CMP1099 - Estrutura de Dados II

Lista de Exercícios - Revisão para prova substitutiva

Max Gontijo de Oliveira

1. Considere os algoritmos de ordenação *selection-sort*, *bubble-sort*, *insertion-sort*, *merge-sort*, *quick-sort* e *heap-sort*.
   (a) Explique, de forma sucinta, como a ordenação é realizada (duas ou três linhas para cada algoritmo).
   (b) Quais são os limites de complexidade assintótica de cada algoritmo? Ou seja, qual é a complexidade *O(...)* de pior caso e a complexidade $\Omega(...)$ de melhor caso?

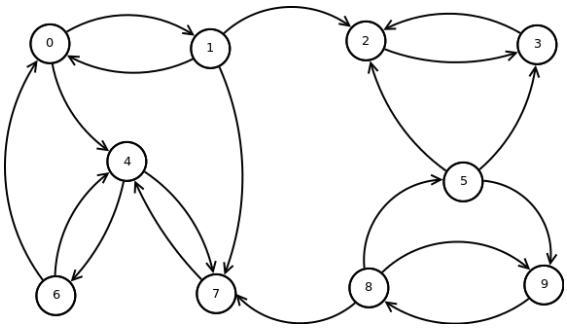
2. Apresente e descreva pelo menos duas técnicas para a criação de uma boa função *hash*. Apresente um exemplo, considerando que a tabela tenha m posições.

3. Sobre o problema de colisão de *hash*:
   (a) Explique o que vem a ser esse problema
   (b) Explique como o tratamento de colisão pode ser resolvido por listas ligadas
   (c) Explique como o tratamento de colisão pode ser resolvido por sondagem (não detalhe os tipos de sondagem).

4. Sobre a sondagem como solução para o problema de colisão de *hash*:
   (a) Explique como funciona a sondagem linear.
   (b) Explique como a sondagem linear é mais suscetível a ter aglomerações de registros e sondagens com sucessivas colisões.
   (c) Explique como funciona a sondagem quadrática e como ela minimiza o problema de aglomerações de registros mas continua com o problema de sucessivas colisões durante a sondagem.
   (d) Explique como funciona a sondagem por *hash* duplo e porque ela consegue minimizar o problema de sucessivas colisões durante a sondagem.

5. Apresente um algoritmo que utilize tabela *hash* para contabilizar quantas vezes cada palavra aparece em um vetor de m strings. Ao final, seu algoritmo deverá imprimir, para cada palavra distinta no vetor, a palavra e a quantidade de vezes que aparece. Utilize qualquer notação: pseudo-código, C++, misturado. Apenas seja claro em seu algoritmo. Não é para apresentar algoritmo de métodos de tabela *hash* ou função *hash* ou qualquer coisa interna à tabela *hash*. Essa questão é para avaliar se você sabe usar a estrutura. Considere que os métodos/-funções disponíveis para utilizar a tabela *hash* sejam os seguintes:
   - `void put(string k, int v)`: armazena o inteiro `v` na tabela *hash* sob a chave `k`.
   - `int get(string k)`: retorna o inteiro armazenado na tabela *hash* sob a chave `k`. Caso não exista a chave na tabela, o método lança uma exceção.
   - `bool exists(string k)`: retorna verdadeiro caso exista um elemento armazenado na tabela *hash* sob a chave `k`; retorna falso, caso contrário.
   - `int size()`: retorna a quantidade de elementos armazenados na tabela *hash*.
   - `string* keys()`: retorna um vetor contendo todas as chaves existentes na tabela *hash*.

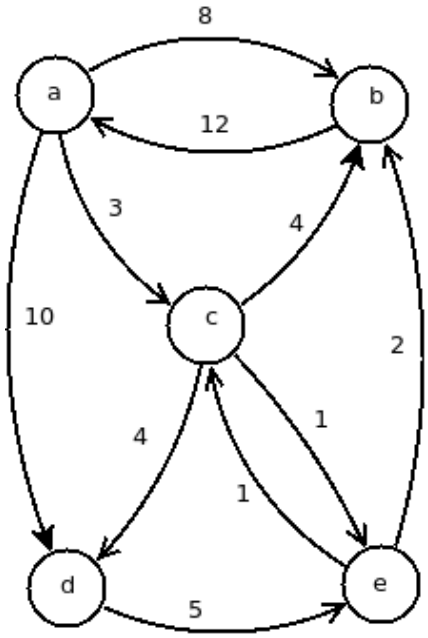
6. Considere o seguinte grafo direcionado:



   (a) Aplique o algoritmo BFS com vértice de origem 1 e mostre as arestas de árvore.
   (b) Aplique o algoritmo DFS com vértice de origem 1 e classifique TODAS as arestas entre: arestas de árvore, arestas de retorno, arestas diretas e arestas cruzadas.
   (c) Após a aplicação do DFS, apresente a ordenação topológica dos vértices.
   (d) O que são os componentes fortemente conectados em um grafo? Descreva, com suas palavras, quais são os passos adicionais para, a partir da aplicação do

DFS e a ordenação topológica, se obter os componentes fortemente conectados do grafo. Não precisa buscar tais componentes. Apenas explicar o que teria que ser feito para obtê-los.

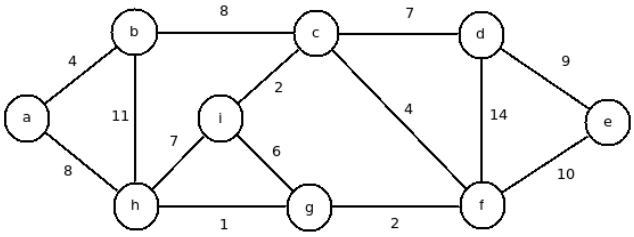
7. Considere o seguinte grafo direcionado e valorado:



   (a) Aplique o algoritmo de Dijikstra a partir de vértice a a apresente a árvore de menores caminhos gerada, bem como o valor da distância de cada vértice alcançado pela origem.

   (b) Aplique o algoritmo Belman-Ford a partir de vértice a a apresente a árvore de menores caminhos gerada, bem como o valor da distância de cada vértice alcançado pela origem. Além disso, mostre o valor das distâncias de cada vértice a cada iteração do algoritmo. Em cada iteração, para selecionar a ordem das arestas que serão avaliadas, considere as arestas de saída de cada vértice em ordem decrescente do identificador. Ou seja, a cada iteração, relaxe primeiro as arestas de saída do vértice e, depois relaxe as arestas de saída do vértice d, depois do c, depois do b e por fim as do a. Quando começar uma nova iteração, siga novamente a mesma ordem.

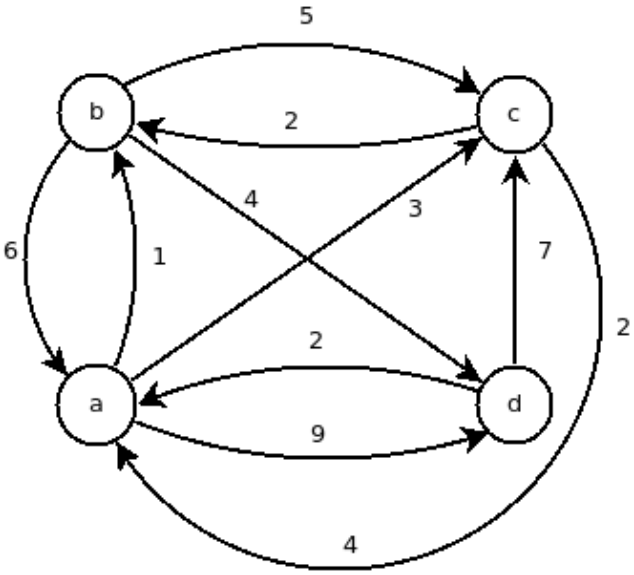
8. Considere o seguinte grafo não direcionado:



   (a) Encontre a árvore espalhada mínima por meio do método de Kruskal.

   (b) Encontre a árvore espalhada mínima por meio do método de Prim.

9. Considere o seguinte grafo direcionado e aplique o algoritmo de Floyd-Warshall para encontrar os menores caminhos entre todos os pares de vértices:



10. Considere o seguinte grafo direcionado e aplique o método de Edmonds-Karp para encontrar o fluxo máximo no grafo: