Computación I
Estructuras de datos II

Curso 2024

Facultad de Ingeniería

Universidad de la República

Temario

- Estructuras de datos
  - Cell arrays
  - Struct
- Manejo de archivos
  - Save, load
  - Funciones de bajo nivel (fopen, fclose)

Estructuras de datos

- Variables que almacenan más de un valor
  - Los valores deberían estar relacionados de alguna forma
  - Ya hemos visto un tipo de estructura: el arreglo o array (vectores y matrices)
    - Rígidas: los valores están organizados en filas y columnas, deben ser del mismo tipo (enteros, reales, etc.)
    - Útiles cuando debemos recorrer los valores (for, while), ordenarlos, etc.
    - Las usamos para representar polinomios y matrices dispersas

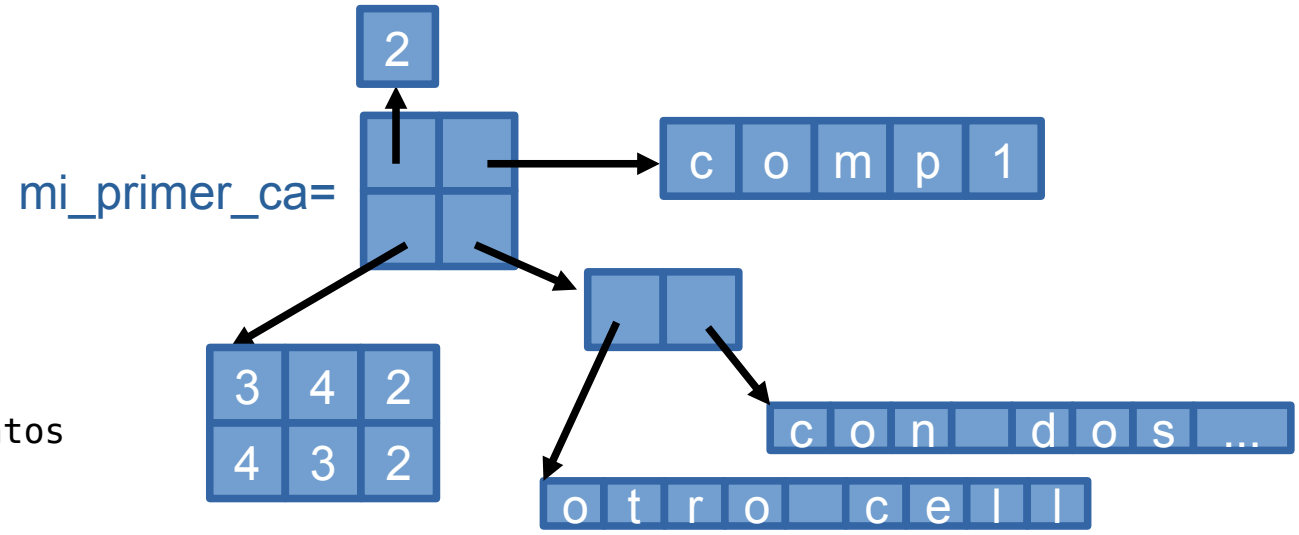
Cell array

- Vectores y matrices con valores de distinto tipo
  - En realidad se almacena un **puntero** al valor:
    - La dirección donde ese valor está almacenado en la memoria
  - Los valores pueden ser números, textos, matrices/vectores, otros cell arrays, etc.
  - Estructura más flexible que conserva todo el poder de matrices y vectores para recorrer los valores

Cell array

- Crear un cell array: como con matrices pero usando { }
  - Ejemplo: cell array de 2x2 que almacena un entero, un texto, una matriz y otro cell array:

```
octave:1> mi_primer_ca={2, 'comp1';[3 4 2; 4 3 2], {'otro cell', 'con dos elementos'}}
mi_primer_ca =
{
  [1,1] = 2
  [2,1] =
     3   4   2
     4   3   2
  [1,2] = comp1
  [2,2] =
  {
    [1,1] = otro cell
    [1,2] = con dos elementos
  }
}
```



Cell array

- Al igual que con las matrices Octave permite agregar elementos al final del cell array:

```
octave:1> mi_primer_ca(3,:)={‘otra’,’fila’}
```

