11X001 – Introduction à la programmation des algorithmes

2.5 Exemples

François Fleuret

https://fleuret.org/francois

12 Octobre, 2020

Le contenu de ce document a été en grande partie repris du cours de Jean-Luc Falcone.

UNIVERSITÉ DE GENÈVE

FACULTY OF SCIENCE

Gestion de données

Base de données musicales

```
struct Album {
  let name: String
  let artist: String
  let year: UInt
  let tracks: [Track]
}

struct Track {
  let name: String
  let duration: UInt
}
```

- La durée est indiquée en secondes
- L'année est indiquée sur 4 chiffres
- La propriété `tracks` ne peut être vide

Base de données musicales

```
func seconds(min: UInt, sec: UInt) -> UInt {
  return min * 60 + sec
}

let albums = [
  Album(
    name: "The Dark Side of the Moon",
    artist: "Pink Floyd",
    year: 1973,
    tracks: [
      Track(name: "Speak to Me", duration: seconds(min: 1, sec: 13)),
      Track(name: "Breathe", duration: seconds(min: 2, sec: 43)),
      Track(name: "On the Run", duration: seconds(min: 3, sec: 36)),
      Track(name: "Time", duration: seconds(min: 6, sec: 53)),
      Track(name: "The Great Gig in the Sky", duration: seconds(min: 4, sec: 36)),
      Track(name: "Money", duration: seconds(min: 6, sec: 23)),
      Track(name: "Us and Them", duration: seconds(min: 7, sec: 49)),
      Track(name: "Any Colour You Like", duration: seconds(min: 3, sec: 26)),
      Track(name: "Brain Damage", duration: seconds(min: 3, sec: 49)),
      Track(name: "Eclipse", duration: seconds(min: 2, sec: 3)),
    ]
  ),
```

Quelle est la durée d'un album ?

```
struct Album {
  let name: String
  let artist: String
  let year: UInt
  let tracks: [Track]
}

struct Track {
  let name: String
  let duration: UInt
}
```

Quelle est la durée d'un album ?

```
struct Album {
  let name: String
  let artist: String
  let year: UInt
  let tracks: [Track]
}

struct Track {
  let name: String
  let duration: UInt
}

func durationOf( album: Album ) -> UInt {
  return album.tracks
    .reduce( 0, { (st, t) in st + t.duration } )
}
```

Quelle est la durée d'un album ?

```
func timeAtom(s: UInt, name: String, range: UInt, unit: UInt) -> String {
  let t = (s / unit) % range
  if t > 0 {
    return String(t) + name
  } else {
    return ""
  }
}

func readableTime(s: UInt) -> String {
  // Should add 'days'?
  return
    timeAtom(s: s, name: "h", range: 24, unit: 3600) +
    timeAtom(s: s, name: "min", range: 60, unit: 60) +
    timeAtom(s: s, name: "s", range: 60, unit: 1)
}
```

Quelle est la durée d'un album ?

On peut utiliser `map` pour des seuls effets de bords en ignorant la valeur de retour avec `_ =`.

```
_ = albums.map(
  {
    (a: Album) in
    print("\"" + a.name + "\"",
          "by", a.artist,
          "(" + readableTime(s: durationOf( album: a )) + ")")
  }
)
```

affiche

```
"The Dark Side of the Moon" by Pink Floyd (42min31s)
"Thriller" by Michael Jackson (42min16s)
"The Slim Shady LP" by Eminem (59min39s)
```

Extraire toutes les pistes d'une collections d'albums ?

```
func allTracksIn( collection: [Album] ) -> [Track]
```

Marche à suivre

1. On extrait les pistes des albums avec un `map`,
2. on concatène les tableaux de tableaux avec un `reduce`.

Marche à suivre

1. On extrait les pistes des albums avec un `map`,
2. on concatène les tableaux de tableaux avec un `reduce`.

```
func allTracksIn( collection: [Album] ) -> [Track] {
  return collection
    .map( { album in album.tracks } )
    .reduce( [], { (a, t) in a + t } )
}
```

Extraire toutes les pistes d'une collection d'albums pouvant passer à la radio

On considère que les chaînes de radio ne passe que des pistes en de plus de 1min30 et moins de 3min30.

```
func radioTracksIn( collection: [Album] )->[Track]
```

Marche à suivre

1. Extraire toutes les pistes (`allTracks`)
2. Garder uniquement les morceaux de la bonne longueur (`filter`)

Marche à suivre

1. Extraire toutes les pistes (`allTracks`)
2. Garder uniquement les morceaux de la bonne longueur (`filter`)

```
func radioTracksIn( collection: [Album] ) -> [Track] {
  return allTracksIn( collection:collection )
    .filter( { track in
               track.duration >= 90 && track.duration <= 210
             } )
}
```

Extraire les noms des pistes d'un artiste à partir d'une collection d'albums

```
func allTracksNameOf(artist: String,
                     fromCollection: [Album] ) -> [String]
```

Marche à suivre

1. Filtrer tous les albums de l'artiste (`filter`),
2. récupérer toutes les pistes (`allTracksIn`),
3. remplacer chaque piste par son mom (`map`).

Marche à suivre

1. Filtrer tous les albums de l'artiste (`filter`),
2. récupérer toutes les pistes (`allTracksIn`),
3. remplacer chaque piste par son mom (`map`).

```
func allTracksNameOf( artist: String,
                      fromCollection: [Album] ) -> [String] {
  let artistAlbums = fromCollection
    .filter( { album in album.artist == artist } )
  return allTracksIn( collection: artistAlbums )
    .map( { t in t.name } )
}
```

Quel est le plus ancien album d'une collection ?

```
func oldestAlbumIn( collection: [Album] )->Album
```

Marche à suivre

Visiter chaque album en gardant le plus ancien (`reduce`)

Marche à suivre

Visiter chaque album en gardant le plus ancien (`reduce`)

```
func oldestAlbumIn( collection: [Album] ) -> Album {
  return collection.reduce(
    collection[0],
    { (album1, album2) in
      if album1.year < album2.year {
        return album1
      } else {
        return album2
      }
    } )
}
```

Polynômes, première tentative

On aimerait implémenter une représentation de polynômes à une variable réels.

Par exemple:

$$P(X) = 3X^4 - 2X^2 + 9X - 2$$

Opérations possibles

1. Additions de polynômes.
2. Multiplication par un facteur.
3. Évaluation en un point.
4. Calcul de dérivée.

Représentation

Par définition, un polynôme est une somme de monômes.

On pourrait donc utiliser la représentation suivante:

```
struct Monomial {
    let factor: Double
    let exponent: UInt
}

struct Polynomial {
    let terms: [Monomial]
}
```

On peut créer le polynôme:

$$P(X) = 3X^4 - 2X^2 + 9X - 2$$

```
let P = Polynomial(
  terms: [
    Monomial( factor: 3, power: 4 ),
    Monomial( factor: -2, power: 2 ),
    Monomial( factor: 9, power: 1 ),
    Monomial( factor: -2 , power: 0 )
  ]
)
```

Pour alléger la syntaxe on peut introduire la fonction suivante:

```
func mon( _ fac: Double, _ exp: UInt )->Monomial {
  return Monomial( factor: fac, power: exp )
}

let Q = Polynomial(
  terms: [ mon( 3, 4 ), mon( -2, 2 ), mon( 9, 1 ), mon( -2, 0 ) ]
)
```

Addition

On peut additionner un monôme à un polynôme en l'ajoutant dans la liste, et un polynôme en concaténant sa liste de monômes:

```
func add( _ p: Polynomial, _ m: Monomial )->Polynomial {
  return Polynomial(
    terms: p.terms + [m]
  )
}

func add( _ p: Polynomial, _ q: Polynomial )->Polynomial {
  return Polynomial(
    terms: p.terms + q.terms
  )
}
```

Addition

```
let P1 = Polynomial(
  terms: [ mon( 3, 1 ), mon( -2, 0 ) ]
)

let P2 = Polynomial(
  terms: [ mon( 3, 2 ), mon( 3, 0 ) ]
)

print(add(P1, P2))
```

