Grafos e *Union-Find*

Parte A

1. **Complete a sentença:** Um grafo é um conjunto de _____ e de _____, as quais armazenam informações sobre as relações entre os _____.
2. Cite 3 representações possíveis para um grafo e dê uma breve descrição de cada uma delas.
3. Qual é a complexidade, em memória, de uma lista de adjacências?
4. No contexto de grafos, defina:
   (a) caminho de um nó u a um nó v
   (b) ciclo
   (c) caminho simples
   (d) grafo conectado
   (e) árvore
5. Qual é a diferença entre um grafo direcionado e um grafo não-direcionado? E entre um grafo ponderado e um grafo não-ponderado? Pode-se assumir que todos os grafos são ponderados?
6. Em quais tipos de grafos se aplicado o conceito de grau de um vértice? O grau de um vértice é definido pelo seu grau de entrada, pelo grau de saída ou por ambos? O que se pode afirmar sobre a soma dos graus de todos os vértices de um grafo?
7. No contexto de vértices vizinhos,
   (a) em que situações o vértice v é vizinho de um vértice u?
   (b) é possível que um vértice u não tenha nenhum vizinho?
   (c) a relação de vizinhança é transitiva, isto é, se u é vizinho de v, então v é vizinho de u?
8. O que caracteriza um grafo regular? E um grafo completo?
9. **Complete a sentença:** Uma estrutura *union-find* mantém um coleção de _____.

Parte B

10. Em quais cenários uma representação de grafo por matrizes de adjacência seria adequada? Qual tipo de problema pode ser resolvido de forma eficiente usando matrizes de adjacência?
11. Qual é a complexidade, em memória, de uma lista de adjacências?
12. Defina, em C++, um grafo representado por
    (a) matriz de adjacências
    (b) lista de adjacências
    (c) lista de arestas
13. Existe uma relação entre um grafo bipartido e os seus ciclos. Que relação é esta?
14. É possível representar, em C++, um grafo ponderado por lista de arestas usando um vetor de tuplas:

```cpp
#include <iostream>
#include <vector>
#include <tuple>

using namespace std;
using graph = vector<tuple<int, int, int>>;
```

Indique como seria feita as inclusões de arestas nesta estruturas e como membros dos elementos individuais seriam acessados.

15. Abaixo temos uma implementação, em pseudocódigo, de uma estrutura *union-find*:

Algoritmo 1 Estrutura *union-find*

Entrada: O número N de subconjuntos

```
1:  procedure INIT(N)
2:      for i ← 1, n do
3:          link[i] ← i
4:          size[i] ← 1
5:      end for
6:  end procedure
7:
8:  procedure FIND(u)
9:      while x ≠ link[x] do
10:         x = link[x]
11:     end while
12:     return x
13: end procedure
14:
15: procedure SAME(u, v)
16:     return FIND(u) == FIND(v)
17: end procedure
18:
19: procedure UNITE(u, v)
20:     a ← FIND(u)
21:     b ← FIND(v)
22:     if size[a] < size[b] then
23:         SWAP(a, b)
24:     end if
25:     size[a]+ = size[b]
26:     link[b] = a
27: end procedure
```

Implemente o pseudocódigo acima em C++.

16. O algoritmo acima pode ser otimizado utilizando se a técnica de compressão de caminhos. Explique o que vem a ser esta técnica e modifique sua implementação anterior para contemplá-la

17. Usando a compressão de caminho é possível descartar o vetor que registra o tamanho de cada subconjunto? Justifique sua resposta.