复杂度 O(mn)，是一个保守的估计，实际往往达不到这么坏的情况。

问题 6：

朴素的 Bellman-Ford 算法把所有边都遍历一遍，遍历 n-1 遍结果收敛，也就是说如果没有负环，遍历 n-1 遍就能求出所有边的最短距离。每次把所有边松弛一遍，访问各条边的顺序是不是一定要一样？这样迭代 n 轮，能不能保证是收敛的？这个利用了最短距离的最优子结构的性质。假设有顶点 v，要求 v->t 的最短距离，假设有一条最短路径，v->w，然后 w->t，性质就是，如果沿着最短路径的各条边的逆序松弛一遍，v 的最短距离就可以求出。假如 v->t 这样走是最短路径，中间顶点 w->t 一定也是 w 的最短路径，如果 w 有一条更短的路径，那么 v->w，再换成 w->t 的更短的路径，就得到 v->t 的更优解。所以最短路径的子路径一定也是最短的。我们沿着最短路径的逆序松弛，这个松弛的过程不一定是连续的，只要在整个算法的运行过程中，能抽出一个子序列是逆序松弛一遍的就可以了。所以如果每一轮处理的边的顺序不一样，第一轮迭代之后，逆序第一个点的最短距离一定能求出，第二轮之后，逆序第二个点的最短距离也求出来了，这样最多经过 n-1 轮，每个点的最短距离都能求出。所以在 Bellman-Ford 算法中，每一轮边的处理顺序是可以发生变化的。下节课的 SPFA 本质上就是利用这一点。