Représentation des nombres, tris, et complexité

Algorithmique et structures de données, 2024-2025

P. Albuquerque (B410) et O. Malaspinas (A401), ISC, HEPIA

2024-11-11

En partie inspirés des supports de cours de P. Albuquerque

La représentation des nombres

Représentation des nombres (1/2)

- Le nombre 247.

Nombres décimaux: Les nombres en base 10

10^2	10^1	10^0
2	4	7

$$247 = 2 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0.$$

Représentation des nombres (2/2)

- Le nombre 11110111.

Nombres binaires: Les nombres en base 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	0	1	1	1

$$1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Représentation des nombres (2/2)

- Le nombre 11110111.

Nombres binaires: Les nombres en base 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	0	1	1	1

$$1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

$$= 247.$$

Conversion de décimal à binaire (1/2)

Convertir 11 en binaire?

Conversion de décimal à binaire (1/2)

Convertir 11 en binaire?

- On décompose en puissances de 2 en partant de la plus grande possible

```
11 / 8 = 1, 11 % 8 = 3
3  / 4 = 0,  3 % 4 = 3
3  / 2 = 1,  3 % 2 = 1
1  / 1 = 1,  1 % 1 = 0
```

- On a donc

$$1011 \Rightarrow 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11.$$

Conversion de décimal à binaire (2/2)

Convertir un nombre arbitraire en binaire: 247?

- Par groupe établir un algorithme.

Conversion de décimal à binaire (2/2)

Convertir un nombre arbitraire en binaire: 247?

- Par groupe établir un algorithme.

Algorithme

1. Initialisation

```
num = 247
N = 0

tant que (2^(N+1) < num) {
    N += 1
}
```

Conversion de décimal à binaire (2/2)

Convertir un nombre arbitraire en binaire: 247?

- Par groupe établir un algorithme.

Algorithme

1. Initialisation

```
num = 247
N = 0

tant que (2^(N+1) < num) {
    N += 1
}
```

2. Boucle

```
tant que (N >= 0) {
    bit = num / 2^N
    num = num % 2^N
    N -= 1
}
```

Les additions en binaire

Que donne l'addition 1101 avec 0110?

- L'addition est la même que dans le système décimal

```
   1101          8+4+0+1 = 13
+  0110    +     0+4+2+0 =  6
-------    -----------------
  10011       16+0+0+2+1 = 19
```

- Les entiers sur un ordinateur ont une précision **fixée** (ici 4 bits).
- Que se passe-t-il donc ici?

Les additions en binaire

Que donne l'addition `1101` avec `0110`?

- L'addition est la même que dans le système décimal

```
   1101          8+4+0+1 = 13
+  0110     +    0+4+2+0 =  6
-------     ----------------
  10011       16+0+0+2+1 = 19
```

- Les entiers sur un ordinateur ont une précision **fixée** (ici 4 bits).
- Que se passe-t-il donc ici?

Dépassement de capacité: le nombre est "tronqué"

- `10011 (19) -> 0011 (3)`.
- On fait "le tour"."

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

$$(0000)_2 = 0 + 0 + 0 + 0 = 0$$

- Quel est l'entier non-signé min/max représentable avec N bit?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

$$(0000)_2 = 0 + 0 + 0 + 0 = 0$$

- Quel est l'entier non-signé min/max représentable avec N bit?

$$0 \text{ et } 2^N - 1.$$

- Donc `uint32_t`? maximal est?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

$$(0000)_2 = 0 + 0 + 0 + 0 = 0$$

- Quel est l'entier non-signé min/max représentable avec N bit?

$$0 \text{ et } 2^N - 1.$$

- Donc `uint32_t`? maximal est?

$$2^{32} - 1 = 4'294'967'295$$

Les multiplications en binaire (1/2)

Que donne la multiplication de 1101 avec 0110?

