

Introduction

IFT436 – Algorithmes et structures de données

Automne 2021



- Plan de cours
- Introduction à l'algorithmique

Plan de cours

Enseignant: Michael Blondin

@ michael.blondin@usherbrooke.ca

🌐 info.usherbrooke.ca/mbлондин

📍 bureau D4-1024-1 au 1^{er} étage

Correcteur: François Ladouceur

@ francois.ladouceur@usherbrooke.ca

Cours	Mercredi	08h30 – 10h30
	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

- 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

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?

1. Notions mathématiques

4h

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

2. Analyse de la complexité des algorithmes

5h

- notations asymptotiques
- analyse des algorithmes itératifs

3. Analyse formelle des algorithmes

4h

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

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

6. Algorithmes gloutons

4h

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

7. Approche diviser-pour-régner

5h

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

8. Force brute

2h

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

9. Programmation dynamique

5h

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

10. Algorithmes probabilistes

4h

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

★ Sujets avancés

2h

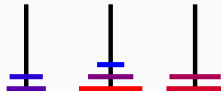
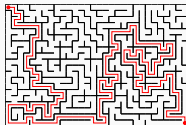
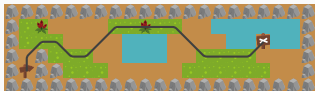
- Complexité du calcul dont P vs. NP
- Informatique quantique?

Cours à saveur théorique

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

Cours à saveur théorique

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



- 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

- 6 devoirs:
- 4 × deux semaines, 2 × une semaine
- À faire en équipes de deux ou individuellement
- Annoncé un mercredi en classe
- À remettre un mardi soir sur Turnin
(sauf une fois, un mercredi, pour compenser Action de grâce)

- 6 devoirs: 1^{er} la semaine prochaine
- 4 × deux semaines, 2 × une semaine
- À faire en équipes de deux ou individuellement
- Annoncé un mercredi en classe
- À remettre un mardi soir sur Turnin
(sauf une fois, un mercredi, pour compenser Action de grâce)

Devoirs	40% $(4 \times 8\% + 2 \times 4\%)$
Examen périodique	25%
Examen final	35%

Rétroaction non officielle vers la
mi-session

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

Sujets	Exercices	Devoirs
10: Diviser-pour-régner + retour examen	✓	Devoir 4 2 semaines
11: Diviser-pour-régner	✓	
12: Force brute, prog. dynamique	✓	Devoir 5 2 semaines
13: Programmation dynamique	✓	
14: Algorithmes probabilistes	✓	Devoir 6 1 semaine
15: Sujets avancés	révision	—
16: Examen final	—	—
17: Examen final	—	—

En cas de n^{ème} annonce du gouvernement

Si la distanciation devient obligatoire:

hybride en alternance (règle du Département)

En cas de $n^{\text{ème}}$ annonce du gouvernement

S'il y a confinement...

IFT 436

```

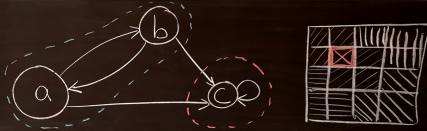
x ← 0
pour i ∈ [1..n]:
    x ← x + 1
retourner x
    
```

$[1, \boxed{2}, \boxed{1}, 3, 2, 4]$

$\underline{O(1)} \leq \underline{O(\log n)} \leq \underline{O(n)} \leq \underline{O(n^2)} \leq \underline{O(2^n)}$

Thm.: $\forall c \in \mathbb{N}_0, O(\log n) \leq O(n^c)$

$$t(n) := \begin{cases} 2 & \text{si } n=0, \\ 3 & \text{si } n=1, \\ 2t(n-1)+5 & \text{sinon.} \end{cases}$$



Sur **rendez-vous**: bureau ou en ligne
et

1h / semaine (à choisir maintenant)

 info.usherbrooke.ca/mblondin/ift436

Introduction à l'algorithmique

Qu'est-ce qu'un algorithme?

- Ensemble d'opérations à exécuter
 - Chaque opération est non ambiguë
 - 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 ambiguë
- Accomplit une tâche précise



Référence :

Qu'est-ce qu'un algorithme?

- Ensemble d'opérations à exécuter
- Chaque opération est non ambiguë
- Accomplit une tâche précise



Référence :



Agence de la santé
publique du Canada

Qu'est-ce qu'un algorithme?

