Algorithmique Avancée et Complexité

Présentation du cours

Sophie Tison
Lille1- Master1 Informatique

OBJECTIFS

Avoir des outils pour concevoir un "bon" algorithme pour résoudre un problème.

Objectifs

Avoir des outils pour concevoir un "bon"? algorithme pour résoudre un problème.

Objectifs

Avoir des outils pour concevoir un "bon" - c.à.d. correct et efficace - algorithme pour résoudre un problème.

Cela pose de nombreuses questions...et demande pas mal de savoir-faire...

Existe-il un algorithme pour résoudre le problème?

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE...

Existe-il un algorithme pour résoudre le problème?

Connaître quelques Notions de Calculabilité et de décidabilité.

Cela pose de nombreuses questions...et demande pas mal de savoir-faire.

Comment modéliser le problème?

Cela pose de nombreuses questions...et demande pas mal de savoir-faire.

Comment modéliser le problème?
Est-ce un problème classique?

Savoir modéliser, Connaître et savoir reconnaître des grands classiques

Cela pose de nombreuses questions...et demande pas mal de savoir-faire.

Comment modéliser le problème?
Est-ce un problème classique?

Savoir modéliser, Connaître et savoir reconnaître des grands classiques

Tris, méthodes de sélection, recherche

Cela pose de nombreuses questions...et demande pas mal de savoir-faire.

Comment modéliser le problème?
Est-ce un problème classique?

Savoir modéliser, Connaître et savoir reconnaître des grands classiques

Tris, méthodes de sélection, recherche
algorithmique des graphes,

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE.

Comment modéliser le problème?
Est-ce un problème classique?

Savoir modéliser, Connaître et savoir reconnaître des grands classiques

Tris, méthodes de sélection, recherche
algorithmique des graphes,
méthodes de hachages

Cela pose de nombreuses questions...et demande pas mal de savoir-faire.

Comment modéliser le problème?
Est-ce un problème classique?

Savoir modéliser, Connaître et savoir reconnaître des grands classiques

Tris, méthodes de sélection, recherche
algorithmique des graphes,
méthodes de hachages
Programmation linéaire...

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE.

Comment modéliser le problème?
Est-ce un problème classique?

Savoir modéliser, Connaître et savoir reconnaître des grands classiques

Tris, méthodes de sélection, recherche
algorithmique des graphes,
méthodes de hachages
Programmation linéaire...
....

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE

Comment concevoir un algorithme?

Cela pose de nombreuses questions...et demande pas mal de savoir-faire

Comment concevoir un algorithme?

Schémas d'algorithmes, Algorithmic design patterns

Cela pose de nombreuses questions...et demande pas mal de savoir-faire

Comment concevoir un algorithme?

Schémas d'algorithmes, Algorithmic design patterns

"Diviser pour régner"

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE

Comment concevoir un algorithme?

Schémas d'algorithmes, Algorithmic design patterns

"Diviser pour régner"
Programmation Dynamique

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE

Comment concevoir un algorithme?

Schémas d'algorithmes, Algorithmic design patterns

"Diviser pour régner"
Programmation Dynamique
Algorithmes gloutons

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE.

L'algorithme est-il correct?

Cela pose de nombreuses questions...et demande pas mal de savoir-faire.

L'algorithme est-il correct?

Savoir prouver un algorithme...
ou tout du moins avoir un minimum de rigueur
pour concevoir des algorithmes

CELA POSE DE NOMBREUSES QUESTIONS... ET DEMANDE PAS MAL DE SAVOIR-FAIRE.

L'algorithme est-il efficace?

Cela pose de nombreuses questions... et demande pas mal de savoir-faire.

L'algorithme est-il efficace?

Savoir analyser la complexité d'algorithmes

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE DE MULTIPLES COMPÉTENCES.

Peut-on trouver un algorithme plus efficace pour le problème?
Est-ce un problème dur?

Cela pose de nombreuses questions...et demande de multiples compétences.

Peut-on trouver un algorithme plus efficace pour le problème?
Est-ce un problème dur?

Avoir quelques notions de Complexité des problèmes

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE.

Si le problème est dur, comment l'appréhender!

Cela pose de nombreuses questions...et demande pas mal de savoir-faire.

Si le problème est dur, comment l'appréhender!

Connaître quelques techniques d' Algorithmique "Avancée":

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE.

Si le problème est dur, comment l'appréhender!

Connaître quelques techniques d' Algorithmique "Avancée":

Backtracking, minmax, séparation-évaluation

CELA POSE DE NOMBREUSES QUESTIONS...ET DEMANDE PAS MAL DE SAVOIR-FAIRE.

Si le problème est dur, comment l'appréhender!

Connaître quelques techniques d' Algorithmique "Avancée":

Backtracking, minmax, séparation-évaluation
Méta-heuristiques, Algorithmes probabilistes,...

Organisation

Programme Prévisionnel

Quelques schémas d'algorithmes (3 cours)

Programme Prévisionnel

Quelques schémas d'algorithmes (3 cours)

Un peu de complexité de problèmes (3 cours)

Programme Prévisionnel

Quelques schémas d'algorithmes (3 cours)

Un peu de complexité de problèmes (3 cours)

Un peu d'algorithmique avancée (2-3 cours)

PROGRAMME PRÉVISIONNEL

Quelques schémas d'algorithmes (3 cours)

Un peu de complexité de problèmes (3 cours)

Un peu d'algorithmique avancée (2-3 cours)

Quelques notions de décidabilité et calculabilité (1-2 cours)

Bibliographie : de nombreuses ressources en ligne

. Le dictionnaire recensant les algorithmes et problèmes classiques du NIST

Bibliographie : de nombreuses ressources en ligne

. Le dictionnaire recensant les algorithmes et problèmes classiques du NIST

. The Stony Brook Algorithm Repository" qui contient des implémentations d'algorithmes pour des dizaines de problèmes classiques,

BIBLIOGRAPHIE : DE NOMBREUSES RESSOURCES EN LIGNE

. Le dictionnaire recensant les algorithmes et problèmes classiques du NIST

. The Stony Brook Algorithm Repository" qui contient des implémentations d'algorithmes pour des dizaines de problèmes classiques,

. Algorithms Courses on the WWW, qui contient une collection de cours d'algorithmique,

BIBLIOGRAPHIE : DE NOMBREUSES RESSOURCES EN LIGNE

. Le dictionnaire recensant les algorithmes et problèmes classiques du NIST

. The Stony Brook Algorithm Repository" qui contient des implémentations d'algorithmes pour des dizaines de problèmes classiques,

. Algorithms Courses on the WWW, qui contient une collection de cours d'algorithmique,

. Le site du cours "Algorithms in the Real World",

BIBLIOGRAPHIE : DE NOMBREUSES RESSOURCES EN LIGNE

. Le dictionnaire recensant les algorithmes et problèmes classiques du NIST

. The Stony Brook Algorithm Repository" qui contient des implémentations d'algorithmes pour des dizaines de problèmes classiques,

