Introduction

IFT436 – Algorithmes et structures de données

Automne 2020

UNIVERSITÉ DE SHERBROOKE

Aujourd'hui

- Plan de cours
- Introduction à l'algorithmique

Plan de cours

Enseignant: Michael Blondin

@ michael.blondin@usherbrooke.ca

info.usherbrooke.ca/mblondin

bureau D4-1024-1 au 1er étage

Correcteur: Etienne D. Massé

@ etienne.dubois.masse@usherbrooke.ca

Horaire

Cours	Mercredi	10h30 – 12h30
	Jeudi	10h30 – 11h30
Exercices	Jeudi	11h30 – 12h30

Dans le cheminement

Préalable:

- IFT339 – Struct. de données

Atout:

- MAT115 – Logique et math. discrètes

Préalable à:

- IFT501 – Recherche d'information et forage de données
- IFT603 – Techniques d'apprentissage
- IFT607 – Traitement automatique des langues naturelles
- IFT608 – Planification en intelligence artificielle
- IFT609 – Informatique cognitive
- IFT615 – Intelligence artificielle
- IGL502 – Techniques de vérification et de validation

Utile pour:

- IFT503 – Théorie du calcul
- IFT580 – Compilation et interprétation des langages
- La majorité des domaines de l'informatique

Grandes questions du cours

- Comment résoudre différents types de problèmes?
- Comment résoudre ces problèmes efficacement?
- Comment analyser l'efficacité d'un algorithme?
- Comment s'assurer qu'un algorithme fonctionne?

Pourquoi étudier l'algorithmique?

Notamment, pour savoir:

- Raisonner indépendamment des technologies
- Identifier des problèmes génériques dans les problèmes concrets
- Repérer ce qui peut/doit être optimisé
- Adapter des algorithmes existants (sans les briser)
- Mettre au point ses propres algorithmes

Contenu

Fondements

1. Notions mathématiques
2. Analyse de la complexité
3. Analyse formelle
4. Tri
5. Graphes

Paradigmes

6. Algorithmes gloutons
7. Approche diviser-pour-régner
8. Force brute
9. Programmation dynamique
10. Algorithmes probabilistes

+ Sujet avancé?

Contenu

1. Notions mathématiques **4h**

- rappels de notions de mathématiques discrètes
- notions de base en probabilités

Contenu

2. Analyse de la complexité des algorithmes **5h**

- notations asymptotiques
- analyse des algorithmes itératifs

Contenu

3. Analyse formelle des algorithmes	4h

- correction et terminaison
- utilisation d'invariants et d'assertions

Contenu

4. Tri **4h**

- Algorithmes de tri
- Algorithmes de sélection

5. Graphes **4h**

- Introduction à la théorie des graphes
- Algorithmes de manipulation et de parcours
- Tri topologique

Contenu

6. Algorithmes gloutons **4h**

- Calcul d’arbre couvrant de poids minimal
- Application à d’autres problèmes

Contenu

7. Approche diviser-pour-régner	5h

- Récurrences
- Analyse des algorithmes récursifs
- Théorème maître
- Application à des problèmes

8. Force brute — 2h

- Recherche exhaustive
- Retour arrière
- Explosion combinatoire et heuristiques
- Application à des problèmes

Contenu

9. Programmation dynamique **5h**

- Décomposition en sous-problèmes
- Approches ascendante et descendante
- Mémoïsation
- Calcul de plus courts chemins
- Application à d'autres problèmes

Contenu

10. Algorithmes probabilistes **4h**

- Algorithmes Las Vegas et Monte Carlo
- Analyse de temps en espérance
- Analyse de probabilité d’erreur

Contenu

★ Sujets avancés **2h**

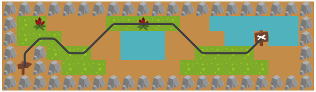
- Algorithmes d'approximation
- Complexité du calcul: P vs. NP
- Informatique quantique?

