Avaliação Final

QXD0010 — Estrutura de Dados Avançada – Turma 01A

Prof. Atílio Gomes

06 de setembro de 2021

Aluno: ______________________________

Matrícula: ______________________________

OBSERVAÇÃO: Logo abaixo, encontram-se cinco exercícios de programação, que devem ser enviados via Moodle até as 23h59h do dia 06/09/2021.

Observação 1: Envie código compilável. Uma questão cujo código não compilar receberá automaticamente nota ZERO.

Observação 2: Qualquer prova com sinal de plágio receberá nota ZERO.

1. (2.5 pontos) Vimos em aula que a implementação mais eficiente da estrutura de dados Conjuntos Disjuntos (Union-Find) é aquela que usa o conceito de floresta de conjuntos disjuntos. Vimos também uma implementação desta ideia que usa um array como estrutura base para a implementação dos Conjuntos Disjuntos. A implementação vista em aula e que também está nos slides foi enviada em anexo juntamente com esta prova. Com base nesta implementação de Conjuntos Disjuntos, resolva o seguinte problema:

   Suponha que nós queiramos adicionar a operação `PrintSet(x)` que faz o seguinte: dado um nó x como entrada ela imprime todos os elementos que estão no mesmo conjunto que o x, em qualquer ordem.

   (a) Mostre como podemos adicionar um único atributo a cada nó em uma floresta de conjuntos disjuntos de modo que: (1) a função `PrintSet(x)` execute em tempo linear no número de elementos do conjunto que contém x; e (2) a complexidade assintótica das demais operações de Conjuntos Disjuntos permaneça inalterada. Escreva o pseudocódigo da sua solução e prove que ela satisfaz os requisitos acima.

   (b) Codifique em C++ a função `PrintSet(x)` que você projetou no item (a) usando como base o código que foi enviado com essa prova.

   (c) Crie um arquivo **main.cpp** que testa a função `PrintSet(x)` que você implementou.

Boa Prova!

2. (3 pontos) O menor ancestral comum de dois nós u e v em uma árvore enraizada T é o nó w que é um ancestral tanto de u quanto de v e que tem a maior profundidade em T. No problema do menor ancestral comum, são dados como entrada uma árvore enraizada T e um conjunto arbitrário $P = \{\{u, v\}\}$ de pares de nós não ordenados de T, e desejamos determinar o menor ancestral comum de cada par em P.

   A fim de resolver o problema do menor ancestral comum, o procedimento a seguir realiza um percurso na árvore T com chamada inicial `MAC(T.root)`. Nós supomos que cada nó é colorido com a cor `WHITE` antes do percurso iniciar.

```
MAC(u)
Make-Set(u)
Find-Set(u).ancestor = u
para cada filho v de u em T faça
    MAC(v)
    UNION(u,v)
    Find-Set(u).ancestor = u
u.color = BLACK
para cada nó v tal que {u,v} ∈ P faça
    se v.color == BLACK então
        print "o menor ancestral comum de"
            u "e" v "é" Find-Set(v).ancestor
```

   As operações `Make-Set`, `Find-Set` e `Union` são operações de Conjuntos Disjuntos. Note que cada nó do conjunto disjunto possui um campo adicional chamado `ancestor` nesse algoritmo. Logo, você deve modificar a implementação de Conjuntos Disjuntos a fim de incluir esse campo adicional juntamente com as operações que o acessam e o modificam.

   (a) Implemente o algoritmo `MAC` visto acima em C++. O algoritmo `MAC` recebe como entrada uma árvore enraizada T e uma sequência de pares não ordenados P de vértices de T e retorna como saída o menor ancestral comum de cada par de vértices de P. A entrada e a saída do programa devem ser fornecidas como explicado abaixo:

   Entrada

   A primeira linha da entrada consiste no número de vértices N da árvore T. A segunda linha contém o vértice r que é a raiz da árvore, $0 \leq r \leq N - 1$. Este número é seguido por uma lista de arestas. Cada linha na lista tem dois inteiros, a e b, que são os vértices extremos de uma aresta ($0 \leq a, b \leq N - 1$). A lista de arestas termina com uma linha com um par de 0 (zeros). Após esta linha, segue uma lista P de pares de vértices da árvore T. Cada linha nessa segunda lista tem dois inteiros, a e b ($0 \leq a, b \leq N-1$). A lista de pares de vértices termina com uma linha com um par de 0 (zeros). Esse par de zeros indica o final da entrada.