. Algorithms Courses on the WWW, qui contient une collection de cours d'algorithmique,

. Le site du cours "Algorithms in the Real World",

. "A compendium of NP optimization problems"

BIBLIOGRAPHIE : DE NOMBREUX LIVRES

. Cormen, Leiserson, Rivest, ”Introduction à l’algorithmique”, Dunod (disponible à la BU) vraiment ”une” référence essentielle en algorithmique,

BIBLIOGRAPHIE : DE NOMBREUX LIVRES

. Cormen, Leiserson, Rivest, ”Introduction à l’algorithmique”, Dunod (disponible à la BU) vraiment ”une” référence essentielle en algorithmique,

. S. Skiena, ”Algorithm Design Manual”, une ”mine”! (Une version on-line proche du livre papier),

BIBLIOGRAPHIE : DE NOMBREUX LIVRES

. Cormen, Leiserson, Rivest, "Introduction à l'algorithmique", Dunod (disponible à la BU) vraiment "une" référence essentielle en algorithmique,

. S. Skiena, "Algorithm Design Manual", une "mine"! (Une version on-line proche du livre papier),

. Jon Kleinberg & Eva Tardos, "Algorithm design", Addison Wesley 2005

Bibliographie : de nombreux livres

. Cormen, Leiserson, Rivest, "Introduction à l'algorithmique", Dunod (disponible à la BU) vraiment "une" référence essentielle en algorithmique,

. S. Skiena, "Algorithm Design Manual", une "mine"! (Une version on-line proche du livre papier),

. Jon Kleinberg & Eva Tardos, "Algorithm design", Addison Wesley 2005

. Sur l'aspect "algorithmic pattern", "Data Structures and Algorithms with object-oriented design patterns in Java". 2000, disponible sur le Web , ...

Organisation: les TPs

Il y aura 6 séances de TP (langage : Java) encadrées pour la mise en oeuvre directe des méthodes étudiées en cours:

ORGANISATION: LES TPS

Il y a 6 séances de TP (langage : Java) encadrées pour la mise en oeuvre directe des méthodes étudiées en cours:

. Programmation dynamique, Algorithmes gloutons(2 séances)

Organisation: les TPs

Il y aura 6 séances de TP (langage : Java) encadrées pour la mise en oeuvre directe des méthodes étudiées en cours:

. Programmation dynamique, Algorithmes gloutons(2 séances)

. Propriétés NP, réductions polynomiales (2 séances)

ORGANISATION: LES TPS

Il y aura 6 séances de TP (langage : Java) encadrées pour la mise en oeuvre directe des méthodes étudiées en cours:

. Programmation dynamique, Algorithmes gloutons(2 séances)

. Propriétés NP, réductions polynomiales (2 séances)

. Heuristiques, Métaheuristiques ou Calculabilité (2 séances)

.... EVALUATION

Le contrôle continu sera basé sur les TPs et un DS en milieu de semestre. La note de contrôle continu sera $1/3 * (note\ DS) + 2/3 * (note\ TP)$ avec éventuellement un bonus donné par des "devoirs maison".

Quelques exemples introductifs

GRAPHES ET CHEMINS

- Problème 1: Trouver le plus court chemin entre deux sommets d'un graphe

Graphes et Chemins

- Problème 1: Trouver le plus court chemin entre deux sommets d'un graphe
- Problème 2: Trouver le chemin le plus long sans cycle entre deux sommets dans un graphe

Graphes et Chemins

- Problème 1: Trouver le plus court chemin entre deux sommets d'un graphe
- Problème 2: Trouver le chemin le plus long sans cycle entre deux sommets dans un graphe

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

- Problème 1: Trouver le plus court chemin entre deux sommets d'un graphe

  C'est un grand classique:

- Problème 2: Trouver le chemin le plus long sans cycle entre deux sommets dans un graphe

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

- Problème 1: Trouver le plus court chemin entre deux sommets d'un graphe

  C'est un grand classique: algorithme de Dijkstra, algorithme de Ford Bellman, algorithme de Floyd-Warshall, ..
- Problème 2: Trouver le chemin le plus long sans cycle entre deux sommets dans un graphe

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

- Problème 1: Trouver le plus court chemin entre deux sommets d'un graphe

  C'est un grand classique: algorithme de Dijkstra, algorithme de Ford Bellman, algorithme de Floyd-Warshall, ..
- Problème 2: Trouver le chemin le plus long sans cycle entre deux sommets dans un graphe
  Si vous trouvez un algorithme "efficace", vous gagnez 1 million de dollars.

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

- Problème 1: Trouver le plus court chemin entre deux sommets d'un graphe

  C'est un grand classique: algorithme de Dijkstra, algorithme de Ford Bellman, algorithme de Floyd-Warshall, ..

- Problème 2: Trouver le chemin le plus long sans cycle entre deux sommets dans un graphe
  Si vous trouvez un algorithme "efficace", vous gagnez 1 million de dollars.

  Si vous montrez qu'on ne peut pas en trouver un, vous gagnez aussi 1 million de dollars!

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

Euler et Hamilton

- Problème 1: Trouver un chemin qui passe une et une seule fois par tous les sommets d'un graphe

Graphes et Chemins

Euler et Hamilton

- Problème 1: Trouver un chemin qui passe une et une seule fois par tous les sommets d'un graphe
- Problème 2: Trouver un chemin qui passe une et une seule fois par tous les arcs d'un graphe

Graphes et Chemins

Euler et Hamilton

- Problème 1: Trouver un chemin qui passe une et une seule fois par tous les sommets d'un graphe
- Problème 2: Trouver un chemin qui passe une et une seule fois par tous les arcs d'un graphe

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

Euler et Hamilton

- Problème 1: Trouver un chemin qui passe une et une seule fois par tous les sommets d'un graphe

  C'est la recherche d'un cycle hamiltonien: encore un problème "à un million de dollars"!

- Problème 2: Trouver un chemin qui passe une et une seule fois par tous les arcs d'un graphe

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

Euler et Hamilton

- Problème 1: Trouver un chemin qui passe une et une seule fois par tous les sommets d'un graphe

  C'est la recherche d'un cycle hamiltonien: encore un problème "à un million de dollars"!

- Problème 2: Trouver un chemin qui passe une et une seule fois par tous les arcs d'un graphe

  C'est la recherche d'un cycle eulerien:

Pour quel problème peut-on trouver un algorithme efficace?

Graphes et Chemins

Euler et Hamilton

- Problème 1: Trouver un chemin qui passe une et une seule fois par tous les sommets d'un graphe

  C'est la recherche d'un cycle hamiltonien: encore un problème "à un million de dollars"!