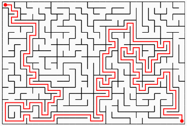
Cours à saveur théorique

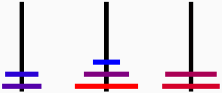
- Pas un cours de programmation
- Encouragé·e·s à implémenter vos algorithmes
- Code source disponible sur GitHub (Python 3)

Cours à saveur théorique

- Pas un cours de programmation
- Encouragé·e·s à implémenter vos algorithmes
- Code source disponible sur GitHub (Python 3)
- Exemples visuels au fil de la session







Matériel

- Aujourd'hui: premier et dernier cours avec diaporama
- Notes électroniques mises à jour chaque semaine
- Références complémentaires:
  - G. Brassard, P. Bratley:
    Fundamentals of Algorithmics. Prentice-Hall, 1996
  - T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein:
    Introduction to Algorithms. The MIT Press, 3rd edition, 2001
  - Notes de cours de Manuel Lafond

Devoirs

- 5 devoirs
- $4 \times$ deux semaines, $1 \times$ une semaine
- À faire en équipes de deux ou individuellement
- Annoncé un mercredi en classe
- À remettre un mercredi avant le cours
(en ligne sur Turnin)

Devoirs

- 5 devoirs (premier semaine prochaine)
- 4 $\times$ deux semaines, 1 $\times$ une semaine
- À faire en équipes de deux ou individuellement
- Annoncé un mercredi en classe
- À remettre un mercredi avant le cours
  (en ligne sur Turnin)

Évaluation

Devoirs	36% $(4 \times 8\% + 1 \times 4\%)$
Examen périodique	28%
Examen final	36%

Rétroaction non officielle vers la mi-session

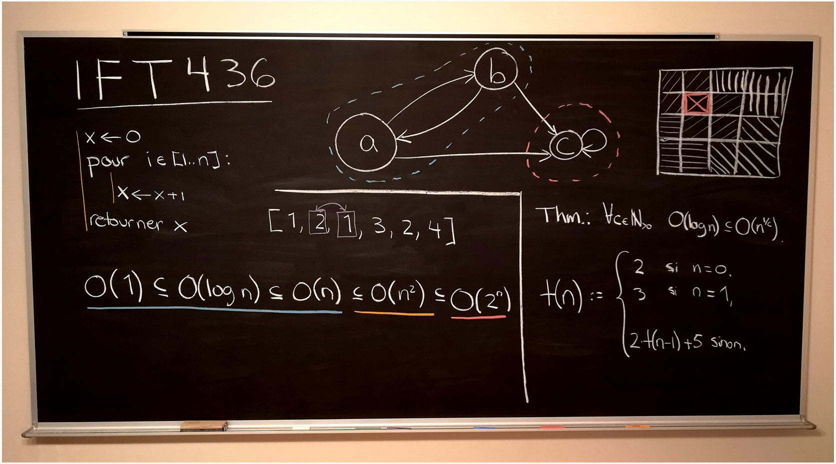
Calendrier

Sujets	**Exercices**	**Devoirs**
1: Introduction	✓	—
2: Notions math., analyse des algo.	✓	Devoir 1 2 semaines
3: Analyse des algorithmes, tri	✓	
4: Tri, graphes	✓	Devoir 2 2 semaines
5: Graphes, algorithmes gloutons	✓	
6: Algorithmes gloutons	révision	—
7: Examen périodique	—	—
8: Relâche	—	—

Calendrier

Sujets	Exercices	Devoirs
9: Approche diviser-pour-régner	✓	Devoir 3 2 semaines
10: Approche diviser-pour-régner	✓	
11: Force brute, prog. dynamique	✓	Devoir 4 2 semaines
12: Programmation dynamique	✓	
13: Algorithmes probabilistes	✓	Devoir 5 1 semaine
14: Sujets avancés (si possible)	révision	—
15: Examen final	—	—
16: Examen final	—	—

En cas de $n^{\text{ème}}$ vague:



Disponibilités

Sur **rendez-vous**: bureau / en ligne

et

1h / semaine (à choisir maintenant)

Page Web

info.usherbrooke.ca/mblondin/ift436

Introduction à l’algorithmique

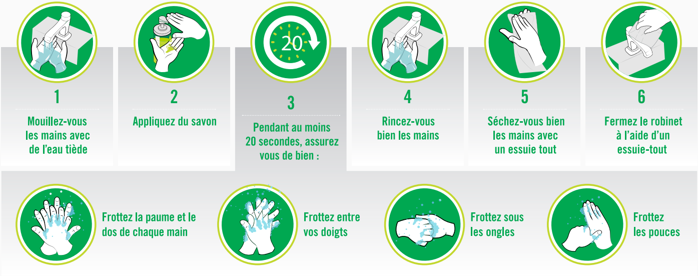
Qu’est-ce qu’un algorithme?

- Ensemble d’opérations à exécuter
- Chaque opération est non ambigüe
- Accomplit une tâche précise

1. Ouvrir le robinet
2. Utiliser du savon
3. ???
4. Mains propres!

Qu’est-ce qu’un algorithme?

