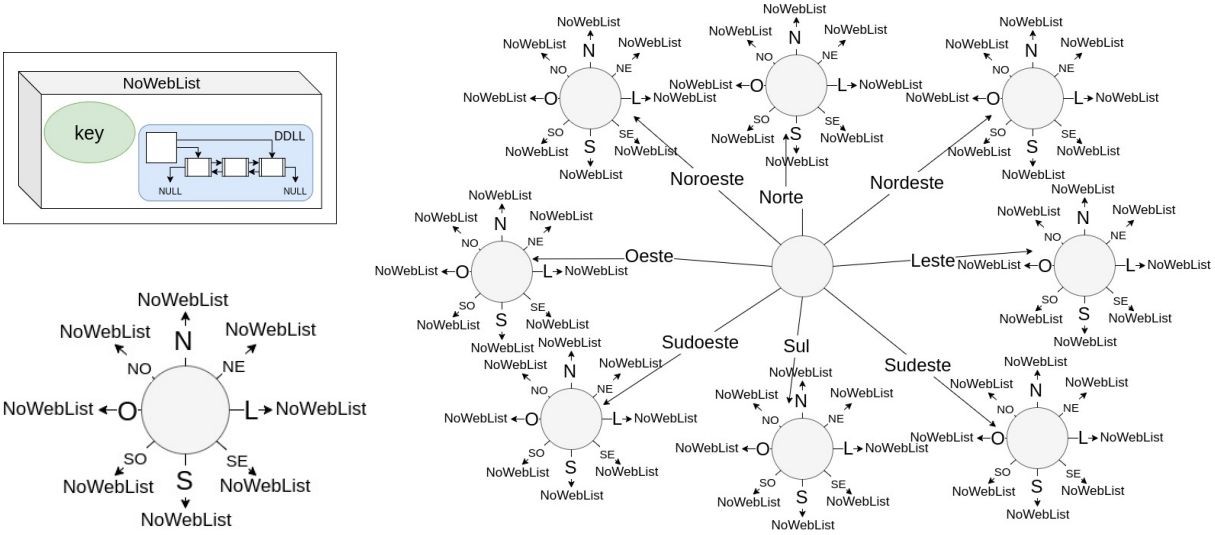
PAA/2024: *WebList* - Teia de listas em Árvore Octógona

Maurício Aronne PILLON

Versão Preliminar: 8 de outubro de 2024*

Ao longo da disciplina, estudamos as estruturas de dados clássicas aplicadas à computação, bem como realizamos a análise assintótica de algumas funções. Até então, os trabalhos da disciplina consolidaram conhecimentos prévios sobre estruturas de dados, sem abordar a modelagem criativa de novas estruturas. A concepção de estruturas adequadas ao problema impacta diretamente no desempenho e na operacionalização de aplicações. A estrutura em teia é uma variação de árvore em que o nó raiz é o nó central da teia (Figura 2). A *Teia de listas em Árvore Octógona* (WebList) é uma árvore completa, cheia e estática. A característica estática se deve pela necessidade de definição do número de níveis na criação e ausência de função que permita a alteração deste número. Portanto, uma WebList que foi criada com nível 0 sempre terá 8 nós folhas e uma com nível 1, 64 nós folhas. Embora o número de nós da WebList e chaves sejam estáticos, a quantidade de dados armazenados é dinâmico. Pois, a cada nó folha associamos um NoWebList (Figura 1a). Um NoWebList é constituído de, pelo menos, uma chave (*key*) e um ponteiro para o descritor de uma lista do tipo *Dynamic Doubly Linked List* (DDLL). O grau da WebList e de todos os seus nós é sempre 8 (oito). O número de *NoWebList*s associadas a uma WebList é descrito pela fórmula 8^{n+1}, cujo n é o nível da WebList.



(a) Nível 0 (8 chaves DDLL).

(b) Nível 1 (64 chaves DDLL).