- Problème 2: Trouver un chemin qui passe une et une seule fois par tous les arcs d'un graphe

  C'est la recherche d'un cycle eulerien: Il faut et il suffit que le graphe soit connexe sans sommet de degré impair.

Pour quel problème peut-on trouver un algorithme efficace?

Groupes en paix

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il s'entend.

- Problème 1: répartir en deux groupes tels que les membres d'un même groupe s'entendent tous entre eux.

Groupes en paix

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il s'entend.

- Problème 1: répartir en deux groupes tels que les membres d'un même groupe s'entendent tous entre eux.
- Problème 2: répartir en trois groupes tels que les membres d'un même groupe s'entendent tous entre eux.

GROUPES EN PAIX

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il s'entend.

- Problème 1: répartir en deux groupes tels que les membres d'un même groupe s'entendent tous entre eux.
- Problème 2: répartir en trois groupes tels que les membres d'un même groupe s'entendent tous entre eux.

Le premier est "facile", le deuxième est encore un problème "dur" à un million de dollars...

BINÔMES OU TRINÔMES

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il veut bien faire équipe.

- Problème 1: Regrouper les étudiants en binômes qui s'entendent

BINÔMES OU TRINÔMES

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il veut bien faire équipe.

- Problème 1: Regrouper les étudiants en binômes qui s'entendent

- Problème 2: Regrouper les étudiants en trinômes qui s'entendent

Binômes ou trinômes

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il veut bien faire équipe.

- Problème 1: Regrouper les étudiants en binômes qui s'entendent

- Problème 2: Regrouper les étudiants en trinômes qui s'entendent

Le problème 1 a un algorithme efficace

BINÔMES OU TRINÔMES

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il veut bien faire équipe.

- Problème 1: Regrouper les étudiants en binômes qui s'entendent

- Problème 2: Regrouper les étudiants en trinômes qui s'entendent

Le problème 1 a un algorithme efficace -non trivial-, c'est un problème de "perfect matching" (algorithme de Edmonds).

BINÔMES OU TRINÔMES

Données: un ensemble d'étudiants. Chacun donne la liste de ceux avec qui il veut bien faire équipe.

- Problème 1: Regrouper les étudiants en binômes qui s'entendent
- Problème 2: Regrouper les étudiants en trinômes qui s'entendent

Le problème 1 a un algorithme efficace -non trivial-, c'est un problème de "perfect matching" (algorithme de Edmonds).

Le deuxième n'a pas d'algorithme efficace connu, c'est encore un problème à un million de dollars.

Un premier rappel: comment mesurer l'efficacité d'un algorithme

Complexité des algorithmes

Analyser la complexité d'un algorithme?

Analyser la complexité d'un algorithme doit permettre de mesurer l'efficacité de l'algorithme.

Analyser la complexité d'un algorithme?

Analyser la complexité d'un algorithme doit permettre de mesurer l'efficacité de l'algorithme.

On pourrait d'abord se poser la question:

Qu'est-ce qu'un algorithme efficace?

Qu'est-ce qu'un algorithme efficace?

Tentative de définition: Un algorithme est efficace si, une fois implémenté, il s'exécute rapidement sur toute donnée "réelle".

Qu'est-ce qu'un algorithme efficace?

Tentative de définition: Un algorithme est efficace si, une fois implémenté, il s'exécute rapidement sur toute donnée "réelle".

- Première remarque: attention à l'implémentation

Qu'est-ce qu'un algorithme efficace?

Tentative de définition: Un algorithme est efficace si, une fois implémenté, il s'exécute rapidement sur toute donnée "réelle".

- Première remarque: attention à l'implémentation
- Deuxième remarque: on devrait pouvoir mesurer l'efficacité indépendamment de la plate-forme

Qu'est-ce qu'un algorithme efficace?

Tentative de définition: Un algorithme est efficace si, une fois implémenté, il s'exécute rapidement sur toute donnée "réelle".

- Première remarque: attention à l'implémentation
- Deuxième remarque: on devrait pouvoir mesurer l'efficacité indépendamment de la plate-forme
- Troisième remarque: qu'est-ce qu'une donnée réelle? Souvent les problèmes d'efficacité surgissent quand la taille des données grandit...

Qu'est-ce qu'un algorithme efficace?

Tentative de définition: Un algorithme est efficace si, une fois implémenté, il s'exécute rapidement sur toute donnée "réelle".

- Première remarque: attention à l'implémentation
- Deuxième remarque: on devrait pouvoir mesurer l'efficacité indépendamment de la plate-forme
- Troisième remarque: qu'est-ce qu'une donnée réelle? Souvent les problèmes d'efficacité surgissent quand la taille des données grandit...
- Quatrième remarque: ne prend en compte que la ressource "temps".

Qu'est-ce qu'un algorithme efficace?

Deuxième Tentative de définition?
Un algorithme efficace est un algorithme qui se comporte de façon raisonnable indépendamment de la plate-forme même pour des données de taille assez grande...

QU'EST-CE QU'UN ALGORITHME EFFICACE?

Deuxième Tentative de définition?
Un algorithme efficace est un algorithme qui se comporte de façon raisonnable indépendamment de la plate-forme même pour des données de taille assez grande...

On proposera une définition un peu plus précise tout à l'heure!

Analyser la complexité d'un algorithme c'est:

Exprimer le coût de l'algorithme - la quantité de ressources utilisées - *en fonction de la taille des données*

Idée: on essaie de prévoir le comportement en fonction de la taille des données.

QUELLES RESSOURCES?

la mémoire: *complexité spatiale*

QUELLES RESSOURCES?

la mémoire: *complexité spatiale*

le temps : *complexité temporelle*

QUELLES RESSOURCES?

la mémoire: *complexité spatiale*

le temps : *complexité temporelle*

ou le nb de processeurs, les communications,....

QUELLES RESSOURCES?

la mémoire: *complexité spatiale*

le temps : *complexité temporelle*

ou le nb de processeurs, les communications,....

COMMENT MESURER LA TAILLE DES DONNÉES

. a priori la taille des données est **le nombre de bits de leur représentation en binaire**;

COMMENT MESURER LA TAILLE DES DONNÉES

. a priori la taille des données est **le nombre de bits de leur représentation en binaire**;

. même si dans certains cas, on étudie la complexité en fonction d'une autre notion de taille, comme le nombre d'éléments pour une liste, la dimension pour une matrice, . . .

COMMENT MESURER LA TAILLE DES DONNÉES

Q? Quelle est la taille d'un entier n?

Comment mesurer la taille des données

Q? Quelle est la taille d'un entier n?

Soit:
quel est le nombre de bits de la représentation en binaire d'un entier n?

COMMENT MESURER LA TAILLE DES DONNÉES

Q? Quelle est la taille d'un entier n?

