Semestre 1 & retour sur trace

Programme de colle

Semaine 19 (03 février 2025 – 08 février 2024)

Programme

Semestre 1

- Preuve de terminaison. Preuve de correction.
- Preuve de complexité pire des cas. L'analyse des fonctions récursives se limite à des cas simples.
- Preuve de complexité amortie. Ex : tableaux dynamiques, compteur binaire.
- Tableaux dynamiques, listes chainées.
- Piles, implémentées comme des listes simplement chainées ou des tableaux (éventuellement dynamiques).
- Files, implémentées comme des listes simplement chainées ou des tableaux circulaires.

Retour sur trace

- Exploration exhaustive : concept, limites de l'approche.
- Exploration exhaustive par une fonction récursive, en construisant une solution choix après choix.
- Fonction de rejet et retour sur trace. *L'importance de l'ordre dans lequel on fait les choix a été illustré sur des exemples, mais n'est pas un attendu.*
- Limites de l'analyse de complexité pire des cas pour un retour sur trace.
- Exemples traités : n dames, subset sum (avec des entiers positifs), tour ouvert du cavalier d'Euler, mot ternaire sans carré.

Induction

L'objectif n'est pas de faire de la théorie des ordres, mais de reconnaitre les contextes où l'on peut faire un raisonnement par induction et de savoir en rédiger un.

- Relation d'équivalence.
- Relation d'ordre. Prédecesseur, prédecesseur strict, prédecesseur immédiat. Ordre partiel, ordre total.
- Relation d'ordre bien fondée.
- Thm : on peut faire des raisonnements par induction si et seulement si l'ordre est bien fondé.
- Ex : nombre d'objets dans un mobile de Calder (un arbre binaire strict à f feuilles a $f-1$ noeuds internes).

Les arbres n'ont pas encore été vus.

Et plus

Tout ce qui a été fait depuis le début de l'année est au programme. De plus, certains exercices peuvent tout à fait s'écarter du programme pour tester votre réactivité face à l'inconnu.

Questions de cours

Toute colle commencera par une de ces questions de cours, notée sur 10/20.

Une tolérance vis-à-vis de la syntaxe pourra être appliquée sur les questions d'implémentations si l'élève n'est pas sur machine.

- Exponentiation rapide : code (C ou OCaml), analyse de la complexité de l'exponentiation rapide. On ne comptera que les multiplications comme opérations, dont on calculera le nombre exact. On se limitera au cas où la puissance est une puissance de 2.

- Compteur binaire : analyse de la complexité d'un INCR au sein d'une suite arbitrairement longue de INCR sur un même compteur binaire, par la méthode du choix de l'élève (état initial : compteur à 0). On ne traite pas le cas d'overflow.

- Files : schéma explicatif de l'implémentation par tableau circulaire d'une file. Proposer un type `struct` correspondant en C, et le code de `enfile` ou `defile`. *La difficulté liée à distinguer la file pleine de la file vide peut-être ignorée.*

- Retour sur trace : proposer un algorithme de retour sur trace pour SubsetSum en OCaml ou C, et un critère de rejet (quand tous les entiers sont positifs). On ne cherchera pas à renvoyer une solution valide, simplement à en déterminer l'existence. *Le problème SubsetSum peut, au besoin, être rappelé pour le colleur.*

1 Exponentiation rapide

Voici une fonction OCaml réalisant l'exponentiation rapide :

```
let rec exprap = fun x n \rightarrow
  if n = 0 then 1
  else
    let y = (exprap x (n/2)) in
    if (n/2)*2 = n then y*y
    else x*y*y
```

Notons $M(n)$ le nombre de multiplications. Si `n` $= 0$, la fonction effectue 0 multiplications. Si `n` $= 1$, on fait 2 multiplications (`x*y*y`). Sinon, posons `n` $= 2^p$ (avec $p \in \mathbb{N}^*$). Dans ce cas on fait 1 multiplications (`y`*`y`). On a donc :

$$M(2^p) = \begin{cases} 2 \text{ si } p = 0 \\ 1 + M(2^{p-1}) \text{ sinon} \end{cases}$$

C'est une suite arithmétique, que l'on résout : $M(2^p) = p+2$. La complexité est donc logarithmique en $n = 2^p$, c'est à dire linéaire en la taille de n.

2 Comptaire binaire : suite de INCR par la méthode du potentiel

Convention de nommages, rappelables par le colleur :

On considère une suite arbitrairement longue $o_1, ..., o_k$ d'opérations INCR commençant sur le compteur vide. On considère qu'un flip coûte exactement 1 et le reste 0. On note c_i les coûts respectifs des o_i, et on l'on veut créer des coûts amortis $\hat{c}_i$.

On note L le nombre de bits du compteur.

Faire un très joli schéma d'un compteur binaire à L bits.

Remarque initiale commune aux deux méthodes :

Découpons le coût d'un INCR en deux parties : chaque INCR flippe exactement un bit de 0 à 1 (*car on ignore l'overflow, qui en flippe 0*). En effet, un tel flip ne propage pas de retenue plus avant, et termine donc l'incrémentation. Ainsi, si un INCR flippe c_i bits, $c_i - 1$ flips sont de 1 à 0 et un est de 0 à 1.

Méthode du potentiel :

Prenons comme potentiel $\Phi_i =$ « Nombre de bits à 1 » .

Alors $\Phi_0 = 0$ car l'état initial est le compteur vide. Par définition de Φ_i, on a bien $\forall i, \Phi_i \geq 0 = \Phi_0$.

On a $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$. Mais d'après l'analyse précédente du comportement de INCR, $\Phi_i = \Phi_{i-1} - (c_i - 1) + 1$. Soit $\hat{c}_i = 2$.

