Algo

- Numero: 7
- Prof: Alexandre Duret-Lutz
- Date: 27 Octobre 2017

Tri Lineaire

Ce sont des tris qui ne font pas de comparaison.

- Counting Sort
- Radix Sort
- Bucket Sort

Complexite amortie

Alloc dynamique de tableau

- Tableau dynamique:
  - Taille max, utilisee
  - Insertion
  - Suppression

Il y a un realloc dans `Insertion`,

```
insert(vector *v, int x)
{
  if (v->used == v->capacity)
  {
    v->capacity += 1;
    int *old = v->data;
    v->data = reallloc(v->data, v->capacity * sizeof(int));
    if (v->data == NULL)
    {
      free(old);
      abort();
    }
  }
  v->data[v->used++] = x;
}
```

Impact de la strategie de realloc sur le cout de `insert()`.

Strategie 1

`v->capacity += 1;`

- `insert()` coute $\Theta(1)$ si pas d'appel a realloc.
- `insert()` coute $[(n)$ si appel a realloc.

Strategie 2

`v->capacity += 4000`

- `insert()` coute $[(n)$ si appel a realloc.
- `insert()` coute $\Theta(1)$ pour les `3999` suivant.

Sur une longue sequence d'insert le coup moyen est $\frac{[(n)\times 1+\Theta(1)+3999}{4000} = [(n) + \Theta(1) = [(n)$

On appelle cela la **complexitee amortie** de insert.

Strategie 3

`v->capacity *= 2` (C'est un peu violent)

- `insert()` coute $[(n)$ lors d'une realloc.
- `insert()` coute $\Theta(1)$ lors des `n - 1` suivant.

En moyenne:

$\frac{1\times[(n)+(n-1)[(n)}{n} = [(1) + \Theta(1) = \Theta(1)$

Insert en temps constant amorti.

CountingSort

2_A	1_B	0_C	1_D	1_E	2_F	0_G	2_H	1_I

0_C	0_G	1_B	1_D	1_E	1_I	2_A	2_F	2_M

$\forall i < n, 0 \leq A[i] \leq k$

```
CountingSort(A, n, k)
{
  for i <- 0 to k
    C[i] <- 0
  for i <- 0 to n - 1
    C[A[i]] <- C[A[i]] + 1
  for i <- 1 to k
    c[i] <- C[i - 1]
  for i <- n - 1 down to 0
    C[A[i]] <- C[A[i]] - 1
    B[C[A[i]]] <- A[i]
}
```

RadixSort

Repeter countingSort sur unite, dizaine, centaine, ect… Dans une base choisie.

BucketSort

Tri d'un ensemble de valeurs dans $[0, 1[$

Pour `n` valeurs, on crees n "buchets" qui divisent $[0, 1[en `n` parties egales.

```
BucketSort(A, n)
  for i <- 0 to n - 1
    B[floor(A[i] * n)].insert(A[i])
  for i <- 0 to n - 1
    InsertSort(B[i])
  return Concat(B[0], B[1], B[2], ..., B[n - 1])
```

Si on a de la chance $\forall i, n_i = 1$ dans ce cas favorable BuchetSort est en

$$\mathcal{O}(n) + \sum_{i=0}^{n-1} \mathcal{O}(1) = \Theta(n) + \mathcal{O}(n) = \Theta(n)$$

Si on na pas de chance $\exists j$ tq $n_j = n$ et $n_i = 0$ ssi $i \neq j$

$$\Theta(n) + \sum_{i=0}^{n-1} \mathcal{O}(n_i^2) = \Theta(n) + \mathcal{O}(n_j^2)$$
$$\Theta(n) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$$

on a `n` valeurs (donc `n` bucket) choisie uniformement dans $[0, 1[$
la proba quune valeur tombe dans un sceaux est $\frac{1}{n}$

$n_i =$ nombre de valeur dans le sceaux `i`.

Pour $(n_i = x) = \binom{n}{x} \left(\frac{1}{n}\right)^x \left(1 - \frac{1}{b}\right)^{n-x}$

Loi binomiale avec $p = \frac{1}{n}$

$$E[n_i] = n \times \frac{1}{n} = 1$$
$$Var[n_i] = n \times \frac{1}{n}(1 - \frac{1}{n}) = 1 - \frac{1}{n}$$
$$E[n_i^2] = E[n_i]^2 + Var[n_i] = 1 + 1 - \frac{1}{n}$$
$$E[T(n)] = E[\Theta(n) + \sum \mathcal{O}(n_i^2)$$
$$= \Theta(n) + \mathcal{O}(\sum_{i=j}^{n-1} E[n_i^2])$$

// A compléter