Soit:
quel est le nombre de bits de la représentation en binaire d'un entier n?

environ $\log_2 n$: $\lceil \log_2 n + 1 \rceil$

Quel est le coût de l'algorithme A pour une donnée d - noté $cout_A(d)$-?

Quel est le coût de l'algorithme A pour une donnée d - noté $cout_A(d)$-?

. Pour évaluer ce coût indépendamment de la machine (physique) , on utilise un *modèle* de machines séquentielles "classiques", en général la RAM ("Random Access Machine").

QUEL EST LE COÛT DE L'ALGORITHME A POUR UNE DONNÉE d - NOTÉ $cout_A(d)$-?

. Pour évaluer ce coût indépendamment de la machine (physique) , on utilise un *modèle* de machines séquentielles "classiques", en général la RAM ("Random Access Machine").

. On considère sauf exception que chaque instruction "simple' ($+$, $*$, $-$, affectation, comparaison, accès mémoire ...) a un coût de 1: coût uniforme, i.e. indépendant de la taille de ses opérandes par opposition au coût logarithmique où on tient compte de la taille des données.

QUEL EST LE COÛT DE L'ALGORITHME A POUR UNE DONNÉE d - NOTÉ $cout_A(d)$-?

. Pour évaluer ce coût indépendamment de la machine (physique) , on utilise un *modèle* de machines séquentielles ”classiques”, en général la RAM (“Random Access Machine”).

. On considère sauf exception que chaque instruction “simple' ($+$, $*$, $-$, affectation, comparaison, accès mémoire ...) a un coût de 1: coût uniforme, i.e. indépendant de la taille de ses opérandes par opposition au coût logarithmique où on tient compte de la taille des données.

. On se borne souvent à ”compter” certaines instructions: comparaisons pour un tri par comparaisons, accès à la mémoire externe pour un tri externe, opérations arithmétiques de base pour des produits de matrices...

QUELLE COMPLEXITÉ?

Pour deux données de taille n, le coût peut être différent!

QUELLE COMPLEXITÉ?

Pour deux données de taille n, le coût peut être différent!

- *La complexité dans le meilleur des cas:*

$$Inf_A(n) = inf\{cout_A(d)/d \text{ de taille } n\}$$

QUELLE COMPLEXITÉ?

Pour deux données de taille n, le coût peut être différent!

- *La complexité dans le meilleur des cas:*

$$Inf_A(n) = inf\{cout_A(d)/d \text{ de taille } n\}$$

- *La complexité dans le pire des cas:*

$$Sup_A(n) = sup\{cout_A(d)/d \text{ de taille } n\}$$

Quelle complexité?

Pour deux données de taille n, le coût peut être différent!

- *La complexité dans le meilleur des cas:*

$$Inf_A(n) = inf\{cout_A(d)/d \text{ de taille } n\}$$

- *La complexité dans le pire des cas:*

$$Sup_A(n) = sup\{cout_A(d)/d \text{ de taille } n\}$$

- *La complexité en moyenne:* pour la définir, il faut disposer pour tout n d'une mesure de probabilité p sur l'ensemble des données de taille n;

$$Moy_A(n) = \sum_{d \text{ de taille } n} p(d) * cout(d)$$

QUELLE COMPLEXITÉ?

Pour deux données de taille n, le coût peut être différent!

- *La complexité dans le meilleur des cas:*

$$Inf_A(n) = inf\{cout_A(d)/d \text{ de taille } n\}$$

- *La complexité dans le pire des cas:*

$$Sup_A(n) = sup\{cout_A(d)/d \text{ de taille } n\}$$

- *La complexité en moyenne:* pour la définir, il faut disposer pour tout n d'une mesure de probabilité p sur l'ensemble des données de taille n;

$$Moy_A(n) = \sum_{d \text{ de taille } n} p(d) * cout(d)$$

EN RÉSUMÉ...

Quand on parle de la complexité d'un algorithme sans préciser laquelle, c'est souvent de la complexité temporelle dans le pire des cas qu'on parle.

Ordres de grandeur...

On ne calcule pas en général la complexité exacte mais on se contente de calculer son *ordre de grandeur asymptotique* voire de borner celui-ci.

Ordres de grandeur…

On ne calcule pas en général la complexité exacte mais on se contente de calculer son *ordre de grandeur asymptotique* voire de borner celui-ci.
Par exemple, si on fait $n^2 + 2n - 5$ opérations, on retiendra juste que l'ordre de grandeur est n^2. On utilisera donc les notations classiques sur les ordres de grandeur.

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in O(g)$ Ssi_{def} $\exists c \in \mathcal{R}^{+*}, \exists \mathcal{A}$, tels que
$\forall n > A, f(n) \leq c * g(n)$

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in O(g)$ Ssi_{def} $\exists c \in \mathcal{R}^{+*}, \exists \mathcal{A}$, tels que
$\forall n > A, f(n) \leq c * g(n)$

On dit que f est dominée asymptotiquement par g.
On notera souvent $f = O(g)$.

ORDRES DE GRANDEUR: DÉFINITIONS

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$$f \in \Omega(g)\ Ssi_{def}\ \exists C \in \mathcal{R}^{+*}, \text{ tels que}$$
$$\forall n > A, C * g(n) \leq f(n))$$

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in \Omega(g)$ Ssi_{def} $\exists C \in \mathcal{R}^{+*}$, tels que
$\forall n > A, C * g(n) \leq f(n))$

On notera souvent $f = \Omega(g)$.
On dit que f domine g.
On a alors $g = O(f)$.

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$$f \in \Theta(g)\ Ssi_{def}$$
$$\exists c, C \in \mathcal{R}^{+*}, \exists \mathcal{A}, \text{tels que}$$
$$\forall n > A, C * g(n) \leq f(n) \leq c * g(n)$$

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in \Theta(g)\ Ssi_{def}$
$\exists c, C \in \mathcal{R}^{+*}, \exists \mathcal{A}$,tels que
$\forall n > A, C * g(n) \leq f(n) \leq c * g(n)$

On dit alors que f et g sont de même ordre de grandeur asymptotique.
On notera souvent $f = \Theta(g)$.
On a $f = \Theta(g)$ Ssi $f = O(g)$ et $f = \Omega(g)$.

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in \Theta(g)\ Ssi_{def}$
$\exists c, C \in \mathcal{R}^{+*}, \exists \mathcal{A}$,tels que
$\forall n > A, C * g(n) \leq f(n) \leq c * g(n)$

On dit alors que f et g sont de même ordre de grandeur asymptotique.
On notera souvent $f = \Theta(g)$.
On a $f = \Theta(g)$ Ssi $f = O(g)$ et $f = \Omega(g)$.

