Tipus estructurat de dades homogènies. Taules

PID_00268290

Autors que han participat col·lectivament en aquesta obra
Lluís Beltrà
Kenneth Capseta
Maria Jesús Marco Galindo
Antonio Ponce

Idea i direcció de l'obra
Maria Jesús Marco Galindo

Disseny i edició gráfica
Asunción Muñoz

Material docent de la UOC

Universitat Oberta de Catalunya

Tercera edició: febrer 2021

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Continguts

Tipus estructurat de dades homogènies

Taules

Els tipus de dades que hem vist fins ara, a mesura que treballem amb problemes més complexos resulten insuficients i ineficients. Per aquest motiu utilitzem un altre tipus de dades més estructurat que ens permetrà representar millor alguns d'aquests problemes. Es tracta de les **taules**.

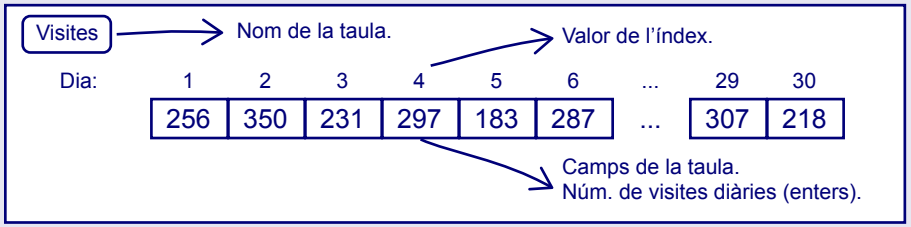
Les taules serveixen per agrupar elements que són del mateix tipus de dades simple (**estructura homogènia**) amb un ordre determinat i sota un únic nom.

Què és una TAULA?

Una TAULA és un tipus estructurat de dades que permet agrupar informació del mateix tipus (homogènia).

Exemple

Imaginem que som els gestors d'una important plataforma web i necessitem tenir un control de les visites que rep la web mensualment. Per emmagatzemar eficientment aquestes dades, segurament les enregistraríem d'una forma similar a:



Observeu que hem creat una taula amb el nom "Visites", que té 30 camps, corresponents a cada dia del mes, i en cadascun d'aquests camps s'hi emmagatzema el nombre de visites rebudes aquell dia que és un valor enter.

Fixeu-vos també que cada camp té assignat un índex, que és la posició d'aquest camp dins la taula.

Com representem aquest registre en notació algorísmica?

Podem representar aquesta estructura amb una taula i les dades a emmagatzemar (el nombre de visites diàries) seran els camps d'aquesta taula:

```
var
    visites: taula[30] de enter;
fvar
```

Observeu que utilitzant una taula hem estat capaços d'emmagatzemar 30 valors en una sola variable. En canvi, sense el tipus taula hauríem d'haver creat 30 variables diferents de tipus enter, una per a cada valor. Així hem guanyat eficiència i claredat.

Sintaxi per declarar una taula

Cal recordar que una taula només pot emmagatzemar dades que siguin del mateix tipus. En aquest cas, hem d'emmagatzemar enters però podríem haver-hi emmagatzemat cadenes, reals, etc. Per exemple, podem utilitzar una taula per emmagatzemar el nom de tots els alumnes d'una aula:

```
var
   alumnes: taula[30] de cadena;
fvar
```

Accés als camps d'una taula

Per referir-nos a cada element simplement haurem d'indicar el nom de la taula i la posició que l'element, ocupa dins d'ella amb l'índex.