- Ensemble d’opérations à exécuter
- Chaque opération est non ambigüe
- Accomplit une tâche précise

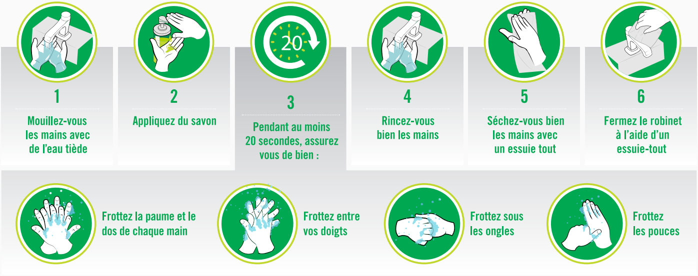


Référence :

Agence de la santé publique du Canada

Qu’est-ce qu’un algorithme?

- Ensemble d’opérations à exécuter
- Chaque opération est non ambigüe
- Accomplit une tâche précise



Référence :

Agence de la santé publique du Canada

UNIVERSITÉ DE SHERBROOKE

Qu'est-ce qu'un algorithme?

Notre intérêt: algorithmes implémentables sur un ordinateur

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$

tant que $i \leq n$ **faire**

  si $x_i > m$ **alors** $m \leftarrow x_i$

  $i \leftarrow i + 1$

retourner m

Qu'est-ce qu'un algorithme?

Notre intérêt: algorithmes implémentables sur un ordinateur

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$

Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$

tant que $i \leq n$ **faire**

    si $x_i > m$ **alors** $m \leftarrow x_i$

    $i \leftarrow i + 1$

retourner m

Description sous pseudocode pour éviter détails techniques

Qu'est-ce qu'un algorithme?

Notre intérêt: algorithmes implémentables sur un ordinateur

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$

tant que $i \leq n$ **faire**

  si $x_i > m$ **alors** $m \leftarrow x_i$

  $i \leftarrow i + 1$

retourner m

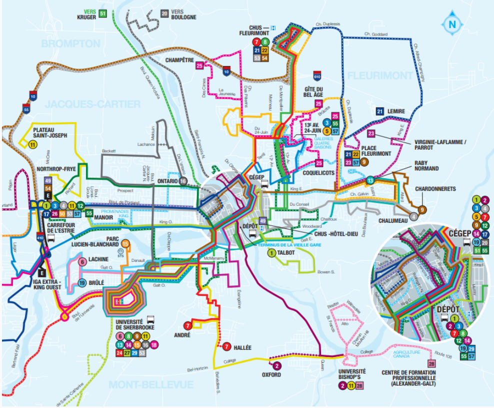
```python
def max(x):
    i, m = 1, x[0]

    while i < len(x):
        if x[i] > m: m = x[i]
        i += 1

    return m
```

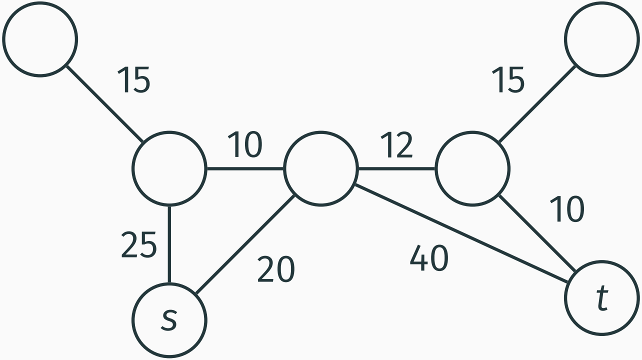
```haskell
max (x1:x') = max' x' x1 where
  max' [] m = m
  max' (xi:x') m | xi > m    = max' x' xi
                 | otherwise = max' x' m
```

Problème abstrait avec un domaine d’entrée et une sortie attendue définis rigoureusement

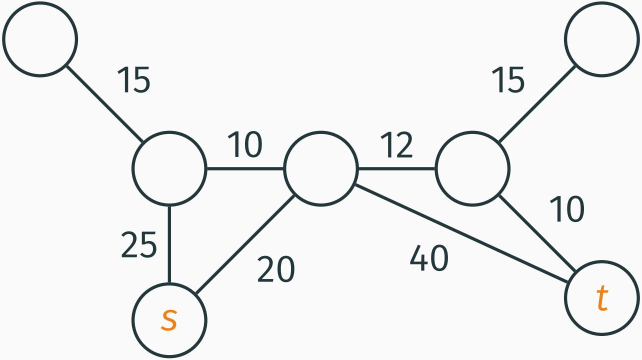


(Tirée de la STS)

