研究生算法课课堂笔记

上课日期：2016 年 12 月 26 星期一　　　第(1)节课

组长学号及姓名：1601214462 李飞

组员学号及姓名：1601214488 吴炜

内容概要：

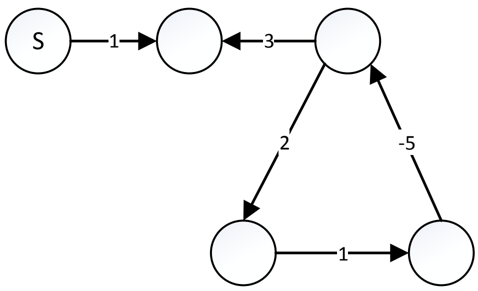
1. 回顾上节课讲的 Bellman-Ford 算法
2. 介绍 Floyd-Warshall 算法
3. 贪心算法习题讲解（最大生成树）

详细内容：

1. Bellman-Ford 算法

上节课讲了 Bellman-Ford 的算法以及改进版本 SPFSA (Shortest Path Faster Algorithm)，和 Dijkstra 算法类似，用来求单源最短路径。如果求多源最短路径，并且图中没有负边，可以使用 Dijkstra 算法。把 Dijkstra 算法运行 n 遍（以下都用 n 表示顶点数，m 表示边数），每次求出一个给顶定点到其它所有顶点的最短路径。一次运行的复杂度是$O(m\,log\,n)$，要运行 n 次，所以一共是$O(n\cdot m\,log\,n)$。如果图有负边可以使用 Bellman-Ford 算法，它和 SPFA 最坏情况下的复杂度都是$O(mn)$，运行 n 遍，所以一共是$O(mn^2)$。

如果是检测负环，目前为止我们只能使用 Bellman-Ford 算法。运行 Bellman-Ford 算法 n 轮，如果第 n 轮与第 n-1 轮相比，最短距离又发生了优化，说明图存在负环。如果图中没有负环，那么此算法最多 n-1 轮就会收敛（实际中往往收敛得比这个快很多）。需要注意的一点是，对于某个源点出发的最短路径，只有源点可达的负环才会被检测到。比如下图中的负环从 S 出发就检测不到，这对求 S 的最短路径没有任何影响，所以我们认为这是合理的。

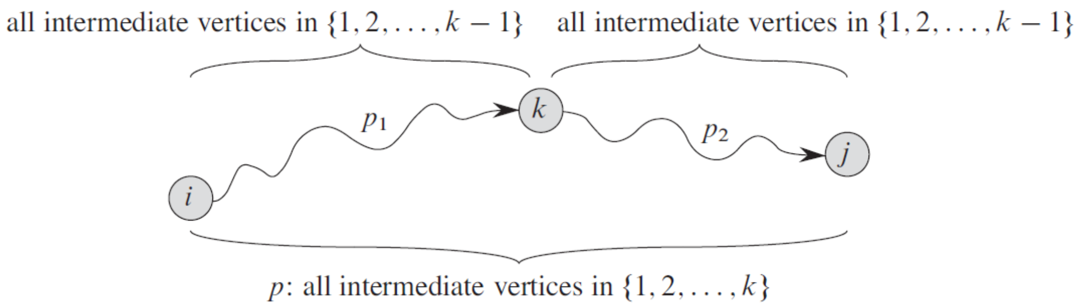


一般使用 Bellman-Ford 算法检测负环会加一个超级源点 S’，它到原图中所有的顶点都有一条权重为 0 的边。这样就能检测到所有的负环，并且不会改变原有的负环，因为加边之后负环不会减少，而新加的顶点 S’只有出边没有入边，即加边之后负环不会增加。如果使用单汇点最短路径算法，只有当负环可达汇点的时候才会对求解有影响。这时如果要检测负环，类似的可以加一个超级汇点，原图中所有的顶点到超级汇点都有一条权重为 0 的边，然后检测可达超级汇点的负环。