Notre intérêt: algorithmes implémentables sur un ordinateur

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

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$

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

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

$i \leftarrow i + 1$

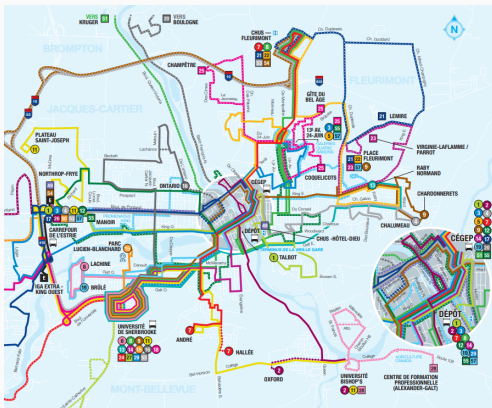
retourner m

```
def max(x):
    i, m = 1, x[0]
    while i < len(x):
        if x[i] > m: m = x[i]
        i += 1
    return m
```

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

Qu'est-ce qu'une tâche?

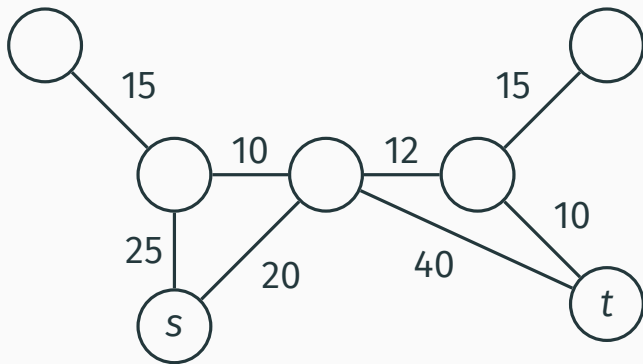
Problème abstrait avec un domaine d'entrée et une sortie attendue définis rigoureusement



(Tirée de la STS)

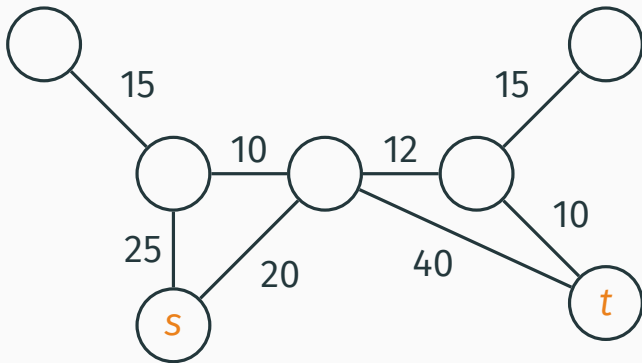
Qu'est-ce qu'une tâche?

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



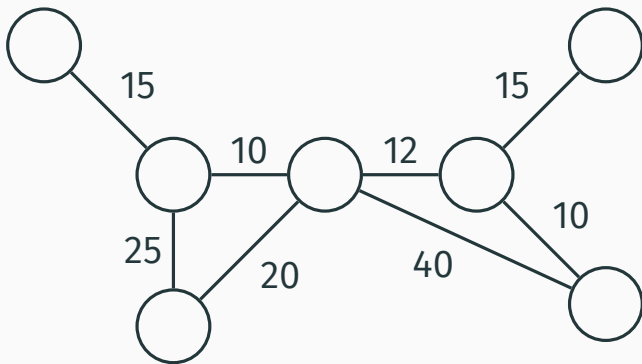
Qu'est-ce qu'une tâche?

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

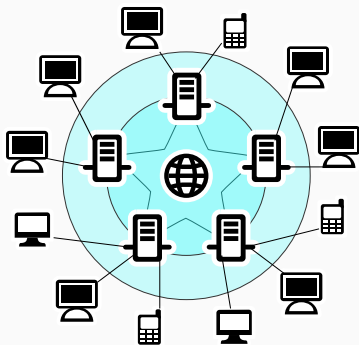


Qu'est-ce qu'une tâche?

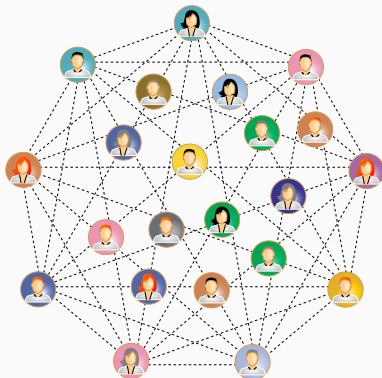
Quels sommets peut-on retirer sans déconnecter le graphe?