Saída

Para cada par de vértices na lista P é impresso numa linha separada o menor ancestral comum do par.

Exemplo de Entrada

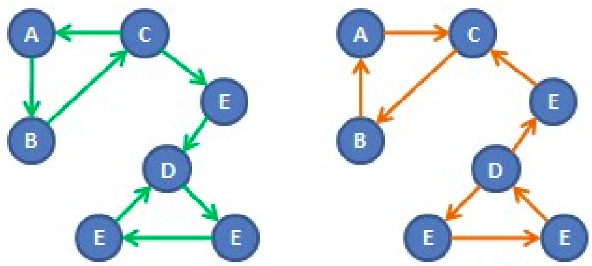
12
0
0 1
0 2
0 3
1 4
1 5
1 6
1 7
4 8
6 9
6 10
10 11
0 0
8 11
9 7
5 3
0 0

Exemplo de Saída para a entrada acima

o menor ancestral comum de 11 e 8 é 1
o menor ancestral comum de 7 e 9 é 1
o menor ancestral comum de 3 e 5 é 0

Boa Prova!

3. (2.5 pontos) O **transposto** de um grafo direcionado $G = (V, E)$ é o grafo $G_T = (V, E_T)$, onde $E_T = \{(v, u) \in V \times V : (u, v) \in E\}$. Assim, G_T é G com todas as suas arestas invertidas. A figura abaixo mostra um grafo direcionado G à esquerda e o seu transposto G_T à direita.



Implemente, em C++, algoritmos eficientes para calcular G_T a partir de G, para a representação por **lista de adjacências** de G e por **matriz de adjacências** de G. Note que você deve fornecer duas implementações distintas do grafo.

Entrada

Suponha que os vértices do grafo G sejam inteiros de 0 a $N - 1$, onde N é o número de vértices de G. A primeira linha da entrada contém o número de vértices no grafo direcionado, N. Este número é seguido por uma lista de arcos de G (arestas). Cada linha na lista tem dois inteiros, a e b, separados por espaço em branco, que são os vértices extremos de um arco de G, o vértice a é a cauda e b é a cabeça do arco $(0 \leq a, b \leq N - 1)$. A lista de arestas termina com uma linha com um par de 0 (zeros). Esse par de zeros indica o final da entrada.

Saída

Cada linha da saída consiste em um par de vértices separados por espaço em branco e indica uma aresta do grafo transposto G_T.

Exemplo de Entrada

4
0 1
1 2
0 3
0 0

Exemplo de Saída

1 0
2 1
3 0

Boa Prova!

4. (2 pontos) Um **heap máximo** é um vetor $A[1..n]$ composto de n inteiros (chaves), satisfazendo a seguinte propriedade:

$$A[i] \leq A[\lfloor i/2 \rfloor], \text{ para todo } i \text{ tal que } 1 < i \leq n. \tag{1}$$

Seja `A` um heap máximo. A função `HEAP-DELETE(int A[], int i)` remove a chave contida na posição i do heap `A` e, após ser executada, garante que a Propriedade (1) de heap máximo continue válida para o vetor `A`, ou seja, o vetor `A` continua sendo um heap máximo após a remoção da chave que estava contida na posição i do vetor. Dê uma implementação (pseudocódigo) de `HEAP-DELETE` que seja executada no tempo $O(\log n)$ para um heap máximo `A` com n elementos. Para essa questão, você pode considerar que o tamanho total do vetor `A` é dado pela variável global `tamVetor` e que o tamanho real do heap contido em `A` é dado pela variável global `tamHeap`. **Observação:** QUALQUER função auxiliar que você usar no seu código também deve ser implementada.

(a) Escreva o pseudocódigo da função `HEAP-DELETE(int A[], int i)`

(b) Prove que a complexidade de pior caso da sua função é $O(\log n)$.

Boa Prova!