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$$f \in o(g) \; Ssi_{def} \; \forall \epsilon \in \mathcal{R}^{+*}, \exists \mathcal{A}, \text{ tels que}$$
$$\forall n > A, f(n) \leq \epsilon * g(n)$$

Ordres de grandeur: définitions

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in o(g)\ Ssi_{def}\ \forall \epsilon \in \mathcal{R}^{+*}, \exists \mathcal{A}$, tels que
$\forall n > A, f(n) \leq \epsilon * g(n)$

On dit que f est négligeable asymptotiquement devant g.
On notera souvent $f = o(g)$

ORDRES DE GRANDEUR: EXEMPLES

$$.n^3 + 3n + 7 \in \Theta(n^3)$$

Ordres de grandeur: exemples

$$.n^3 + 3n + 7 \in \Theta(n^3)$$

$$.5 * n^3 + 3n + 7 \in \Theta(n^3)$$

Ordres de grandeur: exemples

$$.n^3 + 3n + 7 \in \Theta(n^3)$$

$$.5 * n^3 + 3n + 7 \in \Theta(n^3)$$

$$.5 * n^3 + 3n + 7 \in O(n^3)$$

Ordres de grandeur: exemples

$$.n^3 + 3n + 7 \in \Theta(n^3)$$

$$.5 * n^3 + 3n + 7 \in \Theta(n^3)$$

$$.5 * n^3 + 3n + 7 \in O(n^3)$$

$$.5 * n^3 + 3n + 7 \in O(n^4)$$

Ordres de grandeur: exemples

$.n^3 + 3n + 7 \in \Theta(n^3)$

$.5 * n^3 + 3n + 7 \in \Theta(n^3)$

$.5 * n^3 + 3n + 7 \in O(n^3)$

$.5 * n^3 + 3n + 7 \in O(n^4)$

$.\log n \in o(n)$

Ordres de grandeur: exemples

$$.n * \log n \in \Theta(n^2)?$$

ORDRES DE GRANDEUR: EXEMPLES

$.n * \log n \in \Theta(n^2)$? NON!

Ordres de grandeur: exemples

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$?

Ordres de grandeur: exemples

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$? Oui!

Ordres de grandeur: exemples

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$? Oui!

$.2^n \in \Theta(n^3)$?

Ordres de grandeur: exemples

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$? Oui!

$.2^n \in \Theta(n^3)$? NON!

Ordres de grandeur: exemples

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$? Oui!

$.2^n \in \Theta(n^3)$? NON!

$.n^3 \in \Theta(2^n)$?

Ordres de grandeur: exemples

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$? Oui!

$.2^n \in \Theta(n^3)$? NON!

$.n^3 \in \Theta(2^n)$? Non!

ORDRES DE GRANDEUR: EXEMPLES

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$? Oui!

$.2^n \in \Theta(n^3)$? NON!

$.n^3 \in \Theta(2^n)$? Non!

$.n^3 \in O(2^n)$?

ORDRES DE GRANDEUR: EXEMPLES

$.n * \log n \in \Theta(n^2)$? NON!

$.n * \log n \in O(n^2)$? Oui!

$.2^n \in \Theta(n^3)$? NON!

$.n^3 \in \Theta(2^n)$? Non!

$.n^3 \in O(2^n)$? Oui!

Remarque

Evaluer l'ordre de grandeur asymptotique du coût de l'algorithme en fonction de la taille des données plutôt que la complexité exacte se justifie si les données manipulées sont de grande taille.
Il ne faut pas négliger trop vite les constantes .

REMARQUE

Evaluer l'ordre de grandeur asymptotique du coût de l'algorithme en fonction de la taille des données plutôt que la complexité exacte se justifie si les données manipulées sont de grande taille.
Il ne faut pas négliger trop vite les constantes .

Q?: à partir de quelle taille de données, un algorithme de complexité exacte $200 * n * \log_2 n$ sera-t-il plus intéressant qu'un algorithme en n^2 ?

UN PEU DE VOCABULAIRE: UN ALGORITHME EST DIT...

UN PEU DE VOCABULAIRE: UN ALGORITHME EST DIT...

. en temps **constant** si sa complexité dans le pire des cas est bornée par une constante.

UN PEU DE VOCABULAIRE: UN ALGORITHME EST DIT...

. en temps **constant** si sa complexité dans le pire des cas est bornée par une constante.

. **linéaire** (resp. linéairement borné) si sa complexité (dans le pire des cas) est en $\Theta(n)$ (resp. $O(n)$).

UN PEU DE VOCABULAIRE: UN ALGORITHME EST DIT...

. en temps **constant** si sa complexité dans le pire des cas est bornée par une constante.

. **linéaire** (resp. linéairement borné) si sa complexité (dans le pire des cas) est en $\Theta(n)$ (resp. $O(n)$).

. **quadratique** (resp. au plus quadratique) si sa complexité (dans le pire des cas) est en $\Theta(n^2)$ (resp. en $O(n^2)$).

Un peu de vocabulaire: un algorithme est dit…

. en temps **constant** si sa complexité dans le pire des cas est bornée par une constante.

. **linéaire** (resp. linéairement borné) si sa complexité (dans le pire des cas) est en $\Theta(n)$ (resp. $O(n)$).

. **quadratique** (resp. au plus quadratique) si sa complexité (dans le pire des cas) est en $\Theta(n^2)$ (resp. en $O(n^2)$).

. **polynomial** ou polynomialement borné, si sa complexité (dans le pire des cas) est en $O(n^p)$ pour un certain p.

UN PEU DE VOCABULAIRE: UN ALGORITHME EST DIT...

. en temps **constant** si sa complexité dans le pire des cas est bornée par une constante.

. **linéaire** (resp. linéairement borné) si sa complexité (dans le pire des cas) est en $\Theta(n)$ (resp. $O(n)$).

. **quadratique** (resp. au plus quadratique) si sa complexité (dans le pire des cas) est en $\Theta(n^2)$ (resp. en $O(n^2)$).

. **polynomial** ou polynomialement borné, si sa complexité (dans le pire des cas) est en $O(n^p)$ pour un certain p.

. (au plus) **exponentiel** si elle est en $O(2^{n^p})$, pour un certain $p > 0$.

ETRE PRATICABLE OU NE PAS ÊTRE PRATICABLE?...

TELLE EST LA QUESTION QU'ON S EXPOSERA SOUVENT

Par “convention”, un algorithme est dit **praticable** si il est polynomial, c.à.d. si sa complexité temporelle dans le pire des cas est polynomiale.

Les limites de la convention

LES LIMITES DE LA CONVENTION

- Si un algorithme est exponentiel dans le pire des cas, il peut être polynomial en moyenne; si les pires cas sont exceptionnels, il peut être praticable ...en pratique: certains algorithmes impraticables selon la définition ci-dessus sont ... pratiqués tous les jours (comme le simplexe).

Les limites de la convention

- Si un algorithme est exponentiel dans le pire des cas, il peut être polynomial en moyenne; si les pires cas sont exceptionnels, il peut être praticable ...en pratique: certains algorithmes impraticables selon la définition ci-dessus sont ... pratiqués tous les jours (comme le simplexe).