- Ventaja: cómodo, permite construir el CA elemento a elemento cuando no sabemos el largo de antemano.

```
octave:1> mi_primer_ca{4,1}={‘otra fila más, elem (4,2) vacío’}
```

- Desventaja: ineficiente. Si sabemos el largo es mejor reservar primero el espacio usando la función cell (parecida a zeros pero para cell arrays…)

```
octave:1> mi_segundo_ca=cell(4,3) %cell array vacío de tam 4x3
```

Cell array

- Repaso: Los cell arrays almacenan punteros a los valores
  - Por lo tanto hay dos formas de acceder a un elemento:
    - Acceder al puntero - (paréntesis)
    - Acceder al valor apuntado - {llaves}
- Ejemplo:

```
octave:1> mi_primer_ca={2, 'comp1';[3 4 2; 4 3 2], {'otro cell', 'con dos elementos'}};
octave:2> mi_primer_ca{2,1}
ans =
   3   4   2
   4   3   2
octave:3> mi_primer_ca{2,1}(2,3)
ans =
   2
```

```
octave:4> mi_primer_ca(2,1)
ans =
{
  [1,1] =
     3   4   2
     4   3   2
}
```

Cell array

- Para asignar un valor debo usar { }:

```
octave:1> mi_primer_ca={2, 'comp1';[3 4 2; 4 3 2], {'otro cell', 'con dos elementos'}};
octave:2> mi_primer_ca{1,2}=strcat(mi_primer_ca{1,2},' está demás!')
mi_primer_ca =
{
  [1,1] = 2
  [2,1] =
     3   4   2
     4   3   2
  [1,2] = comp1 está demás!
  [2,2] =
  {
    [1,1] = otro cell
    [1,2] = con dos elementos
  }
}
```

Cell array

- Para cambiar lo apuntado por una celda debo usar ( ):

```
octave:1> mi_primer_ca={2, 'comp1';[3 4 2; 4 3 2], {'otro cell', 'con dos elementos'}};
octave:2> mi_primer_ca(1,2)={'asigno un cell array'}
mi_primer_ca =
{
  [1,1] = 2
  [2,1] =
     3   4   2
     4   3   2
  [1,2] = asigno un cell array
  [2,2] =
  {
    [1,1] = otro cell
    [1,2] = con dos elementos
  }
}
```

Cell array

- Se pueden obtener sub-arrays de un cell array...
  - Accediendo a los índices obtengo un cell array:

```
octave:22> otro_ca = mi_primer_ca(1,:);
octave:23> otro_ca
otro_ca =
{
  [1,1] = 2
  [1,2] = asigno un cell array
}
```

  - Accediendo a los elementos obtengo variables separadas:

```
octave:27> [ca11, ca12] = mi_primer_ca{1,:}
ca11 = 2
ca12 = asigno un cell array
```

Cell array

- Obtener el tamaño…

```
octave:22> c1 = {2, 'asdf'}; c2 = {[2 3 1], 'b'};
octave:23> lc1 = length(c1); [nfc2, ncc2] = size(c2);
```

- Concatenar...

```
octave:23> c3 = [c1 ; c2]
```

- Borrar una columna...

```
octave:27> c3(:,2)=[]
```

- Otras…
  - num2cell(), cellstr(), cell2mat(), iscell(), iscellstr(), etc...

Ejercicio

- Obtener la palabra que más aparece en un texto
  - Usaremos la función strsplit(texto,separador) para separar el texto en un cell array de palabras
  - Para comparar palabras usamos strcmp(pal1, pal2)
- Procedimiento:

Ejercicio

- Obtener la palabra que más aparece en un texto
  - Usaremos la función strsplit(texto,separador) para separar el texto lo representaremos como un cell array de palabras
  - Para comparar palabras usamos strcmp(pal1, pal2)
- Procedimiento:
  - Crear otro cell array de dos columnas {palabra, apariciones} para contar las apariciones de cada palabra
  - Para cada palabra del texto busco en el CA apariciones
    - Si aparece aumento la cantidad y actualizo el máximo
    - Si no aparece la agrego con 1 aparición

