# Trabalho Prático 3

Interação Ligante Proteína

**Entrega:** –/–/2015

17 de novembro de 2015

## 1 Descrição

De forma simplificada, *ligantes* são pequenas moléculas que interagem com *proteínas*. Essas moléculas podem ter, entre outras funções, a função de fármaco que se lposiciona dentro do sítio catalítico de uma enzima inibindo-a e não permitindo que desempenhe sua função biológica. Veja um exemplo do complexo *proteína-ligante* na Figira 1.

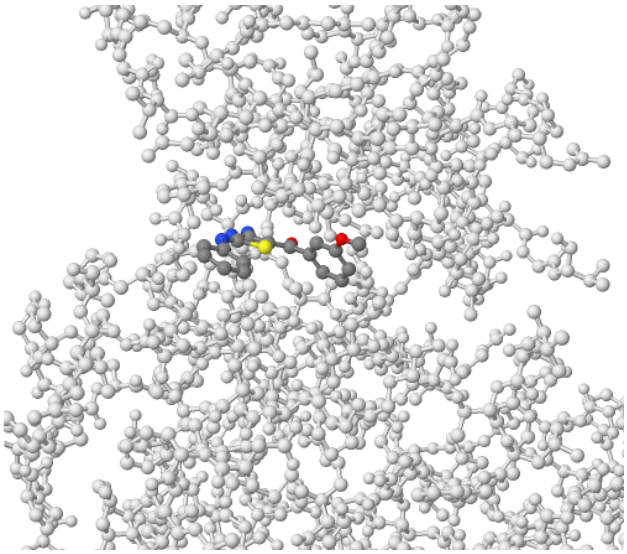

Figura 1: Exemplo de uma proteína interagindo com um ligante. Os átomos do ligante estão coloridos enquanto os da proteína estão em tom de cinza mais claro.

Estratégias computacionais para o desenvolvimento de fármacos envolvem o cálculo de possíveis interações que possam ser estabelecidas entre a proteína e o ligante. No caso da busca de um fármaco que interaja fortemente com uma proteína, normalmente, o que se busca são ligantes capazes de estabelecer um grande número de ligações, tendo maior afinidade com a proteína.

Uma das maneiras mais simples de se estudar essa afinidade entre o ligante e a proteína é através da contagem do número de átomos da proteína que estão a uma certa distância pré-definida do ligante. Isto pode ser feito, criando-se um cubo imaginário, com dimensão $d$, centrado no átomo do ligante e verificando quantos átomos da proteína se encontram dentro deste cubo.

# 2 O que deve ser feito

Neste trabalho vocês devem desenvolver um programa que:

1. Receba as coordenadas dos átomos de uma proteína e de vários ligantes;
2. Calcule a interação entre a proteína e cada ligante;
3. Retorne uma lista ordenada dos ligantes, de acordo o número de interações que estabelece com a proteína

Dado um cubo $k$ centrado em um átomo $a_l$ e com arestas de dimensões $d$, o número de átomos $a_p$ da proteína $p$ que interagem com este átomo é dado por:

$$f(a_l, p) = \sum_{\forall\ a_p\ \in\ p} \begin{cases} 1 & \text{se } a_p \text{ está no interior de } k \\ 0 & \text{caso contrário} \end{cases} \tag{1}$$

A interação total entre uma proteína $p$ e um ligante $l$ deve ser calculado da seguinte maneira:

$$Interact(p, l) = \sum_{\forall\ a_l\ \in\ l} f(a_l, p); \tag{2}$$

A fórmula 1 indica que para cada átomo do ligante, devemos percorrer todos os átomos da proteína verificando se ele está ou não dentro do cubo. Geralmente, o ligante possui algumas dezenas de átomos e a proteína alguns milhares, e por isso, esta busca exaustiva pode ser relativamente lenta.

Desta forma, neste trabalho deverá ser utilizado uma estrutura de dados especial que permite uma busca eficiente por átomos dentro de um determinado cubo. A estrutura a ser implementada se chama octree e é um tipo especial de árvore.

## 2.1 Octree

Dado um subespaço do $R^3$ representado por um cubo, uma octree é utilizada para subdividir recursivamente este em cubos menores. Ela pode ser utilizada para armazenar pontos em 3 dimensões e permitir que buscas por vizinhos próximos ou por pontos dentro de um determinado cubo sejam feitas de maneira eficiente. Seu processo de criação é o seguinte

1. Inicialmente, define-se um cubo delimitador no espaço $R^3$ para ser o nó raiz da octree;

2. Esta raiz será então utilizada para representar o espaço definido por este cubo delimitador;
3. Este espaço é agora dividido em 8 octantes iguais, cada um representando um sub-cubo da cubo raiz. Estes octantes são filhos do nó raiz.
4. Cada sub-cubo será agora subdivido em 8 novos sub-cubos ainda menores;
5. O processo se repete a cada nível da árvore;

De maneira geral, em uma octree, cada nó interior (não folha) representa uma região do espaço definida por um cubo delimitador e possui 8 filhos, que subdividem este cubo em 8 octantes iguais. A figura 2 ilustra a divisão recursiva de um cubo e uma octree associada a esta divisão.

