MAC 2166 – Introdução à Ciência da Computação
Grande área Elétrica – Primeiro Semestre de 2021

Quarto Exercício-programa — Politech e a evolução dos Poli technicum viri

versão com exemplos de execução, programa principal e alterações nas especificações de `BuscaMarcadores` *e de* `HaIncompatibilidade`.

Um vírus de computador é um programa que tem a propriedade de eventualmente infectar outros programas e/ou sistemas computacionais quando executado, copiando-se a si mesmo e instalando-se nestes novos hospedeiros. Diferentes técnicas podem ser empregadas por estes programas visando ocultar sua presença, inclusive a criptografia e a geração de variantes diversas através de mutações, como se dá no caso de vírus polimórficos, que modificam o próprio código. Uma forma de produzir estas modificações pode ser feita através da execução de algoritmos genéticos, que se inspiram no sistema de replicação das cadeias de DNA. Eventualmente algum erro neste processo provoca uma mutação, produzindo pequenas modificações pontuais nalguns poucos lugares do programa. Este EP segue um modelo de vírus de computador polimórfico inspirado na biologia que é bastante específico para a nossa disciplina. Face às múltiplas técnicas normalmente empregadas, nosso vírus politécnico será carinhosamente chamado de *Politech*, um clássico *Poli technicum virus*. De nosso particular interesse é o levantamento de características que, em condições ideais, nos permitem inferir a história da evolução dos Politech, bem como verificar se existem eventuais incompatibilidades que nos impedem de realizar perfeitamente esta inferência.

Se nas sequências biológicas temos cadeias nucleicas como RNA e DNA, nosso *Politech* é formado por um *vetor não natural de opcodes*, ou simplesmente *ONA* (*Opcode non Natural Array*). Lembra que *array*, um arranjo organizado, uma armada, é o termo do inglês usado em computação para designar um vetor. Ademais, um *opcode* é abreviatura de um código numérico de uma operação realizada na CPU. Por limitações visuais, os opcodes utilizados pelos Politech são inteiros entre 33 e 126, sendo representados pelo caractere correspondente na tabela ASCII. Estes são justamente os caracteres imprimíveis e visualizáveis da tabela. Por comporem os Politech são aqui chamados de *politípicos*, e denotamos seu conjunto por Σ: o conjunto de caracteres da tabela ASCII com valores numéricos no intervalo fechado de 33 a 126.

Figura 1: Logotipo de um *Poli technicum virus*, um Politech.

Assim, a partir de agora, trataremos um Politech e seu correspondente ONA como uma cadeia de caracteres de Σ.

Na biologia, o trecho de um gene numa cadeia de DNA possui uma funcionalidade privilegiada ao codificar uma proteína a ser usada nas reações bioquímicas necessárias à vida. Também no caso de um agente infeccioso como um vírus ou uma bactéria devemos identificar sua presença dentro das cadeias de DNA (ou RNA) que portam sua informação genética. É bem sabido que o *código genético* transcreve $4^3 = 64$ triplas de quatro nucleotídeos nos vinte aminoácidos a compor as proteínas e cada tripla destas é chamada de *códon*. Pois cada gene presente numa

sequência de DNA começa com o códon `ATG` — que faz o papel de um *marcador de início* do gene — e termina com um dos três códons de parada (`TAA`, `TGA` ou `TAG`), que fazem o papel de um *marcador de fim* do gene. Observa que nem toda tripla de nucleotídeos que ocorra dentro de um gene e que forme um códon de parada é reconhecida como término do gene. Somente são reconhecidos os códons que ocorrem em posições cuja diferença com a posição do marcador de início do gene é um múltiplo de três.[1] O primeiro códon de parada reconhecido é o marcador de fim do gene considerado.

Em paralelo, uma função computacional desempenha um papel semelhante ao de um gene e a identificação das linhas de código ou dos trechos de opcodes que formam as funções algorítmicas é de grande importância na compreensão do funcionamento de qualquer programa, o que vale também para um Politech. Para nossa felicidade, os conjuntos de instruções de uma CPU foram concebidos de forma a permitir sua implementação e os próprios compiladores costumam gerar código de máquina com padrões especiais de opcodes que marcam tanto o início de uma função quanto seu término. Dentro de um Politech, estes marcadores constituídos de pequenos fragmentos de código também são chamados de *marcador de início* e *marcador de fim* da função.