Ejercicio: primera solución

```
function pmf = palabraMasFrecuente_simple(texto)

    maxApariciones = 0; pmf = '';
    % Dividir el texto en un cell array de palabras
    palabras = strsplit(texto, ' ');

    % cell array para las apariciones de cada palabra
    apariciones = {};

    for i = 1:length(palabras)
        palabra = palabras{i};
        % Para cada palabra del texto busco en el CA apariciones
        j=1;
        palabraEsta=0;
        while j <= size(apariciones,1) && ~palabraEsta
            if strcmp(palabra, apariciones{j,1})
                palabraEsta=1;
            else
                j=j+1;
            end
        end

        if palabraEsta
            apariciones{j,2} = apariciones{j,2} + 1;
        else
            apariciones=[apariciones;{palabra,1}];
        end

        if apariciones{j,2} > maxApariciones
            maxApariciones = apariciones{j,2};
            pmf = palabra;
        end
    end
end
```

- Por cada palabra del texto debo recorrer el CA de apariciones hasta encontrar la palabra…
  - La recorrida puede ser costosa
  - Si supiera aprox. dónde está la palabra podría recorrer menos elementos...

Implementar un *hash* con CAs

- Hash: Especie de tabla a la que accedemos a través de una función fácil de calcular (función de hash)
  - Lo usaremos para acceder rápido a la cantidad de apariciones de cada palabra en lugar de buscarla en todo el CA
  - En nuestro caso podemos usar la suma del valor de las letras de una palabra módulo el tamaño del hash.
  - Ejemplo: si el hash se llama *palabras* la palabra “hola” se guarda en la posición mod(sum(‘hola’),length(palabras))+1
- En cada posición del hash puede haber más de una palabra (la función de hash puede dar el mismo resultado para dos palabras distintas)
  - Cada posición del hash puede implementarse como un cell array igual al que usamos antes.

Ejercicio: solución con hash

```
function pmf = palabraMasFrecuente(texto, tamHash)

    maxApariciones = 0; pmf = '';
    palabras = strsplit(texto, ' ');

    hash_apariciones = cell(tamHash,1);

    for i = 1:length(palabras)
        palabra = palabras{i}; j=1; palabraEsta=0;
        apariciones=hash_apariciones{H(palabra,tamHash)};

        while j <= size(apariciones,1) && ~palabraEsta
            if strcmp(palabra, apariciones{j,1})
                palabraEsta=1;
            else
                j=j+1;
            end
        end

        if palabraEsta
            apariciones{j,2} = apariciones{j,2} + 1;
        else
            apariciones=[apariciones;{palabra,1}];
        end

        if apariciones{j,2} > maxApariciones
            maxApariciones = apariciones{j,2};
            pmf = palabra;
        end

        hash_apariciones{H(palabra,tamHash)}=apariciones;
    end
end
```

- H(palabra, tamHash) es la función mod(sum(palabra),tamHash)+1

Structs

- Agrupan los datos en campos con nombre
  - Los datos de cada campo pueden ser de distinto tipo
  - Los structs permiten recordar fácilmente la información que está almacenada en una estructura
  - A diferencia de los arreglos, los campos de los structs no tienen índice, por lo que no es posible iterar sobre ellos
  - Sí es posible crear arreglos de structs.
- Ejemplo: Datos de una persona
  - Nombre - texto
  - Edad - número
  - Foto - imagen

Structs

- Los campos de un struct se acceden a través del punto (.)
- Crear un struct:
  - Campo a campo

    persona.nombre = ’Ernesto’

    persona.edad = 35
  - Usando la función struct()

    persona = struct(‘nombre’,’Ernesto’,’edad’,35)
- Al igual que con matrices y CAs, es más eficiente utilizar la función struct() si se conocen los campos que contendrá el struct.

Structs

- Como cualquier variable pueden asignarse a otra (creando una copia):

  otra_persona = persona

  otra_persona.nombre = ‘Martín’

- Para eliminar un campo es necesario utilizar rmfield():

  persona = rmfield(persona,‘nombre’)

  (se asigna a persona una copia del struct sin el campo nombre)

- Es posible pasar structs a funciones (como con cualquier variable)
  - los nombres de los campos accedidos dentro de la función deben coincidir con los del struct...

