

Teste 11 de Janeiro 2022

Algoritmos e Complexidade

Universidade do Minho

🔗 Questão 1 [4 valores]

Considere uma tabela de *hash* implementada por *open addressing* e *linear probing* sobre um array dinâmico cujo comprimento é um número primo N , utilizando-se a habitual função de hash $h(x) = x \% N$. O array é realocado quando fica totalmente preenchido e se faz nova inserção, passando a ter como tamanho o número primo imediatamente superior ao dobro do tamanho actual. Ao realocar é necessário inserir todos os elementos na nova tabela, com a função de hash actualizada. Considere que se utiliza a chave especial `EMPTY` para sinalizar as posições inicialmente vazias.

Simule a evolução de uma tabela de comprimento inicial 3 ao longo da sequência de inserção dos seguintes números, mostrando claramente o seu tamanho e conteúdo em cada passo:

| | | | | |
|----|----|----|----|----|
| 10 | 17 | 25 | 27 | 38 |
|----|----|----|----|----|

Questão 2 [5 valores]

Considere uma noção de árvore binária em que os nós contêm um campo adicional `weight` em que se armazena o **peso** de cada nó. O peso de um nó N define-se como o número total de nós contidos na árvore que tem N como raiz.

Escreva uma função `int checkWeight(Btree t)` que verifica se os pesos de uma árvore se encontram correctamente preenchidos (devolvendo 1 caso estejam, e 0 caso não estejam).

Analise depois o tempo de execução da função.

Por exemplo, nas figura seguinte, os pesos estão bem preenchidos na árvore da esquerda, mas não na da direita (os número nos nós correspondem ao campo `weight`, omitindo-se outra informação).



Assuma o seguinte tipo de dados para as árvores:

```
1 struct btree {
2     int info;
3     int weight;
4     struct btree *left, *right;
5 };
6 typedef struct btree *Btree;
```

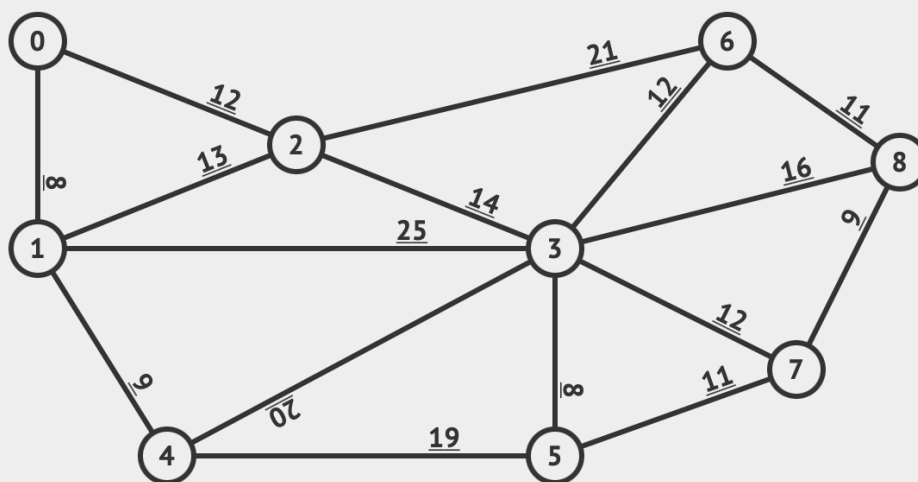
(obs: para quem estiver autorizado a codificar em Python, sugere-se a seguinte representação simples para as árvores binárias: `None` para a árvore vazia, e um tuplo `(info, weigth, esq, dir)` para um nó intermédio — e.g. a árvore da esquerda seria representada como

```
(10, 4, (5, 2, (3, 1, None, None), None), (22, 1, None, None)))
```


Questão 3 [4 valores]

Aplique o algoritmo de Prim para calcular uma árvore geradora mínima do seguinte grafo não-orientado, tomando o vértice 4 como inicial, e:

1. desenhe a árvore intermédia contendo apenas os 3 primeiros vértices, e explique o passo seguinte de expansão da árvore
2. desenhe a árvore geradora mínima (final).



Questão 4 [5 valores]

Considere os seguintes tipos de dados para a representação de grafos orientados, sem pesos, por listas de adjacências:

```
1 struct edge {  
2     int dest;  
3     struct edge *next;  
4 };  
5 typedef struct edge *GraphL[MAX];
```

Defina em C a função `int shortestBy (GraphL g, int s, int d, int i)` que recebe um grafo orientado sem pesos e três vértices `s`, `d`, `i`, e:

- devolve -1 caso `d` não seja alcançável a partir de `s`
- devolve 0 caso nenhum caminho mais curto de `s` até `d` contenha o vértice `i`
- devolve 1 caso um caminho mais curto de `s` até `d` passe pelo vértice `i`

Questão 5 [2 valores]

Seja α uma constante tal que $0.5 \leq \alpha < 1$ e relembre o conceito de peso de um nó da Questão 2.

Uma árvore binária de procura diz-se α —balanceada se em toda a sub-árvore, os rácios dos números dos seus nós da contidos à sua esquerda e à sua direita são ambos $\leq \alpha$.

Por outras palavras, para todo o nó X , $\frac{\text{peso}(X.\text{esq})}{\text{peso}(X)} \leq \alpha$ e $\frac{\text{peso}(X.\text{dir})}{\text{peso}(X)} \leq \alpha$. O valor $\alpha = 0.5$ corresponde ao balanceamento ideal.

Demonstra-se que o tempo de execução de uma operação de procura numa árvore α —balanceada é logarítmico no tamanho da árvore.

1. Quando, na sequência de uma operação de inserção ordenada seguindo o algoritmo habitual, uma sub-árvore de tamanho M deixa de ser α —balanceada, é possível reajustá-la, por forma a torná-la 0.5—balanceada, em tempo $\Theta(M)$ (desde que se possa utilizar espaço auxiliar também de tamanho $\Theta(M)$).

Mostre (por palavras ou utilizando pseudo-código) como pode ser implementado este reajuste (baseie-se na estrutura definida na Questão 2).

2. O tempo *amortizado* deste reajuste é no entanto $\mathcal{O}(1)$, o que significa que o tempo amortizado da operação de inserção numa árvore α —balanceada de tamanho N é de facto $\mathcal{O}(\lg N)$ (uma vez que os reajustes necessários são todos feitos em nós de um caminho desde a raiz até ao nó mais profundo em que o balanceamento foi infringido, e este caminho tem comprimento logarítmico).

Considere a seguinte função de potencial sobre árvores:

$$\Phi(t) = c \cdot \sum_{X \in t} \Delta(X)$$

com

$$\Delta(X) = \begin{cases} | \text{peso}(X.\text{esq}) - \text{peso}(X.\text{dir}) | & \text{se } | \text{peso}(X.\text{esq}) - \text{peso}(X.\text{dir}) | \geq 2 \end{cases}$$

e

$$\Delta(X) = 0 \text{ se } | \text{peso}(X.\text{esq}) - \text{peso}(X.\text{dir}) | < 2$$

Determine um valor apropriado para a constante c em função do valor de α . Mostre que a função Φ satisfaz os requisitos de uma função de potencial e mostre que o custo amortizado resultante para a operação de reajuste é efectivamente constante.