-L'exécution d'un algorithme dont la complexité moyenne est de l'ordre de grandeur de n^5 microsecondes prendrait 30 ans si $n = 1000$.

Mais ce n'est pas une si mauvaise convention:

Mais ce n'est pas une si mauvaise convention:

+ En pratique, il s'avère que la plupart des algorithmes polynomiaux ont un comportement asymptotique équivalent à un polynôme de faible degré, 2 ou 3.

Mais ce n'est pas une si mauvaise convention:

+ En pratique, il s'avère que la plupart des algorithmes polynomiaux ont un comportement asymptotique équivalent à un polynôme de faible degré, 2 ou 3.

+ De plus, la classe des algorithmes polynomiaux a de bonnes propriétés de clôture (par exemple, une "séquence" de deux algorithmes polynomiaux est polynomiale).

Mais ce n'est pas une si mauvaise convention:

+ En pratique, il s'avère que la plupart des algorithmes polynomiaux ont un comportement asymptotique équivalent à un polynôme de faible degré, 2 ou 3.

+ De plus, la classe des algorithmes polynomiaux a de bonnes propriétés de clôture (par exemple, une "séquence" de deux algorithmes polynomiaux est polynomiale).

+ Enfin, et surtout, elle est indépendante du modèle séquentiel "classique" choisi: un algorithme polynômial pour le modèle des machines de Turing, le sera pour une RAM et vice-versa .

Ordres de grandeur: exemples

Exemples de temps d'exécution en fonction de la taille de la donnée et de la complexité de l'algorithme, si on suppose qu'une instruction est de l'ordre de la μs;

T.\C.	$\log n$	n	$n \log n$	n^2	2^n
10	$3\mu s$	$10\mu s$	$30\mu s$	$100\mu s$	$1000\mu s$
100	$7\mu s$	$100\mu s$	$700\mu s$	$1/100s$	10^{14} *siècles*
1000	$10\mu s$	$1000\mu s$	$1/100s$	$1s$	*astronomique*
10000	$13\mu s$	$1/100s$	$1/7s$	$1,7mn$	*astronomique*
100000	$17\mu s$	$1/10s$	$2s$	$2,8h$	*astronomique*

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 int max= t[0];
  for (i=1; i<=n-1; i++)
     if (max <t[i]) max=t[i];
```

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 int max= t[0];
  for (i=1; i<=n-1; i++)
     if (max <t[i]) max=t[i];
```

Algorithme en $\Theta(n)$, donc linéaire.

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
  for (i=0; i<n-1; i++)
     for (j=0; j<n-1-i; j++)
       if (t[j+1] <t[j]) echanger(t[j],t[j+1]);
```

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
  for (i=0; i<n-1; i++)
     for (j=0; j<n-1-i; j++)
       if (t[j+1] <t[j]) echanger(t[j],t[j+1]);
```

Algorithme en $\Theta(n^2)$, donc quadratique.

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 boolean permute=true;
 int last=n-1;
  while permute {
     permute=false;
     for (j=0; j<last; j++)
       if (t[j+1] <t[j])
          {permuter=true; echanger(t[j],t[j+1]);
     last=last-1;}
}
```

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 boolean permute=true;
 int last=n-1;
  while permute {
     permute=false;
     for (j=0; j<last; j++)
       if (t[j+1] <t[j])
          {permuter=true; echanger(t[j],t[j+1]);
     last=last-1;}
}
```

dans le meilleur des cas en $\Theta(n)$

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 boolean permute=true;
 int last=n-1;
  while permute {
     permute=false;
     for (j=0; j<last; j++)
       if (t[j+1] <t[j])
         {permute=true; echanger(t[j],t[j+1]);
     last=last-1;}
}
```

dans le meilleur des cas en $\Theta(n)$
dans le pire des cas en $\Theta(n^2)$
Algorithme en $\Theta(n^2)$, donc quadratique.

QUELQUES EXEMPLES: MULTIPLICATION À LA RUSSE ... OU À L'ÉGYPTIENNE

Quelle est la complexité de:

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

Quelques exemples: multiplication à la russe ... ou à l'égyptienne

Quelle est la complexité de:

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

Le nombre d'opérations élémentaires est $\Theta(log_2(y))$.

QUELQUES EXEMPLES: MULTIPLICATION À LA RUSSE ... OU À L'ÉGYPTIENNE

Quelle est la complexité de:

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
    if ( b %2 =1) R=R+a;
    a=2*a;
    b=  b/2; }
\\ R= a*b
```

Le nombre d'opérations élémentaires est $\Theta(log_2(y))$.

L'algorithme est linéaire en coût uniforme, quadratique en coût logarithmique.....

Quelques exemples:

Soit l'algorithme suivant pour tester si un nombre est premier:

```
//n entier >1
boolean Premier(int n){
  int j =sqrt(n); \\racine carrée entière
  for (int i = 2; i<=j; i++)
    if (n mod i=0)  return false;
  return true;
}
```

Attention à la taille de la donnée!

```
//n entier >1
boolean Premier(int n){
  int j =sqrt(n); \\racine carree entiere
  for (int i = 2; i<=j; i++)
    if (n mod i=0)  return false;
  return true;}
```

Cet algorithme est-il polynomial? Non!!

Attention à la taille de la donnée!

```
//n entier >1
boolean Premier(int n){
  int rac =sqrt(n); \\racine carrée entière
  for (int i = 2; i<=rac; i++)
    if (n mod i=0)  return false;
  return true;
}
```

supposons que le coût d'une division soit constant, indépendant de la taille de la donnée, donc en $\Theta(1)$, alors la complexité dans le pire des cas de l'algo est $\Theta(\sqrt{(}n))$..

ATTENTION À LA TAILLE DE LA DONNÉE!

```
//n entier >1
boolean Premier(int n){
  int rac =sqrt(n); \\racine carrée entière
  for (int i = 2; i<=rac; i++)
    if (n mod i=0)  return false;
  return true;
}
```

supposons que le coût d'une division soit constant, indépendant de la taille de la donnée, donc en $\Theta(1)$, alors la complexité dans le pire des cas de l'algo est $\Theta(\sqrt(n))$..

mais la taille de la donnée $|d|$ est $\log_2 n$

donc le coût est en $\Theta(2^{|d|/2})$: l'algorithme est exponentiel.