Structs

- Pueden formarse arreglos de structs (vectores y matrices):
  - Elemento a elemento (ineficiente):

```
persona(1)=struct('Nombre','Juan','Edad',20)
persona(2)=struct('Nombre','María','Edad',23)
```

  - Si se conoce el tamaño final puede crearse un arreglo de structs de ese tamaño asignando el último elemento:

```
persona(20)=struct('Nombre',[],'Edad',[]) % crea un arreglo vacío de 20 personas
```

    - Luego puedo llenarlo accediendo a los campos usando el .

```
persona(1).Nombre='Juan'
persona(1).Edad=20
```

- Tres niveles: persona, persona(i), persona(i).Nombre/Edad

Structs

- Ejercicio: escribir en octave una función que reciba un vector de structs de personas con los campos Nombre (texto) y Edad (entero) y devuelva el nombre de la persona más joven:

Structs

- Ejercicio: escribir en octave una función que reciba un vector de structs de personas con los campos Nombre (texto) y Edad (entero) y devuelva el nombre de la persona más joven:

```
function nombre = mas_joven(personas)
        lp = length(personas);
        if lp > 0
                nombre=personas(1).Nombre;
                min_edad=personas(1).Edad;
                for i = 2:lp
                        if personas(i).Edad < min_edad
                                min_edad=personas(i).Edad;
                                nombre=personas(i).Nombre;
                        endif
                endfor
        else
                nombre='';
        endif
```

Structs

- Los campos de un struct pueden ser de cualquier tipo, incluso un arreglo/matriz, CA u otro struct:
  - Ejemplo: colección de matrices dispersas. Cada matriz es un struct de tres vectores:

```
mat(1).fil=[1 2 3 4 5];
mat(1).col=[3 1 3 4 4];
mat(1).val=ones(1,5);
mat(2).fil=[1 2 3 4 5 5 7 7 7];
mat(2).col=[3 1 3 4 4 8 3 6 7];
mat(2).val=ones(1,10);
```

    - Índice de columna del 4to no-cero de la 2da matriz:
      - `mat(2).col(4)`
  - Matriz disp. como arreglo de structs: mat(j).fil/col/val son las coordenadas del elem. j de la matriz. Cómo almaceno la colección? Cómo accedo al elem j de mat i?
    - Opción 1: Matriz de stucts –
    - Opción 2: Usar un cell array –

Structs

- Los campos de un struct pueden ser de cualquier tipo, incluso un arreglo/matriz u otro struct:
  - Ejemplo: colección de matrices dispersas. Cada matriz es un struct de tres vectores:

```
mat(1).fil=[1 2 3 4 5];
mat(1).col=[3 1 3 4 4];
mat(1).val=ones(1,5);
mat(2).fil=[1 2 3 4 5 5 7 7 7];
mat(2).col=[3 1 3 4 4 8 3 6 7];
mat(2).val=ones(1,10);
```

    - Índice de columna del 4to no-cero de la 2da matriz:
      - `mat(2).col(4)`
  - Matriz disp. como arreglo de structs: mat(j).fil/col/val son las coordenadas del elem. j de la matriz. Cómo almaceno la colección? Cómo accedo al elem j de mat i?
    - Opción 1: Matriz de stucts – mat(i,j).fil/col/val
    - Opción 2: Usar un cell array – colecc_mat{i}(j).fil/col/val

Structs

- Es importante diferenciar las distintas partes de la variable
  - En el ejemplo anterior: `mat(2).col(4)`
    - `mat es un vector de structs (de tamaño al menos 2)`
    - `mat(2) es el struct almacenado en la posición 2 del vector`
    - `mat(2).col es un vector que contiene los índices de columna de mat(2)`
    - `mat(2).col(4) es el índice de columna del 4to no-cero de mat(2)`
  - Ejemplo con matriz de structs:
    - `mat es una matriz de structs`
    - `mat(i,j) es un struct que contiene 3 campos (fil, col y val)`
    - `mat(i,j).fil es un entero`
  - Ejemplo con cell arrays
    - `colecc_mat es un cell array`
    - `colecc_mat(i) apunta a un vector de structs`
    - `colecc_mat{i} es un vector de structs`
    - `colecc_mat{i}(j) es el j-ésimo struct del vector`

