算法导论复习6——图算法

Chapter 22 基本图算法

广度优先算法

- 节点的颜色：
  - 白色：节点未被发现
  - 灰色：节点被发现但还未遍历完成（在队列中或在栈中）
  - 黑色：节点被遍历完成（已经出队或出栈）
- 代码实现：

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
```

```
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.color = BLACK
```

- 最短路径
  - 在无权图中，对于任何边(u, v)，s到v的最短距离一定大于等于s到u的最短距离加一（三角形法则）
  - BFS中，若v在u之前进队，则在u进队时，v的最短距离小于等于u的最短距离
  - BFS结束后，**计算所得的v.d是最短距离**，且**一条最短路径一定是s到v的直接前驱的最短路径加上v直接前驱到v这条边**
- 广度优先树：BFS构建的前驱子图（通过前驱关系构建出的子图）是一棵广度优先树
- 时间复杂度：$\Theta(V+E)$

深度优先搜索

- 发现时间：刚发现节点的时间（边变灰时刻）

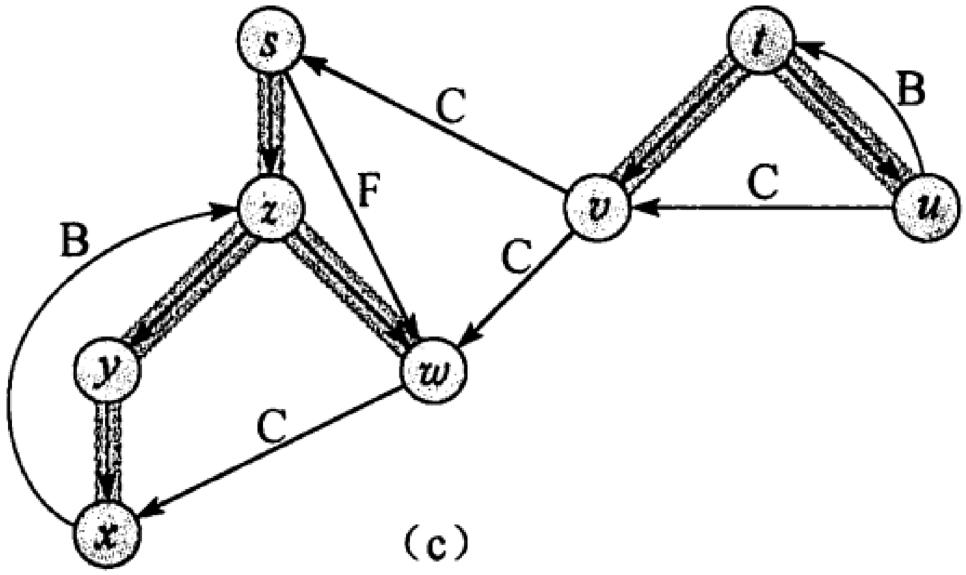
  完成时间：遍历完节点的时间（边变黑时刻）
- 代码实现

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1                // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]          // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                // blacken u; it is finished
time = time + 1
u.f = time
```

- 时间复杂度：$\Theta(V+E)$
- 括号化定理
  - 如果两个节点的d-f区间不重合，那么深度优先森林中互相不为祖先关系
  - 如果v的区间包含于u的区间，那么深度优先树中v为u的祖先（反之也成立）
  - **不存在不重合且不包含关系**
- 白色路径定理

  v是u的后代，当且仅当**在发现u的时刻，存在一条从u到v的全白路径**
- 边的分类：



(c)

- 树边：在深度优先森林中的边，如果节点v是因为算法对(u, v)的探索而发现，那么(u, v)是一条树边
- 后向边：在深度优先森林中，连接节点和其祖先的边
- 前向边：在深度优先森林中，连接节点和其后代的边
- 横向边：在深度优先森林中，连接的两个节点互不为祖先的边

在无向图中，只存在树边和后向边

拓扑排序

- 算法描述
    - 使用DFS计算完成时间
    - 当一个节点完成时，将其插入拓扑序列的头部
- 只有当图为有向无环图时，拓扑排序才能成功

强连通分量

- 求强连通分量的算法描述
    - 进行一次DFS
    - 将图中所有边转置
    - 按照完成时间递减的顺序来对转置图DFS，每次遍历完一棵树后就输出这棵树
- 算法正确性：

  从完成时间最晚的强连通分量开始DFS，那么这个强连通分量的转置图中不存在到任何其他强连通分量的边

  引理 22.13 设 C 和 C' 为有向图 $G=(V, E)$ 的两个不同的强连通分量，设结点 u，$v\in C$，结点 u'，$v'\in C'$，假定图 G 包含一条从结点 u 到结点 u' 的路径 $u \rightsquigarrow u'$。那么图 G 不可能包含一条从结点 v' 到结点 v 的路径 $v' \rightsquigarrow v$。

  引理 22.14 设 C 和 C' 为有向图 $G=(V, E)$ 的两个不同的强连通分量。假如存在一条边 $(u, v)\in E$，这里 $u\in C$，$v\in C'$，则 $f(C)>f(C')$。

推论 22.15 设 C 和 C' 为有向图 $G=(V, E)$ 的两个不同的强连通分量，假如存在一条边 $(u, v)\in E^{T}$，这里 $u\in C$，$v\in C'$，则 $f(C)<f(C')$。