- La multiplication est la même que dans le système décimal

```
      1101                  13
*     0110      *            6
---------      --------------
      0000                  78
     11010
    110100
+ 0000000
---------      --------------
   1001110      64+0+0+8+4+2+0
```

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

- Décalage de un bit vers la gauche!

```
      0110
*     0010
---------
      0000
+    01100
---------
     01100
```

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

- Décalage de un bit vers la gauche!

```
      0110
*     0010
---------
      0000
+    01100
---------
     01100
```

Que fait la multiplication par 2^N?

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

- Décalage de un bit vers la gauche!

```
      0110
*     0010
---------
      0000
+    01100
---------
     01100
```

Que fait la multiplication par 2^N?

- Décalage de N bits vers la gauche!

Entiers signés (1/2)

Pas de nombres négatifs encore...

Comment faire?

Entiers signés (1/2)

Pas de nombres négatifs encore...

Comment faire?

Solution naïve:

- On ajoute un bit de signe (le bit de poids fort):

```
00000010: +2
10000010: -2
```

Problèmes?

Entiers signés (1/2)

Pas de nombres négatifs encore...

Comment faire?

Solution naïve:

- On ajoute un bit de signe (le bit de poids fort):

```
00000010: +2
10000010: -2
```

Problèmes?

- Il y a deux zéros (pas trop grave): 10000000 et 00000000
- Les additions différentes que pour les non-signés (très grave)

```
  00000010              2
+ 10000100            + -4
----------            ----
  10000110 = -6   !=    -2
```

Entiers signés (2/2)

Beaucoup mieux

- Complément à un:
  - on inverse tous les bits: `1001 => 0110`.

Encore un peu mieux

- Complément à deux:
  - on inverse tous les bits,
  - on ajoute 1 (on ignore les dépassements).

Entiers signés (2/2)

Beaucoup mieux

- Complément à un:
  - on inverse tous les bits: `1001 => 0110`.

Encore un peu mieux

- Complément à deux:
  - on inverse tous les bits,
  - on ajoute 1 (on ignore les dépassements).
- Comment écrit-on -4 en 8 bits?

Entiers signés (2/2)

Beaucoup mieux

- Complément à un:
  - on inverse tous les bits: `1001 => 0110`.

Encore un peu mieux

- Complément à deux:
  - on inverse tous les bits,
  - on ajoute 1 (on ignore les dépassements).
- Comment écrit-on −4 en 8 bits?

```
 4 =  00000100

        --------
-4 =>   00000100


        11111011
      + 00000001
      ----------
        11111100
```

Le complément à 2 (1/2)

Questions:

- Comment on écrit +0 et −0?
- Comment calcule-t-on 2 + (−4)?
- Quel est le complément à 2 de 1000 0000?

Le complément à 2 (1/2)

Questions:

- Comment on écrit +0 et -0?
- Comment calcule-t-on 2 + (-4)?
- Quel est le complément à 2 de 1000 0000?

Réponses

- Comment on écrit +0 et -0?

```
+0 = 00000000
-0 = 11111111 + 00000001 = 100000000 => 00000000
```

- Comment calcule-t-on 2 + (-4)?

```
  00000010            2
+ 11111100        +  -4
----------        -----
  11111110           -2
```

- En effet

```
11111110 => 00000001 + 00000001 = 00000010 = 2.
```

Le complément à 2 (2/2)

Quels sont les entiers représentables en 8 bits?

Le complément à 2 (2/2)

Quels sont les entiers représentables en 8 bits?

```
01111111 =>  127
10000000 => -128 // par définition
```

Quels sont les entiers représentables sur N bits?

Le complément à 2 (2/2)

Quels sont les entiers représentables en 8 bits?

```
01111111 =>  127
10000000 => -128 // par définition
```

Quels sont les entiers représentables sur N bits?

$$-2^{N-1}...2^{N-1}-1.$$

Remarque: dépassement de capacité en `C`

- Comportement indéfini!

