Instituto Politécnico Nacional
Escuela Superior de Cómputo

Teoría de la Computación

Práctica 1. Alfabetos y Cadenas

Juárez Martínez Yunuen

4CM2

Introducción

Definiciones

Alfabeto: conjunto finito, no vacío.

Símbolos: Elementos de un alfabeto.

Palabra o cadena (x ó y): Secuencia finita formada con los símbolos de un alfabeto.

Longitud de una cadena $|x|$: Número de símbolos que la componen.

Cadena vacía: λ.

Lenguaje universal W(∑): Conjunto de todas las cadenas que se pueden formar con las letras de un alfabeto.

Lenguaje: Conjunto de palabras.

Operaciones con Cadenas

Concatenación de dos cadenas: ω = uv

Propiedades

- No es conmutativa, en general no es lo mismo uv que vu.
- Es asociativa, es decir cualesquiera que sean las cadenas u, v y w sobre el mismo alfabeto, se tiene que (uv)w = u(vw).
- Esta propiedad nos permite concatenar cualquier número finito de cadenas sin tener que poner los paréntesis. Escribiremos uvw.
- |uv|=|u|+|v| es decir la longitud de la cadena formada por la concatenación de dos cadenas, es la suma de las longitudes de cada una de ellas.
- La cadena vacía es el elemento neutro de la concatenación. En efecto uλ=λu =u.

Prefijos y sufijos

- Sea ω una cadena sobre cierto alfabeto Σ.
- Sean u y v dos cadenas sobre Σ tales que ω=uv.
- Decimos que u es un prefijo y que v es un sufijo de ω.
- Cualquier cadena que se obtiene al eliminar cero o más símbolos del final (prefijos) o del principio (sufijos).

Subcadena

Se obtiene al eliminar cualquier prefijo y cualquier sufijo.

Los prefijos, sufijos y subcadenas propios de una cadena S son esos prefijos, sufijos y subcadenas, respectivamente, de S que no son λ ni son iguales a la misma S.

Subsecuencia

Es cualquier cadena que se forma mediante la eliminación de cero o más posiciones no necesariamente consecutivas de S.

Inversión: ω^{-1}

Propiedades

$(uv)^{-1} = v^{-1}u^{-1}$es decir la cadena inversa (o reflejada) de la concatenación de dos cadenas es la concatenación de las cadenas inversas (o reflejadas) en orden contrario.

$|\omega^{-1}| = |\omega|$ es decir, la longitud de una cadena y su inversa coinciden siempre.

Potencia

Sea ω una cadena y k un número entero, definimos: ω^k

- k>0, ω …$^{k)}$ … ω
- k=0,
- k<0, ω^{-1} …$^{-k)}$ … ω^{-1}

Desarrollo

Funciones

En esta parte se muestran los prototipos de las funciones que usaremos y la estructura de nuestra lista.

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

typedef struct Nodo {

    char* dato;
    struct Nodo* siguiente;

}Nodo;

void concatenacion(Nodo** cabeza);
void potencia(Nodo** cabeza, char cad[]);
void longitudCadena(char cad[]);
void prefijos(char cad[]);
void sufijos(char cad[]);
void subcadenas(char cad[]);
char* crearCadenaSinCaracter(const char* original, int indice);
int existeEnLista(Nodo* cabeza, const char* cadena);
void generarSubsecuenciasRecursivo(Nodo** cabeza, const char* cad_actual);
void mostrarListaSubsecuencias(Nodo* cabeza);
void subsecuencias(Nodo** cabeza, char cad[]);
void menu(Nodo** cabeza, char cad[]);
```

```c
void menu(Nodo** cabeza, char cad[]){

    int opc;

    printf("\n\n\n\t Elige una opcion: \n");
    printf("\n 1. Concatenacion");
    printf("\n 2. Potencia");
    printf("\n 3. Calculo de la longitud");
    printf("\n 4. Generacion de prefijos");
    printf("\n 5. Generacion de sufijos");
    printf("\n 6. Generacion de subcadenas");
    printf("\n 7. Generacion de subsecuencias");
    printf("\n 8. Salir");
    printf("\n\n Opcion? ");
    scanf("%i", &opc);

    printf("\n");

    switch(opc){

        case 1:
            concatenacion(cabeza);
            menu(cabeza, cad);
        break;

        case 2:
            potencia(cabeza, cad);
            menu(cabeza, cad);
        break;

        case 3:
            longitudCadena(cad);
            menu(cabeza, cad);
```

Aquí solo se pide al usuario la función que desea realizar donde se implementa un switch y manda llamar a cada opción y al menú para hacer el programa recursivo.