Addition

```
let P1 = Polynomial(
  terms: [ mon( 3, 1 ), mon( -2, 0 ) ]
)

let P2 = Polynomial(
  terms: [ mon( 3, 2 ), mon( 3, 0 ) ]
)

print(add(P1, P2))
```

```
Polynomial(terms: [
  Monomial(factor: 3.0, exponent: 1),
  Monomial(factor: -2.0, exponent: 0),
  Monomial(factor: 3.0, exponent: 2),
  Monomial(factor: 3.0, exponent: 0)
])
```

Multiplication par une constante

Multiplier un polynôme par une constante revient à multiplier chaque monôme par une constante:

$$a \sum_{d=0}^{D} \alpha_d x^d = \sum_{d=0}^{D} (a\alpha_d) x^d$$

```
func multCst( _ m: Monomial, _ a: Double )->Monomial {
  return mon( m.factor * a, m.power )
}

func multCst( _ p: Polynomial, _ a: Double )->Polynomial {
  return Polynomial(
    terms: p.terms.map( { m in multCst( m, a ) } )
  )
}
```

Soustraction de polynômes

On peut utiliser le fait que:

$$P - Q = P + (Q * (-1))$$

pour implémenter la soustraction.

```
func sub( _ p: Polynomial, _ q: Polynomial )->Polynomial {
  return add( p, multCst( q, -1 ) )
}
```

Évaluation en un point

On peut évaluer un polynôme en un point (par exemple pour tracer une fonction):

$$\begin{aligned} P(X) &= 3X^4 - 2X^2 + 9X - 2 \\ P(1) &= 3 - 2 + 9 - 2 = 8 \\ P(2) &= 48 - 8 + 18 - 2 = 56 \\ P(0.5) &= 2.1875 \end{aligned}$$

Évaluation en un point

Il faut évaluer chaque monôme (`map`), puis sommer les résultats (`reduce`):

```
func eval( _ m: Monomial, x: Double )->Double {
  return pow( x, Double( m.power ) ) * m.factor
}

func eval( _ p: Polynomial, x: Double )->Double {
  return p.terms
    .map( { m in pow( x, Double( m.power ) ) * m.factor } )
    .reduce( 0, { ( sum, y ) in sum+y } )
}

print(eval(P1, x: 1.5))
```

Dérivée d'un polynôme

Dériver un polynôme consiste à dériver et sommer chaque monôme:

$$\left(\sum_{d=0}^{D} \alpha_d x^d\right)' = \sum_{d=0}^{D} \left(\alpha_d x^d\right)'$$

Dérivée d'un polynôme (cont.)

```
func derivative( _ m: Monomial )->Monomial {
  if m.power == 0 {
    return mon( 0, 0 )
  } else {
    return Monomial(
      factor: m.factor * Double( m.power ),
      power: m.power - 1
    )
  }
}

func derivative( _ p: Polynomial )->Polynomial {
  return Polynomial(
    terms: p.terms
      .map( derivative )
      .filter( { m in m.factor != 0.0 } )
  )
}
```

Dérivée d'un polynôme (cont.)

Avec

$$R(X) = X^4 - X$$

```
let R = Polynomial(
  terms: [ mon( 1, 4 ), mon( -1, 1 ) ]
)

print(derivative(R))
```

affiche

```
Polynomial(terms: [ Monomial(factor: 4.0, power: 3),
                    Monomial(factor: -1.0, power: 0) ])
```

Problème

Les polynômes obtenus avec cette représentation ne sont pas réduits:

```
let p = Polynomial(
  terms: [mon(2,1)]  // 2x
)

add(p, p) != multCst(p, 2)
```

Il faut soit:

- écrire une fonction de réduction,
- soit trouver une meilleure représentation

Polynômes, deuxième tentative

Polynômes, représentation alternative

On peut représenter un polynôme par une liste de coefficients. Par exemple le polynôme:

$$\begin{aligned} P(x) &= 3x^4 - 2x^2 + 9x - 2 \\ &= (-2)x^0 + 9x^1 + (-2)x^2 + 0x^3 + 3x^4 \end{aligned}$$

peut être écrit comme le tableau: `[-2, 9, -2, 0, 3]`.