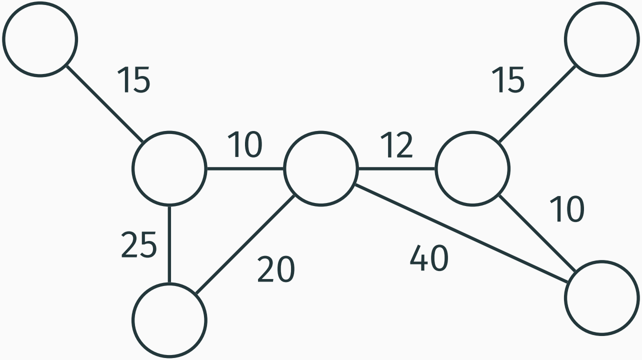
Problème abstrait avec un domaine d'entrée et une sortie attendue définis rigoureusement



Quel est le coût du plus court chemin de s vers t dans ce graphe?

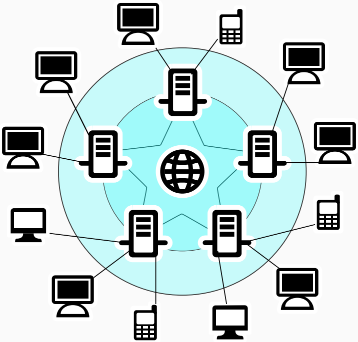


Quels sommets peut-on retirer sans déconnecter le graphe?

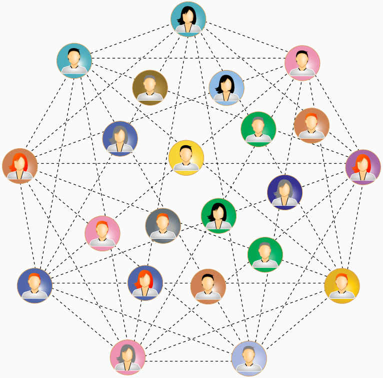


Qu’est-ce qu’une tâche?

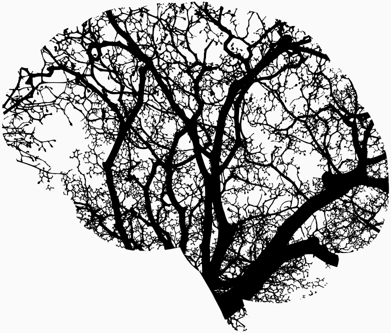
Une bonne abstraction permet souvent plusieurs applications



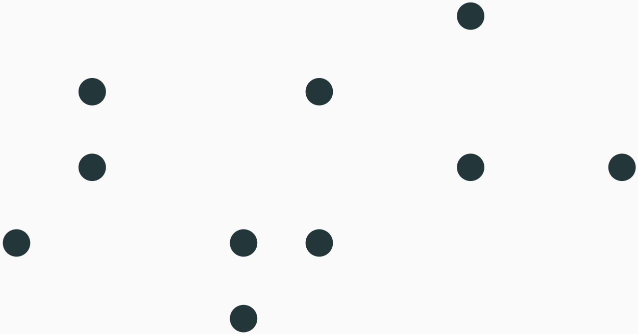
Une bonne abstraction permet souvent plusieurs applications



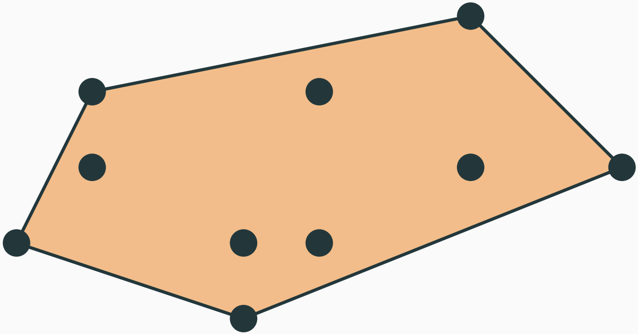
Une bonne abstraction permet souvent plusieurs applications



Qu’est-ce qu’une tâche?

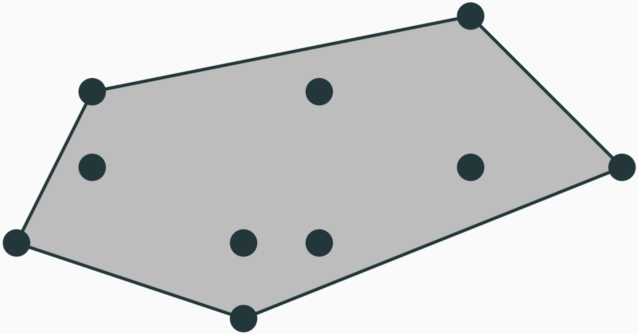


Quel est le polygone convexe qui contient tous les points et possède la plus petite surface?