A título de exemplo, o utilitário GNU/Linux `objdump` com a opção `-d` faz uma engenharia reversa parcial ao desmontar um arquivo executável montado na última etapa da compilação e permite-nos verificar que as funções geradas pelo `gcc` com opção de otimização desabilitada `-O0` iniciam com duas instruções cujo código assembly para CPU intel de 64 bits é:

```
pushq   %rbp         # salva na pilha de ativação o registrador Base Pointer
movq    %rsp, %rbp    # move o registrador Stack Pointer para o Base Pointer
```

De fato, o registrador `%rbp` (Register Base Pointer) serve de base que aponta para o início da região da pilha em que são automaticamente alocadas as variáveis locais da instância da função chamada. O endereço de cada variável local é calculado pela soma de uma constante ao conteúdo do registrador `%rbp`. Depois do código que efetivamente realiza os comandos da função chamada, esta termina com duas instruções: `popq %rbp` (ou `leaveq`), que restabelece o valor de `%rbp` da função chamante; e `ret`, que desvia para o endereço de retorno automaticamente armazenado na pilha pela instrução `call` da função chamante que transferiu a execução para a função chamada.

Lembra que um programa em linguagem de máquina divide a memória em código e dados:

- A seção de código é reservada para armazenar o conjunto de instruções do programa que serão decodificadas na unidade de controle da CPU. Se na arquitetura de von Neumann cada instrução é codificada numericamente, no Politech esta codificação se dá pelos caracteres politípicos de Σ.

- A seção de dados divide a memória disponível em quatro subseções: dados constantes; dados globais; monte (heap); e pilha (stack). As duas primeiras efetuam alocações estáticas,

[1] Se esta é uma regra quase universal, há notáveis exceções, como as variantes do HIV tipo 1 e os diversos coronavírus conhecidos desde a primeira SARS em 2003, pois desenvolvem *deslocamentos programados da janela de transcrição ribossômica* (cf. <`https://en.wikipedia.org/wiki/Ribosomal_frameshift`>). Conforme Luc Montaignier, prêmio Nobel de medicina de 2008 que descobriu o vírus HIV, o vírus causador da pandemia que se alastrou a partir de Wuhan possui 16 trechos significativos altamente semelhantes (homólogos) a diversas variantes de HIV coletadas em diferentes localidades do mundo (cf. Perez, J. C. Montaignier, L.. (2020). COVID-19, SARS AND BATS CORONAVIRUSES GENOMES PECULIAR HOMOLOGOUS RNA SEQUENCES. International Journal of Research -GRANTHAALAYAH, 8(7), 217-263. `https://doi.org/10.29121/granthaalayah.v8.i7.2020.678`

ao passo que o monte e a pilha são regiões da memória usadas na alocação dinâmica das variáveis e demais estruturas de dados.

A existência de técnicas de polimorfismo na replicação de um Politech p impele-nos a detectar o início da área de código. Assim, definimos $\gamma(p)$ como *a primeira ocorrência de um marcador de início*, pois é a partir desta posição num Politech que são verificadas eventuais mutações presentes nas diversas funções que compõem a seção de código. (Caso não ocorra nenhum marcador de início em p, adota-se como valor de $\gamma(p)$ uma posição inválida maior que o comprimento de p.) Assim,

> Escreva em C a função `BuscaMarcador`, que busca a partir da posição `ini` na string `Politech` pela primeira ocorrência de um `marcador` e devolve a posição desta ocorrência caso encontre. Caso contrário, devolve o índice inválido `MAXP`. Seu protótipo é:
>
> ```
> int BuscaMarcador( int ini, char politech[MAXP], char marcador[MAXM] );
> ```

Observe que a mesma função `BuscaMarcador` pode ser usada para procurar pela primeira ocorrência de um marcador de início ou de fim. Ademais, convém termos uma versão que busque por mais de um marcador. Assim,

> Escreva em C a função `BuscaMarcadores`, que busca a partir da posição `ini` na string `politech` pela primeira ocorrência de qualquer das `k` strings armazenadas nas linhas da matriz de `marcadores`. Ela retorna: a posição desta primeira ocorrência, caso encontre; o índice inválido `MAXP`, caso não. Ela devolve na variável apontada por `compr`: o comprimento do primeiro marcador encontrado; ou zero, se não encontrou nenhum. Ela possui o seguinte protótipo:
>
> ```
> int BuscaMarcadores( int ini, char politech[MAXP],
>                      int k, char marcadores[MAXK][MAXM], int *compr );
> ```