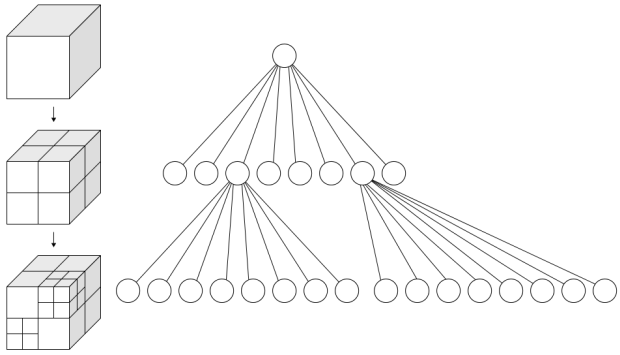

Figura 2: Esquerda uma divisão recursiva de um cubo em octantes. Direita: Uma octree associada a esta divisão

No contexto deste trabalho, os pontos serão armazenados nas folhas da octree e cada folha só poderá armazenar um único ponto. A criação da octree deverá ser feita sob-demanda durante a inserção dos pontos. Assim, inicialmente a árvore será formada somente pela raiz, os demais nós serão criados a medida que forem necessários.

Durante a inserção de um ponto $p$ em um nó da octree podem acontecer três possibilidades:

- **O nó é um nó interior (não folha):** Como os pontos são armazenados somente nas folhas, deve-se encontrar o filho adequado para este ponto, e chamar a função de inserção recursivamente neste filho;
- **O nó é uma folha e está vazia:** Este é o caso mais simples. Basta armazenar o ponto $p$ neste nó.
- **O nó é uma folha mas não está vazio:** Como cada folha só é capaz de armazenar um único ponto e já existe um ponto $p0$ armazenado, este nó deve ser convertido em um nó interior. Para isto, a região do espaço que ele representa (um cubo) deve ser quebrada em 8 octantes (sub-cubos) de mesmo tamanho. Estes 8 octantes serão filhos deste nó. Com o espaço sub-divido, deve-se chamar recursivamente a função de inserção para o ponto antigo $p0$ e para o ponto $p$. Note que foi necessário redistribuir $p0$ pois o nó que ele se encontrava foi transformado em um nó interior.

Outra função que será necessária é a função $getPointsInsideBox$. Esta função recebe um cubo delimitador como parâmetro e retorna quais dos pontos armazenados na octree que estão dentro deste cubo.

### 2.2 Octree e a interação ligante-proteína

Neste trabalho a octree deverá ser utilizado da seguinte maneira:

- Cada átomo da proteína deverá ser armazenado em uma octree;
- A função $getPointsInsideBox$ deverá ser utilizada para calcular o número de átomos da proteína que interagem com cada átomo do ligante.

**Não se esqueça que você também deverá implementar um algoritmo de ordenação para ordenar os ligantes de acordo com o número de interações com a proteína.**

## 3 Entrada e saida

Como especificado, o seu programa deverá ser capaz de receber os átomos de uma proteína e de vários ligantes, calcular a interação entre a proteína e cada ligante e retornar uma lista ordenada dos ligantes segundo o número de interações que estabelece com a proteína (do menor para o maior).

Os dados de entrada utilizados, são dados reais retirados do *Protein Data Bank*[1] (PDB). Estes dados foram gerados a partir de vários experimentos e as posições dos átomos foram capturadas. Assim, apesar de ser a mesma proteína em cada um dos experimentos, estes átomos estão em posições diferentes (a proteína pode estar rotacionada de infinitas maneiras diferentes) e em alguns casos esta proteína pode apresentar uma quantidade maior ou menor de átomos. Portanto, **para cada ligante será informada novamente a posição de cada átomo da proteína e o espaço de dimensões da árvore.**

A entrada será realizada através da entrada padrão. O primeiro valor informado indica qual o comprimento das arestas do cubo que será criado ao redor dos átomos dos ligantes.

A seguir, para cada par proteína-ligante será informado:

- Uma linha contendo o nome do ligante;
- Uma linha contendo as coordenadas do ponto que representa o menor vértice do cubo delimitador do espaço de busca do programa (menores coordenadas de x,y e z). Figura 3;
- Uma linha contendo as coordenadas do ponto que representa o maior vértice do cubo delimitador do espaço de busca do programa (maiores coordenadas de x,y e z). Figura 3;
- As demais linhas apresentam as informações dos átomos e são compostas por: um identificador da estrutura a qual o átomo pertence (proteína ou ligante), o tipo do átomo e as coordenadas x,y e z.

[1] http://www.rcsb.org/pdb/home/home.do

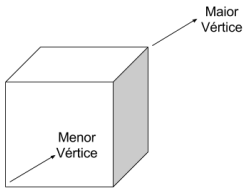


Figura 3: Ilustração do menor e do maior vértice de um cubo

- A entrada termina com um linha contento ”-1”.