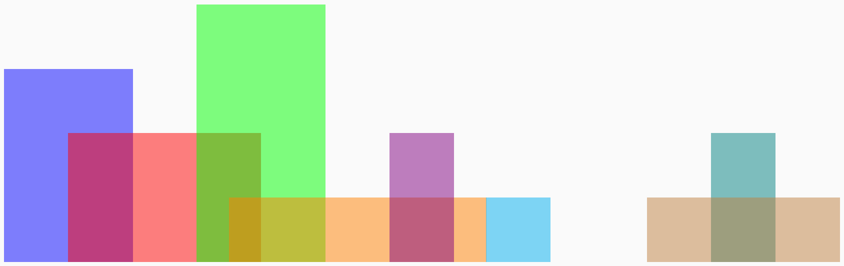
Quel est le polygone convexe qui contient tous les points et possède la plus petite surface?

Qu’est-ce qu’une tâche?

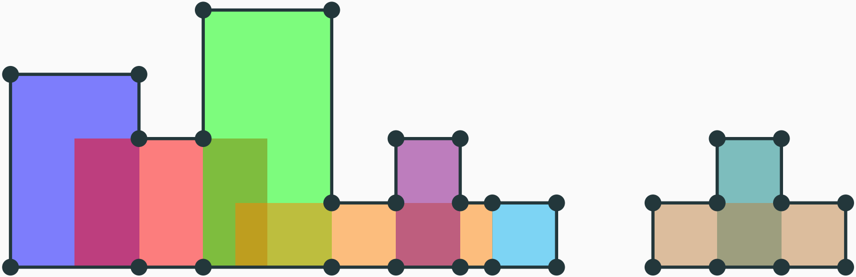


Applications: traitement d’images, géométrie computationnelle, etc.

Qu’est-ce qu’une tâche?



Qu’est-ce qu’une tâche?



Qu’est-ce qu’une tâche?

Quelle est la distance entre deux chaînes: nombre d’insertions, suppressions et modifications pour passer d’une à l’autre?

AACAGATTTGAGACCCATTGAGACCCAT
AACAAATTTGAGACTCATTGAGACCATT

Quelle est la distance entre deux chaînes: nombre d’ insertions, suppressions et modifications pour passer d’une à l’autre?

AACAGATTTGAGACCCATTGAGACCCAT
AACAAATTTGAGACTCATTGAGACCATT

Quelle est la distance entre deux chaînes: nombre d’ insertions, suppressions et modifications pour passer d’une à l’autre?

AACAGATTTGAGACCCATTGAGACCCAT
AACAAATTTGAGACTCATTGAGACCATT

Applications: biologie computationnelle, traitement des langues, auto-correction, etc.

Une formule propositionnelle est-elle satisfaisable?

$$(x_1 \lor x_2) \land (\neg x_2 \lor \neg x_3) \land (\neg x_3 \lor \neg x_1) \land (\neg x_2 \lor x_3)$$

Une formule propositionnelle est-elle satisfaisable?

$$(x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_1) \wedge (\neg x_2 \vee x_3)$$

Une formule propositionnelle est-elle satisfaisable?

$$(x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_1) \wedge (\neg x_2 \vee x_3)$$

Applications: vérification de programmes et de circuits, recherche opérationnelle, intelligence artificielle, etc.

Quelle est la plus longue sous-chaîne commune?

abdcababad
abaccabcaa

Quelle est la plus longue sous-chaîne commune?

abd**cab**abad

abac**cab**caa

Quelle est la plus longue sous-chaîne commune?

a b d c a b a b a d
a b a c c a b c a a

Applications: détection de plagiat, (différence de code source, bio-informatique, etc.)

Efficacité d'un algorithme

- Une tâche peut être accomplie par plusieurs algorithmes
- La puissance calculatoire varie selon l'ordinateur
- Le langage machine varie selon l'architecture

Comment évaluer et comparer l'efficacité d'algorithmes?

Efficacité d'un algorithme

- Une tâche peut être accomplie par plusieurs algorithmes
- La puissance calculatoire varie selon l'ordinateur
- Le langage machine varie selon l'architecture

Comment évaluer et comparer l'efficacité d'algorithmes?

Efficacité d'un algorithme

- Une tâche peut être accomplie par plusieurs algorithmes
- La puissance calculatoire varie selon l'ordinateur
- Le langage machine varie selon l'architecture

Comment évaluer et comparer l'efficacité d'algorithmes?