Les types énumérés

Types énumérés (1/2)

- Un **type énuméré**: ensemble de *variantes* (valeurs constantes).
- En C les variantes sont des entiers numérotés à partir de 0.

```
enum days {
    monday, tuesday, wednesday,
    thursday, friday, saturday, sunday
};
```

- On peut aussi donner des valeurs "custom"

```
enum days {
    monday = 2, tuesday = 8, wednesday = -2,
    thursday = 1, friday = 3, saturday = 12, sunday = 9
};
```

- S'utilise comme un type standard et un entier

```
enum days d = monday;
(d + 2) == monday + monday; // true
```

Types énumérés (2/2)

- Très utile dans les `switch ... case`

```
enum days d = monday;
switch (d) {
    case monday:
        // trucs
        break;
    case tuesday:
        printf("0 ou 1\n");
        break;
}
```

- Le compilateur vous prévient qu'il en manque!

Utilisation des types énumérés

Réusiner votre couverture de la reine avec des `enum`

A faire à la maison comme exercice!

Tri rapide ou quicksort

Tri rapide ou quicksort (1/8)

Idée: algorithme `diviser pour régner` (`divide-and-conquer`)

- Diviser: découper un problème en sous problèmes;
- Régner: résoudre les sous-problèmes (souvent récursivement);
- Combiner: à partir des sous problèmes résolu, calculer la solution.

Le pivot

- Trouver le **pivot**, un élément qui divise le tableau en 2, tels que:
  1. Éléments à gauche sont **plus petits** que le pivot.
  2. Élements à droite sont **plus grands** que le pivot.

Tri rapide ou quicksort (2/8)

Algorithme `quicksort(tableau)`

1. Choisir le pivot et l'amener à sa place:
   - Les éléments à gauche sont plus petits que le pivot.
   - Les éléments à droite sont plus grand que le pivot.
2. `quicksort(tableau_gauche)` en omettant le pivot.
3. `quicksort(tableau_droite)` en omettant le pivot.
4. S'il y a moins de deux éléments dans le tableau, le tableau est trié.

Tri rapide ou quicksort (2/8)

Algorithme `quicksort(tableau)`

1. Choisir le pivot et l'amener à sa place:
   - Les éléments à gauche sont plus petits que le pivot.
   - Les éléments à droite sont plus grand que le pivot.
2. `quicksort(tableau_gauche)` en omettant le pivot.
3. `quicksort(tableau_droite)` en omettant le pivot.
4. S'il y a moins de deux éléments dans le tableau, le tableau est trié.

Compris?

Tri rapide ou quicksort (2/8)

Algorithme `quicksort(tableau)`

1. Choisir le pivot et l'amener à sa place:
   - Les éléments à gauche sont plus petits que le pivot.
   - Les éléments à droite sont plus grand que le pivot.
2. `quicksort(tableau_gauche)` en omettant le pivot.
3. `quicksort(tableau_droite)` en omettant le pivot.
4. S'il y a moins de deux éléments dans le tableau, le tableau est trié.

Compris?

Non c'est normal, faisons un exemple.

Tri rapide ou quicksort (3/8)

Deux variables sont primordiales:

```
entier ind_min, ind_max; // les indices min/max des tableaux à trier
```