2. Floyd-Warshall 算法

这个算法比较巧妙，简单易写，没有递归，没有数据结构，可以检测负环。思想是把所有顶点以任意的顺序从 1 开始编号，假如要求顶点 i 到顶点 j 的最短路径，那么这两个顶点要么直连，要么经过某个中间顶点连起来，用$f(i,j,k)$ 表示从顶点 i 到顶点 j 中间经过的顶点序号不超过 k 的最短路径长度，即中间作为跳板的顶点只能取自集合{1, 2, ..., k}。当 k=0 时，表示从顶点 i 到顶点 j 中间经过的顶点需要小于等于 0，即若图中 i 到 j 有边直连，f(i, j, 0) 为 i 到 j 的权重，否则就是无穷大。下面给出递推公式，从顶点 i 到顶点 j 所经过中间顶点序号不超过 k 的最短路径上：

- 如果没有用到顶点 k，则路径上的中间顶点序号不超过 k-1。
- 如果用到了顶点 k，因为没有负环，所以最短路径肯定是简单路径，用到的顶点只会出现一次，因此顶点 i 到顶点 k 的最短路径 p_1 和顶点 k 到顶点 j 的最短路径 p_2 就不会再出现序号为 k 的中间顶点，即 p_1 和 p_2 上的中间顶点序号不超过 k-1，如下图（引用自《算法导论》）所示：



我们可以得到如下的递推公式：

$$f(i,j,k) = min\begin{cases} f(i,j,k-1) & \text{没有用到第}k\text{个顶点} \\ f(i,k,k-1)+f(k,j,k-1) & \text{用到了第}k\text{个顶点} \end{cases}$$

其中 f(i, j, n)即为顶点 i 到顶点 j 的最短路径长度。算法的时间复杂度和空间复杂度都是$O(n^3)$，实际实现时空间复杂度可优化到$O(n^2)$，如下所示：

```
for i = 1 to n
  f(i, i) = 0 自己到自己的距离初始化为 0，有自环（非负）不用考虑
for (i, j) ∈ E  所有 i 到 j 的边
  f(i, j) = Wi,j
for k = 1 to n
    for i = 1 to n
      for j = 1 to n
        dist = f(i, k) + f(k, j)
        if (dist < f(i, j))
          f(i, j) = dist
```

需要注意的一点是，实现时得有一个无穷大表示不可达（注意无穷大相加会溢出的问题）。或者另开一个 Boolean 数组，用 b(i, j)表示顶点 i 是否可达顶点 j，得到上面的算法改进版（新增部分加粗表示）：

```
for i = 1 to n
  f(i, i) = 0 自己到自己的距离初始化为0，有自环（非负）不用考虑
  b(i, i) = 1
for (i, j) ∈ E  所有i到j的边
  f(i, j) = W_{i,j}
  b(i, j) = 1
for k = 1 to n
    for i = 1 to n
      for j = 1 to n
        if(b(i, k) && b(k, j))
          dist = f(i, k) + f(k, j)
          if (!b(i, j) || dist < f(i, j))
            f(i, j) = dist
            b(i, j) = 1
```

Bellman-Ford算法的时间复杂度为$O(mn^2)$，其中稀疏图为$O(n^3)$，稠密图为$O(n^4)$；Floyd-Warshall算法的时间复杂度为$O(n^3)$。看起来Floyd-Warshall算法比Bellman-Ford算法快，但实际上Bellman-Ford的时间复杂度是一个很悲观的复杂度。而Floyd-Warshall的时间复杂度是优化不了的，不管什么样的图都是$O(n^3)$，与边数无关，适合稠密图。

使用上面的算法还可以检测负环，方法是检查最后得到的$n*n$矩阵f的对角线，如果对角线上有负数，说明这个顶点到自己有一条长度为负数的路径，即有负环。

3. 贪心算法练习题第19题。

问题：证明沿着最大生成树上的路径走可以得到最大带宽。

证明：假如s到t沿着最大生成树有一条路径p_1，但是还存在一条与之不完全重合的最优路径p_2，p_2的最大带宽大于p_1的最大带宽，即p_2上带宽最小的边比p_1上带宽最小的边大。p_1和p_2可以构成一个环，环上带宽最小的边在p_1上，也在最大生成树上，这跟最大生成树的red rule（环上权值最小的边一定不在最大生成树上）矛盾，所以不存在这样的路径p_2，即沿着最大生成树上的路径就可以得到最大的带宽。

另外，对于图G有最小生成树MST，有一条边E不在MST上，将E放入MST中会得到一个环，那么E是环上**权重最大的边之一**。