Efficacité d'un algorithme

- Une tâche peut être accomplie par plusieurs algorithmes
- La puissance calculatoire varie selon l'ordinateur
- Le langage machine varie selon l'architecture

Comment évaluer et comparer l'efficacité d'algorithmes?

En faisant abstraction de l’ordinateur:

- Définir des opérations élémentaires
- Compter nombre d’opér. élém. $f(x)$ pour chaque entrée x
- Considérer $t(n) = \max \{f(x) : \text{entrée } x \text{ de taille } n\}$ (pire cas)
- Analyser $t(n)$ asymptotiquement, c.-à-d. lorsque $n \to \infty$

En faisant abstraction de l'ordinateur:

- Définir des opérations élémentaires
- Compter nombre d'opér. élém. $f(x)$ pour chaque entrée x
- Considérer $t(n) = \max \{f(x) : \text{entrée } x \text{ de taille } n\}$ (pire cas)
- Analyser $t(n)$ asymptotiquement, c.-à-d. lorsque $n \to \infty$

En faisant abstraction de l'ordinateur:

- Définir des opérations élémentaires
- Compter nombre d'opér. élém. $f(x)$ pour chaque entrée x
- Considérer $t(n) = \max\{f(x) : \text{entrée } x \text{ de taille } n\}$ (pire cas)
- Analyser $t(n)$ asymptotiquement, c.-à-d. lorsque $n \to \infty$

En faisant abstraction de l'ordinateur:

- Définir des opérations élémentaires
- Compter nombre d'opér. élém. $f(x)$ pour chaque entrée x
- Considérer $t(n) = \max \{f(x) : \text{entrée } x \text{ de taille } n\}$ (pire cas)
- Analyser $t(n)$ asymptotiquement, c.-à-d. lorsque $n \to \infty$

En faisant abstraction de l'ordinateur:

- Définir des opérations élémentaires
- Compter nombre d'opér. élém. $f(x)$ pour chaque entrée x
- Considérer $t(n) = \max \{f(x) : \text{entrée } x \text{ de taille } n\}$ (pire cas)
- Analyser $t(n)$ asymptotiquement, c.-à-d. lorsque $n \to \infty$

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire**
  si $x_i > m$ **alors** $m \leftarrow x_i$
  $i \leftarrow i + 1$
retourner m

Opérations élémentaires?

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire**
  si $x_i > m$ **alors** $m \leftarrow x_i$
  $i \leftarrow i + 1$
retourner m

Opérations élémentaires:

- Comparaison
- Affectation
- Arithmétique dont l'addition
- Accès au $i^{\text{ème}}$ élément d'une séquence

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire**
    si $x_i > m$ **alors** $m \leftarrow x_i$
    $i \leftarrow i + 1$
retourner m

Opérations élémentaires:

- Comparaison
- Affectation
- Arithmétique dont l'addition
- Accès au $i^{\text{ème}}$ élément d'une séquence

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire**
    si $x_i > m$ **alors** $m \leftarrow x_i$
    $i \leftarrow i + 1$
retourner m

Opérations élémentaires:

- Comparaison
- Affectation
- Arithmétique dont l'addition
- Accès au $i^{\text{ème}}$ élément d'une séquence

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire**
  si $x_i > m$ **alors** $m \leftarrow x_i$
  $i \leftarrow i + 1$
retourner m

Opérations élémentaires:

- Comparaison
- Affectation
- Arithmétique dont l'addition
- Accès au $i^{\text{ème}}$ élément d'une séquence

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$	//	3
tant que $i \leq n$ **faire**	//	$n - 1$
si $x_i > m$ **alors** $m \leftarrow x_i$	//	$[2(n-1),\ 4(n-1)]$
$i \leftarrow i + 1$	//	$2(n-1)$
retourner m	//	`Total:` $[5n-2,\ 7n-4]$

Opérations élémentaires:

- Comparaison
- Affectation
- Arithmétique dont l'addition
- Accès au $i^{\text{ème}}$ élément d'une séquence

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$ // $\mathcal{O}(1)$
tant que $i \leq n$ **faire** // $\mathcal{O}(n)$
    si $x_i > m$ **alors** $m \leftarrow x_i$ // $\mathcal{O}(n)$
    $i \leftarrow i + 1$ // $\mathcal{O}(n)$
retourner m // Total: $\mathcal{O}(n)$

Opérations élémentaires:

- Comparaison
- Affectation
- Arithmétique dont l'addition
- Accès au $i^{\text{ème}}$ élément d'une séquence