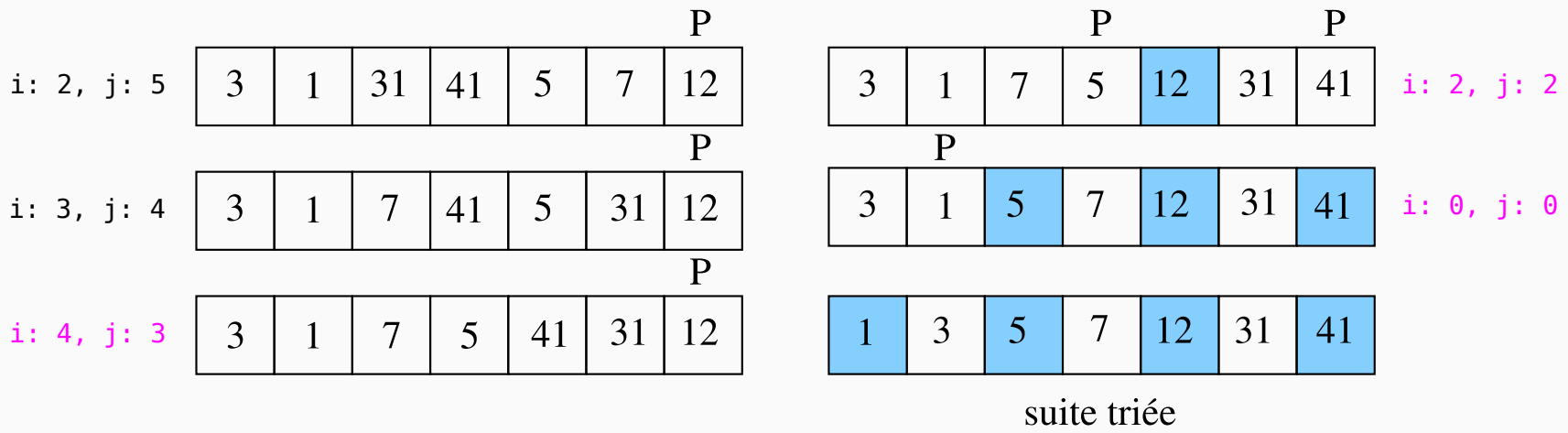

Figure 1: Un exemple de quicksort.

Tri rapide ou quicksort (4/8)

Deux variables sont primordiales:

```
entier ind_min, ind_max; // les indices min/max des tableaux à trier
```

Pseudocode: quicksort

```
rien quicksort(entier tableau[], entier ind_min, entier ind_max)
    si (longueur(tab) > 1)
        ind_pivot = partition(tableau, ind_min, ind_max)
        si (longueur(tableau[ind_min:ind_pivot-1]) != 0)
            quicksort(tableau, ind_min, pivot_ind - 1)
        si (longueur(tableau[ind_pivot+1:ind_max]) != 0)
            quicksort(tableau, ind_pivot + 1, ind_max)
```

Tri rapide ou quicksort (5/8)

Pseudocode: partition

```
entier partition(entier tableau[], entier ind_min, entier ind_max)
    pivot = tableau[ind_max] // choix arbitraire
    i = ind_min
    j = ind_max-1
    tant que i < j:
        en remontant i trouver le premier élément > pivot
        en descendant j trouver le premier élément < pivot
        échanger(tableau[i], tableau[j])
        // les plus grands à droite
        // mettre les plus petits à gauche

    // on met le pivot "au milieu"
    échanger(tableau[i], tableau[ind_max])
    retourne i // on retourne l'indice pivot
```

Tri rapide ou quicksort (6/8)

Exercice: implémenter les fonctions `quicksort` et `partition`

Tri rapide ou quicksort (6/8)

Exercice: implémenter les fonctions `quicksort` et `partition`

```
void quicksort(int size, int array[size], int first,
               int last)
{
    if (first < last) {
        int midpoint = partition(size, array, first, last);
        if (first < midpoint - 1) {
            quicksort(size, array, first, midpoint - 1);
        }
        if (midpoint + 1 < last) {
            quicksort(size, array, midpoint + 1, last);
        }
    }
}
```

Tri rapide ou quicksort (7/8)

Exercice: implémenter les fonctions `quicksort` et `partition`

```
int partition(int size, int array[size], int first, int last) {
    int pivot = array[last];
    int i = first - 1, j = last;
    do {
        do {
            i += 1;
        } while (array[i] < pivot && i < j);
        do {
            j -= 1;
        } while (array[j] > pivot && i < j);
        if (j > i) {
            swap(&array[i], &array[j]);
        }
    } while (j > i);
    swap(&array[i], &array[last]);
    return i;
}
```

L'algorithme à la main

Exercice *sur papier*

- Trier par tri rapide le tableau `[5, -2, 1, 3, 10, 15, 7, 4]`