MO417 – Ata do Exercício 22.2-6

Alexandre Toshio Hirata

27 de junho de 2010

Enunciado: *Há dois tipos de lutadores de luta livre profissional: "bons" e "maus". Entre qualquer par de lutadores profissionais, pode ou não haver rivalidade. Suponha que temos n lutadores profissionais e temos uma lista de r pares de lutadores dos quais há rivalidade. Dê um algoritmo que roda em tempo $O(n+r)$ que determina se é possível rotular alguns dos lutadores como bons e o restante como maus de forma que cada rivalidade é entre um bom e um mau. Se é possível realizar tal rotulação, seu algoritmo deveria produzí-la.*

Seja $G = (V, E)$ um grafo de rivalidades onde $v \in V$ é um lutador e $(u, v) \in E$ indica que existe rivalidade entre os lutadores u e v, tal que $u, v \in V$. Assim, o problema de rivalidade proposto também pode ser interpretado como verificar se G é bipartido ou então verificar se é possível colorir G com até 2 cores.

Para resolver este problema vamos modificar o algoritmo de busca em largura para "colorir" o grafo com duas "cores" ($GOOD$ e BAD) e, caso haja alguma inconsistência durante a colaração retornamos um erro e concluímos que a coloração não é possível e, portanto, não é possível realizar a rotulação desejada. Entretanto, é sabido que a busca em largura produz uma árvore, logo depende do grafo de entrada ser conexo. Para contornar isso, aplicaremos a busca em largura modificada em cada componente conexa de G e, assim, garantimos que todos os nós são testados. A seguir, segue a implementação proposta pelo aluno Fabian van't Hooft com uma pequena modificação para que seja executado em todas as componentes conexas.

O algoritmo 1 inicializa os tipos de todo $v \in V$ para $UNKNOWN$ (ou seja, não tem tipo conhecido) e depois executa o algortimo 2 para cada componente conexa. Note que se não for possível colorir com duas cores em alguma componente conexa de G então podemos concluir que não podemos realizar a rotulação necessária.

Algorithm 1: RIVALRY(G)

```
1  foreach u ∈ V[G] do
2      type[u] ← UNKNOWN
3  foreach u ∈ V[G] do
4      if type[u] = UNKNOWN then
5          if BIPARTITE(G,u) = FALSE then
6              return FALSE
7  return TRUE
```

Algorithm 2: BIPARTITE(G, s)

```
1  type[s] ← GOOD
2  Q = ∅
3  ENQUEUE(Q,s)
4  while Q ≠ ∅ do
5      u ← DEQUEUE(Q)
6      foreach v ∈ Adj[u] do
7          if type[v] = UNKNOWN then
8              if type[u] = GOOD then
9                  type[v] ← BAD
10             else
11                 type[v] ← GOOD
12             ENQUEUE(Q,v)
13         else
14             if type[u] = type[v] then
15                 return FALSE
16 return TRUE
```

Já o algoritmo 2 inicializa a raiz da busca como $GOOD$ e a coloca na fila Q. Depois, para cada nó desenfileirado (este obrigatoriamente é $GOOD$ ou BAD de acordo com a construção do algoritmo), é percorrida sua lista de adjacências rotulando os nós com tipo desconhecido de acordo com um tipo diferente do nó que acabou de ser desenfileirado e os coloca na fila. Caso o nó da lista de adjacências já possua um tipo, é verificado se este não viola a "coloração", caso viole, então retorna que a rotulação falhou.