Algoritmos e Complexidade — Universidade do Minho
Teste 3 de Janeiro de 2024

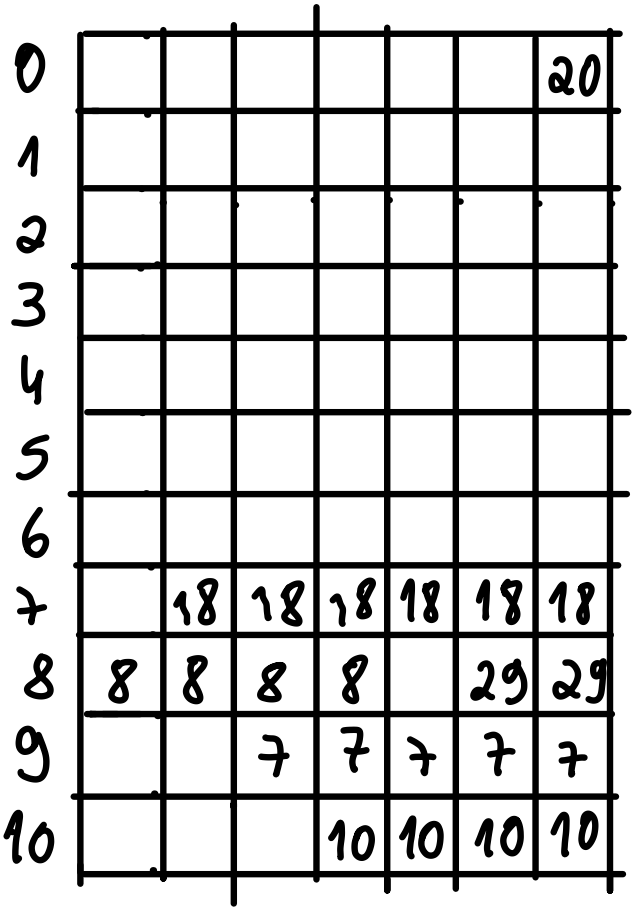
Número: _____ Nome: _____

1. Tabelas de *Hash* (5 valores)

Considere uma tabela de números inteiros implementada com endereçamento aberto e *linear probing,* sobre um array de comprimento 11, com a função hash(x) = x%11, e com reaproveitamento de posições entretanto libertadas. Simule a evolução do seu estado a partir da tabela inicialmente vazia, quando executa a seguinte sequência de operações:

Inserir 8; 8%11 = 8
Inserir 18; 18%11 = 7
Inserir 7; 7%11 = 7
Inserir 10; 10%11 = 10
Remover 8; 8%11 = 8
Inserir 29; 29%11 = 7
Inserir 20; 20%11 = 9

Deve desenhar a tabela em todos os passos da sequência acima.

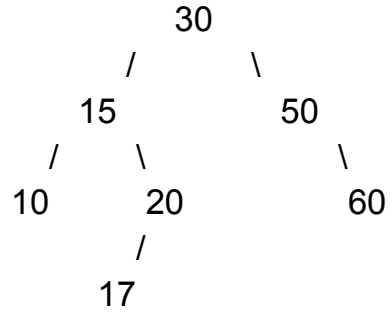


Algoritmos e Complexidade — Universidade do Minho
Teste 3 de Janeiro de 2024

Número: _____ Nome: _____

2. Árvores Binárias / AVL (5 valores)

Recorde que as estruturas de dados conhecidas como *heaps* são árvores binárias implementadas fisicamente sobre arrays. Embora não habitual, é possível implementar quaisquer outras árvores binárias sobre arrays, bastando para isso utilizar um valor especial para representar as lacunas, i.e. posições da árvore em que não se encontra qualquer nó. Por exemplo a seguinte árvore de números inteiros



pode ser representada por arrays preenchidos desta forma:
`| 30 | 15 | 50 | 10 | 20 | H | 60 | H | H | 17 | ... H ... |`

a) Escreva uma função `checkBST (int t [], int N)` que recebe um array `t` de comprimento `N` e verifica (devolvendo `0` ou `1`) se representa ou não uma **árvore binária de procura** de números inteiros. Considere que a constante `H` é utilizada para representar lacunas.

b) Escreva uma função `checkAVL (int t [], int N)` que verifica se `t` representa ou não uma árvore balanceada (de acordo com o critério das árvores AVL). Analise em termos assintóticos o tempo de execução desta função.

