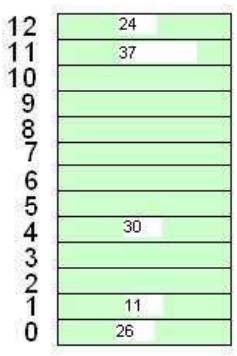
Algorithmes et structures de données : TD 10 Corrigé

Tables de hachage - Fonctions de hachage

Rappel :
`SetLength(tableau, n)` est de complexité O(n)
`SetLength(tableau, 1)` est de complexité O(1)
`New(element)` est de complexité O(1) quand `element` est d'un type de taille fixe

Exercice 10.1 *Fonction de hachage*

Considérer la fonction de hachage $h(c) = cmod13$ et une table de hachage avec $m = 13$ adresses.
1. Insérer les clés 26,37,24,30, et 11 dans la table de hachage ci-dessus en utilisant la résolution des collisions par adressage ouvert et sondage linéaire avec la fonction $h_i(c) = (h(c)+i)modm$.



2. Rajouter maintenant les clés 1,2,3,4,5,6,7,8,9,10. Quel problème rencontrez-vous ? Quelles solutions proposez-vous ?

La taille de la table de hachage n'est pas suffisante. Dans ce cas, il y a deux solutions :

1. On aggrandi la taille de la table de hachage. Dans ce cas là, tous les éléments doivent être insérés de nouveau. Il est courant de doubler la taille de la table de hachage.

2. On utilise la résolution des collision par chaînage externe.

Exercice 10.2 *Table de hachage*

```
program td_9;

{$APPTYPE CONSOLE}

uses
```


```
  SysUtils;

const m = 13;

type element = record
  cle       : string;
  date      : string;
end;

var newelement : element;
var cle        : string;
var HT : array[0..m-1] of element;

function hash(cle : string) : integer;
var j : integer;
var i : integer;
begin
  if length(cle) = 0 then
    result := 0
  else
  begin
    j := ord(cle[1]) mod m;
    for i := 2 to length(cle) do
      j := (j * 256 + ord(cle[i])) mod m;
    result := j;
  end;
end;

function findHT(cle : string) : integer;
var i : integer;
begin
  i := hash(cle);
  result := -1;
  while (result=-1) do
```

```
  begin
    if (length(HT[i].cle)=0) or (HT[i].cle = cle) then
      result := i;
    i := (i + 1) mod m;
  end;
end;

function exists(cle : string) : boolean;
var i : integer;
begin
     i := findHT(cle);
     if NOT(HT[i].cle='') then
       result:= true
     else                      // key is not in table
       result:= false;
end;

function lookup(cle : string) : element;
var i : integer;
begin
     i := findHT(cle);
     result:= HT[i];
end;

 procedure put(elem : element);
 var i : integer;
 begin
     i := findHT(elem.cle);
     if i>0 then
       HT[i] := elem
     else
     begin

         { COMMENTAIRE
         if the table is almost full
             rebuild the table larger (note 1)
             i := findHT(key)
       HT[i] := elem;
       FIN DU COMMENTAIRE}

       WriteLn('Enlarge table ');
     end;
 end;

procedure showtable;
var i : integer;
begin
```

```
  { à faire }

end;

begin
  newelement.cle := 'SAM';
  newelement.date := '20.03.84';
  put(newelement);
  newelement.cle := 'EMMA';
  newelement.date := '13.02.86';
  put(newelement);
  newelement.cle := 'LEO';
  newelement.date := '21.02.85';
  put(newelement);
  newelement.cle := 'AXEL';
  newelement.date := '28.03.82';
  put(newelement);

  if (exists('AXEL')) then
  begin
        newelement := lookup('AXEL');
        WriteLn('Birthdate',newelement.date);
  end
  else
        WriteLn('Cette entrée n existe pas');
  showtable;

  ReadLn;
end.
```

1. Ecrire la fonction `showtable` qui affiche la table de hachage avec la clé et la date d'anniversaire.

```
procedure showtable;
var i : integer;
begin
  for i:=0 to m-1 do
  begin
    Write(i);
    WriteLn(' : ',HT[i].cle, '  ', HT[i].date);
  end;
end;
```

2. Faites tourner le programme entier dans un tableau. Utiliser une nouvelle colonne pour chaque nouvelle variable locale.

i 1er `put`	i 1er `findHT`	result 1er `findHT`	i 1er `hash`	j 1er `hash`	result 1er `hash`
				5	
			2		
				6	
			3		
				1	
					1
	1				
		-1			
		1			
	2				
1					

i 2ème `put`	i 2ème `findHT`	result 2ème `findHT`	i 2ème `hash`	j 2ème `hash`	result 2ème `hash`
				4	
			2		
				9	
			3		
				2	
			4		
				5	
					5
	5				
		-1			
		5			
	6				
5					

i 3ème `put`	i 3ème `findHT`	result 3ème `findHT`	i 3ème `hash`	j 3ème `hash`	result 3ème `hash`
				11	
			2		
				12	
			3		
				5	
					5
	5				
		-1			
	6				
		6			
	7				
6					

i 4ème `put`	i 4ème `findHT`	result 4ème `findHT`	i 4ème `hash`	j 4ème `hash`	result 4ème `hash`
				0	
			2		
				10	
			3		
				3	
			4		
				12	
					12
	12				
		-1			
		12			
	0				
12					

i 1er `exists`	result 1er `exists`	i 5ème `findHT`	result 5ème `findHT`	i 5ème `hash`	j 5ème `hash`	result 5ème `hash`
					11	
				2		
					12	
				3		
					5	
						5
		5				
			-1			
		6				
			6			
		7				
6						
	true					

i 1er `lookup`	i 6ème `findHT`	result 6ème `findHT`	i 6ème `hash`	j 6ème `hash`	result 6ème `hash`
				11	
			2		
				12	
			3		
				5	
					5
	5				
		-1			
	6				
		6			
	7				
6					

La procédure showtable affichera :

```
0 :
1 : SAM  20.03.84
2 :
3 :
4 :
5 : EMMA  13.02.86
6 : LEO  21.02.85
7 :
8 :
9 :
10 :
11 :
12 : AXEL  28.03.82
```