



TRABALHO 2: BUSCA POR *Substrings* (2021-07-11)

PROFESSOR: PABLO MAYCKON SILVA FARIAS

1 Resumo

Você deverá implementar em C++ um programa que apresente os tempos de execução do algoritmo de busca por subsequências de caracteres (*substrings*) por força bruta e do algoritmo Knuth-Morris-Pratt (KMP) em diferentes tipos de instâncias. A implementação será entregue ao professor através de tarefa no SIGAA.

Atenção: Colabore para o bom andamento da disciplina e do aprendizado de todos, não copiando códigos de outras pessoas nem compartilhando seu próprio código. Em caso de dificuldade sua ou com algum colega, fale com o professor.

2 Compilação do Programa

O seu programa deverá ser escrito em C++17 padrão e compilável, sem erros ou avisos, através de uma linha como

```
g++ -Wall -Wextra -std=c++17 -pedantic -o programa main.cpp
```

Caso algo além disso seja necessário para compilar o programa (por exemplo, caso o seu programa possua mais de um arquivo que precise ser explicitamente fornecido como entrada para o compilador), por favor inclua um arquivo `explicacoes.txt` com as devidas explicações.

3 Declarações das Funções dos Algoritmos

Tanto o algoritmo de força bruta quanto o KMP têm que ter a seguinte forma:

```
bool nome_da_funcao (const char *P, const char *T, int *O)
```

sendo P o padrão, T o texto e O o (ponteiro para o primeiro elemento do) vetor onde as ocorrências devem ser informadas. Tanto P quanto T são “strings de C”, ou seja, sequências de caracteres finalizadas por um `'\0'`; observe que a função não recebe o tamanho m de P nem o tamanho n de T .

O vetor O deve ser preenchido com os índices iniciais (em ordem crescente, naturalmente) das ocorrências de P em T , seguidos ao final por um `-1`. Assim, por exemplo, se P for “ata” e T for “Matata”, então o início de O deve ser preenchido com `[1 3 -1]`, já que as ocorrências de P ocorrem nos índices 1 e 3 de T , respectivamente. Observe que o tamanho de O também não é recebido pela função, que deve simplesmente partir da pré-condição de que O é grande o suficiente para armazenar os índices das ocorrências e o `-1` ao final; garantir que esse é o caso será responsabilidade de quem chamar a função. **Importante:** a função não deve escrever em posições de O posteriores àquela em que for gravado o `-1`.

A função deve retornar `false` caso haja algum erro de alocação de memória, e deve retornar `true` em contrário (isto é, em caso de execução bem-sucedida).

4 Instâncias Aleatórias e de Pior Caso

A entrada do programa será necessariamente informada através dos argumentos fornecidos em sua chamada. O primeiro argumento determinará o tipo das instâncias a serem fornecidas aos dois algoritmos. Caso esse argumento seja um `A`, então as instâncias serão (pseudo)aleatórias e a chamada terá o formato (abaixo, os colchetes apenas delimitam os nomes de *metavariáveis*; eles não devem estar presentes nas chamadas ao programa)

```
./programa A [l] [m] [n] [I]
```

onde `l` é uma letra minúscula de “a” a “z” (escrita sem aspas), `m` é o tamanho do padrão (neste trabalho, o *tamanho* de uma *string* é o seu número de símbolos, sem contar o `'\0'`), `n` é o tamanho do texto e `I` é o número de instâncias a serem fornecidas aos algoritmos, sendo `m`, `n` e `I` números naturais positivos (este trabalho não faz exigências sobre o comportamento do programa nos casos em que a chamada não estiver no formato esperado). Nesse caso, o programa deverá fazer o seguinte:

1. Criar a instância aleatória número 1, isto é, criar um padrão e um texto aleatórios, de tamanhos `m` e `n`, respectivamente, e contendo letras aleatórias de “a” a `l` (assim, por exemplo, se `l` for a letra “e”, então o padrão e o texto deverão conter letras aleatórias de “a” a “e”). Em particular, observe que, caso `l` seja a letra “a”, então o padrão será necessariamente uma sequência de `m` a’s e o texto uma sequência de `n` a’s, o que é justamente uma instância de pior caso para o algoritmo de força bruta.
2. Fornecer essa instância ao algoritmo de busca por força bruta, aferindo e salvando o tempo utilizado, **mas sem imprimi-lo na tela por enquanto**. A tomada de tempo realizada pode ser feita em termos de *tempo de CPU* ou *tempo real*, a seu critério, mas tem, obrigatoriamente, que ter frações de segundo como precisão, e não apenas segundos inteiros.
3. Fazer o análogo do passo anterior para o algoritmo KMP, salvando o tempo utilizado separadamente do tempo do algoritmo anterior.
4. Comparar as saídas dadas pelos dois algoritmos, avisando ao usuário e encerrando o programa caso elas tenham sido diferentes.
5. Gerar a instância aleatória número 2 e repetir os passos acima para ela. O tempo gasto pelo algoritmo de força bruta nessa instância deve ser somado ao tempo gasto por esse algoritmo na instância anterior; idem para o KMP.
6. Repetir o processo para as instâncias 3, 4, ..., `I`.
7. Ao final, o tempo total utilizado pelo algoritmo de força bruta para as `I` instâncias deve ser impresso na tela, **como um número fracionário e em segundos** (deve ser impressa a soma dos tempos e não o tempo individual por instância). Idem para o KMP.

Atenção:

1. Não utilize vetores diferentes para as várias instâncias: ao invés disso, reutilize os mesmos vetores, desalocando-os apenas ao fim do programa. Observe que o vetor que num momento armazena o padrão da primeira instância pode (e, neste trabalho, tem que) ser o mesmo que, posteriormente, armazenará o padrão da segunda instância (não sendo

- necessário, nem permitido, alocar um segundo vetor para isso). O mesmo vale para o vetor do texto, bem como para os vetores que guardarão as saídas dos dois algoritmos.
2. Você pode escolher o tamanho dos vetores que guardarão as saídas dos dois algoritmos analisados no programa. Observe que $n+1$ é sempre um valor seguro, mas você pode fazer outras escolhas, como alguma baseada na divisão de n por m . A única restrição é que o tamanho escolhido seja, além de grande o suficiente, computável em tempo constante (em relação aos valores de m e n).
 3. Ao fim do programa, não esqueça de desalocar os vetores dinamicamente alocados durante a execução.

5 Instâncias Reais

A chamada do programa para as instâncias reais terá a seguinte forma:

```
./programa R [x] [y]
```

sendo x e y números naturais tais que $0 \leq x \leq y \leq 35129$. Nesse caso, o comportamento do programa deverá ser análogo ao descrito anteriormente, com as seguintes diferenças:

1. Para cada índice i de x a y , o programa fornecerá como entrada para os dois algoritmos a **instância real de índice i** , composta pelo padrão `Padroes_Palavras[i]` e o texto `Texto_Livros`, sendo `Padroes_Palavras` e `Texto_Livros` variáveis definidas no arquivo `instancias_Reais_Trabalho_2.hpp`, que é fornecido pelo professor (com base no conteúdo de um **livro de domínio público**) juntamente com o enunciado deste trabalho e que deve ser incluído no código-fonte do seu programa (`#include "..."`).
2. O tempo total exibido ao final do programa para cada um dos algoritmos será então aquele necessário para resolver as $y - x + 1$ instâncias reais indexadas de x a y , conforme definido no item anterior.

6 Submissão do Trabalho

A solução do trabalho deverá consistir num arquivo `[Matrícula].zip` (sem colchetes) entregue através do SIGAA em tarefa a ser cadastrada pelo professor. Naturalmente, apenas o código-fonte deve ser submetido, sem qualquer arquivo executável. Prezando pelo tamanho do arquivo zip final, **também não deve ser incluído o arquivo `instancias_Reais_Trabalho_2.hpp`**. Quaisquer explicações adicionais podem ser incluídas através de um arquivo `explicacoes.txt`.

Em caso de dúvida, contate o professor rapidamente.

– Eu espero que você se divirta e aprenda. Bom trabalho! –