Révision pour l'examen final

Claude-Guy Quimper

Analyse d'algorithmes

$$C_{best}(n) = \min_{x:|x|=n} C(x)$$

$$C_{worst}(n) = \max_{x:|x|=n} C(x)$$

$$C_{avg}(n) = \mathbb{E}_{x:|x|=n} C(x) = \sum_{x:|x|=n} P_n(x) C(x)$$

Pas à l'examen

$$\mathcal{C}_{best}(n) = \min_{x:|x|=n} \mathbb{E}_\rho C(x,\rho) = \min_{x:|x|=n} \sum_\rho p(\rho) C(x,\rho)$$

$$\mathcal{C}_{worst}(n) = \max_{x:|x|=n} \mathbb{E}_\rho C(x,\rho) = \max_{x:|x|=n} \sum_\rho p(\rho) C(x,\rho)$$

$$\mathcal{C}_{avg}(n) = \mathbb{E}_{x:|x|=n} \mathbb{E}_\rho C(x,\rho) = \sum_{x:|x|=n} P_n(x) \sum_\rho p(\rho) C(x,\rho)$$

- Vous devrez analyser vos algorithmes.
- ~~Analyse amortie~~

Chapitre 6
Transformer pour régner

- Le pré triage et les monceaux sont très utiles pour la conception d'algorithmes voraces (Chapitre 9)
- Sachez utiliser le pré triage.
  - Quel algorithme de tri utiliser?
- Monceau
  - Sachez vous servir d'un monceau.
  - Sachez la complexité de chaque algorithme qui manipule un monceau.

Chapitre 6.5

- L’analyse amortie
- Pour l’examen, il est suffisant de savoir ce qu’est l’analyse amortie.
- Aucune analyse amortie ne sera demandée.

Chapitre 7
Compromis espace-temps

- Mettre en cache les résultats d'une fonction
- Tri par dénombrement
  - Fonctionne seulement avec des entiers
  - Efficacité: $\Theta(\max\{u-l, n\})$

Chapitre 8
Programmation dynamique

- Comment résoudre un problème à l'aide de la programmation dynamique?
- 1) Identifier le sous-problème
  - Imaginez que vous codez votre algorithme en C++. Quels sont les paramètres de votre fonction?
  - Pour chaque paramètre qui est une liste, considérez seulement les i premiers éléments de la liste.
  - Pour chaque paramètre qui est un entier n, considérez toutes les valeurs entre 0 et n.

Chapitre 8
Programmation dynamique

- 2) Définissez les dimensions du tableau
  - Une dimension par paramètre défini en 1
  - Pour chaque dimension, déterminez la longueur du tableau sur cette dimension.
- 3) Définissez le contenu du tableau
  - Le contenu est toujours la valeur objectif de la solution et rarement (voire jamais) la solution elle-même.
- 4) Récurrence
  - Commencez par les cas de base, ce sont les plus faciles.
  - Si le problème vous demande de faire des choix, calculez le coût de chacun de ces choix. Prenez la valeur du choix qui maximise ou qui minimise la valeur objective.

Chapitre 8
Programmation dynamique

- 5) Écrivez le pseudo-code
  - Vous n'avez qu'à remplir le tableau en suivant la récurrence définie au point 4.
- 6) Trouvez la solution
  - Le tableau vous donne la valeur objective de la solution optimale.
  - Il faut remonter le tableau pour découvrir comment cette valeur a été calculée.
- 7) Analysez l'algorithme
  - Comptez le nombre de cellules et multipliez par le temps requis pour le calcul d'une seule cellule.

Chapitre 8

- Exemples que nous avons vus:
  - Coefficients binomiaux
  - Problème du sac à dos
  - Distance d'édition
  - Rendre la monnaie
  - Joutes de sport
  - La compression d'une image
  - Coupe d'une planche
  - La question du devoir

Chapitre 9: Les algorithmes voraces

- Ces algorithmes construisent une solution par ajouts successifs de composantes.
- Le choix vorace a trois caractéristiques:
  - Il est réalisable
  - Il est optimal
  - Il est irrévocable

Utilisation du monceau

- L'algorithme de Prim ajoute dans un monceau tous les nœuds constituant des choix réalisables.
  - En extrayant la racine, Prim choisit le nœud optimal à ajouter à l'arbre de recouvrement.