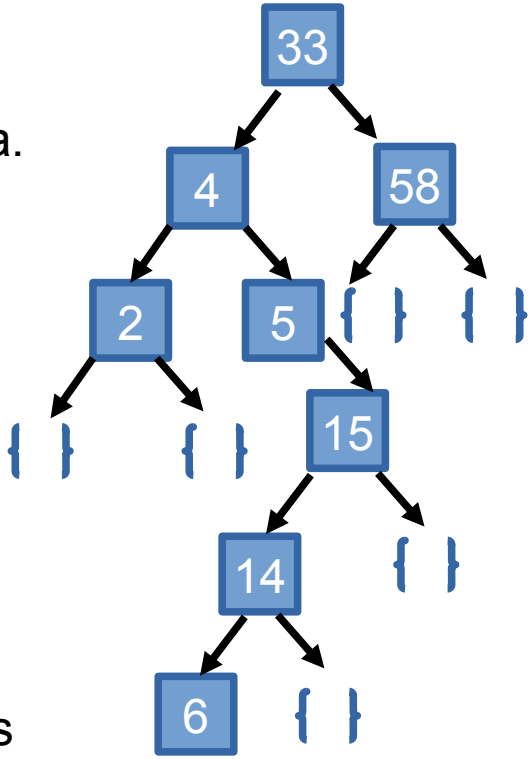
Structs

- Un struct puede anidarse dentro de otro...
  - Ejemplo: Representar un segmento de recta.
    - Cada campo de la estructura es un punto, que a su vez tiene tres coordenadas (x,y,z).
    - ```
      seg_rect=struct('ini',struct('x',3,'y',34,'z',93),
                      'fin',struct('x',1,'y',4 ,'z',333));
      ```
    - ```
      Para cambiar la coordenada 'y' del punto de fin uso dos veces el punto: seg_rect.fin.y=0;
      ```
- Esto puede combinarse con vectores de structs:
  - Por ejemplo, un polígono puede representarse como un struct que contenga un número de lados y un vector con un segmento de recta por cada lado:
    - ```
      pol.lados=3;
      ```
    - ```
      pol.lado(1).ini.x=1; pol.lado(1).ini.y=0; pol.lado(1).ini.z=1;
      ```
    - ```
      pol.lado(1).fin.x=1; pol.lado(1).fin.y=1; pol.lado(1).fin.z=1;
      ```
    - ```
      pol.lado(2).ini.x=...
      ```

Structs

Estructuras recursivas

- Al menos un campo tiene la misma estructura.
- Ejemplo: árbol binario de búsqueda ABB
  - Una estructura útil para buscar rápidamente valores numéricos.
  - Cada nodo del árbol tiene 3 campos:
    - un valor numérico,
    - un ABB izquierdo con valores menores al valor del nodo
    - un ABB derecho con valores mayores al valor del nodo



Ejercicio:

- Representamos un nodo de ABB como un struct con tres campos: val, izq, der.
- Escribir en Octave la función recursiva insertarABB que dado un ABB (representado con structs) y un número, inserte el número en el ABB.
  - Caso base: si recibo un número y un árbol vacío...
  - Caso recursivo: insertar el número en un árbol más chico...

Ejercicio:

```
function ABB = insertarABB(ABB, num)
    % Insertar un valor en un árbol binario de búsqueda
    if isempty(ABB)
        % Si el árbol está vacío devuelvo un ABB compuesto de un solo nodo
        ABB = struct('val',num, 'izq', [], 'der', []);
    else
        % Elijo el ABB donde debo insertar el nodo
        if num < ABB.val
            ABB.izq = insertarABB(ABB.izq, num);
        elseif num > ABB.val
            ABB.der = insertarABB(ABB.der, num);
        end
    end
end
```

Entrada y salida

- Interacción con el usuario
  - Obtener entrada directamente del usuario / desplegar en pantalla
  - Almacenar y obtener datos de archivos
  - Transmitir o recibir de internet?
- Cada lenguaje tiene sus herramientas
  - Octave brinda varias opciones con más o menos complejidad
  - Más control por parte del usuario = más complejidad...