Per exemple, tot seguint la declaració que hem vist anteriorment on emmagatzemàvem les visites diàries de la web:

Aquestes referències **serien correctes**

```
visites[2]    → Accedim al segon camp de la taula.
visites[num]  → En cas que num sigui una variable de tipus i que contingui un valor comprés entre 1 i 30, accedirem al camp de la taula que tingui la posició indicada.
visites[30]   → Accedim al trentè camp de la taula. En el nostre exemple, és on es guarda el número de visites de l'últim dia del mes.
```

En canvi, aquestes referències **NO seran correctes**

```
visites       → No podem fer referència a la taula sense indicar algun camp concret.
visites[32]   → No podem fer referència a un índex que sobrepassa els límits marcats per la mida màxima de la taula.
visites["dues"] → L'índex ha de ser de tipus enter.
```

Concepte Clau

Les **taules** ens permeten agrupar un seguit d'elements tots ells del mateix tipus. D'aquesta manera podem fer certs tractaments de manera més eficient. Les taules ens obren una nova possibilitat amb l'accés directe als seus elements.

Ens quins casos ens són útils les taules?

Per a representar dades formades per un seguit d'elements (per una seqüència) del mateix tipus. És el cas per exemple de vectors, matrius o frases.

Com podem assignar i consultar els camps d'una taula?

Exemple

Podem utilitzar l'assignació com fèiem amb altres tipus de dades per assignar valors a cada camp de la taula per separat:

```
visites[1] := 520;   → Assignem a la primera posició de la taula visites el número 520.
```

Això significa que el primer dia del mes, la web ha obtingut 520 visites.

Pel que fa a la consulta, si volem utilitzar el valor d'un camp concret, hi podem accedir en la taula de forma directa:

```
x := 2 * visites[1];   → Consultem el número de visites del primer dia del mes i el dupliquem, de manera que x valdrà 1040.
```

També podem utilitzar les operacions de lectura i escriptura:

```
escriurePantalla(visites[1]);   → Escrivim per pantalla el nombre de visites del primer dia del mes.
visites[12] := llegirTeclat();  → Llegim a través del teclat el nombre de visites del 12è dia del mes i el guardem a la posició corresponent de la taula.
```

Com podem consultar la longitud d'una taula?

La longitud d'una taula ha quedat definida en la seva declaració:

```
var
    alumnes: taula[30] de cadena;
fvar
```

En aquest cas la taula pot contenir fins a 30 elements, que seria la seva longitud.

Alguns llenguatges de programació permeten utilitzar taules sense indicar la seva longitud inicial. Són taules que van creixent a mesura que s'hi van afegint elements. En aquest cas, és necessari poder saber d'alguna manera la longitud de la taula en un moment donat.

En notació algorísmica per comoditat disposem també també de la següent funció predefinida que retorna la longitud de la taula:

```
l := longitudTaula();
```

Retorna la longitud de la taula *t*, això és, la mida amb la que l'hem definit inicialment, i la guarda a la variable entera *l*.

Taules de diverses dimensions

Fins ara hem treballat amb taules d'una sola dimensió, que hem representat com un vector o seqüència. Doncs bé, en algunes situacions ens convindrà treballar amb taules de dues o més dimensions, el que podem representar com a **matrius**.

Exemple

Seguint amb l'exemple inicial, on s'hi emmagatzemen les visites diàries que rep la pàgina web en un mes, si volem tenir un control de les visites rebudes de tot l'any hauríem d'utilitzar una matriu similar a:

Visites → Nom de la taula.

Valor de l'índex horitzontal.

Dia: / Mes:	1	2	3	4	5	6	...	29	30
Gener	256	350	231	297	183	287	...	307	218
Febrer	315	198	197	216	187	214	...	297	246
Març	248	378	245	310	232	265	...	365	202
Abril	218	316	273	263	169	224	...	386	265
...									
Novembre	269	305	243	219	194	237	...	360	246
Desembre	276	363	211	248	146	230	...	309	286

Valor de l'índex vertical.

Camps de la taula. Núm. de visites diàries (enter).

Com veiem, estem treballant amb una matriu de dues dimensions: en un eix tindrem els dies que té un mes i en l'altre eix representarem els mesos de l'any.

Així doncs, tindrem dos índexs diferents que representaran, cadascun, un dels eixos.

Com representem la matriu bidimensional en algorísmica?

Amb la notació algorísmica, la podem representar com una taula bidimensional així:

```
var
    visites: taula[12,30] de enter;
fvar
```

Nom de la taula.

Tipus de dades que emmagatzema la taula.

Índex que representa els mesos.

Índex que representa els dies.

Sintaxi per declarar una taula multidimensional

Hem vist un exemple amb una taula de dues dimensions però podríem treballar amb taules de tres, quatre o més dimensions.

La declaració d'una taula **n-dimensional** es fa amb la sintaxi següent:

```
var
   nom: taula[mida1, mida2, ..., midaN] de tipus;
fvar
```

Les operacions d'assignació i consulta es realitzen de forma similar a com ho fèiem amb les taules d'una sola dimensió.

Exemple

```
visites[2,3] := 230;
```

→ Assignem un total de 230 visites al dia 3 de febrer.

Com representem les taules en JavaScript?

En llenguatge JavaScript (JS), les taules es representen amb el tipus **Array**. Igual que les taules, un **Array** és un conjunt ordenat de valors als quals s'accedeix mitjançant un índex (que indica la posició dins l'**Array** i que s'identifica per un nom).

Sintaxi per declarar una taula en JS

```
var nomTaula=Array(midaTaula);
```

A diferència de la definició de taula que hem vist en notació algorísmica, un mateix **Array** en JavaScript pot contenir elements de diferents tipus, per això no s'indica el tipus dels camps en la declaració.

De totes maneres, encara que el JS ho permeti, no és una bona pràctica definir un **Array** amb valors diferents tipus. És millor evitar-ho.

Accés als camps d'una taula en JS

Accedirem als valors a través de l'índex, tenint en compte que l'índex del primer element d'un **Array** sempre és zero (0), és dir, les posicions d'un **Array** de longitud n són 0,1,2,3,...n-1. És molt habitual en els llenguatges de programació que l'índex comenci a comptar a partir del 0.

Assignarem valors a l'**Array** accedint-hi de la mateixa manera:

```
nomTaula[índex]=valorCamp;
```

Recordeu que el llenguatge JavaScript té moltes més opcions que la notació algorísmica. Aquí adoptarem la més senzilla.

Trobareu altres opcions per a declarar i inicialitzar un **Array** a la guia de JavaScript de Mozilla.

Exemple

```
1  //Aquesta definició ens tornarà una taula buida de longitud 30
2  var visites = Array(30);
3  console.log(visites.length);
4  //Omplim la taula amb les dades dels set primers dies
5  //Atenció! El primer element sempre és 0
6  visites[0] = 256;
7  visites[1] = 350;
8  visites[2] = 231;
9  visites[3] = 297;
10 visites[4] = 183;
11 visites[5] = 267;
12 visites[6] = 321;
13 //Mostrem tots els camps de la taula
14 console.log(visites);
15 //Mostrem les visites del dia 4
16 console.log(visites[3]);
17 //Mostra les visites del darrer dia de la setmana
18 console.log(visites[6]);
19 //Mostra el primer camp de la taula
20 console.log(visites[0]);
21 //Mostrem el darrer camp de la taula que no hem omplert encara
22 console.log(visites[29]);
23
```

L'accés a cadascun dels elements d'un **Array** el fem tot indicant la seva posició a través de l'índex.

Què passaria si ...

...intentem afegir un valor a la posició següent a la que hem establert com a la longitud màxima de l'**Array** en la seva definició? En aquest cas, l'**Array** s'amplia automàticament i allarga la seva longitud en una posició més.

Exemple

En el nostre cas, atès que l'hem definit amb longitud 30, aquesta seria la 31a posició, el que és el mateix que dir **visites[30]** (recordeu que comencem numerant per 0 i per això la darrera posició té l'índex 30).