A RETENIR SUR LA COMPLEXITÉ DES ALGORITHMES:

. Quand on parle de la Complexité c'est souvent la complexité temporelle dans le pire des cas

A RETENIR SUR LA COMPLEXITÉ DES ALGORITHMES:

. Quand on parle de la Complexité c'est souvent la complexité temporelle dans le pire des cas

. Praticable=polynomial

A retenir sur la complexité des algorithmes:

. Quand on parle de la Complexité c'est souvent la complexité temporelle dans le pire des cas

. Praticable=polynomial

. Evaluer l'ordre de grandeur de la complexité d'un algorithme ... n'exempte pas de tester et d'expérimenter

A RETENIR SUR LA COMPLEXITÉ DES ALGORITHMES:

. Quand on parle de la Complexité c'est souvent la complexité temporelle dans le pire des cas

. Praticable=polynomial

. Evaluer l'ordre de grandeur de la complexité d'un algorithme ... n'exempte pas de tester et d'expérimenter

. Attention à la taille de la donnée!

Un deuxième rappel: comment prouver qu'un algorithme est correct

Prouver qu'un programme est correct???

PROUVER QU'UN PROGRAMME EST CORRECT???

"Program testing can be used to show the presence of bugs, but never to show their absence !" Edsger W. Dijkstra

PROUVER QU'UN PROGRAMME EST CORRECT???

"Program testing can be used to show the presence of bugs, but never to show their absence !" Edsger W. Dijkstra

"One does not need to give a formal proof of an obviously correct program; but one needs a thorough understanding of formal proof methods to know when correctness is obvious." John C. Reynolds

Prouver la correction d'un programme,

PROUVER LA CORRECTION D'UN PROGRAMME,

c'est prouver qu'il correspond à la spécification donnée.

PROUVER LA CORRECTION D'UN PROGRAMME,

c'est prouver qu'il correspond à la spécification donnée.

La Logique de Hoare-Floyd a été introduite à la fin des années 60 pour formaliser la relation entre langage de programmation (impératif) et langage de spécification.

PROUVER LA CORRECTION D'UN PROGRAMME,

c'est prouver qu'il correspond à la spécification donnée.

La Logique de Hoare-Floyd a été introduite à la fin des années 60 pour formaliser la relation entre langage de programmation (impératif) et langage de spécification.

Elle est basée sur la notion de *triplet de Hoare*, de la forme $\{P\}$ `A` $\{Q\}$;

PROUVER LA CORRECTION D'UN PROGRAMME,

c'est prouver qu'il correspond à la spécification donnée.

La Logique de Hoare-Floyd a été introduite à la fin des années 60 pour formaliser la relation entre langage de programmation (impératif) et langage de spécification.

Elle est basée sur la notion de *triplet de Hoare*, de la forme $\{P\}$ `A` $\{Q\}$;
P est la précondition,

Prouver la correction d'un programme,

c'est prouver qu'il correspond à la spécification donnée.

La Logique de Hoare-Floyd a été introduite à la fin des années 60 pour formaliser la relation entre langage de programmation (impératif) et langage de spécification.

Elle est basée sur la notion de *triplet de Hoare*, de la forme $\{P\}$ `A` $\{Q\}$;

P est la précondition,

Q la postcondition,

PROUVER LA CORRECTION D'UN PROGRAMME,

c'est prouver qu'il correspond à la spécification donnée.

La Logique de Hoare-Floyd a été introduite à la fin des années 60 pour formaliser la relation entre langage de programmation (impératif) et langage de spécification.

Elle est basée sur la notion de *triplet de Hoare*, de la forme $\{P\}$ `A` $\{Q\}$;

P est la précondition,

Q la postcondition,

A étant une expression du langage de programmation.

Spécification

$$\{P\}\ \mathtt{A}\ \{Q\};$$

SPÉCIFICATION

$$\{P\}\ \texttt{A}\ \{Q\};$$

P et Q sont des assertions, i.e. des formules du langage de spécification décrivant les valeurs des variables, l'état de la mémoire ou du système.

SPÉCIFICATION

$$\{P\}\ \mathtt{A}\ \{Q\};$$

P et Q sont des assertions, i.e. des formules du langage de spécification décrivant les valeurs des variables, l'état de la mémoire ou du système.

En toute rigueur, le langage de spécification devrait être un langage mathématique comme la logique des prédicats, mais dans un objectif de documentation plus que de preuve de programmes, le langage de spécification peut être le langage naturel (en évitant les ambiguités!).

INTERPRÉTATION:

L'interprétation de $\{P\}$ `A` $\{Q\}$ est:

INTERPRÉTATION:

L'interprétation de $\{P\}$ `A` $\{Q\}$ est:

si le système vérifie $\{P\}$, on peut assurer qu'il vérifie $\{Q\}$ après l'exécution de `A`;

INTERPRÉTATION:

L'interprétation de $\{P\}$ `A` $\{Q\}$ est:

si le système vérifie $\{P\}$, on peut assurer qu'il vérifie $\{Q\}$ après l'exécution de `A`;

en fait, c'est un peu plus compliqué que cela; en effet, il se peut que l'exécution de `A` ne termine pas! On a donc deux interprétations possibles:

INTERPRÉTATION:

L'interprétation de $\{P\}$ `A` $\{Q\}$ est:

si le système vérifie $\{P\}$, on peut assurer qu'il vérifie $\{Q\}$ après l'exécution de `A`;

en fait, c'est un peu plus compliqué que cela; en effet, il se peut que l'exécution de `A` ne termine pas! On a donc deux interprétations possibles:

si $\{P\}$ est vérifiée, après l'exécution de `A`, si celle-ci termine, on peut assurer $\{Q\}$;

INTERPRÉTATION:

L'interprétation de $\{P\}$ `A` $\{Q\}$ est:

si le système vérifie $\{P\}$, on peut assurer qu'il vérifie $\{Q\}$ après l'exécution de `A`;

en fait, c'est un peu plus compliqué que cela; en effet, il se peut que l'exécution de `A` ne termine pas! On a donc deux interprétations possibles:

si $\{P\}$ est vérifiée, après l'exécution de `A`, si celle-ci termine, on peut assurer $\{Q\}$;

si $\{P\}$ est vérifiée, la terminaison de `A` est assurée et après son exécution, on peut assurer $\{Q\}$;

INTERPRÉTATION:

L'interprétation de $\{P\}$ `A` $\{Q\}$ est:

si le système vérifie $\{P\}$, on peut assurer qu'il vérifie $\{Q\}$ après l'exécution de `A`;

en fait, c'est un peu plus compliqué que cela; en effet, il se peut que l'exécution de `A` ne termine pas! On a donc deux interprétations possibles:

si $\{P\}$ est vérifiée, après l'exécution de `A`, si celle-ci termine, on peut assurer $\{Q\}$;

si $\{P\}$ est vérifiée, la terminaison de `A` est assurée et après son exécution, on peut assurer $\{Q\}$;

Dans le premier cas, on parle de correction partielle.

Prouver un programme revient donc à prouver:

Prouver un programme revient donc à prouver:

{Précondition sur les données} `Programme` {condition exprimant les résultats attendus}

Prouver un programme revient donc à prouver:

{Précondition sur les données} `Programme` {condition exprimant les résultats attendus}

à partir des axiomes et des règles d'inférences de Hoare qui montrent comment inférer la correction d'instructions composées à partir de celles de base.

