《算法设计与分析》
第四次作业

姓名：曹建钦

学号：20375177

1 对下面的每个描述，请判断其是正确或错误，或无法判断正误。对于你判为错误/无法判断的描述，请说明它为什么是错误/无法判断的

1. P 类问题为 NP 类问题的真子集。

此描述是无法判断的。因为 P 是否等于 NP 无法确定，因此只能肯定 P 类问题一定是 NP 类问题的子集而无法判断是否为真子集，若假设 $P \neq NP$，则此描述为正确的。

2. 如果假设 $P \neq NP$，则 NP 完全问题可以在多项式时间内求解。

此描述是错误的。如果假设 $P \neq NP$，则 NP 完全问题不存在一个多项式时间算法，则不可以在多项式时间内求解。

3. 若 SAT 问题可以用复杂度为 $O(n^9)$ 的算法来解决，则所有的 NP 完全问题都可以在多项式时间内被解决。

此描述是正确的。

4. 对于一个 NP 完全问题，其所有种类的输入均需要用指数级的时间求解。

此描述是错误的。NP 完全问题一定是一个 NP 类问题，而 NP 类问题是多项式时间可验证的，那么存在一种输入使得只需要多项式时间求解。

2 颜色交错最短路问题

2.1 Main Idea

给定一个无权有向图 $G =< V, E >$（所有边长度为 1），其中 $V = \{v_0, v_1, \ldots v_{n-1}\}$,且这个图中的每条边不是红色就是蓝色 ($\forall e \in E, e.color = red$ 或 $e.color = blue$)，图 G 中可能存在自环或平行边。现给定图中两点 v_x, v_y，求出一条从 v_x 到 v_y，且红色和蓝色边交替出现的最短路径。如果不存在这样的路径，则输出-1。

分析过程：对于无权有向图求两点之间最短路径的问题一般采取广度优先搜索的办法，但本题有些特殊，其特殊之处在于:

- 图 G 中存在自环或平行边
- 两点间的路径需要红色和蓝色边交替出现

可以看出，一个点最多可以访问两次（对应自环的情况，走自环边以切换路径颜色），且在寻找下个入队节点时需要考虑该边的颜色，需要据此对 BFS 进行修正，算法的伪代码如下：

2.2 Pseudo Code

Algorithm 1: $Color-Cross-BFS(G, v_x, v_y)$

Input: 无权有向图 G，起点 v_x，终点 v_y

Output: 一条从 v_x 到 v_y，且红色和蓝色边交替出现的最短路径或-1

新建数组 $color[0..|V|-1]$，$visitTimes[0..|V|-1]$，$pred[0..|V|-1]$
新建空队列 Q
// 初始化
for $u \in V$ **do**
    $color[u] \leftarrow WHITE$
    $pred[u] \leftarrow NULL$
    $visitTimes[u] \leftarrow 0$
end
$color[v_x] \leftarrow GRAY$
$pred[0].append(noColor)$
$Q.Enqueue(v_x)$
$visitTimes[v_x] \leftarrow 1$
while 等待队列 Q 非空 **do**
    $u \leftarrow Q.Dequeue()$
    for $v \in G.Adj[u]$ **do**
        if $color[v] = WHITE$ *or* $color[v] = GRAY$ **then**
            if u, v 间存在和 $pred[u].tail$ 异色的边 $< u, v >$ **then**
                $eColor \leftarrow < u, v > .color$
                $color[v] \leftarrow GRAY$
                $pred[v].head \leftarrow u$
                $pred[v].append(eColor)$
                $Q.Enqueue(v)$
                $visitTimes[v] \leftarrow visitTimes[v] + 1$
                if $visitTimes[v] = 2$ **then**
                    $color[v] \leftarrow BLACK$
                end
            end
        end
    end
end
$backTracking(v_y)$

