Lista 11 - MAC0122 Princípios de Desenvolvimento de Algoritmos - POLI

Prof. Ronaldo Fumio Hashimoto

1 Exercícios

1. Em uma árvore binária, quantos nós podem existir no nível 0? E no nível 1? E em um nível arbitrário i?

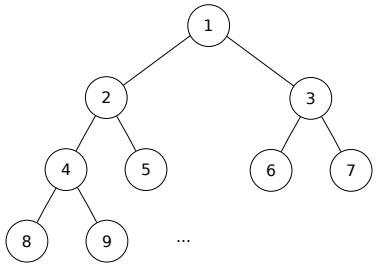
2. Repita o exercício anterior para uma árvore em que cada nó pode ter até k filhos.

3. Considere todas as configurações possíveis de árvores binárias com 5 nós. Qual é altura[1] mínima possível? E a altura máxima?

4. (Exercício retirado de [1]) Para o conjunto: $\{1, 4, 5, 10, 16, 17, 21\}$ de chaves, desenhe árvores binárias de busca[2] com alturas: 3, 4, 5 e 6.

5. (Exercício retirado de [2]) Implemente uma função que encontra o valor máximo e uma função que encontra o valor mínimo em uma árvore binária de busca.

6. Escreva uma função que adiciona nós a uma árvore binária criando nós da esquerda para a direita, de forma que um novo nível é criado apenas quando o nível anterior estiver cheio (segue a ordem mostrada pelos números na figura abaixo).



   Dica: note que a ordem de criação dos nós é exatamente igual ao índice de um *heap* binário.

[1] A altura de uma árvore é definida como o nível de seu nó mais profundo

[2] em uma árvore binária de busca, o filho esquerdo tem valor $\leq$ ao valor do nó pai e o filho direito tem valor $>$ que nó pai

7. (Exercício retirado de [1]) Implemente o algoritmo abaixo (percurso *in-order*) de forma iterativa (ou seja, sem usar recursão. Dica: use uma pilha).

```
typedef struct BTNode BTNode;
struct BTnode { int value; BTNode *left, *right; }

void inorder_print(BTNode *node)
{
   if (node != NULL) {
      inorder_print(node->left);
      printf("%d ", node->value);
      inorder_printf(node->right);
   }
}
```

8. Desenvolva uma estrutura de dados que mapeia uma *string* (`char *`) em um número inteiro. Essa estrutura deve utilizar uma árvore binária de busca de maneira semelhante à vista em aula e que possua a seguinte definição:

```
typedef struct BTNode BTNode;
  struct BTNode {
  char *key;
  int value;
  BTNode *left;
  BTNode *right;
};

typedef struct Map Map;
struct Map {
  BNode *root;
};
```

Então implemente as seguintes operações para a tipo de dados `Map`:

```
/* aloca memoria para um map */
Map *init_map();
/* insere um item com chave key e valor value */
void put(Map* map, char *key, int value);
/* devolve o valor associado a chave key */
int get(Map *map, char *key);
/* libera memoria de um map */
void destroy_map(Map** map);
```

Implemente funções extras para auxiliar seu desenvolvimento, se achar necessário e/ou altere as estruturas `BTNode` e `Map`, mas não altere as operações na interface. Sua implementação deve realizar a busca (`get`) e inserção (`put`) em tempo $O(\lg n)$ quando uma sequência de n itens são inseridos com chaves que não seguem uma ordem lexicográfica.

Referências

[1] T. H. Cormen, E. C. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms, third edition," p. 289, 2009.

[2] T. H. Cormen, E. C. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms, third edition," p. 293, 2009.