

Interação Ligante-Proteína

Nome: Luiz Otávio Resende Vasconcelos
Matrícula: 2015042142

1 . Introdução

O presente documento visa descrever a atividade prática desenvolvida na disciplina AEDS-II no curso de Ciência da Computação 2015/2.

Foi solicitado a que “mapeássemos” as posições dos átomos de uma proteína.

Após isso, recebemos posições de ligantes que será comparado com os átomos de proteína e então descobrir uma possível interação dependendo da distância entre os dois.

O objetivo era a implementação de estruturas de dados abstratas(TAD) em forma de Árvore(octree) para fins de busca → com todos os átomos da proteína mapeados(distribuídos na árvore) se faz necessário muito menos comparações para encontrar os que estão próximos.

Também foi necessário implementar algoritmo de ordenação.

O presente trabalho pode ser encontrado em

→ https://github.com/LuizOtavio/interacao_ligante_proteina

2 . Implementação

Estruturas abstratas de dados utilizadas:

→ Estrutura de um cubo:

```
// - max/min [0] = x
// - max/min [1] = y
// - max/min [2] = z
typedef struct cube_str {
    double max[3];
    double min[3];
} Cube;
```

→ Estrutura da folha:

```
typedef struct leaf_st {
    // boolena que verifica se a folha é de fato
    // uma folha ou um nó (se == 1 -> folha)
    int is_leaf;
    // ponteiro que aponta para essa struct
    // ou seja, dentro de Leaf, temos um
    // "array de Leaf" com 8 filhos.
    struct leaf_st* sons[8];
    // cada folha também terá uma proteína.
    Protein protein;
    // cada folha também terá coordenadas de máx e mín
    // para saber-mos onde procurar por determinada
    // proteína.
    Cube coords;
} Leaf;
```

→ Estrutura da proteína:

```
typedef struct prot_st {
    // point[] refere-se à cordenada da da Proteína:
    // Sendo:
    // - point[0] = x
    // - point[1] = y
    // - point[2] = z
    double point[3];
    int isSet;
} Protein;
```

→ Estrutura do ligante:

```
typedef struct lig_st {  
    // point[] refere-se à cordenada do Ligante:  
    // Sendo:  
    // - point[0] = x  
    // - point[1] = y  
    // - point[2] = z  
    double point[3];  
    char name[11];  
    int sum;  
    struct lig_st *prox;  
} Ligante;
```

→ Lista para ordenar os ligantes:

```
typedef struct ligante_list_st{  
    Ligante *header;  
} LiganteList;
```

Funções:

→ Leaf* **findLeaf**(Leaf* root, double* point);

Encontra a folha na árvore a partir do 'point'

Caso encontre um nó, é chamada novamente recursivamente.

→ void **freeLeafs**(Leaf *root);

Limpa toda a árvore ao final da execução.

→ void **setProtein**(Leaf* leaf, Protein new_protein);

Pega as coordenadas da proteína inserida no STDIN e seta numa nova proteína.

→ void **setCubeCoords**(Leaf *leaf);

Recebe no STDIN as coordenadas do um novo cubo e insere nele.

→ void **splitCubes**(Leaf* leaf, double half_edge[]);

Divide uma folha em folhas e a torna um nó.

→ Ligante `getNewLigante(char* str);`

Cria um novo ligante e insere nele as coordenadas inseridas no STDIN

→ void `insertOnMiddle(Ligante* new_ligante, Ligante* aux);`

Insere o ligante no meio da lista de ligantes ordenada.

→ void `putLiganteOnLiganteList(Ligante new_ligante, LiganteList* ligant_list);`

Insere o ligante na lista de ligantes de forma ordenada.

→ Protein `getNewProtein(char* str);`

Cria um nova proteína e insere nela as coordenadas inseridas no STDIN

→ void `getPointsInsideBox(Leaf* leaf, Ligante lig, double cubelig_edge, int *sum);`

Função solicitada na descrição.

Conta e soma de forma recursiva todas as interações Ligante/Proteína.

→ void `printResult(LiganteList* ligant_list);`

Função de STDOUT exibe na tela os resultados

3 . Testes

A maioria dos testes são feitos com 'printf()' e utilizando o GDB. O primeiro teste com imput executado fora afim de descobrir se a árvore estava armazenando corretamente os átomos de proteína.

→ primeiro fora inserido valores pequenos e fáceis de calcular

Ex: min<0, 0, 0> máx<10, 10, 10>

após verificar a inserção correta na árvore

→ verifiquei se estava contando as interações corretamente.

Novamente: inserindo valores redondos, 3 proteínas e 1 ligante. Valores que eu pudesse calcular facilmente e ver se batem.

→ utilizei também os arquivos fornecidos no Moodle.

4. Análise de complexidade

Leaf* **findLeaf**(Leaf* root, double* point) ;

→ função recursiva. No melhor caso, faço apenas $O(1)$: quando a folha a ser encontrada é a raiz, por exemplo. No pior caso preciso fazer $n \log_8(n) \rightarrow O(n \log n)$.

void **freeLeafs**(Leaf *root) :

→ novamente. Função recursiva que desaloca todos os filhos. No melhor caso, faço apenas $O(1)$: No pior caso preciso fazer $n \log_8(n) \rightarrow O(n \log n)$.

void **setProtein**(Leaf* leaf, Protein new_protein) :

→ função recursiva. Contém duas chamadas recursivas $n \log n \rightarrow$ melhor caso.

Porém, contém a função , e a chama, **splitCubes** que é $O(n^2)$ no pior caso.

void **insertOnMiddle**(Ligante* new_ligante, Ligante* aux);

void **setCubeCoords**(Leaf *leaf)

Ligante **getNewLigante**(char* str) ;

void **insertOnMiddle**(Ligante* new_ligante, Ligante* aux);

Protein **getNewProtein**(char* str) ;

→ $O(1)$ apenas atribuições.

void **putLiganteOnLiganteList**(Ligante new_ligante, LiganteList* ligant_list)

→ função com 1 loop. Chama a função **insertOnMiddle** que é $O(1)$, só faz atribuições.

Logo ela é $O(n)$.

void **getPointsInsideBox**(Leaf* leaf, Ligante lig, double cubeLig_edge, int *sum) ;

→ novamente. No melhor caso, o cubo do ligante está apenas dentro de um cubo da árvore.

No pior caso preciso fazer $n \log_8(n) \rightarrow O(n \log n)$.

void **printResult**(LiganteList* ligant_list);

→ um loop : $O(n)$

5. Conclusão

Árvore é um excelente método de busca. Te permite dividir o espaço em subespaços e você precisará fazer muito menos comparações do que você teria que fazer se não a implementasse.

Após o entendimento na disciplina, consigo pressupor muitos problemas práticos que envolvem pesquisa sendo muito melhores selecionados empregando árvores de buscas.

6 . Referências

→ Projetos de Algoritmos - com implementações em Java e C++ Nivio Ziviani

→ Slides - minhaUFMG Moodle.