Como é sabido, eventuais erros na replicação de uma cadeia de DNA podem provocar mutações na clonagem de um vírus invasor, principalmente através da substituição de um dos quatro nucleotídeos `A`, `G`, `C` ou `T` por outro. Outros tipos de alterações como a inserção ou remoção de símbolos ocorrem na biologia com menos frequência e não são aqui considerados. Por razão de simplicidade neste exercício programa, a clonagem de um Politech aqui considerada usa uma técnica simples de polimorfismo, uma forma elementar de algoritmo genético que sorteia algumas posições da área de código e faz substituição de opcode em cada uma delas.

Assim, suponha que um Politech p seja clonado para um novo Politech q. Como regra, cada caractere politípico $p[i + \gamma(p)]$ é copiado fielmente para $q[i + \gamma(q)]$. As exceções se devem às mutações introduzidas pelas técnicas de polimorfismo, de modo que uma *mutação* na posição $i + \gamma(p)$ em p é simplesmente a substituição do caractere $p[i + \gamma(p)]$ por outro: $q[i + \gamma(q)]$.

Por este novo valor em $q[i + \gamma(q)]$, uma tal mutação introduz uma nova característica que q possui mas p não possui na posição correspondente $i + \gamma(p)$, característica que é herdada pelos eventuais descendentes de q. Como no caso da replicação de um vírus biológico, a reprodução de um Politech é 'assexuada' e o Politech *filho* q não recebe transferência de um trecho significativo de outro vírus que não de seu *pai* p, seu único ancestral direto. Um conjunto de Politechs que

descendem de um mesmo ancestral comum é chamado de *família* e este ancestral comum é chamado de *patriarca*. Seja $\mathcal{F}$ uma família de Politechs e p seu patriarca. Para todo Politech $q \in \mathcal{F}$, dizemos que a posição $i+\gamma(p)$ no código de p é uma *característica* de q que o diferencia de p se $p[i+\gamma(p)] \neq q[i+\gamma(q)]$. Assim, definimos

$$\Delta(q,p) = \{i+\gamma(p) \text{ tal que } p[i+\gamma(p)] \neq q[i+\gamma(q)]\}$$

como sendo o *conjunto das características de* q, características que o diferenciam do patriarca p. Caso p seja claro e fixo, podemos omiti-lo e denotar $\Delta(q,p)$ simplesmente por $\Delta(q)$. Assim,

> Escreva em C a função `Delta` que calcula o conjunto das características que diferenciam um Politech `q` do seu patriarca `p` e que pertencem ao intervalo fechado [`ini` .. `fim`], armazenando-o em ordem crescente no vetor `delta`. Para caracterizar o término da sequência de posições, também armazena-se ao final de `delta` a posição inválida `MAXP`. O parâmetro `desloc` recebe o deslocamento $\gamma(\texttt{q}) - \gamma(\texttt{p})$ que deve ser somado às posições de `p` para obter as correspondentes de `q`. `Delta` retorna a quantidade de características encontradas entre as posições `ini` e `fim` de `p`, inclusive. Ela possui o seguinte protótipo:
>
> ```
> int Delta( char q[MAXP], char p[MAXP], int delta[MAXD],
>            int desloc, int ini, int fim );
> ```
>
> Sugestão: esboça primeiro um código que imprime em ordem crescente todos os inteiros no intervalo fechado [`ini` .. `fim`].

Se no jogo do bixo visto no EP2 vimos fomos introduzidos a um método que nos permite periciar a lisura de uma sucessão de sorteios, a mesma abordagem nos leva aqui a inquirir sobre quão natural ou anômalo é um conjunto de características extraído de uma família de Politechs. Tanto mais improvável é um tal conjunto, tanto mais preciosa é a informação nele contida...

