Instituto de Computação - Unicamp

MC102 - Algoritmos e Programação de Computadores

Laboratório 19 - Todos os Livros do Mundo

Prazo de entrega: **23/06/2017 23:59:59**

Peso: **2**

Professor: Eduardo C. Xavier
Professor: Guido Araújo

Monitor: Arthur Pratti Dadalto
Monitor: Cristina Cavalcante
Monitor: Klairton de Lima Brito
Monitor: Luís Felipe Mattos
Monitor: Paulo Finardi
Monitor: Paulo Lucas Rodrigues Lacerda
Monitor: Pedro Alves
Monitor: Renan Vilas Novas
Monitor: Vinicius de Novaes Guimarães Pereira

Descrição

Dado um limite máximo de páginas, é possível criar um algoritmo para gerar todos os livros já escritos e ainda por serem escritos neste mundo, desde clássicos de Machado de Assis até as novelas populares. A ideia do algoritmo é gerar todas as combinações de letras possíveis incluindo espaços. Obviamente o número de combinações é astronômico. Se por exemplo usarmos 50 possíveis caracteres, incluindo letras com acentos, pontuação e espaço, e num livro tivermos espaço para 500000 caracteres teremos 50^(500000) possíveis livros.

O nosso objetivo neste laboratório é mais modesto, consistindo em gerar todos os anagramas de uma palavra. Um anagrama de uma palavra é uma permutação das suas letras afim de formar uma outra palavra. Por exemplo, *ator* é um anagrama da palavra *rota*.

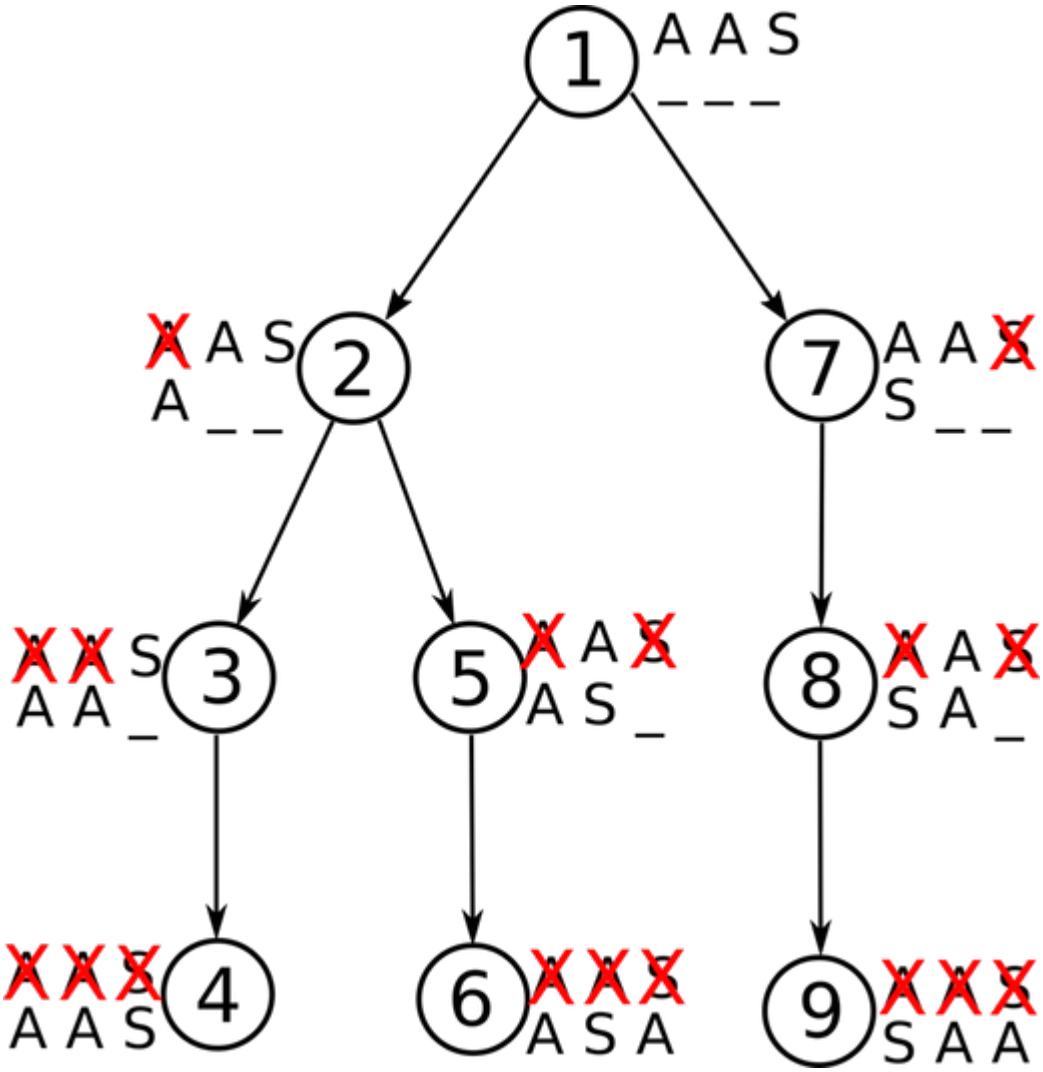
Note porém que não basta gerar todas as permutações possíveis das letras de uma palavra. Se gerassemos todas as permutações de *asa* obteríamos

```
aas
aas
asa
asa
saa
saa
```

pois repetiríamos as permutaçoes das duas letras a's. Mas os anagramas da palavra *asa* são apenas *aas*, *asa* e *saa*, em ordem alfabética.

Neste laboratório você deve fazer um programa que usa uma função recursiva que gera todos os anagramas de uma palavra dada na entrada.

Uma maneira de se resolver este problema é recursivamente ir montando um anagrama. Inicialmente ordenamos a palavra de entrada, por exemplo, para `asa` temos `aas`. Agora de forma recursiva vamos montar os anagramas. A cada chamada recursiva temos um anagrama em formação que salvamos em *word*, e um indicador de quais letras já foram usadas. Na chamada recursiva atual devemos incluir cada uma das letras ainda não usadas no final de *word*, e para cada uma destas inclusões marcar a letra como usada e chamar recursivamente a função para posicionar a próxima letra. Um detalhe importante para não gerar palavras repetidas é que as letras a serem incluídas no final de word numa chamada recursiva atual devem ser distintas. Como exemplo do processo veja a figura abaixo. Na parte de cima de cada nó temos as letras usadas/livres e embaixo a palavra *word* em formação.



Considere cada nó da árvore de recursão:

- 1: Inicialmente temos as letras ordenadas e *word* vazia.
- 2: Incluímos a letra `A` em *word* e a marcamos como usada.
- 3: Incluímos a próxima letra livre em *word*, que também é `A`, e marcamos ela como usada.
- 4: Incluímos a próxima letra em *word*, e como foram usadas todas as letras, temos um anagrama que deve ser impresso.
- Note que ao voltar para 3, não temos mais letras sobrando (só havia o `S`) e portanto voltamos para 2.
- 5: Incluímos a próxima letra distinta, `S` em *word* marcando ela como usada.
- 6: Incluímos a letra `A`, e como usamos todas as letras, imprimimos um novo anagrama.
- Note que ao voltar para 5 não há mais letras livres (só havia o `A`) e

voltamos para 2. Em 2, também não temos mais letras livres (já foram posicionadas o `A` e o `S`) portanto voltamos para 1.

- 7: Em 1, escolhemos a próxima letra livre distinta da anterior (`A` foi usada para ir para 2), portanto incluímos o `S` em *word* e a marcamos como usada.
- 8: Incluímos a letra `A` em *word* e a marcamos como usada.
- 9: Incluímos o `A` em *word*, usando todas as letras e portanto imprimimos mais um anagrama.
- Ao voltar para 8, não há outra letra livre e voltamos para 7. Em 7 não há uma outra letra distinta de `A` livre, portanto voltamos para 1, e não havendo mais letras livres encerramos o processo.

Use os códigos auxiliares abaixo para guiar a sua função que se chamrá *generate*. Linguagem C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/*
 * Entrada:
 *          string_ordenada: a string original, que devemos proce
 *                           para encontrarmos todos os anagramas
 *          letra_usada: um vetor de booleanos (implementado da f
 *                       de um vetor de inteiros) para marcarmos
 *                       posições da string original já foram usa
 *          word: o anagrama que estamos formando
 *          n: o numero de letras do anagrama
 *          k: a posição onde incluiremos a proxima letra em word
 *
 * A ideia para gerar sem repeticoes eh que setada a letra da pos
 * e retornado da chamada recursiva, temos
 * que colocar uma letra diferente na posição k, pois senão gerar
 * as repetições. Quando n==k incluimos a letra faltante e imprim
 * anagrama.
 */
void generate(char *string_ordenada, int *letra_usada, char *word
  printf("%s\n",word);
  return;
}
```

```c
char * sort(char string[]);

int main(){

  char *string_inicial;
  scanf("%ms", &string_inicial);
  char *string_ordenada =  sort(string_inicial);
  int n  = strlen(string_ordenada);
  int *letra_usada = malloc(n*sizeof(int));
  char *anagrama = malloc((n+1)*sizeof(char));

  int i=0;
  for(i=0; i<n; i++)
    letra_usada[i]=0;
  generate(string_ordenada, letra_usada, anagrama, n, 0);

  free(anagrama);
  free(string_ordenada);
  free(letra_usada);
}

char * sort(char string[]){
  int count[256], i;
  char *sr;

  for(i=0; i<256; i++)
    count[i] = 0;

  i=0;
  while(string[i] != '\0'){
    count[(int)string[i]]++;
    i++;
  }

  sr = malloc((i+1)*sizeof(char));

  int j=0, k=0;
  for(i=0; i<256; i++){
    if(count[i]!=0){
      for(k=0; k<count[i]; k++){
        sr[j] = i;
        j++;
      }
    }
  }
  sr[j] = '\0';
```

```
  return sr;
}
```

Linguagem C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * Entrada:
 *          string_ordenada: a string original, que devemos proce
 *                           para encontrarmos todos os anagramas
 *          letra_usada: um vetor de booleanos (implementado da f
 *                       de um vetor de inteiros) para marcarmos
 *                       posições da string original já foram usa
 *          word: o anagrama que estamos formando
 *          n: o numero de letras do anagrama
 *          k: a posição onde incluiremos a proxima letra em worc
 *
 * A ideia para gerar sem repeticoes eh que setada a letra da pos
 * e retornado da chamada recursiva, temos
 * que colocar uma letra diferente na posição k, pois senão gerar
 * as repetições. Quando n==k incluimos a letra faltante e imprim
 * anagrama.
 */
void generate(char *string_ordenada, int *letra_usada, char *worc
  printf("%s\n",word);
  return;
}
```

Objetivo

O seu objetivo é fazer um programa com uma função recursiva que imprime todos os anagramas de uma palavra em ordem alfabética.

Você também não deve submeter o arquivo `lab19_main.c` para o SuSy, **somente o arquivo** `lab19.c` .

Entrada

A entrada consiste em uma palavra composta apenas de, no máximo,

10 letras do alfabeto português, sem acento.

Saída

Como saída o seu programa deverá imprimir todos os anagramas da palavra em ordem alfabética.

Exemplos

Teste 01

Entrada

```
oi
```

Saída

```
io
oi
```

Teste 02

Entrada

```
ave
```

Saída

```
aev
ave
eav
eva
vae
vea
```

Teste 03

Entrada

```
asa
```

Saída

```
aas
asa
saa
```

Para mais exemplos, consulte os testes abertos no Susy.

Observações

- O número máximo de submissões é **15**;
- O seu programa deve estar completamente contido em um único arquivo denominado `lab19.c` ;
- Para a realização dos testes do SuSy, a compilação dos programas desenvolvidos considerar o comando:__ `gcc -std=c99 -pedantic -Wall -o lab19 lab19.c` ;
- Você deve incluir, no início do seu programa, uma breve descrição dos objetivos do programa, da entrada e da saída, além do seu nome e do seu RA;
- Indente corretamente o seu código e inclua comentários no decorrer do seu programa.

Critérios importantes

Independentemente dos resultados dos testes do SuSy, o não cumprimento dos critérios abaixo implicará em nota zero nesta tarefa de laboratório.

- Os únicos headers aceitos para esta tarefa são os `stdio.h` `stdlib.h` e o `string.h` .