Implementação de Árvore Rubro-Negra com Inserção e Remoção - CI1057 - 2025/1

Pedro Pierobon Marynowski
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
pedro.pierobon@ufpr.br

Pedro Endrigo
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
endrigopedro@ufpr.br

***Resumo*—Este trabalho apresenta uma implementação funcional de uma árvore Rubro-Negra (Red-Black Tree), com suporte às operações de inserção e remoção. A entrada é processada via `stdin`, e a saída formatada é gerada via `stdout` conforme especificado. A estrutura foi implementada em linguagem C e alocada dinamicamente.**

I. Introdução

Árvores Rubro-Negras são estruturas de dados balanceadas binárias que garantem tempo logarítmico para operações de busca, inserção e remoção. Este projeto teve como objetivo aplicar os conceitos teóricos da estrutura, conforme apresentados na disciplina de Algoritmos III, utilizando a linguagem C.

II. Metodologia

A implementação segue o algoritmo clássico descrito no livro Algoritmos Teoria e Prática 3º edição, Thomas H. Cormen, com as devidas adaptações ao contexto da tarefa.

II.I Estruturas

Foram adicionadas duas estruturas. A primeira delas representa um sentinela "árvore"que contêm a raiz da árvore e um nodo nulo que representa filhos pretos das folha. A segunda delas é o nodo que guarda a chave, cor, ponteiros para filho esquerdo, direito e pai.

II.II Inserção

A operação de inserção foi implementada com o uso de um nó `t.nil`. Após a inserção, é realizada uma rotina de correção (*insert-fixup*) com rotações e recolorações para manter as propriedades da árvore Rubro-Negra. Foi utilizada abordagem recursiva para as operações e não foi implementado o tratamento iterativo para o caso do duplo preto.

II.III Remoção

A remoção foi realizada utilizando o algoritmo de *transplante*, substituindo o nó a ser removido por seu antecessor quando necessário. O balanceamento pós-remoção também segue a lógica do fix-up descrito por Cormen, com aplicação de rotações e recolorações.

II.IV Saída

A saída da árvore ocorre após o processamento completo da entrada. É realizado um percurso *em ordem*, imprimindo para cada nó uma linha no formato `(valor, nível, cor)`, onde a cor é codificada com 0 para preto e 1 para vermelho. O nível é calculado recursivamente durante a impressão:

```
void print_tree(struct node *root,
                int h, struct tree *t) {
    if (root == t->nil) return;
    print_tree(root->left, h + 1, t);
    printf("%d,%d,%d\n", root->key, h, root->color);
    print_tree(root->right, h + 1, t);
}
```

II.V Exclusão da Árvore

Para evitar vazamentos de memória, a árvore é desalocada ao final da execução por meio de um percurso em *pós-ordem*. A função percorre recursivamente os filhos antes de liberar o nó atual:

```
void rb_destroy(struct node *n, struct tree *t){
    if(n == t->nil) return;
    rb_destroy(n->left, t);
    rb_destroy(n->right, t);
    free(n);
}
```

Essa abordagem garante que nenhum nó seja liberado antes de seus descendentes, evitando acesso a memória já liberada.

III. Compilação e Execução

A compilação do projeto é feita por meio de um `Makefile`, com o comando:

```
$ make
```

Isso gera o executável `myrb`, conforme exigido. A execução se dá pelo redirecionamento de arquivos:

```
$ ./myrb < teste.in > teste.out
```

Além disso, foi implementado o comando make test que compara os testes com as saídas do programa indicando se a execução foi bem sucedida, e caso tenha falhado mostra as diferenças.

```
$ make test
```

E para limpar os arquivos executáveis e os resultados dos testes do make test, foi implementado o comando:

```
$ make clear
```

IV. Conclusão

A árvore Rubro-Negra foi implementada com sucesso, respeitando as propriedades da estrutura e o formato de entrada e saída especificados no enunciado. A abordagem com sentinela e nó nulo simplificou o tratamento de casos base. A implementação passou corretamente pelos testes previstos.