Observa que a primeira e a última das posições devolvidas em `delta[MAXD]` são respectivamente a menor e a maior das posições de `p` do intervalo de `ini` a `fim` onde ocorre mutação que diferencia q de p. Sendo $n = \texttt{fim} - \texttt{ini} + 1$ e k a quantidade de posições de $\Delta(\texttt{q},\texttt{p}) \cap [\texttt{ini}..\texttt{fim}]$, analogamente ao EP2 definimos o *diâmetro* deste conjunto com k características como sendo: um mais a diferença entre a maior e a menor das posições de $\Delta(\texttt{q},\texttt{p})) \cap [\texttt{ini}..\texttt{fim}]$, devolvidas em `delta[MAXD]` pela função `Delta`.

Dado um natural d tal que $k \leq d \leq n$, definimos $T(n,k,d)$ como o conjunto de sequências de k sorteios sem repetição no intervalo [`ini`..`fim`] que possuem diâmetro d. Como visto na fórmula (8) do EP2, temos que:

$$|T(n,k,d)| = \begin{cases} n, & \text{se } d = k = 1, \\ (n-d+1)k(k-1)\frac{(d-2)!}{(d-k)!}, & \text{se } 2 \leq k \leq d \leq n. \end{cases} \quad (1)$$

Por outro lado, o número total de sequências de k sorteios sem repetição no intervalo [1 .. n] é $\frac{n!}{(n-k)!}$. Supondo uma distribuição de probabilidades uniforme, isto nos permite calcular a probabilidade de que uma sequência de k sorteios sem repetição no intervalo [1 .. n] possua diâmetro d. Assim, também podemos acumular a soma destas probabilidades de forma a calcular a probabilidade de que uma sequência de k sorteios sem repetição no intervalo [1 .. n] possua diâmetro *não superior* a d. No caso de *conjuntos* de k sorteios, as quantidades devem ser divididas por $k!$, mas as probabilidades não se alteram porque os fatores $k!$ se cancelam. Feita esta rápida exposição,

Escreva a função `DiametroEProbabilidades`, que recebe como parâmetros os mesmos `ini`, `fim` e `delta` de `Delta`, mais os endereços de três variáveis onde devolverá três valores: o diâmetro do conjunto armazenado em `delta`; a probabilidade de que um conjunto de k (o tamanho de `delta`) inteiros no intervalo [`ini..fim`] possua o mesmo diâmetro; a probabilidade de que um conjunto de k inteiros no intervalo [`ini..fim`] possua diâmetro não superior.

Seja $\mathcal{C} = \cup_{q \in \mathcal{F}} \Delta(q, p)$ a classe das características que distinguem qualquer membro da família $\mathcal{F}$ do patriarca p. O conjunto de características $\mathcal{C}$ é fundamental para inferir a história da evolução da família $\mathcal{F}$ a partir de seu patriarca p, o que normalmente é feito através da elaboração de uma *árvore filogenética* para a família $\mathcal{F}$, uma espécie de árvore genealógica que leva em consideração o fato de que cada membro da família que não o patriarca possui um único pai. Supomos que o evento de uma mutação seja raro o suficiente para afirmar que 'o raio não cai duas vezes no mesmo lugar', ou seja, que não existem duas mutações distintas envolvendo a mesma posição a não ser no caso de transmissão hereditária. Em particular, não existe nem a *reversão* de uma mutação nem tampouco o fenômeno de *evolução paralela* ou *convergência*.

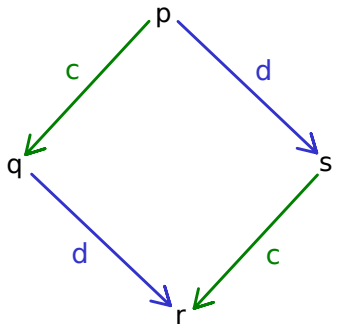

Figura 2: Convergência ou evolução paralela. O Politech q possui a característica c e o Politech s possui a característica d. Nem q possui a característica d, nem s possui c. O patriarca p não possui nenhuma, mas o Politech r possui ambas. De quem r descende: de q ou de s? O primeiro caso requer uma evolução paralela da característica d na evolução de q a r; o segundo, uma evolução paralela da característica c na evolução de s para r.