```c
Nodo* crearNodo(const char* dato){

    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));

    if(nuevo == NULL){

        printf("Error: memoria insuficiente\n");
        exit(1);

    }

    nuevo->dato = (char*)malloc(strlen(dato) + 1);

    if(nuevo->dato == NULL){

        printf("Error: Memoria insuficiente\n");
        free(nuevo);
        exit(1);

    }

    strcpy(nuevo->dato, dato);

    nuevo->siguiente = NULL;
    return nuevo;

}
```

Para implementar la lista tenemos que crear un espacio de memoria para la lista y un espacio de memoria para añadirle algo a esa lista o el primer elemento por medio de apuntadores.

```c
void mostrarLista(Nodo* cabeza){

    Nodo* actual = cabeza;

    if(actual == NULL){

        printf("Cadena vacia\n");
        return;
    }

    printf("Resultado: ");

    while(actual != NULL){
        printf("%s", actual->dato);
        actual = actual->siguiente;
    }

    printf("\n");

}
```

También ocuparemos mostrar la lista en varias funciones, por lo que es mejor solo mandar llamar la función, entonces imprimimos los datos mediante un bucle.

```c
void liberarLista(Nodo* cabeza){

    Nodo* actual = cabeza;

    while(actual != NULL){
        Nodo* temp = actual;
        actual = actual->siguiente;
        free(temp->dato);
        free(temp);
    }

}
```

Ahora bien, en este programa se trabaja con una cadena inicial, así que para poder obtener todos los datos de esta cadena tenemos que liberar los cambios y regresar a la lista original.

Aquí ocupamos un auxiliar que es nuestro Nodo* temp = actual;

Comenzamos con la función de concatenación, aquí nos pide ingresar una cadena para concatenar, sin embargo, tenemos dos opciones, al concatenar una cadena con 0 que hace que solo muestre la lista con nuestra cadena inicial sin hacer cambios, de lo contrario concatenara la cadena que introduzcamos agregándola en la lista en donde tenemos nuestra cadena original.

```c
void concatenacion(Nodo** cabeza){

    char cad2[50];

    printf("Ingresa la segunda cadena para concatenar (si deseas concatenar una cadena vacia ingresa 0): ");
    scanf("%s", cad2);

    Nodo* nuevo = crearNodo(cad2);

    if(strcmp(cad2, "0") == 0){

        mostrarLista(*cabeza);
        return;

    }else if(*cabeza == NULL){
        *cabeza = nuevo;
    }else{
        Nodo* actual = *cabeza;
        while(actual->siguiente != NULL){
            actual = actual->siguiente;
        }
        actual->siguiente = nuevo;
    }

    mostrarLista(*cabeza);

}
```

Para poder sacar la potencia negativa necesitaremos la función invertir cadena, que hace un cambio de variable algo parecido a como liberamos la lista, solo que en este caso copiamos los valores de la cadena original a nuestro auxiliar para hacer la inversión y guardamos los nuevos valores en la cadena original.

```c
void invertirCadena(char* cad){

    int lon = strlen(cad);
    char temp;

    for(int i = 0, j = lon-1; i<j; i++, j--){
        temp = cad[i];
        cad[i] = cad[j];
        cad[j] = temp;
    }

}
```

```c
void potencia(Nodo** cabeza, char cad[]){

    int pot;

    printf("Ingresa el valor de la potencia: ");
    scanf("%i", &pot);

    int absPot = abs(pot);

    if(*cabeza != NULL){

        liberarLista(*cabeza);
        *cabeza = NULL;
    }

    if(pot == 0){

        printf("Potencia: Cadena vacia");
        return;

    }if(pot < 0){

        invertirCadena(cad);

    }
```

Para la potencia, nuevamente si la potencia es 0, únicamente se regresa un texto que es equivalente a decir que nuestra potencia es λ, si la potencia es negativa, se obtiene el valor absoluto para poder pasárselo al ciclo for e invertimos la cadena para finalmente ir agregando a una nueva lista esta cadena invertida las n veces que ingreso el usuario.