Entrada y salida

- Interacción con el usuario
  - input(... , “s”) : Obtener entrada directamente del usuario
  - disp( ) : desplegar en pantalla
- Ejemplo:

```
octave:12> a =input("ingrese texto: ","s")
ingrese texto: hola
a = hola
octave:13> a =input("ingrese un número: ")
ingrese un número: 4
a = 4
octave:14> a =input("ingrese una matriz: ")
ingrese una matriz: [ 34, 3, 4; 1, 3, 11]
a =

   34    3    4
    1    3   11
```

Entrada y salida: archivos

save / **load**: permiten guardar/cargar en/desde un archivo las variables del entorno de octave

- También permiten leer archivos de texto con datos regulares (almacenables en una matriz)
- Fáciles de usar pero limitadas

```
octave:15> whos
Variables visible from the current scope:


variables in scope: top scope


  Attr   Name        Size                     Bytes  Class
  ====   ====        ====                     =====  =====
         a           2x3                         48  double
         ans         2x3                         48  double


Total is 12 elements using 96 bytes


octave:16> save('variables.m')
octave:17> clear all
octave:18> whos
```

```
octave:19> load('variables.m')
octave:20> whos
Variables visible from the current scope:

variables in scope: top scope

  Attr   Name        Size                     Bytes  Class
  ====   ====        ====                     =====  =====
         a           2x3                         48  double
         ans         2x3                         48  double

Total is 12 elements using 96 bytes
```

Entrada y salida: archivos

- Funciones de bajo nivel
  - Más control del usuario (parecido a lenguajes como C++)
  - Abrir un archivo: fid = fopen(‘nombre archivo’,’permiso’)
    - Si el archivo se leyó bien devuelve el identificador fid
    - Si hubo error devuelve -1
    - Permiso puede ser ‘r’ si se abre para lectura, ‘w’ para escritura, ‘a’ si es para agregar al final.
  - Cerrar un archivo: fclose(fid)
    - Si es exitosa devuelve 0

Entrada y salida: archivos

- Funciones de bajo nivel: Leer
  - linea=fgetl(fid) : lee un archivo de texto línea por línea
    - Es útil usarla dentro de un loop
    - Se puede combinar con funciones para procesar cadenas de caracteres (ej: strtok ...)
    - Cuando el archivo se termina devuelve -1
  - mat = fscanf(fid, 'formato', [dimensiones])
    - Lee por columnas y guarda en una matriz (todos los datos deben poder almacenarse en una matriz)
    - "formato" especifica el formato de cada fila (ej: "%d, %c" significa que cada línea tiene un número, una coma, un espacio y un carácter)
    - Dimensiones especifica las dimensiones de mat (si no se conocen puede ser Inf)
  - cell = textscan(fid,'format')
    - Parecida a fscanf pero devuelve un cell array
    - Los datos de cada columna pueden ser de distinto tipo
  - fscanf y textscan pueden leer varias líneas

Entrada y salida: archivos

- Ejemplo: leer una planilla de datos
  - Los datos están en columnas y son todos numéricos
    - Usar load
  - El archivo tiene varias líneas, con una fecha y una cotización de moneda en forma “dd/mm/aaaa : $##,##” y se quiere generar una matriz con una columna para día, otra para mes, año, y cotización:
    - Usar M = fscanf(fid,’%d/%d/%d : $%.2f’,[5,Inf]);
  - El mismo archivo pero se quiere generar un cell array de dos columnas con la fecha en formato texto (igual que en el archivo):
    - Usar C = textscan(fid,’%s : $%.2f’);

Resumen

- Los datos pueden agruparse en distintas estructuras
  - De la estructura elegida dependerá el tamaño ocupado en la memoria y la velocidad en el acceso a los datos
  - En Octave disponemos de matrices/vectores, cell arrays y structs.
  - Otros lenguajes tienen estructuras básicas similares
  - Con estas estructuras básicas podemos armar estructuras más complejas (arreglos de structs, árboles, hash,…)
- Octave (y otros lenguajes) proporciona funciones de E/S
  - Permiten interactuar con el usuario
  - Leer o escribir datos en archivos
  - Usarlas en combinación con matrices, structs, cell arrays, funciones de Octave y programas propios dan una capacidad enorme de procesar datos!