Structure des ensembles disjoints

- L'algorithme de Kruskal ordonne les arêtes du graphe en ordre croissant de poids.
- L'algorithme choisit les arêtes les plus petites en premier.
- On s'assure que l'ajout de l'arête dans l'arbre ne crée pas un cycle.
- La structure de données des ensembles disjoints regroupe dans un même ensemble les nœuds faisant partie du même arbre.
- On peut donc vérifier si l'ajout d'une arête ne crée pas de cycle en vérifiant si les deux nœuds appartiennent à des ensembles distincts.

Chapitre 10: Les minorants triviaux

- Pour certains problèmes, il faut lire chaque élément des données fournies en entrée.
  - Ex.: Trouver le minimum d'un ensemble de n nombres.
  - Ex.: Calculer la somme de deux matrices n x n
- Pour tout problème, le temps d'exécution doit être au moins proportionnel à la taille des données produites en sortie
  - Ex.: Imprimer toutes les permutations de n éléments

Chapitre 10: Les minorants triviaux

- Les minorants s'énoncent toujours avec la notation Omega.
- S'il existe plus d'un minorant, il est possible de les combiner en prenant le plus gros.
  - Si on a un minorant de $\Omega(g(n))$ et un deuxième minorant de $\Omega(f(n))$, alors nous avons un minorant de $\Omega(f(n) + g(n))$.
  - Ex.: Soit A et B deux ensembles de n nombres. Énumérer toutes les paires de nombres dans A x B.

$$\Omega(2n), \Omega(n^2) \Rightarrow \Omega(2n + n^2) = \Omega(n^2)$$

Chapitre 10: minorant informationnel

- On énumère toutes les sorties possibles.
- On sélectionne une opération de base que tout algorithme doit exécuter.
- On suppose que l'opération de base élimine la moitié des sorties possibles.
  - Recherche binaire
  - Tri

Chapitre 10: Classe P

- Un **problème de décision** est un problème dont la solution est « oui » ou « non ».
- La **classe P** est l'ensemble des problèmes de décision solubles en temps polynomial.
- Comment prouver qu'un problème est dans P?
  - Démontez qu'il s'agit d'un problème de décision.
  - Donnez un algorithme qui donne la solution « oui » ou « non ».
  - Analysez votre algorithme et montrez qu'il est à temps polynomial.

Chapitre 10: Certificat et vérification

- Un **certificat** est une donnée qui prouve que la réponse à l'instance du problème de décision est « oui ».
- Un **algorithme de vérification** est un algorithme qui prend une instance x d'un problème de décision et une donnée c et qui retourne « oui » si c est bien le certificat qui démontre que la solution au problème est « oui ». L'algorithme est à temps polynomial s'il s'exécute en temps $O((|x|+|c|)^k)$.

Chapitre 10: La classe NP

- La **classe NP** est l'ensemble des problèmes de décision qui admettent un algorithme de vérification à temps polynomial.
- Comment démontrer qu'un problème est dans NP?
  - Démontrez qu'il s'agit d'un problème de décision.
  - Décrivez ce qu'est le certificat pour ce problème.
  - Donnez un algorithme qui vérifie le certificat
  - Analysez l'algorithme de vérification et prouvez qu'il est à temps polynomial.

Chapitre 10: Problèmes NP-Complets

- Un **problème NP-Complet** est un problème dans NP tel que tout autre problème dans NP peut s'y réduire de façon polynomiale.
  - Ex.: Existe-t-il un cycle hamiltonien dans le graphe?
  - Ex.: Existe-t-il une solution à la formule SAT?

Chapitre 11: Fouille avec retours arrière

- On construit une solution en joignant des composantes.
- Pour chaque composante, il faut faire un choix.
- L'arbre de recherche permet d'explorer ces choix.
- La fouille avec retours arrière ne considère que des choix qui ne violent pas les contraintes du problème.

Chapitre 11: Branch & Bound

- Le branch & bound est aux problèmes d'optimisation ce que la fouille avec retours arrière est aux problèmes de satisfaction.
- En plus de faire une fouille avec retours arrière, les algorithmes de type branch & bound calculent une borne optimiste au problème.
- On explore le nœud qui offre la meilleure borne.
- Si la borne optimiste d'un nœud n'est pas meilleure que la dernière solution trouvée, on « ferme » ce nœud, c'est-à-dire qu'on n'explore pas ses enfants.

Chapitre 11: Calcul d'une borne

- Il existe plusieurs façons de calculer une borne optimiste au problème.
- Il y a généralement un compromis entre la qualité de la borne et le temps requis pour la calculer.
- Une partie de la borne est calculée en fonction des choix qui ont été faits depuis la racine de l'arbre, et l'autre partie est calculée de façon optimiste à partir des choix restant à faire.