```c
    for(int i=1; i <= absPot; i++){

        Nodo* nuevo = crearNodo(cad);

        if(*cabeza == Nodo *nuevo
            *cabeza = nuevo;
        }else{
            Nodo* actual = *cabeza;
            while(actual->siguiente != NULL){
                actual = actual->siguiente;
            }
            actual->siguiente = nuevo;
        }

    }

    mostrarLista(*cabeza);


}
```

Para la potencia positiva se salta lo anterior y llega a este paso final, y por último se muestra la cadena con la función anteriormente explicada.

```c
void longitudCadena(char cad[]){

    int lon = 0;

    while(cad[++lon] != 0); // incrementar  la variable y regresar  el valor despu s de incrementarla
    printf("La longitud de %s es %d", cad, lon);

}
```

Aquí tenemos un ciclo while que recorre la cadena y va contando el número de elementos para obtener la longitud.

```c
void prefijos(char cad[]){

    int l = strlen(cad);
    char cadMod[50];

    for(int i=0; i<=l; i++){
        cadMod[i] = cad[i];
    }

    for(int i=0; i<=l; i++){
        printf("Prefijo %d: ", i);
        for(int j=0; j<i; j++){
            printf("%c", cadMod[j]);
        }
        printf("\n");
    }

}
```

Para la función prefijos utilizamos un ciclo para primero hacer un cambio a otra cadena y así no tener problemas con la cadena original, posteriormente se genera otro ciclo for para recorrer cada carácter y obtener el número de prefijos y un segundo ciclo anidado para limitar las cadenas y así obtener los prefijos.

```c
void sufijos(char cad[]){

    int l = strlen(cad);
    char cadMod[50];

    for(int i=0; i<=l; i++){
        cadMod[i] = cad[i];
    }

    invertirCadena(cadMod);

    for(int i=0; i<=l; i++){
        printf("Sufijo %d: ", i);
        for(int j=i-1; j<l && j>=0; j--){
            printf("%c", cadMod[j]);
        }
        printf("\n");
    }

}
```

Para los para los sufijos aplicamos la misma lógica que con los prefijos, sólo que en este caso hay que invertir la cadena, pero como ya ocupamos una función para la parte de potencia negativa, únicamente mandamos llamar, la cadena se invierte y obtiene los sufijos.

```c
void subcadenas(char cad[]){

    int l = strlen(cad);

    printf("\n");

    for(int i=0; i<=l; i++){

        for(int j=0; j<l-i; j++){

            for(int k=0; k<=i; k++){

                printf("%c", cad[j+k]);

            }

            printf("\n");

        }

    }

}
```

Para las subcadenas, aplicamos el primer for para recorrer la cadena, el segundo para decidir cuantos elementos vamos a ir trabajando y el último para que se impriman.

Ahora bien, para la parte de las subsecuencias, como vamos a utilizar la lista que ya tenemos, implementaremos algunas funciones que nos apoyarán en el desarrollo del código.

```c
void eliminarElemento(Nodo** cabeza, const char* datoEli){

    Nodo* actual = *cabeza;
    Nodo* anterior = NULL;

    while(actual != NULL && strcmp(actual->dato, datoEli) != 0) {
        anterior = actual;
        actual = actual->siguiente;
    }

    if (actual == NULL) {
        printf("El elemento no se encontro en la lista.\n");
        return;
    }

    if (anterior == NULL) {
        *cabeza = actual->siguiente;
    } else {
        anterior->siguiente = actual->siguiente;
    }

    free(actual->dato);
    free(actual);
    printf("Elemento eliminado exitosamente.\n");

}
```

En eliminar elemento lo que va a hacer es que si la cadena esta llena y es diferente de una cadena vacía, va a asignar nuestra lista a un nodo nuevo y si ese nodo tiene un elemento, este se elimina y se libera ese espacio de memoria.

En estas dos funciones:

La primera crea cadenas sin carácter (subsecuencias) utilizando un nuevo espacio de memoria en donde concatenara la cadena que recordemos de la que ya se ha eliminado un elemento.

