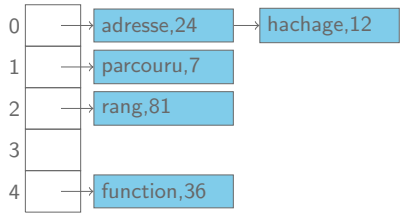
Schéma de principe

La table est un tableau de listes chaînées de couples.



k	v	$h(k)$
hachage	12	0
fonction	36	4
parcouru	7	1
rang	81	2
adresse	24	0

Recherche d'une clé

```
Require: t a hashtable
Require: k a key
  function LOOKUP(t, k)
      a ← h(k)
      p ← t[a]
      while p not empty && p.key ≠ k do
          p ← tail(p)
      end while
      if p not empty then
          return head(p).value
      else
          raise NotFound
      end if
  end function
```


Coût de la recherche infructueuse d'un élément

- l'accès à la bonne alvéole est en $\Theta(1)$
- la recherche de l'élément dans la liste va dépendre de la longueur de la liste :
  - $\Theta(1)$ pour le meilleur des cas (où aucune clé ayant même adresse n'a été insérée dans la table)
  - $\Theta(n)$ pour le pire des cas (où toutes les clés insérées ont même adresse, donc dans une seule liste, et la clé recherchée a même adresse)

en moyenne :

- si on suppose que la fonction de hachage est uniforme, les n clés auront été hachées équitablement dans les alvéoles
- les listes auront donc une longueur de l'ordre de $\frac{n}{M} = \tau$
- le temps moyen de la recherche est en $\Theta(1 + \tau)$ (le 1 est pour l'accès à l'alvéole)

Coût de la recherche fructueuse d'un élément

- l'accès à la bonne alvéole est en $\Theta(1)$
- la recherche de l'élément dans la liste va dépendre de la longueur de la liste :
  - $\Theta(1)$ pour le meilleur des cas (clé en première position de la liste)
  - $\Theta(n)$ pour le pire des cas (où toutes les clés insérées ont même adresse, donc dans une seule liste, et la clé recherchée est en fin de liste)

en moyenne :

- si on suppose que la fonction de hachage est uniforme, les n clés auront été hachées équitablement dans les alvéoles
- les listes auront donc une longueur de l'ordre de $\frac{n}{M} = \tau$
- le temps moyen de la recherche est aussi en $\Theta(1 + \tau)$

Coût de l'insertion et de la suppression

- l'insertion et la suppression se font comme dans une liste chaînée (le comportement de l'insertion dépend de la bibliothèque utilisée : on change la valeur, on lève une exception, on ne fait rien)
- insertion :
  - trouver l'alvéole : $\Theta(1)$
  - tester l'existence : $\Theta(\frac{n}{M})$ (en moyenne)
  - ajouter en tête de liste : $\Theta(1)$
- suppression :
  - trouver l'alvéole : $\Theta(1)$
  - trouver la position dans la liste : $\Theta(\frac{n}{M})$ (en moyenne)
  - supprimer : $\Theta(1)$ (on se souvient que la suppression d'un élément dans une liste, une fois l'élément trouvé, est en temps constant)

II Faisons le point

sur l'adressage par chaînage

- permet de stocker autant d'éléments qu'on veut
- la performance est liée à la fois à la fonction de hachage mais aussi à la taille de la table
- l'insertion est très efficace
- la suppression est réalisable facilement

`http://groups.engin.umd.umich.edu/CIS/course.des/cis350/hashing/WEB/HashApplet.htm`