Figura 1: Estrutura de Teia

1 Descrição

A estrutura clássica de árvore, quando balanceada, é uma das estruturas de dados mais eficientes para busca de elementos. O objetivo deste trabalho é consolidar o conhecimento sobre estruturas do tipo árvore e aplicá-lo a uma estrutura adaptada, denominada de WebList. As principais características da WebList são:

- homogeneidade, ou seja, o tipo dos dados é único e definido na criação da estrutura;
- balanceamento, a soma do número de dados incluídos nas DDLLs por nó da árvore não pode variar mais do que 1 elemento; e

*Sujeita revisão! Aguardando solicitação esclarecimentos e retorno/sugestão dos alunos sobre as regras.

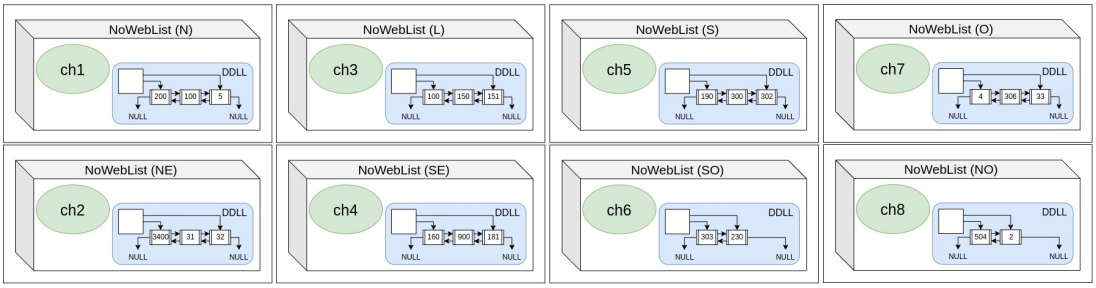


Figura 2: Um exemplo de distribuição de dados inteiros na WebList balanceada.

- generalidade, qualquer dado que possua uma regra de precedência pode ser armazenado na estrutura.

Algoritmo 1: API pública da WebList

```
#define SUCCESS 0
#define FAIL -1

typedef struct weblist *pweblist, **ppweblist;

// Funcoes operacionais
int cWL (...); // criar a estrutura;
int dWL (...); // destruir a estrutura;

// Funcoes focada nos dados
int iDado (...); // inserir um novo dado na estrutura;
int rDado (...); // remover um dado da estrutura;
int bDado (...); // buscar um dado na estrutura;
int pLista (...); // percorrer a lista de dados;

// Funcoes focada nos nos-folha
int cpLista (..., int chave, ppDDLL retorno); // retornar uma
    copia da DDLL correspondente a chave;
int sbLista (..., int chave, pDDLL novaLista); // substituir a
     lista DDLL correspondente a chave pela lista recebida por
     parametro (novaLista);
int rmLista (..., int chave, ppDDLL rmLista); // retornar a
    lista 'rmLista' por parametro e remover a lista DDLL
    correspondente a chave;
int nvLista (..., int chave); // criar uma DDLL vazia para a
    chave recebida como parametro;


// Funcoes da WebList
int nroEleNoFolha (..., int* retorno); // retornar o numero de
     elementos em um no-folha especifico (soma de elementos de
     cada lista do no-folha)
int nroNoFolha (..., int* retorno); // retornar o numero total
     de nos-folha da estrutura
int nroEleWL (..., int* retorno); // retornar o numero total
    de elementos cadastrados na webList
int lstChaves (..., ppDDLL retorno); // retornar uma lista com
     todas as chaves da WebList.
int WLbalanceada (...); // retornar SUCCESS se a webList
    estiver balanceada e, FAIL, caso contrario.
```

A WebList é composta pelas operações de construção e destruição, acesso e manipulação dos dados, conforme descrito na API pública (Algoritmo 1). As chaves, identificadas como *key* na Figura 2, são únicas, estáticas, e devem ser definidas no momento de criação da WebList. Inicialmente, essas chaves imutáveis estão associadas a DDLLs vazias, as quais podem ser modificadas, substituídas ou removidas. Em caso de remoção de uma DDLL, a chave correspondente fica associada a uma lista inexistente (apontando para $NULL$). A operação de criação de uma DDLL (nvLista) só é possível se houver uma chave associada a uma lista inexistente.