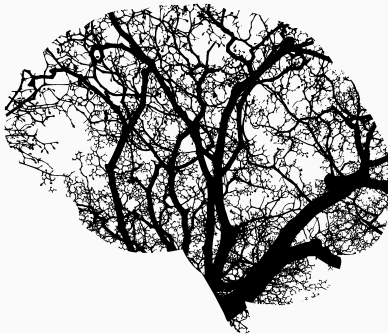
Une bonne abstraction permet souvent plusieurs applications



**Une bonne abstraction permet souvent
plusieurs applications**



**Une bonne abstraction permet souvent
plusieurs applications**

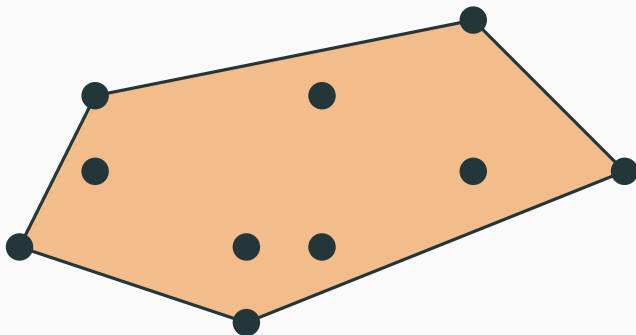


Qu'est-ce qu'une tâche?



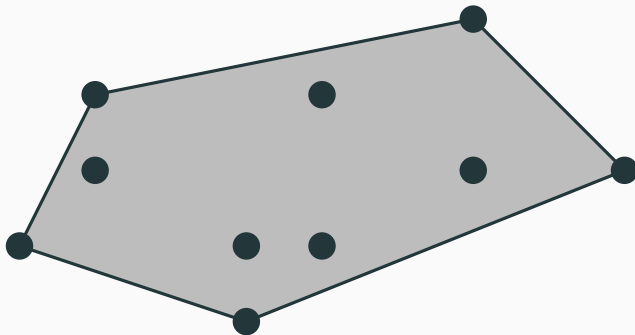
Quel est le polygone convexe qui contient tous les points et possède la plus petite surface?

Qu'est-ce qu'une tâche?



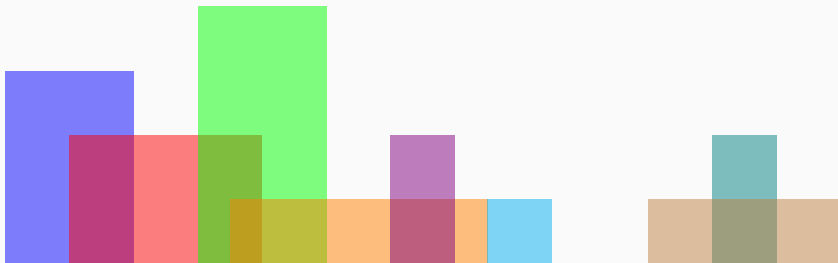
Quel est le polygone convexe qui contient tous les points et possède la plus petite surface?

Qu'est-ce qu'une tâche?

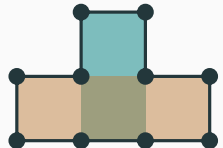
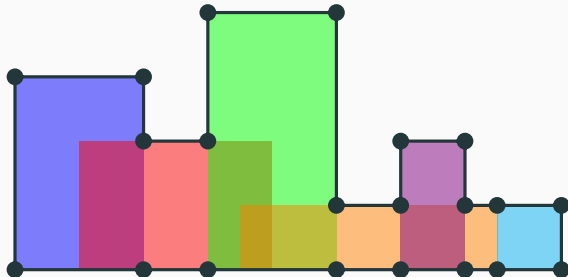


Applications: traitement d'images, géométrie computationnelle, etc.

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

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?**

AACAGATTTGAGACCATTGAGACCAT
AACAAATTTGAGACTCATTGAGACCATT

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

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

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?

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

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

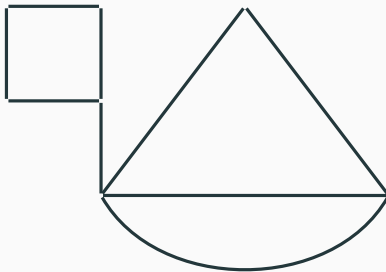
Quelle est la plus longue sous-chaîne commune?

abdcababad
abaccabcaa

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

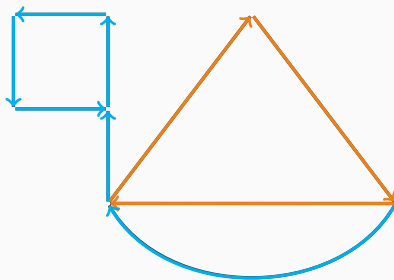
Qu'est-ce qu'une tâche?

Possible de dessiner sans lever le crayon?



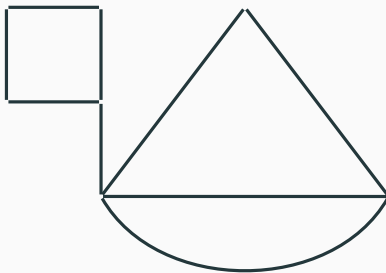
Qu'est-ce qu'une tâche?