Chaque composante de ce tableau représente un monôme, dont le degré est l'indice.

Polynômes, exemples

- $x^2 \rightarrow$ `[0, 0, 1]`
- $19 \rightarrow$ `[19]`
- $x^2 - x + 1 \rightarrow$ `[1, -1, 1]`
- $x^{10} \rightarrow$ `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]`

Remarque

Les zéros à droite peuvent être ignorés:

- $x^2 \rightarrow$ `[0, 0, 1] = [0, 0, 1, 0, 0]`

Polynômes, représentation

```
struct Polynomial {
  let coeffs: [Double]
}

let p = Polynomial(coeffs: [ 1, 0, 1 ])
let q = Polynomial(coeffs: [ -1, 1, 1, 3, 4 ])
print(p, q)

Polynomial(coeffs: [1.0, 0.0, 1.0])
Polynomial(coeffs: [-1.0, 1.0, 1.0, 3.0, 4.0])
```

On peut programmer l'évaluation d'un polynôme avec une réduction, où la valeur que l'on propage est une paire composée de la somme partielle des monômes et de l'exponentiation de x.

Avec

$$y_0 = (0, 1)$$
$$f : ((s, z), a) \mapsto (s + a \cdot z, z \cdot x)$$

$$y_1 = f(y_0, a_0) = f((0, 1), a_0) = (a_0, x)$$
$$y_2 = f(y_1, a_1) = f((a_0, x), a_1) = (a_0 + a_1 x, x^2)$$
$$y_3 = f(y_2, a_2) = f((a_0 + a_1 x|, x^2), a_2) = (a_0 + a_1 x + a_2 x^2, x^3)$$
$$y_4 = f(y_3, a_3) = \ldots$$

Polynômes, évaluation (cont.)

Dans `reduce`, la valeur `u.0` est la somme partielle et `u.1` l'exponentiation de `x`.

```
func eval( _ p: Polynomial, _ x: Double )->Double {
  return p.coeffs.reduce(
    (0.0, 1.0),
    { ( u, c ) in (u.0 + c * u.1, u.1 * x) }
  ).0
}
```

Polynômes, évaluation (cont.)

Dans `reduce`, la valeur `u.0` est la somme partielle et `u.1` l'exponentiation de `x`.

```
func eval( _ p: Polynomial, _ x: Double )->Double {
  return p.coeffs.reduce(
    (0.0, 1.0),
    { ( u, c ) in (u.0 + c * u.1, u.1 * x) }
  ).0
}

let s = Polynomial(coeffs: [ 5, 4, 3, 2, 1])
print(s, eval(s, 2.0))

Polynomial(coeffs: [5.0, 4.0, 3.0, 2.0, 1.0])
57.0
```

Polynômes, fonction générique de création

```
func build( count: Int, with: ( Int )->Double ) -> Polynomial {
  func doBuild( _ i: Int, _ result: [Double], _ work: [Double] ) -> [Double] {
    if i >= count {
      return result
    } else {
      let x = with( i )
      let w = work + [x]
      if x == 0.0 {
        return doBuild( i + 1, result, w )
      } else {
        return doBuild( i + 1, w, w)
      }
    }
  }

  return Polynomial(coeffs: doBuild( 0, [], [] ))
}
```

Polynômes, fonction générique de création (cont.)

```
let a = build( count: 3, with: { d in Double(d) } )
print(a)

let b = build( count: 4, with: { d in 1 } )
print(b)

let c = build( count: 3, with: { d in a.coeffs[d] + b.coeffs[d] } )
print(c)

Polynomial(coeffs: [0.0, 1.0, 2.0])
Polynomial(coeffs: [1.0, 1.0, 1.0, 1.0])
Polynomial(coeffs: [1.0, 2.0, 3.0])
```