Algorithm 2: $backTracking(v_y)$

Input: 点 v_y

Output: 节点 v_x 到节点 v_y 的路径

```
1  新建空栈 stack
2  stack.push(v_y)
3  u ← pred[v].head
4  while u ≠ v_x do
5  |  if u = NULL then
6  |  |  return -1
7  |  else
8  |  |  stack.push(u)
9  |  |  u ← pred[u].head
10 |  end
11 end
12 stack.push(v_x)
13 while stack ≠ NULL do
14 |  print stack.pop()
15 end
16 return
```

2.3 Complexity Analysis

算法 $Color-Cross-BFS$ 中 4-8 行初始化需要遍历图中所有点，复杂度为 $O(V)$；13-30 行的循环本质上是图的广度搜素，复杂度同 BFS 为 $O(V+E)$；31 行调用的 $backTracking$ 函数是一个回溯的过程，复杂度最多为 $O(V)$。综上则有总复杂度：

$$O(V+E)$$

3 最小闭合子图问题

3.1 Main Idea

给定一个包含 n 个点的有向图 $G =< V, E >$，求出该图中的闭合子图至少应包含几个顶点。

分析过程：以图 G 中任意一个点开始，执行 DFS 后得到的点集都是一个闭合子图，那么便可以通过遍历得到闭合子图最少包含的顶点个数。

3.2 Pseudo Code

Algorithm 3: $Min - Close - Graph(G)$

Input: 有向图 G，其包含 n 个点

Output: 闭合子图至少包含的顶点个数 num

```
// 图 G 本身是最大闭合子图
num ← n
for s ∈ V do
    aNum ← BFS(G, s)
    if aNum < num then
        num ← aNum
    end
end
return num
```

Algorithm 4: $BFS(G, s)$

Input: 有向图 G，源点 s

Output: 以 S 为源点得到的闭合子图中顶点个数 $aNum$

```
1  新建一维数组 color[1...|V|]
2  // 初始化
3  for u ∈ V do
4  |  color[u] ← WHITE
5  end
6  color[s] ← GRAY
7  Q.Enqueue(s)
8  aNum ← 0
9  while 等待队列 Q 非空 do
10 |  u ← Q.Dequeue()
11 |  aNum ← aNum + 1
12 |  for v ∈ G.Adj[u] do
13 |  |  if color[v] = WHITE then
14 |  |  |  color[v] ← GRAY
15 |  |  |  Q.Enqueue(v)
16 |  |  end
17 |  end
18 |  color[u] ← BLACK
19 end
20 return n
```

3.3 Complexity Analysis

算法 $Min-Close-Graph$ 中 3-8 行的循环结构需要执行 $|V|$ 次 BFS，而执行一次 BFS 的复杂度为 $O(V+E)$，因此，算法总复杂度为 $O[V(V+E)]$

4 食物链问题

4.1 Main Idea

一个食物网，包含 n 个动物，m 个捕食关系，第 i 个捕食关系使用 (s_i, t_i) 表示，s_t 为捕食者 t_i 表示被捕食者，根据生物学定义，食物网中不会存在环。长度为 k 的食物链指包含 k 个动物的链：$a_1, a_2, \ldots, a_k$，其中 a_i 会捕食 a_{i+1}，一个食物链为最大食物链当且仅当 a_1 不会被任何动物捕食，且 a_k 不会捕食任何动物，需要计算食物网中最大食物链的数量。

分析过程：食物网即为一张图，而根据生物学原理，这张图为有向无环图。同时，最大食物链的起点不被任何动物捕食，则入度为零，终点不会捕食任何动物，则出度为零。那么对所有入度为零的点进行深度优先搜索即可找到最大食物链的数量。

伪代码如下：

4.2 Pseudo Code

Algorithm 5: $Food - Chain - DFS(G)$

Input: 食物网 G

Output: 食物网中最大食物链的数量 num

```
num ← 0
新建数组 color[1...V]
// 初始化
for v ∈ V do
    color[v] ← WHITE
end
for v ∈ V and v.in_dergee = 0 do
    DFS − Visit(G, v)
end
return num
```