Possible de dessiner sans lever le crayon?



Qu'est-ce qu'une tâche?

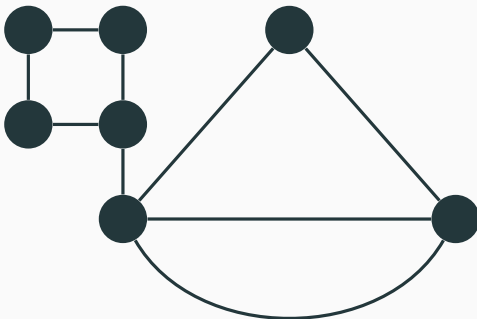
Possible de dessiner sans lever le crayon?



**Applications: passer le temps sur
Instagram et TikTok...**

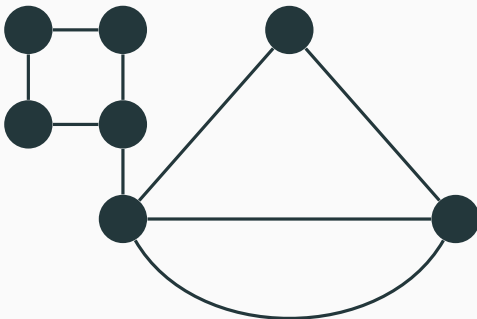
Qu'est-ce qu'une tâche?

Possible d'utiliser chaque arête une seule fois?



Qu'est-ce qu'une tâche?

Possible d'utiliser chaque arête une seule fois?



**Applications: faire du porte-à-porte
aux élections...**

- 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?**

- 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?**

- 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?**

- 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 $n \rightarrow \infty$

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 $n \rightarrow \infty$

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 $n \rightarrow \infty$

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 $n \rightarrow \infty$

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 $n \rightarrow \infty$

Efficacité d'un algorithme

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

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

Sorties : maximum de la séquence

$\max(x_1, \dots, x_n)$:

si $n = 1$ **alors**

retourner x_1

sinon

$m \leftarrow \max(x_1, \dots, x_{n \div 2})$

$m' \leftarrow \max(x_{n \div 2 + 1}, \dots, x_n)$

si $m > m'$ **alors retourner** m

sinon retourner m'

Comment analyser cet algorithme?

Efficacité d'un algorithme

Entrées : séquence d'entiers $x_1, \dots, x_n$ t.q. $|\{x_1, \dots, x_n\}| = 2$

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?

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.

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

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 $x_1, x_2, \dots, x_n$

Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$

tant que $i \leq n$ **faire** // $m = \max\{x_1, \dots, x_{i-1}\}$

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 $x_1, x_2, \dots, x_n$

Sorties : maximum de la séquence

$i \leftarrow 2, m \leftarrow x_1$

tant que $i \leq n$ **faire** // $m = \max\{x_1, \dots, x_{i-1}\}$

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¢



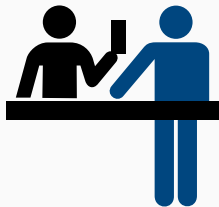
10¢



5¢



1¢



Comment rendre la monnaie
avec le moins de pièces?

Exemple: rendre la monnaie



2\$



1\$



25¢



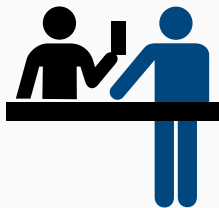
10¢



5¢



1¢



$$582¢ - 2 \times 200¢ = 182¢$$

$$182¢ - 1 \times 100¢ = 82¢$$

$$82¢ - 3 \times 25¢ = 7¢$$

$$7¢ - 1 \times 5¢ = 2¢$$

$$2¢ - 2 \times 1¢ = 0¢$$

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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**

$sol \leftarrow sol + 1$

$r \leftarrow r - v_i$

sinon

$i \leftarrow i + 1$

retourner sol

Sous forme de pseudocode

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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**

$sol \leftarrow sol + 1$

$r \leftarrow r - v_i$

sinon

$i \leftarrow i + 1$

retourner sol

Temps d'exécution polynomial?

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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**

$sol \leftarrow sol + 1$

$r \leftarrow r - v_i$

sinon

$i \leftarrow i + 1$

retourner sol

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

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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**

$sol \leftarrow sol + 1$

$r \leftarrow r - v_i$

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 $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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

**Même idée, mais fonctionne maintenant en
temps polynomial**

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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?

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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?

...

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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 $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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

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

Exemple: rendre la monnaie

Entrées : sys. monétaire $v_1, \dots, v_n \in \mathbb{N}_{>0}$, 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

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!