Chapter 23 最小生成树

Kruskal算法

- 算法描述
  - 将所有边按照权重进行排序
  - 对所有点建立并查集
  - 对所有边进行遍历，如果这条边两个端点不在同一个集合中，就将这条边加入MST边集合中

```
MST-KRUSKAL(G,w)
A=∅
for each vertex v∈G.V
      MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge(u,v)∈G.E, taken in nondecreasing order by weight
      if FIND-SET(v)≠FIND-SET(v)
            A=A∪{(u,v)}
            UNION(u,v)
return A
```

- 时间复杂度
  - 对边排序$O(ElgE)$
  - 建立并查集和查找合并操作$O((V+E)\alpha(V))$
  - 连通图，有E > V - 1，因此并查集操作时间复杂度为$O(E\alpha(V))$
  - 由于$\alpha(V)=O(lgV)$，时间变为$O(ElgE)$
  - 再注意到E < V^2，最终时间复杂度为$O(ElgV)$

Prim算法

- 算法描述
  - 将所有节点的距离初始化为无穷，按照距离建立优先级队列
  - 从优先级队列中抽取最小值，并将队列中所有的节点距离按照抽出的节点进行更新，直至队列为空

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
    if v ∈ Q and w(u, v) < v.key
        v.π = u
        v.key = w(u, v)
```

- 时间复杂度
  - 抽取循环执行V次
  - 抽取最小值总需要$O(VlogV)$
  - 关键字减值，使用斐波那契堆为$O(E)$，使用二叉堆为$O(ElogV)$
  - 总时间复杂度，斐波那契堆为$O(E+VlogV)$，二叉堆为$O(ElogV)$

Chapter 24 单源最短路径

松弛操作

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

最短路径贪心的数学保证

引理 24.1(最短路径的子路径也是最短路径)　给定带权重的有向图 $G=(V, E)$ 和权重函数 $w: E\to\mathbf{R}$。设 $p=\langle v_0, v_1, \cdots, v_k\rangle$ 为从结点 v_0 到结点 v_k 的一条最短路径，并且对于任意的 i 和 j，$0\leqslant i\leqslant j\leqslant k$，设 $p_{ij}=\langle v_i, v_{i+1}, \cdots, v_j\rangle$ 为路径 p 中从结点 v_i 到结点 v_j 的子路径。那么 p_{ij} 是从结点 v_i 到结点 v_j 的一条最短路径。

Bellman-Ford算法

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge(u, v) ∈ G.E
        RELAX(u, v, w)
for each edge(u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

- **当存在负环时，Bellmanford算法返回false**

  证明：假设返回true，找出图中负环，利用三角不等式将环上所有的距离之和相加，即可反证
- 时间复杂度：$O(VE)$

利用拓扑排序对有向无环图求单源最短路径

- 由于没有环路，拓扑排序后，边松弛的顺序就是从源到目的节点的顺序，所以恰好是正确的

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
  for each vertex v ∈ G.Adj[u]
      RELAX(u, v, w)
```

- 时间复杂度：$O(V+E)$

Dijkstra 算法

- Dijkstra算法不能运行在有负边权的图上

```
DIJKSTRA. (G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

- 时间复杂度
  - 数组实现：$O(V^2+E)=O(V^2)$
  - 二叉堆：$O(VlogV+ElogV)=O(ElogV)$
  - 斐波那契堆：$O(VlogV+E)$

Chapter 25 多源最短路径

矩阵乘法

- 每乘一次邻接矩阵，都是计算多一步转移的最短路径，因此只需要乘n次即可得到最短路径矩阵，时间复杂度为$O(V^4)$
- 可以通过自乘的方法快速逼近正确解，时间复杂度变为$O(V^3logV)$
- 最短路径的恢复方法见下方Floyd算法

Floyd-Warshall算法

FLOYD-WARSHALL(W)

$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
    let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
return $D^{(n)}$

- 该算法可以只构建一个矩阵，因为单个k循环中只会使用一次ij，且即使ik和kj的值被改变了，因为k是任意一个中间节点，通过遍历所有的k，一定可以找到这样的一个k，所以只用一个矩阵就可以求解
- 恢复最短路径，可以**在每一次更新dij的时候看使用了哪个量，来更新pij**即可，**其中存储的变量表示为以i为起点，j的最短路径上的前一个节点。**
- 时间复杂度：$O(V^3)$，适合稠密图

Johnson算法

```
JOHNSON(G, w)
    compute G', where G'.V = G.V ∪ {s},
            G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
            w(s, v) = 0 for all v ∈ G.V
    if BELLMAN-FORD(G', w, s) == FALSE
        print "the input graph contains a negative-weight cycle"
    else for each vertex v ∈ G'.V
                set h(v) to the value of δ(s, v)
                    computed by the Bellman-Ford algorithm
            for each edge (u, v) ∈ G'.E
                ŵ(u, v) = w(u, v) + h(u) − h(v)
            let D = (d_uv) be a new n × n matrix
            for each vertex u ∈ G.V
                run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
                for each vertex v ∈ G.V
                    d_uv = δ̂(u, v) + h(v) − h(u)
            return D
```

- Johnson算法适合稀疏图，通过Bellmanford算法计算的最小值来定义顶点修正因子，从而修正了每一条边的权值，使其非负，从而可以使用Dijkstra算法。
- 修正公式：

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

- 时间复杂度：
    - 二叉堆：$O(VE + VElogV) = O(VElogV)$
    - 斐波那契堆：$O(VE + V^2logV)$