Algorithm 6: $DFS-Visit(G,v)$

Input: 图 G，顶点 v

```
1  if v.out_degree = 0 then
2  |  num ← num + 1
3  end
4  color[v] ← GRAY
5  for w ∈ G.Adj[v] do
6  |  if color[w] = WHITE then
7  |  |  DFS − Visit(G, w)
8  |  end
9  end
10 color[v] ← WHITE
```

4.3 Complexity Analysis

算法复杂度与课上所讲深度优先搜索相似。在算法 $DFS-Visit$ 中，搜索顶点 v 的开销为 $O(1+|Adj[v]|)$，而执行 $DFS-Visit$ 时最多需要搜索全图，即 $|V|$ 次，复杂度为 $O(V+E)$；同时，算法 $Food-Chain-DFS$ 的 4-6 行复杂度为 $O(V)$，7-8 行最多循环 $|V|-1$ 次，则算法总时间复杂度为 $O[V(V+E)]$

5 景区限流问题

5.1 Main Idea

已知某市有热门景区 m 个，可以表示为集合 $S=\{s_1,s_2,\ldots,s_m\}$，有游客 n 人，可以表示为 $T=\{t_1,t_2,\ldots,t_n\}$。每名游客有 k 个心仪的景区，但每个景区最多容纳 l 人，需要求出最多有多少人能够去到自己心仪的景区，对于游客与景区的偏好关系，用 H 表示，其中 (t_u,s_v) 表示游客 t_u 偏好景点 s_v。

分析过程：此问题为二部图多重匹配的一对多问题，可以建立一个超级源点和一个超级汇点，将每名游客与源点间建立一条容量为 1 的边，再将每名游客和其心仪的 k 个景区间连接一条容量为 1 的边，最后将 m 个景区和超级汇点间依次连接容量为 l 的边，那么问题“求出最多有多少人

能够去到自己心仪的景区”便可以转化为最大流问题，可利用课上所讲的 $Ford-Fulkerson$ 算法进行求解。伪代码如下：

5.2 Pseudo Code

Algorithm 7: $flowLimit(m, S, n, T, H, k, l)$

Input: m 个热门景区集合构成的集合 S，n 名游客构成的集合 T，游客和景区间的偏好关系 H，每个景区最多容纳人数为 l

Output: 能够去到自己心仪景区的最大人数 f^*

```
// 构造二部图
作二部图 G_1 =< T, S, H >
新建两节点 s, t
V ← T ∪ S ∪ {s, t}
E ← H
C ← {}
for u ∈ T do
    e ←< s, u >
    E ← E ∪ e
    C(e) ← 1
end
for v ∈ S do
    e ←< v, t >
    E ← E ∪ e
    C(e) ← l
end
for u ∈ T do
    for v ∈ S do
        e ←< u, v >
        C(e) ← 1
    end
end
作图 G =< V, E, C >
rerutn Ford − Fulkerson(G, s, t)
```

Algorithm 8: $Ford-Fulkerson(G,s,t)$

Input: 图 $G=<V,E,C>$，源点 s，汇点 t

Output: 最大流 f^*

```
for eachedgee ∈ G.E do
    e.f ← 0
end
G_f ← bulidresidualnetwork
while there exists a path p from s to t in the residual network G_f do
    c_f(p) ← min{c_f(e) : eisinp}
    if e ∈ E then
        e.f ← e.f + c_f(p)
    else
        e.f ← e.f − c_f(p)
    end
    update G_f
end
f* ← f
return f*
```

5.3 Complexity Analysis

算法 $flowLimit$2-23 行是造流网络的过程，其中 7-11 行为连接源点 s 到集合 T 中的所有点，并赋以容量，其复杂度为 $O(n)$；12-16 行为连接集合 S 的所有点到汇点 t，并赋以容量，其复杂度为 $O(m)$；17-22 行双重循环给集合 S、T 中两两相连的边赋以容量，其复杂度为 $O(m\cdot n)$；第 24 行调用算法 $Ford-Fulkerson$，时间复杂度为 $O(E\cdot|f^*|)$，其中 $E=m+n+m\cdot n$，综上，算法的总时间复杂度为：

$$O(E\cdot|f^*+1|)$$