En aquest cas, l'**Array** s'amplia automàticament i tindrà una longitud de 31:

```
//Inicialment hem definit una taula de 30 elements
var visites = Array (30);
console.log(visites.length);
//Ens adonem que el mes té 31 dies i volem afegir que el dia 31
//ha tingut 400 visites a la pàgina web
visites[30] = 400;
//Comprova que s'ha afegit automàticament una posició
console.log(visites.length);
console.log(visites[30]);
```

I si fem el mateix a la posició 33 (visites[32])? Doncs ara també s'amplia la seva longitud fins a 33, encara que visites[31] la deixarà buida:

```

//I si omplim per error la posició 33?
visites[32] = 325;
console.log(visites[32]);
//Comprovem que la longitud s'ha ampliat a 33
console.log(visites.length);
//però la posició 31 quedarà undefined
console.log(visites[31]);

```

Per inicialitzar la taula de visites mensuals a la pàgina web a zero podem utilitzar l'estructura iterativa:

```
var visites = Array(30);
var i;
//Iterem per recórrer totes les posicions de l'Array
//i inicialitzem els camps
for (i = 0; i <30; i++){ //Recordeu: la primera posició és 0
  visites[i] = 0;
  console.log(visites[i]);
}

```

També podem utitlizar la propietat predefinida d'un **Array *length***. Una propietat és un atribut que s'aplica a un tipus de dades i, per tant, la podem consultar. Aquest és un concepte relacionat amb el paradigma d'orientació a objectes, que s'introdueix en etapes posteriors de l'aprenentatge de la programació.

```

//Iterem per recórrer totes les posicions de l'Array
//i inicialitzem els camps utilitzant la propietat
//que ens indica la seva longitud
for (i = 0; i <visites.length; i++){ //Recordeu: la primera posició és 0
  visites[i] = 0;
  console.log(visites[i]);
}

```

Aquesta propietat aplicada a un **Array** retorna la longitud d'un **Array**.

Taules de diverses dimensions

La correspondència de les t**aules de taules** de la notació algorísmica són els **Array multidimensionals**.

Exemple

Hem vist que en **JavaScript** un **Array** pot contenir qualsevol objecte; sí, fins i tot un altre **Array**.

Per emmagatzemar les visites rebudes a una web, durant un any en una matriu, doncs, podem fer:

```
//Definim la matriu anual amb 12 taules de mesos de 30 dies
var visitesAnuals = Array(12);
var i;
for (i = 0; i <12; i++){
  visitesAnuals[i] = Array(30); //Cada element de l'array de 12 mesos és
                                //un array de 30 dies
}

//Si volem accedir a les visites obtingudes el dia 3 del mes 4
//mes abril està a la posició 3 i dia 3 està a la posició 2
console.log(visitesAnuals[3][4]); //retornarà undefined perquè encara
                                  // no li hem assignat cap valor
//també podem fer aquí assignacions de valors directament:
//assignem a dia 3 d'abril un total de 360 visites
 visitesAnuals[3][2] = 360;
 console.log(visitesAnuals[3][2]); //ara tornarà 360

```

Podem inicialitzar els elements de la taula a 0 tot utilitzant l'estructura iterativa.

```
//Definim la matriu anual amb 12 taules de mesos de 30 dies
var visitesAnuals = Array(12);
var i, j;
for (i = 0; i <12; i++){
  visitesAnuals[i] = Array(30); //Cada element de l'array de 12 mesos és
                                //un array de 30 dies
}

//Podem inicialitzar tots els camps de la matriu a 0
for(i = 0; i<12; i++){
  for (j = 0; j<30; j++){
    visitesAnuals[i][j]=0;
    console.log(visitesAnuals[i][j]);
  }
}

```

Fixeu-vos que el comportament del tipus **Array** en JS permet representar les taules tal com les hem definit algorísmicament, però també altres possibilitats que de moment no utitlitzarem.

Podeu provar aquests codis a la web de PythonTutor.