Serão valorizadas as soluções mais eficientes.

```
int checkBST(int t[], int N){
    for(int i=0; i<N; i++){
        if (t[i]==H) continue;
        left=2*i+1;
        right=2*i+2;
        if(t[left]!=H && t[left]>=t[i])
            return 0;
        if(t[right]!=H && t[right]<=t[i])
            return 0;
    }
    return 1;
}
```

```
int checkAVL(int t[], int N){
    int *isBalanced = 1;
    isAVL(t, N, 0, isBalanced);
    return *isBalanced;
}
int isAVL(int t[], int N, int idx, int *isBalanced){
    if(idx >= N || t[idx] == H) return 0;
    int left = 2*idx + 1;
    int right = 2*idx + 2;
    int leftH = isAVL(t, N, left, isBalanced);
    int rightH = isAVL(t, N, right, isBalanced);
    if(abs(leftH - rightH) > 1) *isBalanced = 0;
    return 1 + (leftH > rightH ? leftH : rightH);
```

↓

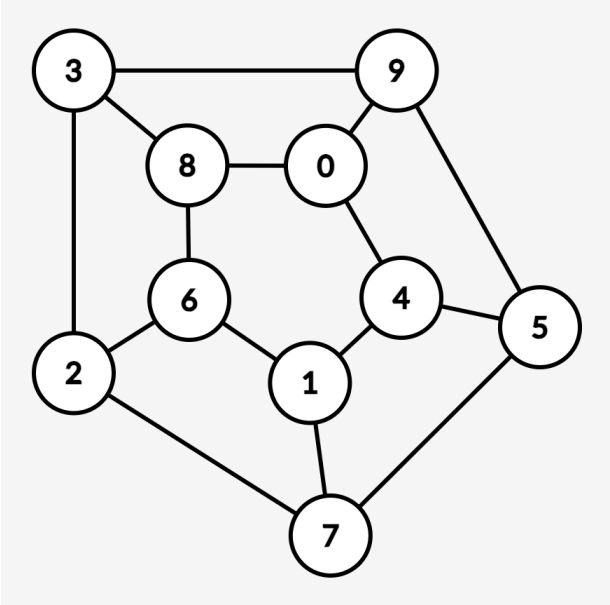
max(leftH, rightH);

Algoritmos e Complexidade — Universidade do Minho
Teste 3 de Janeiro de 2024

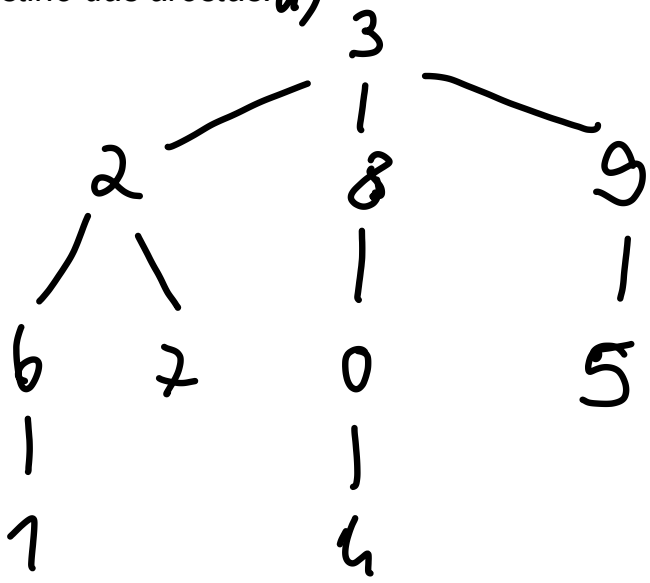
Número: _____ Nome: _____

3. Grafos (5 valores)

Considere que o grafo não-orientado seguinte é representado por listas de adjacências da forma usual (i.e. cada aresta é representada por duas arestas orientadas). Considere ainda que as listas de adjacência são mantidas ordenadas por *ordem crescente dos nós destino das arestas.*