```c
void generarSubsecuenciasRecursivo(Nodo** cabeza, const char* cad_actual) {
    int lon = strlen(cad_actual);

    if (lon <= 1) {
        if (!existeEnLista(*cabeza, cad_actual)) {
            Nodo* nuevo_nodo = crearNodo(cad_actual);
            if (*cabeza == NULL) {
                *cabeza = nuevo_nodo;
            } else {
                Nodo* actual = *cabeza;
                while (actual->siguiente != NULL) {
                    actual = actual->siguiente;
                }
                actual->siguiente = nuevo_nodo;
            }
        }
        return;
    }

    if (!existeEnLista(*cabeza, cad_actual)) {
        Nodo* nuevo_nodo = crearNodo(cad_actual);
        if (*cabeza == NULL) {
            *cabeza = nuevo_nodo;
        } else {
            Nodo* actual = *cabeza;
            while (actual->siguiente != NULL) {
                actual = actual->siguiente;
            }
            actual->siguiente = nuevo_nodo;
        }
    }
    for (int i = 0; i < lon; i++) {
        char* sub_cadena = crearCadenaSinCaracter(cad_actual, i);
        if (sub_cadena != NULL) {
            generarSubsecuenciasRecursivo(cabeza, sub_cadena);
            free(sub_cadena);
        }
    }
}
```

```c
char* crearCadenaSinCaracter(const char* original, int indice) {
    int lon = strlen(original);

    if (indice < 0 || indice >= lon) {
        return NULL;
    }


    char* nueva = (char*)malloc(lon);

    if (nueva == NULL) {
        return NULL;
    }

    strncpy(nueva, original, indice);
    strcpy(nueva + indice, original + indice + 1);

    return nueva;
}

int existeEnLista(Nodo* cabeza, const char* dato) {
    Nodo* actual = cabeza;
    while (actual != NULL) {
        if (strcmp(actual->dato, dato) == 0) {
            return 1;
        }
        actual = actual->siguiente;
    }
    return 0;
}
```

La segunda función únicamente es para que no se repitan las subsecuencias que ya existen y su única función es comparar la generada con la lista.

También vamos a tener la función que nos genere las subsecuencias por medio de los nodos de nuestra lista, en donde si el elemento no existe crea un espacio para agregarlo a la lista, pero si existe continua con otra comparación.

Y añadimos una función para mostrar únicamente las subsecuencias ya que se necesitaba de un espacio o algo para identificarlas.

```c
void mostrarListaSubsecuencias(Nodo* cabeza) {
    Nodo* actual = cabeza;
    while (actual != NULL) {
        printf("%s", actual->dato);
        if (actual->siguiente != NULL) {
            printf(", ");
        }
        actual = actual->siguiente;
    }
    printf("\n");
}

void subsecuencias(Nodo** cabeza, char cad[]) {
    int l = strlen(cad);

    if (l > 14) {
        printf("La cadena excede el limite de caracteres (máximo 14).\n");
        return;
    }

    liberarLista(*cabeza);
    *cabeza = NULL;

    generarSubsecuenciasRecursivo(cabeza, cad);

    printf("Las subsecuencias generadas son:\n");
    mostrarListaSubsecuencias(*cabeza);

}
```

```c
int main(){

    int numcad;
    char cad1[50];
    char cad2[50];

    //Variables para concatenar cadenas
    char cadCon[100];
    char *ptrcadCon = cadCon;
    char *ptrcad1 = cad1;
    char *ptrcad2 = cad2;