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$	//	$\mathcal{O}(1)$
tant que $i \leq n$ **faire**	//	$\mathcal{O}(n)$ fois
si $x_i > m$ **alors** $m \leftarrow x_i$	//	$\mathcal{O}(1)$
$i \leftarrow i + 1$	//	$\mathcal{O}(1)$
retourner m	//	Total: $\mathcal{O}(n)$

Opérations élémentaires:

- Comparaison
- Affectation
- Arithmétique dont l'addition
- Accès au $i^{\text{ème}}$ élément d'une séquence

Efficacité d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$\texttt{max}(x_1, \ldots, x_n)$:
- **si** $n = 1$ **alors**
  - **retourner** x_1
- **sinon**
  - $m \leftarrow \texttt{max}(x_1, \ldots, x_{n \div 2})$
  - $m' \leftarrow \texttt{max}(x_{n \div 2 + 1}, \ldots, x_n)$
  - **si** $m > m'$ **alors retourner** m
  - **sinon retourner** m'

Comment analyser cet algorithme?

Efficacité d'un algorithme

Entrées : séquence d'entiers $x_1, \ldots, x_n$ t.q. $|\{x_1, \ldots, x_n\}| = 2$
Sorties : maximum de la séquence

piger $i \in [1..n]$ aléatoirement de façon uniforme
répéter
    piger $j \in [1..n]$ aléatoirement de façon uniforme
    si $x_j > x_i$ **alors** $i \leftarrow j$
jusqu'à $x_i \neq x_j$
retourner x_i

Et celui-ci?

Efficacité d’un algorithme

Plusieurs paramètres peuvent être analysés:

- Temps (nombre d’opérations)
- Quantité de mémoire
- Probabilité de succès
- Nombre de processeurs
- Nombre d’écritures en mémoire
- Nombre de qubits
- etc.

Efficacité d’un algorithme

Plusieurs paramètres peuvent être analysés:

- Temps (nombre d’opérations)
- Quantité de mémoire
- Probabilité de succès
- Nombre de processeurs
- Nombre d’écritures en mémoire
- Nombre de qubits
- etc.

Exactitude d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire**
  si $x_i > m$ **alors** $m \leftarrow x_i$
  $i \leftarrow i + 1$
retourner m

Comment s'assurer qu'un algorithme fonctionne bel et bien?

Exactitude d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire**
  si $x_i > m$ **alors** $m \leftarrow x_i$
  $i \leftarrow i + 1$
retourner m

Comment s'assurer qu'un algorithme fonctionne bel et bien?

- **Terminaison:** termine sur toute entrée
- **Correction:** bonne sortie sur toute entrée

Exactitude d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire** //
  si $x_i > m$ **alors** $m \leftarrow x_i$ //
  $i \leftarrow i + 1$ // i croît strictement
retourner m

- **Terminaison:** termine sur toute entrée
- **Correction:** bonne sortie sur toute entrée

Exactitude d'un algorithme

Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire** // $m = \max\{x_1, \ldots, x_{i-1}\}$
  si $x_i > m$ **alors** $m \leftarrow x_i$
  $i \leftarrow i + 1$
retourner m

- **Terminaison:** termine sur toute entrée
- **Correction:** bonne sortie sur toute entrée

Exactitude d'un algorithme

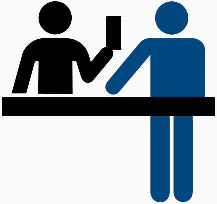
Entrées : séquence non vide d'entiers $x_1, x_2, \ldots, x_n$
Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$
tant que $i \leq n$ **faire** // $m = \max\{x_1, \ldots, x_{i-1}\}$
  si $x_i > m$ **alors** $m \leftarrow x_i$ // (preuve par induction sur i)
  $i \leftarrow i + 1$
retourner m

- **Terminaison:** termine sur toute entrée
- **Correction:** bonne sortie sur toute entrée

Exemple: rendre la monnaie

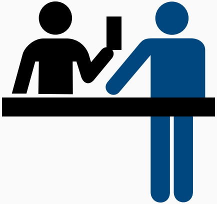
2\$ 1\$ 25¢ 10¢ 5¢ 1¢



Comment rendre la monnaie
avec le moins de pièces?