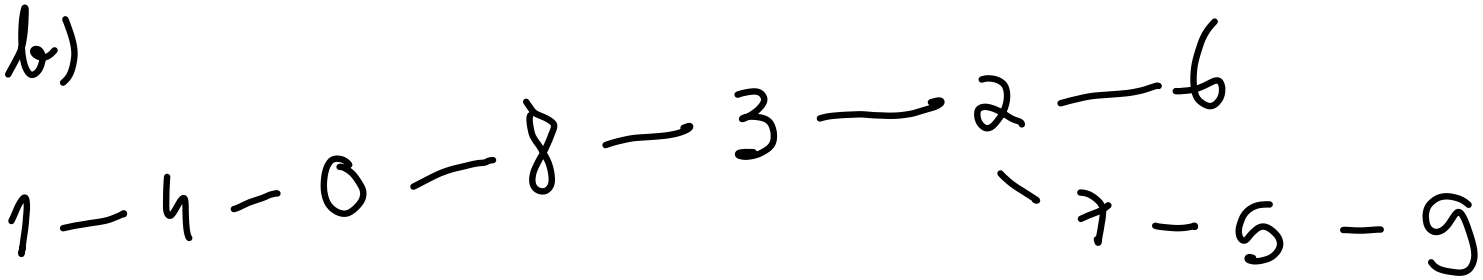
a)



Desenhe:

a) A árvore de travessia em largura com origem no vértice 3
b) A árvore de travessia em profundidade com origem no vértice 1

b)

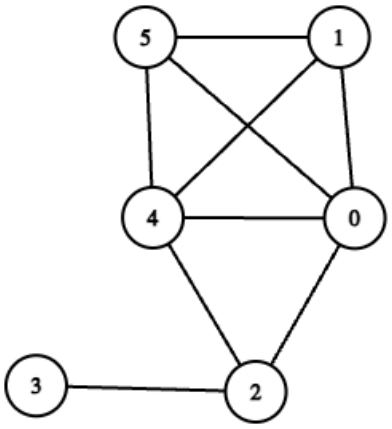


Algoritmos e Complexidade — Universidade do Minho
Teste 3 de Janeiro de 2024

Número: _____ Nome: _____

4. Grafos (5 valores)

Um “clique” de um grafo não-orientado é um conjunto de vértices que são todos adjacentes uns dos outros. No grafo em baixo, os conjuntos {0, 1, 4, 5} e {0, 2, 4} são exemplos de cliques.



Complete a seguinte definição da função `is_clique` que determina se o conjunto de vértices contidos no array `v`, de comprimento `k`, constitui um clique do grafo `g` (`NV` é o número de vértices de `g`).

```
struct edge { int dest; struct edge *next; };
typedef struct edge *GrafoL[MAX];

int is_clique (GrafoL g, int NV, int v[], int k) {
    ...
}
```

Analise o tempo de execução da função que escreveu.

Serão valorizadas as soluções mais eficientes.

```
int is_clique (Grafo g, int NV, int v[], int K) {
  for (int i = 0; i < K; i++)
    for (int j = i + 1; j < K; j++) {
      int a = v[i];
      int b = v[j];
      int connected = 0;
      struct edge *adj;
      for (adj = g[a]; adj != NULL; adj = adj->next)
        if (adj->dest == b) {
          connected = 1;
          break
        }
      }
      if (!connected) return 0;
    }
  return 1;
```

Existem $O(K^2)$ pares (a, b) de vértices
Para cada par é necessário percorrer a lista de adjacências de a cujo comprimento é pior caso NV e no melhor caso K.

Assim, a complexidade no pior caso é:
$O(K^2 \cdot NV)$

Uma possível outra solução é converter o grafo representado por um array de lista ligadas numa matriz

exemplo: