Algorithmique Appliquée

BTS SIO SISR

Conclusion et révisions

Yvo Solutions
Loïc Yvonnet

Plan

- Retour sur les points essentiels
- Conseils pour l'examen
- Questions/réponses
- Travail dans une base de code réelle
- Recherche opérationnelle

Retours sur les points essentiels et attendus pour l'examen

Expressions et variables

- Variables.
- Assignation.
- Expressions.
- Opérateurs.

Structures de contrôle

- Conditions (`if`).
- Boucles (`while` et `for`).

Fonctions

- Définition de fonctions.
- Appel de fonctions.
- Retour de fonctions.

Structures de données

- `list` : liste/tableau.
- `dict` : dictionnaire de données.
- `str` : chaîne de caractères.
- `class` : vos propres structures de données.

Complexité

- Notation Landau $O(\ldots)$.
- Compter les boucles et évaluer la complexité.
- Reconnaître une complexité logarithmique.

Recherche et tri

- Dichotomie.
- Recherche binaire.
- Arbre de recherche binaire.
- Connaître au moins un algorithme de tri (idéalement en $O(N \log N)$).

Théorie des graphes

- Représentation d'un graphe (matrice d'adjacence ou liste de listes).
- Parcours d'un graphe (en profondeur et largeur).
- Identification d'un cycle.
- Plus court chemin.
- Chemin critique.

Critères d'évaluation (1/2)

Les compétences attendues sont évaluées notamment sur la base des critères suivants :

- **Maîtrise des connaissances** liées au module d'Algorithmique Appliquée.
- **Efficacité** et **pertinence** de la solution proposée.
- **Qualité de la mise œuvre**, notamment la lisibilité, l'indentation, et les commentaires.

Critères d'évaluation (2/2)

Autres critères :

- Pertinence de l'utilisation des **composants logiciels disponibles**.
- Adéquation des **tests de validation** effectués.
- Aptitude à proposer des **éléments de correction pertinants**.

Conseils pour l'examen

Conseils (1/4)

- Lisez **très attentivement** l'énoncé jusqu'au bout.
- Identifiez **quelles parties du cours** sont abordées : graphe, arbre binaire, tri, complexité, etc.
- Utilisez un **brouillon**.
- Prenez un **exemple** : déroulez l'approche avec cet exemple pour arriver au résultat souhaité.

Conseils (2/4)

- Faites une **ébauche d'algorithme** au brouillon.
- **Exécutez manuellement** votre algorithme sur votre exemple.
- Notez les valeurs de vos variables à chaque itération dans **un tableau**.
- **Corrigez** si nécessaire votre algorithme suite à vos observations pendant l'exécution.

Conseils (3/4)

- Utilisez des **noms de variable pertinents** et sémantiquement riches.
- **Commentez** chaque fonction avec une docstring.
- Commentez votre code.

Conseils (4/4)

- Ajoutez au moins un **test unitaire** par fonction.
- A minima, si vous manquez de temps, indiquez que vous ajouteriez des tests unitaires dans une base de code industrielle.
- Enfin, **reportez au propre votre solution**.

Sujets récurrents

- Multiplication matricielle.
- Utilisation de la dichotomie.
- Parcours d'un graphe (en profondeur ou en largeur).
- Tri d'une collection (liste ou chaîne de caractères).
- Plus court chemin.
- Chemin critique.

Questions/Réponses

Sur l'ensemble du cours

Des questions ?

Travail dans une base de code réelle

Un million de lignes de code

- Les projets de plus d'un million de lignes de code **ne sont pas rares**.
- On a souvent une **compréhension partielle** d'une base de code.
- On utilise des outils dédiés pour **naviguer dans le code**.
- On documente l'**architecture logicielle** dans des formalismes tels que **UML** ou **Archimate**.

Base de code "legacy"

- De nombreuses bases de code n'ont pas du tout, ou **très peu de tests automatiques**.
- Les bases de code de produits qui ont du succès ne sont **pas toujours écrites par des informaticiens**.
- Il existe des techniques pour *rentrer* dans une base de code.
- Ces techniques sont basés sur la **rétro-ingénierie**.

Amélioration continue

- Chaque modification de code doit **améliorer** la base de code.
- La **qualité logicielle** nécessite de l'attention et de la discipline.
- Il est toujours possible d'ajouter des tests automatiques, même dans une base de code qui n'en comporte aucun.
- Il faut avoir un **tableau de bord** et des **indicateurs de performance clés**.

Quelques outils essentiels

- Gestionnaire de contrôle de version (ex: `git`).
- Environnement de développement intégré (ex : VS Code).
- Outil d'intégration continue (ex : Jenkins).
- Outil de déploiement continu (ex : Bamboo).

Processus classique (1/2)

- **Collection des besoins** : une enquête auprès des utilisateurs finaux permet de collecter les besoin.
- **Spécifications fonctionnelles** : on traduit les besoins en un ensemble de fonctionnalités.
- **Besoins non-fonctionnelles** : certains besoins sont transversaux (ex : performance).
- **Cahier des charges** : on écrit un document formel qui ressemblent les spécifications fonctionnelles et non-fonctionnnelles.

Processus classique (2/2)

- **Décompositions** : on décompose les fonctionnalités en ensemble de fonctions élémentaires que l'on peut programmer.
- **Plan de tests** : on établi un plan pour vérifier et valider le bon fonctionnement de chaque fonctionnalité.
- **Implémentation** : implémentation du code.
- **Tests** : déroulement des tests.
- **Déploiement** : mise en production de la solution.

Agile

- Des méthodologies comme **SCRUM** ou **XP** peuvent booster certaines typologies de projets.
- On rassemble les différents métiers au sein d'une **équipe projet**.
- On effectue des **itérations courtes** (nommée "sprints") pour converger progressivement vers une solution qui convient à tous.
- On suit de **bonnes pratiques** et on favorise un environnement de travail bienveillant.

Ne pas refaire la roue

- Connaître les algorithmes est une excellente chose.
- Il vaut mieux chercher une bibliothèque Open Source plutôt que de toujours tout réimplémenter.

Performance ou clarté du code ?

- On privilégie par défaut la **clarté du code**.
- Il faut garder à l'esprit le **ratio 80/20**.
- Seul 20% du code doit être optimisé.

Recherche opérationnelle

Ce n'est qu'un début...

- Flot maximal, recouvrement, etc.
- Algorithmes spécialisés dans les chaînes de caractères.
- Data Mining.
- Compilateurs, Moteurs d'inférences.
- Solveurs d'équations numériques ou formels.
- Modeleur 3D (lancé de rayon, etc.) ou géométrique.
- Réseaux de neurones (profonds, convolutionnels, etc.).
- Etc.

Merci