O assunto é suficientemente complexo para ser aqui abordado em detalhes mas o fato é que é possível fazer uma tal reconstituição da sequência de eventos mutacionais através da obtenção de uma *árvore filogenética perfeita* desde que não exista nenhuma incompatibilidade entre as características observadas entre os membros da família.[2] Duas características distintas $c, d \in \mathcal{C}$ são ditas *incompatíveis entre si* se existirem $q, r, s \in \mathcal{F}$ tais que:

- $c \in \Delta(q)$ e $d \notin \Delta(q)$;
- $c \in \Delta(r)$ e $d \in \Delta(r)$;
- $c \notin \Delta(s)$ e $d \in \Delta(s)$.

Observe que $\Delta(p) = \emptyset$ e que, portanto, $c \notin \Delta(p)$ e $d \notin \Delta(p)$. Assim,

Escreva em C a função `DiferencaDeConjuntos` que recebe dois conjuntos de características armazenados nos vetores `delta_q` e `delta_s`, calcula o conjunto diferença `delta_q` − `delta_s`, e armazena suas características em ordem crescente no vetor `q_menos_s`. Tal como na função `Delta`, as sequências armazenadas nos vetores `delta_q`, `delta_s` e `q_menos_s` são terminadas pela inserção de `MAXP` ao final. A função devolve o número de posições no conjunto diferença e possui o protótipo:

[2]Um eventual interessado pode referir-se às páginas 175-186 e 36 do livro Introduction to Computational Molecular Biology escrito pelos professores João Meidanis e João Carlos Setúbal.

```
int DiferencaDeConjuntos( int delta_q[MAXD], int delta_s[MAXD],
                          int q_menos_s[MAXD] );
```

Assim, estamos aptos a verificar se existem duas características incompatíveis entre aquelas associadas aos Politechs de nossa família.

> Seja `n` o número de Politechs da família que está armazenada na matriz de caracteres `F[MAXF][MAXP]`, que mantém em cada linha `F[i]` a string do $i+1$-ésimo Politech. Seja `j_p` o índice tal que `F[j_p]` seja o patriarca da família. Seja `D[MAXF][MAXD]` uma matriz que guarda em cada linha `D[i]` o conjunto de características do Politech `F[i]` computado pela função `Delta` e devolvido em `delta`. Seja `G[MAXF]` um vetor que guarda em cada `G[i]` o valor de γ(`F[i]`). Escreva em C a função `HaIncompatibilidade` que verifica se há em `D` duas características incompatíveis e devolve `TRUE` caso haja ou `FALSE` caso não haja. Ademais, ela deve produzir uma listagem com todos os eventuais exemplos de características incompatíveis c e d e politechs q, s e r que configuram uma prova de incompatibilidade, segundo a definição vista acima.[3] Use o protótipo abaixo (nem todos os parâmetros são necessários, mas há soluções que os usam):
>
> ```
> int HaIncompatibilidade( int n, char F[MAXF][MAXP], int j_p,
>                          int G[MAXF], int D[MAXF][MAXD] );
> ```

Caso o conjunto $\mathcal{C}$ não possua duas características incompatíveis entre si, é possível computar e exibir uma árvore filogenética perfeita, o que não será cobrado neste EP. Um interessado pode consultar as páginas já citadas do livro de Meidanis e Setúbal para ver a descrições de algoritmos para testar se há características incompatíveis ou não e para listar uma árvore filogenética perfeita, caso exista. E se não existir? Quão anômala é a existência de características incompatíveis? Afinal, o que é incompatível: as características observadas ou a modelagem desenvolvida pelo cientista? A experimentação com dados da realidade pode por à tona questões fundamentais ligadas à evolução dos Politechs, da vida humana, das sequências genômicas, e virtualmente por tudo aquilo que pode ser representado através de sequências de símbolos.

* * *

Anexo ao arquivo com este enunciado[4] encontra-se o código da função `main()` no arquivo `EP04.c`, com os comandos de entrada e saída e chamadas das funções antes especificadas. Os arquivos `.dat` fornecidos apresentam sete casos de dados a serem oferecidos como entrada ao programa. A saída produzida por eles pode ser vista no arquivo `SeteSaidas.txt`, que se encontra dentro do arquivo `SeteExemplos.zip` junto com alguns arquivos usados na elaboração dos exemplos.

[3] O formato da impressão de uma eventual listagem deve ser como o do exemplo da tabela 6.1, que segue em anexo.

[4] Confira na página <`https://edisciplinas.usp.br/mod/assign/view.php?id=3568516`>.