03 - Busca em largura

Marco A L Barbosa malbarbo.pro.br

1. Exercício 22.2-1 de CLRS3 ou CLRS2.
2. Exercício 22.2-2 de CLRS3 ou CLRS2.
3. Exercício 22.2-3 de CLRS3. (De acordo com a errata, é para considerar a remoção da linha 18, e não das linhas 5 e 14)
4. Exercício 22.2-4 CLRS3 ou 22.2-3 CLRS2.
5. Exercício 22.2-5 CLRS3 ou 22.2-4 CLRS2.
6. Exercício 22.2-6 CLRS3 ou 22.2-5 CLRS2.
7. Exercício 22.2-7 CLRS3 ou 22.2-6 CLRS2. (Dica: pense nos níveis da árvore da busca em largura - os vértices que têm o mesmo valor d estão no mesmo nível)
8. Exercício 22.2-8 CLRS3 ou 22.2-7 CLRS2 (opcional). (Dica: e se o problema fosse encontrar o vértice mais distante de um outro vértice dado?)
9. Exercício 22.2-9 CLRS3 ou 22.2-8 CLRS2. (Dica: e se o grafo fosse uma árvore da busca em largura? Pense neste caso e tente estender a solução para o caso geral.)

Referências

- [CLRS2] - Thomas H. Cormen et al. Introduction to Algorithms. 2nd edition. Capítulo 22.2.
- [CLRS3] - Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 22.2.

Soluções

Exercício 7

Representamos os lutadores e as rivalidades por um grafo não orientado $G = (V, E)$. Os lutadores são os vértices e as rivalidades as arestas. Desta forma $n = |V|$ e $r = |E|$. Temos que verificar se é possível atribuir a cada vértice o rótulo "bom" ou "mau" de forma que cada aresta conecte vértices com rótulos diferentes. De fato, os rótulos em si não são importantes, precisamos é classificar o vértices em duas classes de forma que não existam arestas entre vértices da mesma classe. Este é o problema de verificar se um grafo é bipartido.

Podemos usar os valores $v.d$ obtidos pela execução do BFS para particionar os vértices. O vértice com $d = 0$, fica na primeira partição (dos "bons"), os vértices com $d = 1$ precisam ficar na segunda partição (dos "maus") pois estão conectados com vértices na primeira partição. Os vértices com $d = 2$ devem ficar na primeira partição (pois estão conectados com vértices da segunda partição) e assim por diante. Esta classificação é feita analisando apenas as arestas da árvore da busca em profundidade, no final precisamos verificar se as demais arestas conectam vértices em partições diferentes. A execução do BFS é $O(n + r)$ e o custo de verificar é arestas é $O(n + r)$ (no caso de lista de adjacências), portando o tempo de execução do algoritmo é $O(n + r)$. Segue o pseudo código do algoritmo.

```
BONS-E-MAUS(G)
1  Execute BFS no grafo G partindo de um vértice qualquer
2  for u ∈ G.V
3       for v ∈ G.Adj[u]
4            if paridade de u.d == paridade de v.d
5                 return FALSE
6  Designe cada vértice v com v.d par como "bom"
7  Designe cada vértice v com v.d ímpar como "mau"
8  return TRUE
```