A plataforma disponibiliza acesso a uma DDLL padrão do sistema, cuja API pública é descrita no Algoritmo 2. A entrada de dados é realizada por meio da leitura de um arquivo passado como parâmetro

no *prompt*, i.e., *./main.exe dados1.in* e *./main.exe dados2.in*. O arquivo deve ser em formato de texto, e os dados devem ser separados por espaços (conforme ilustrado na Figura 3).

```
20007 16597 3641 16140
8622 29947 9148
13861 27783
6104 12185 8843
1750 30231 18761 15407 14890
25032 11469 17267
14033
23155 11582
21359 10487
28663 7913
```

(a) Entrada de inteiros.

```
Livros e flores

Teus olhos são meus livros.
Que livro há aí melhor,
Em que melhor se leia
A página do amor?

Flores me são teus lábios.
Onde há mais bela flor,
Em que melhor se beba
O bálsamo do amor?

Machado de Assis
Falenas. Rio de Janeiro: B. L. Garnier, 1870.
```

(b) Entrada de *Strings*

Figura 3: Exemplos de Arquivos de entrada

Algoritmo 2: API pública da DDLL

```c
#include <string.h>
#define SUCCESS 0
#define FAIL 1

typedef struct DDLL *pDDLL, **ppDDLL;

// Funcoes basicas de uma DDLL
int cDDLL(ppDDLL pp, int sizedata);
int dDDLL(ppDDLL pp);
int cleanDDLL(pDDLL p);

int iBegin(pDDLL p, void *element);
int iEnd(pDDLL p, void *element);
int iPosition(pDDLL p, int N, void *element);

int rBegin(pDDLL p, void *element);
int rEnd(pDDLL p, void *element);
int rPosition(pDDLL p, int N, void *element);

int sBegin(pDDLL p, void *element);
int sEnd(pDDLL p, void *element);
int sPosition(pDDLL p, int N, void *element);

// Funcoes adicionais
int empty(pDDLL p);
int countElements(pDDLL p);
```

O formato de saída dos dados dependerá do algoritmo de manipulação utilizado por cada equipe. Portanto, considere que o conjunto $C_n = \{3, 2, 5, 4, 1\}$ de dados, cujo $n = 5$, é incluído na WebList por meio de um arquivo de entrada. Após a leitura dos dados, o usuário chama quatro vezes a operação *rDado*(...) para os seguintes dados: 5, 2, 3 e 4. Em seguida, o usuário chama *iDado*(...) para o dado 15. As respostas corretas para este exemplo são: $\{15, 1\}$ ou $\{1, 15\}$. Somente os dados de entrada, que não foram excluídos ou incluídos ao longo das manipulação, devem aparecer. Qualquer outro texto de depuração ou explicação não pode estar na saída da aplicação.

2 Procedimentos do Trabalho Final

Este trabalho final é composto por duas etapas:

1. **Desenvolvimento da biblioteca WebList**: Esta fase é assíncrona, em duplas e com consulta. Terá a duração mínima de 15 dias e não será atribuída nota.

2. **Adequação da WebList**: Esta fase é individual e sem consulta[1]. Contará com $4h$ para desenvolvimento em sala de aula. O conceito do trabalho final será o resultado da VPL desta avaliação.

Na etapa 2 será considerado o tempo de execução das soluções. A versões das equipes serão comparadas entre si e classificadas em 3 faixas, denominadas A, B e C. Preocupem-se com a elaboração de algoritmos com complexidades assintóticas mais eficientes. Apresente a análise assintótica do seu algoritmo. Todas as equipes terão acesso a mesma biblioteca DDLL, portanto, a complexidade interna desta biblioteca pode ser desconsiderada na análise.

[1] Exceto código da biblioteca WebList desenvolvida por cada dupla na fase 1.