Exemple: rendre la monnaie

2\$ 1\$ 25¢ 10¢ 5¢ 1¢



$$
\begin{aligned}
582¢ - 2 \times 200¢ &= 182¢ \\
182¢ - 1 \times 100¢ &= 82¢ \\
82¢ - 3 \times 25¢ &= 7¢ \\
7¢ - 1 \times 5¢ &= 2¢ \\
2¢ - 2 \times 1¢ &= 0¢
\end{aligned}
$$

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$, $i \leftarrow 1$
tant que $r > 0$ **faire**
    si $r \geq v_i$ **alors**
        $sol \leftarrow sol + 1$
        $r \quad \leftarrow r - v_i$
    sinon
        $i \leftarrow i + 1$
retourner sol

Sous forme de pseudocode

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$, $i \leftarrow 1$
tant que $r > 0$ **faire**
  si $r \geq v_i$ **alors**
    $sol \leftarrow sol + 1$
    $r \quad \leftarrow r - v_i$
  sinon
    $i \leftarrow i + 1$
retourner sol

Temps d'exécution polynomial?

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$, $i \leftarrow 1$
tant que $r > 0$ **faire**
    si $r \geq v_i$ **alors**
        $sol \leftarrow sol + 1$
        $r \quad \leftarrow r - v_i$
    sinon
        $i \leftarrow i + 1$
retourner sol

Temps d'exécution polynomial? $\mathcal{O}(m)$

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$, $i \leftarrow 1$
tant que $r > 0$ **faire**
  si $r \geq v_i$ **alors**
    $sol \leftarrow sol + 1$
    $r \quad \leftarrow r - v_i$
  sinon
    $i \leftarrow i + 1$
retourner sol

Temps d'exécution polynomial? $\mathcal{O}(2^{\log m})$

Exponentiel dans le nombre de bits...

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$
pour $i \in [1..n]$ **faire**
    $sol \leftarrow sol + (r \div v_i)$
    $r \quad \leftarrow r \bmod v_i$
retourner sol

Même idée, mais fonctionne maintenant en temps polynomial

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$
pour $i \in [1..n]$ **faire**
    $sol \leftarrow sol + (r \div v_i)$
    $r \quad \leftarrow r \bmod v_i$
retourner sol

Comment prouver que l'algorithme fonctionne?

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$
pour $i \in [1..n]$ **faire**
  $sol \leftarrow sol + (r \div v_i)$
  $r \quad \leftarrow r \bmod v_i$
retourner sol

Comment prouver que l'algorithme fonctionne?

...

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$
pour $i \in [1..n]$ **faire**
  $sol \leftarrow sol + (r \div v_i)$
  $r \leftarrow r \bmod v_i$
retourner sol

Comment prouver que l'algorithme fonctionne?

Impossible puisqu'il ne fonctionne pas en général!

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$
pour $i \in [1..n]$ **faire**
  $sol \leftarrow sol + (r \div v_i)$
  $r \quad \leftarrow r \bmod v_i$
retourner sol

1 000 000$ si vous trouvez un algorithme qui fonctionne (correctement) en temps polynomial!

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \ldots, v_n \in \mathbb{N}_{>0}$, montant $m \in \mathbb{N}$
Sorties : nombre minimal de pièces qui somment à m

trier v en ordre décroissant
$r \leftarrow m$, $sol \leftarrow 0$
pour $i \in [1..n]$ **faire**
  $sol \leftarrow sol + (r \div v_i)$
  $r \leftarrow r \bmod v_i$
retourner sol

Pas toujours évident de trouver des algorithmes efficaces et corrects!

Problème

- Généralement une **abstraction** de problèmes concrets
- **Entrées/sorties:** $\in$ domaines définis rigoureusement
- **Sortie attendue:** propriété ou objet défini rigoureusement

Algorithme

- **Opérations** et structures de contrôle bien définies
- Résout un **problème** précis
- **Termine** sur toute entrée
- Retourne la **sortie attendue** sur toute entrée

Pseudocode

- « Langage universel » de **description** d'algorithme
- Fait **abstraction** des langages de programmation
- Pas de standard, mais doit être clairement **implémentable**
- Peut invoquer des « **boîtes noires** » si elles sont connues

Efficacité

- **Mesure** d'une quantité en fonction de la taille de l'entrée
- Le temps dépend d'**opérations élémentaires** à définir
- Emphase mise sur le **comportement asymptotique**
- **Différents cas:** meilleur cas, pire cas, cas moyen

À venir: rappel

Fondements

1. Notions mathématiques
2. Analyse de la complexité
3. Analyse formelle
4. Tri
5. Graphes

Paradigmes

6. Algorithmes gloutons
7. Approche diviser-pour-régner
8. Force brute
9. Programmation dynamique
10. Algorithmes probabilistes

⚠ Pas de cours demain

À mercredi prochain!