Polynômes, addition

```
func add( _ p: Polynomial, _ q: Polynomial )->Polynomial {
  func coeff( _ pol: Polynomial, _ i: Int) -> Double {
    if(i < pol.coeffs.count) {
      return pol.coeffs[i]
    } else {
      return 0.0
    }
  }

  return build(
    count: max(p.coeffs.count, q.coeffs.count),
    with: { d in coeff(p, d) + coeff(q, d) }
  )
}
```

Polynômes, addition (cont.)

```
print(p, q, add(p, q))

let p1 = Polynomial(coeffs: [  1, -1, -1,  3, -4 ])
let p2 = Polynomial(coeffs: [ -1,  1,  1, -3,  4 ])

print(p1, p2, add(p1, p2))

Polynomial(coeffs: [1.0, 0.0, 1.0])
Polynomial(coeffs: [-1.0, 1.0, 1.0, 3.0, 4.0])
Polynomial(coeffs: [0.0, 1.0, 2.0, 3.0, 4.0])

Polynomial(coeffs: [1.0, -1.0, -1.0, 3.0, -4.0])
Polynomial(coeffs: [-1.0, 1.0, 1.0, -3.0, 4.0])
Polynomial(coeffs: [])
```

Polynômes, soustraction

```
func cstMult( _ p: Polynomial, _ a: Double )->Polynomial {
  return build(
    count: p.coeffs.count,
    with: { d in p.coeffs[d] * a }
  )
}

func sub( _ p: Polynomial, _ q: Polynomial )->Polynomial {
  return add( p, cstMult( q, -1.0 ) )
}


print(p, q, sub(p, q))

Polynomial(coeffs: [1.0, 0.0, 1.0])
Polynomial(coeffs: [-1.0, 1.0, 1.0, 3.0, 4.0])
Polynomial(coeffs: [2.0, -1.0, 0.0, -3.0, -4.0])
```

On a

$$\left(\sum_d a_d x^d\right)\left(\sum_d b_d x^d\right) = \sum_d \left(\sum_{i=0}^{d} a_i b_{d-i}\right) x^d$$

```
func mult( _ p: Polynomial, _ q: Polynomial ) -> Polynomial {
  func coeff(_ i: Int, _ d: Int, _ acc: Double) -> Double {
    if i <= d {
      let j = d - i
      if i < p.coeffs.count && j < q.coeffs.count {
        return coeff(i + 1, d, acc + p.coeffs[i] * q.coeffs[j])
      } else {
        return coeff(i + 1, d, acc)
      }
    } else {
      return acc
    }
  }

  return build(
    count: p.coeffs.count + q.coeffs.count - 1,
    with: { d in coeff(0, d, 0) }
  )
}
```

Polynômes, multiplication (cont.)

```
let q1 = Polynomial(coeffs: [ 1.0, 1.0 ])
let q2 = Polynomial(coeffs: [ 0.0, 2.0, 1.0 ])
print(q1, q2, mult(q1, q2))

Polynomial(coeffs: [1.0, 1.0])
Polynomial(coeffs: [0.0, 2.0, 1.0])
Polynomial(coeffs: [0.0, 2.0, 3.0, 1.0])
```

Polynômes, dérivation

```
func derivative( _ p: Polynomial )->Polynomial {
  return build(
    count: max(p.coeffs.count - 1, 0),
    with: { d in p.coeffs[d + 1] * Double(d + 1) }
  )
}
```

Polynômes, dérivation (cont.)

```
print(s, derivative(s))
```

```
Polynomial(coeffs: [5.0, 4.0, 3.0, 2.0, 1.0])
Polynomial(coeffs: [4.0, 6.0, 6.0, 4.0])
```

Conclusion

- Cette représentation permet de toujours travailler avec des polynômes réduits.
- On pourrait "facilement" ajouter les opérations manquantes. Par exemple:
  - Calcul de la dérivée seconde
  - Produits de polynômes
  - Intégrales définies ou indéfinies
  - Recherche de zéros

FIN