La règle pour la Séquence

$$\{P\}\ \texttt{A ; B}\ \{Q\}?$$

La règle pour la Séquence

$$\{P\}\ \texttt{A ; B}\ \{Q\}?$$

si

$$\{P\}\ \texttt{A}\ \{Intermede\}$$

La règle pour la Séquence

$$\{P\}\ \texttt{A ; B}\ \{Q\}?$$

si

$$\{P\}\ \texttt{A}\ \{Intermede\} \text{ et } \{Intermede\}\ \texttt{B}\ \{Q\}$$

alors:

La règle pour la Séquence

$$\{P\}\ \texttt{A ; B}\ \{Q\}?$$

si

$$\{P\}\ \texttt{A}\ \{Intermede\}\ \text{ et }\ \{Intermede\}\ \texttt{B}\ \{Q\}$$

alors:

$$\{P\}\ \texttt{A ; B}\ \{Q\}$$

La règle de la conditionnelle

$\{P\}$ `si cond alors A sinon B` $\{Q\}$?

La règle de la conditionnelle

$$\{P\}\ \texttt{si cond alors A sinon B}\ \ \{Q\}\,?$$

si

$$\{P \wedge cond\}\ \texttt{A}\ \ \{Q\}\ \text{et}\ \{P \wedge \neg cond\}\ \texttt{B}\ \ \{Q\}$$

alors:

La règle de la conditionnelle

$$\{P\}\ \texttt{si cond alors A sinon B}\ \ \{Q\}\ ?$$

si

$$\{P \wedge cond\}\ \texttt{A}\ \ \{Q\}\ \text{et}\ \{P \wedge \neg cond\}\ \texttt{B}\ \ \{Q\}$$

alors:

$$\{P\}\ \texttt{si cond alors A sinon B}\ \ \{Q\}$$

La règle pour le "tant que'

$\{P\}$ `tant que cond faire A` $\{Q\}$??

LA RÈGLE POUR LE "TANT QUE'

$$\{P\}\ \texttt{tant que cond faire A}\ \ \{Q\}??$$

si on trouve une assertion i, qu'on appelle *invariant*, telle que:
$P \Rightarrow i$

La règle pour le "tant que'

$$\{P\}\ \texttt{tant que cond faire A}\ \ \{Q\}??$$

si on trouve une assertion i, qu'on appelle *invariant*, telle que:
$P \Rightarrow i$

$\{i \wedge cond\}\ \texttt{A}\ \ \{i\}$

LA RÈGLE POUR LE "TANT QUE'

$$\{P\}\ \texttt{tant que cond faire A}\ \ \{Q\}??$$

si on trouve une assertion i, qu'on appelle *invariant*, telle que:

$P \Rightarrow i$

$\{i \wedge cond\}\ \texttt{A}\ \ \{i\}$

$\{i \wedge \neg cond\} \Rightarrow \{Q\}$

LA RÈGLE POUR LE "TANT QUE'

$$\{P\}\ \texttt{tant que cond faire A}\ \ \{Q\}??$$

si on trouve une assertion i, qu'on appelle *invariant*, telle que:
$P \Rightarrow i$

$\{i \wedge cond\}$ $\texttt{A}$ $\{i\}$

$\{i \wedge \neg cond\} \Rightarrow \{Q\}$

alors:

LA RÈGLE POUR LE "TANT QUE'

$$\{P\}\ \texttt{tant que cond faire A}\ \ \{Q\}??$$

si on trouve une assertion i, qu'on appelle *invariant*, telle que:
$P \Rightarrow i$

$\{i \wedge cond\}$ `A` $\{i\}$

$\{i \wedge \neg cond\} \Rightarrow \{Q\}$

alors:

$\{P\}$ `tant que cond faire A` $\{Q\}$

La règle pour le "tant que'

$$\{P\}\,\texttt{tant que cond faire A}\;\;\{Q\}??$$

si on trouve une assertion i, qu'on appelle *invariant*, telle que:
$P \Rightarrow i$

$\{i \wedge cond\}\,\texttt{A}\;\;\{i\}$

$\{i \wedge \neg cond\} \Rightarrow \{Q\}$

alors:

$\{P\}\,\texttt{tant que cond faire A}\;\;\{Q\}$

Attention: pour avoir une preuve de correction totale, il faut aussi faire une preuve d'arrêt de la boucle!

L'AFFECTATION

$$P[x \leftarrow exp]\ \texttt{x:=exp}\quad P$$

L'AFFECTATION

$P[x \leftarrow exp]$ `x:=exp` P

en supposant que exp ne contient pas de fonctions à effet de bord.

L'AFFECTATION

$P[x \leftarrow exp]$ `x:=exp` P

en supposant que exp ne contient pas de fonctions à effet de bord.

Par exemple, $x + y < 5$ `x:=x+y` $x < 5$

Le renforcement de la précondition

si

$$P \Rightarrow Q$$

et

$$\{Q\}\ \mathtt{A}\ \{R\}$$

alors

Le renforcement de la précondition

si

$$P \Rightarrow Q$$

et

$$\{Q\}\ \mathtt{A}\ \{R\}$$

alors

$$\{P\}\ \mathtt{A}\ \{R\}$$

L'AFFAIBLISSEMENT DE LA POSTCONDITION

si

$\{P\}$ `A` $\{Q\}$

et

$Q \Rightarrow R$

alors

L'AFFAIBLISSEMENT DE LA POSTCONDITION

si

$$\{P\}\ \mathtt{A}\ \ \{Q\}$$

et

$$Q \Rightarrow R$$

alors

$$\{P\}\ \mathtt{A}\ \ \{R\}$$

L'AFFAIBLISSEMENT DE LA POSTCONDITION

si

$\{P\}\ \mathtt{A}\ \ \{Q\}$

et

$Q \Rightarrow R$

alors

$\{P\}\ \mathtt{A}\ \ \{R\}$

Soit: Qui peut le plus, peut le moins!

UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
    if ( b %2 =1) R=R+a;
    a=2*a;
    b=  b/2; }
\\ R= a*b?
```

UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
\\ R=0, a=x, b=y
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b?
```

UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
\\ R=0, a=x, b=y
while (b>0){\\invariant R+a*b=x*y
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

REMARQUE: VERSION RÉCURSIVE

```
\\ x >=y >=0
int Mult (int x, int y)
 if y==0 retrun 0
else return  Mult(2*x, y / 2) +x* y % 2;
```

Preuve par induction sur y!

- Pas de TD/TP cette semaine
- Semaine prochaine:
  - Cours et TD pas de TP
  - Attention! le groupe 5 aura deux TDs un au créneau habituel, l'autre au créneau de TP.
- Semaine prochaine: Pas de cours d'amphi le 24 septembre