    Nodo* cabeza = NULL;

    printf("Con cuantas cadenas desea trabajar: ");
    scanf("%i", &numcad);
    fflush(stdin);

    if(numcad == 1){

    printf("Ingresa la cadena alfanumerica: ");
    scanf("%s", cad1);
    Nodo* nodoMain = crearNodo(cad1);
    nodoMain->siguiente = cabeza;
    cabeza = nodoMain;
    menu(&cabeza, cad1);
}if(numcad == 2){

printf("Ingresa la primer cadena alfanumerica: ");
scanf("%s", cad1);
printf("Ingresa la segunda cadena alfanumerica: ");
scanf("%s", cad2);

printf("Cadenas ingresadas: \n 1- %s \n 2- %s", cad1, cad2);


        while(*ptrcad1 != '\0'){

            *ptrcadCon = *ptrcad1;
            ptrcadCon++;
            ptrcad1++;
        }

        *ptrcadCon = '\0';

        while(*ptrcadCon != '\0'){

            ptrcadCon++;

        }
```

En la función principal de las subsecuencias se mandan llamar las anteriores para que ya haga el procedimiento completo además de que se delimitó a solo recibir cadenas menores de 14 caracteres.

Para terminar, aquí tenemos el main, donde se manda llamar al menú y de ahí a cada función respectivamente, solo que para dos cadenas, el programa las concatena para posteriormente poder trabajar con ellas en cada una de las opciones del menú.

```c
            while(*ptrcad2 != '\0'){

                *ptrcadCon = *ptrcad2;
                ptrcadCon++;
                ptrcad2++;
            }

            *ptrcadCon = '\0';

            printf("\n La cadena concatenada es: %s\n", cadCon);
            Nodo* nodoMain = crearNodo(cadCon);
            nodoMain->siguiente = cabeza;
            cabeza = nodoMain;
            menu(&cabeza, cadCon);
    }

        return 0;
}
```

Resultados

```
Con cuantas cadenas desea trabajar: 2
Ingresa la primer cadena alfanumerica: casa
Ingresa la segunda cadena alfanumerica: verde
Cadenas ingresadas:
 1- casa
 2- verde
 La cadena concatenada es: casaverde



        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 1

Ingresa la segunda cadena para concatenar (si deseas concatenar una cadena vacia ingresa 0): 0
Resultado: casaverde
```

```
        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 2

Ingresa el valor de la potencia: 0
Potencia: Cadena vacia
```

```
        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 2

Ingresa el valor de la potencia: 3
Resultado: casaverdecasaverdecasaverde
```

```
        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 3

La longitud de casaverde es 9
```

```
        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 8

Vuelve pronto! :)
```

```
        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 4

Prefijo 0:
Prefijo 1: c
Prefijo 2: ca
Prefijo 3: cas
Prefijo 4: casa
Prefijo 5: casav
Prefijo 6: casave
Prefijo 7: casaver
Prefijo 8: casaverd
Prefijo 9: casaverde
```

```
        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 5

Sufijo 0:
Sufijo 1: e
Sufijo 2: de
Sufijo 3: rde
Sufijo 4: erde
Sufijo 5: verde
Sufijo 6: averde
Sufijo 7: saverde
Sufijo 8: asaverde
Sufijo 9: casaverde
```

```
        Elige una opcion:

 1. Concatenacion
 2. Potencia
 3. Calculo de la longitud
 4. Generacion de prefijos
 5. Generacion de sufijos
 6. Generacion de subcadenas
 7. Generacion de subsecuencias
 8. Salir

 Opcion? 2

Ingresa el valor de la potencia: -2
Resultado: edrevasacedrevasac
```

```
        Elige una opcion:

  1. Concatenacion
  2. Potencia
  3. Calculo de la longitud
  4. Generacion de prefijos
  5. Generacion de sufijos
  6. Generacion de subcadenas
  7. Generacion de subsecuencias
  8. Salir

  Opcion? 6


c
a
s
a
v
e
r
d
e
ca
as
sa
av
ve
er
rd
de
```

```
de
cas
asa
sav
ave
ver
erd
rde
casa
asav
save
aver
verd
erde
casav
asave
saver
averd
verde
casave
asaver
saverd
averde
casaver
asaverd
saverde
casaverd
asaverde
casaverde
```