Exemplo (desconsidere os comentários):

```
15.00        //comprimento das arestas do cubo a ser criado ao redor dos atomos do ligante
Nome: 3QQF                //nome do ligante
-25.5 -31.00 -20.05       //menor vértice do cubo que delimita o espaço de busca
39.5 34.00 44.95        //menor vértice do cubo que delimita o espaço de busca
PROTEINA MET1:N 11.054 23.6985 1.288
PROTEINA MET1:CA 12.255 24.5605 1.479
...                       //várias outras linhas com átomos da proteína
PROTEINA ASN3:OD1 13.473 26.4795 -3.342
PROTEINA PHE4:N 9.014 26.8715 -1.592
PROTEINA ASN3:ND2 13.272 27.1025 -5.488
LIGANTE X07:C1 -1.528 12.5915 2.876
LIGANTE X07:C2 -0.237 12.1365 3.217
...                       //várias outras linhas com átomos do ligante
LIGANTE X07:N11 2.097 11.8815 2.64
LIGANTE X07:O12 3.14 12.0475 1.798
Nome: 3R7U               //nome do outro ligante
-20.5 -15.70 -10.05      //menor vértice do cubo que delimita o espaço de busca
28.9 33.70 39.35       //maior vértice do cubo que delimita o espaço de busca
PROTEINA MET1:N 1.054 13.6985 11.288
PROTEINA MET1:CA -12.255 -14.5605 1.479
...                       //várias outras linhas com átomos da proteína
PROTEINA ASN3:OD1 3.473 28.5 -3.342
PROTEINA PHE4:N -9.014 6.8715 -1.592
PROTEINA ASN3:ND2 13.272 27.1025 -5.488
LIGANTE X07:C1 -15.528 -12.5915 2.876
LIGANTE X07:C2 -0.237 22.1365 -3.217
...                       //várias outras linhas com átomos do ligante
LIGANTE X07:N11 -2.097 11.8815 -2.64
LIGANTE X07:O12 13.14 2.0475 1.798
Nome: 3S00               //nome do outro ligante
...
Nome: 3QL8               //nome do outro ligante
...
-1
```

A saída deve ser a saída padrão e deve apresentar de maneira ordenada (do menor para o maior) o nome do ligante e o grau de interação com a proteína. Exemplo:

```
3R7U 137
3QL8 142
3QQF 173
3S00 189
```

# 4 Detalhes de Implementação

**Como dito, para cada ligante são informadas novamente as posições dos átomos da proteína. Assim, para cada ligante deve-se utilizar uma nova octree par armazenar os novos átomos da proteína**

A boa prática de programação indica que o código desenvolvido deve ser legível, bem modularizado e comentado.

- Legível: Nomes intuitivos para funções e variáveis, identação, quebras de linhas adequadas, etc.
- Modularização: O código deve ser decomposto em funções e estruturas semanticamente coesas.
- Comentários: Trechos relevantes e/ou complexos do código devem ser comentados para facilitar a compreensão do código.

Ao final da execução, todas as estruturas alocadas devem ser liberadas (utilizando *free*).

Uma boa ferramenta para avaliar se a alocação e liberação de memória foi realizada corretamente é o *valgrind*[2].

# 5 O que deve ser entregue

Você deve submeter uma documentação de até 10 páginas contendo uma descrição da solução proposta, detalhes relevantes da implementação, além de uma análise de complexidade de tempo dos algoritmos envolvidos e de complexidade de espaço das estruturas de dados utilizadas.

A documentação deve conter:

1. Introdução: descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
2. Implementação: descrição sobre a implementação do programa. Devem ser detalhados as estruturas de dados utilizadas (de preferência com diagramas ilustrativos) e o funcionamento das principais funções.
3. Estudo de complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados.

[2] http://cs.ecs.baylor.edu/~donahoo/tools/valgrind/

4. Avaliação experimental: estudo do comportamento empírico da solução proposta. Vocês deverão analisar o impacto da dimensão do problema no tempo de execução do algoritmo.

5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.

6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet, se for o caso.

Utilizando o sistema VPL do MOODLE, você deverá submeter todos os seus códigos fonte (todos os arquivos .c e .h) do simulador.
Também deverá ser entregue arquivo **.pdf** contendo a documentação (máximo 10 páginas).

# 6 Comentários gerais

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.

- Clareza, indentação e comentários no programa também vão valer pontos.

- Siga atentamente os formatos de entrada e saída especificados. Fique atento ao número de casas decimais.

- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiverem problemas para entender o que está escrito aqui, pergunte aos monitores.

- Você **não** precisa de utilizar uma IDE como Netbeans, Eclipse, Code::Blocks ou QtCreator para implementar esse trabalho prático. No entanto, se o fizer,

  **não** inclua os arquivos extras que foram gerados por essa IDE em sua submissão.

- Trabalhos copiados serão penalizados com a nota zero. Serão utilizadas ferramentas automáticas de detecção de cópias.

- Penalização por atraso: (2d - 1) pontos, em que d é o número de dias de atraso.