```
        Elige una opcion:

  1. Concatenacion
  2. Potencia
  3. Calculo de la longitud
  4. Generacion de prefijos
  5. Generacion de sufijos
  6. Generacion de subcadenas
  7. Generacion de subsecuencias
  8. Salir

  Opcion? 7

Las subsecuencias generadas son:
casaverde, asaverde, saverde, averde, verde, erde, rde, de, e, d, re, r, rd, ede, ee, ed, ere, er, erd, vrde, vde, ve, v, vd, vre, vr, vrd, vede, vee, ved, vere, ver, verd, aerde, arde,
 ade, ae, a, ad, are, ar, ard, aede, aee, aed, aere, aer, aerd, avrde, avde, ave, av, avd, avre, avr, avrd, avede, avee, aved, avere, aver, averd, sverde, serde, srde, sde, se, s, sd, s
re, sr, srd, sede, see, sed, sere, ser, serd, svrde, svde, sve, sv, svd, svre, svr, svrd, svede, svee, sved, svere, sver, sverd, saerde, sarde, sade, sae, sa, sad, sare, sar, sard, saed
e, saee, saed, saere, saer, saerd, savrde, savde, save, sav, savd, savre, savr, savrd, savede, savee, saved, savere, saver, saverd, aaverde, aaerde, aarde, aade, aae, aa, aad, aare, aar
, aard, aaede, aaee, aaed, aaere, aaer, aaerd, aavrde, aavde, aave, aav, aavd, aavre, aavr, aavrd, aavede, aavee, aaved, aavere, aaver, aaverd, asverde, aserde, asrde, asde, ase, as, as
d, asre, asr, asrd, asede, asee, ased, asere, aser, aserd, asvrde, asvde, asve, asv, asvd, asvre, asvr, asvrd, asvede, asvee, asved, asvere, asver, asverd, asaerde, asarde, asade, asae,
 asa, asad, asare, asar, asard, asaede, asaee, asaed, asaere, asaer, asaerd, asavrde, asavde, asave, asav, asavd, asavre, asavr, asavrd, asavede, asavee, asaved, asavere, asaver, asaver
d, csaverde, caverde, cverde, cerde, crde, cde, ce, c, cd, cre, cr, crd, cede, cee, ced, cere, cer, cerd, cvrde, cvde, cve, cv, cvd, cvre, cvr, cvrd, cvede, cvee, cved, cvere, cver, cve
rd, caerde, carde, cade, cae, ca, cad, care, car, card, caede, caee, caed, caere, caer, caerd, cavrde, cavde, cave, cav, cavd, cavre, cavr, cavrd, cavede, cavee, caved, cavere, caver, c
averd, csverde, cserde, csrde, csde, cse, cs, csd, csre, csr, csrd, csede, csee, csed, csere, cser, cserd, csvrde, csvde, csve, csv, csvd, csvre, csvr, csvrd, csvede, csvee, csved, csve
re, csver, csverd, csaerde, csarde, csade, csae, csa, csad, csare, csar, csard, csaede, csaee, csaed, csaere, csaer, csaerd, csavrde, csavde, csave, csav, csavd, csavre, csavr, csavrd,
 csavede, csavee, csaved, csavere, csaver, csaverd, caaverde, caaerde, caarde, caade, caae, caa, caad, caare, caar, caard, caaede, caaee, caaed, caaere, caaer, caaerd, caavrde, caavde, c
aave, caav, caavd, caavre, caavr, caavrd, caavede, caavee, caaved, caavere, caaver, caaverd, casverde, caserde, casrde, casde, case, cas, casd, casre, casr, casrd, casede, casee, cased,
 casere, caser, caserd, casvrde, casvde, casve, casv, casvd, casvre, casvr, casvrd, casvede, casvee, casved, casvere, casver, casverd, casaerde, casarde, casade, casae, casa, casad, cas
are, casar, casard, casaede, casaee, casaed, casaere, casaer, casaerd, casavrde, casavde, casave, casav, casavd, casavre, casavr, casavrd, casavede, casavee, casaved, casavere, casaver,
 casaverd
```