On a donc prouvé par la méthode du potentiel que le coût amorti d'un INCR au suite d'une suite arbitrairement longue de INCR est constant.

Méthode du comptable :

Prépayons le coût d'un flip $1 \to 0$ lors du flip $0 \to 1$. Initialement, tous les bits sont à 0. Les états successifs d'un bit sont donc $0 \to 1 \to 0 \to 1 \to 0...$.

En payant 2 lors de chaque flip $0 \to 1$, on peut alors payer 0 lors du flip $1 \to 0$ qui le suit. Il s'ensuit de la remarque initiale que chaque INCR coûte 2, soit $\hat{c}_i = 2$.

On a donc prouvé par la méthode du comptable que le coût amorti d'un INCR au suite d'une suite arbitrairement longue de INCR est constant.

3 Files par tableaux circulaires

Faire un joli schéma. Il est important que sur ce schéma apparaissent `sortie` (l'indice du prochain élément défilé) et `entree` (l'indice de l'autre extrémité de la file; on pensera à préciser s'il est inclus ou exclu), ainsi que le sens de parcours.

Code : cf `https://nuage04.apps.education.fr/index.php/s/EQX2QtdN2kJTW2X` . Dossier TP/09-tabCircC/solution/ . J'y utilise un booléen pour distinguer la file vide de la file pleine; mais c'est un détail qui n'est pas attendu.

4 Retour sur trace

On représentera un ensemble par une liste. J'appelle `ens` la liste contenant l'ensemble, `target` l'entier-cible, et `subsetsum` la fonction récursive.

Le critère de rejet est que lorsque les entiers sont tous positifs, une target négative est impossible.

```
let rec subsetsum ens t =
  if t = 0 then true
  else if t < 0 then false
  else match ens with
  | x :: ens' -> subsetsum ens' (t-x) || subsetsum ens' t
  | [] -> false
```

Pour plus d'explications, cf TP SubsetSum.

Retour sur trace & Induction

Programme de colle

Semaine 20 (10 février 2025 – 15 février 2024)

Programme

Retour sur trace

- Exploration exhaustive : concept, limites de l'approche.
- Exploration exhaustive par une fonction récursive, en construisant une solution choix après choix.
- Fonction de rejet et retour sur trace. *L'importance de l'ordre dans lequel on fait les choix a été illustré sur des exemples, mais n'est pas un attendu.*
- Limites de l'analyse de complexité pire des cas pour un retour sur trace.
- Exemples traités : n dames, subset sum (avec des entiers positifs), tour ouvert du cavalier d'Euler, mot ternaire sans carré.

Induction

L'objectif n'est pas de faire de la théorie des ordres, mais de reconnaitre les contextes où l'on peut faire un raisonnement par induction et de savoir en rédiger un.

- Relation d'équivalence.
- Relation d'ordre. Prédecesseur, prédecesseur strict, prédecesseur immédiat. Ordre partiel, ordre total.
- Relation d'ordre bien fondée.
- Thm : on peut faire des raisonnements par induction si et seulement si l'ordre est bien fondé.
- Ex : nombre d'objets dans un mobile de Calder (un arbre binaire strict à f feuilles a $f-1$ noeuds internes).
- Ordre produit : définition, conservation du caractère bien fondé, non-totalité.
- Ordre lexicographique : définition, conservation du caractère bien fondé, conservation de la totalité.
- Ex : preuve du fait que la fonction Ackerman est au-moins sur-linéaire en son second argument.
- Ordre engendré par une relation acyclique.
- Ensembles inductifs : un ensemble défini apr induction est l'ensemble des éléments que l'on obtient en appliquant un nombre fini de fois les règles de construction.
- Induction structurelle.

Les arbres n'ont pas encore été vus.

Et plus

Tout ce qui a été fait depuis le début de l'année est au programme. De plus, certains exercices peuvent tout à fait s'écarter du programme pour tester votre réactivité face à l'inconnu.

Questions de cours

Toute colle commencera par une de ces questions de cours, notée sur 10/20.

Une tolérance vis-à-vis de la syntaxe pourra être appliquée sur les questions d'implémentations si l'élève n'est pas sur machine.

- Les questions de cours des semaines précédentes.

- On conidère la fonction suivante :

$$A : (n, x) \in \mathbb{N} \times \mathbb{N} \mapsto \begin{cases} x + 1 \text{ si n} = 0 \\ A(n-1, 1) \text{ si } x = 0 \\ A(n-1, A(n, x-1)) \text{sinon} \end{cases}$$

  Prouver par induction bien fondée sur $(\mathbb{N}^2, \leq_{lex})$ que pour tous n et x on a $A(n, x) > x$.

- Proposer en OCaml une fonction qui réalise la fusion triée de deux listes triées `l0` et `l1` . Prouver sa terminaison, et expliquer à l'oral sa correction.

1 Semaines précédentes

Cf semaines précédentes.

2 Ackerman est au moins sur-linéaire en son second argument

Notons $H_{n,x}$ la propriété « $Ack(n,x) > x$ ». Montrons la vraie pour toute paire (n,x) par induction bien fondée sur (n,x) dans $(\mathbb{N}^2, \leq_{lex})$.

- **Initialisation** Le seul élément minimal de $\leq_{lex}$ est $(0,0)$. Nous allons traiter plus de cas ici : tous les cas $(0,x)$ (c'est plus confortable).

  Soit $(0,x) \in \mathbb{N}^2$. Alors $A(0,x) = x+1 > x$.

- **Hérédité** Soit $(n,x) \in \mathbb{N}^2$ avec $n > 0$. Supposons H vraie pour tout $(n',x') <_{lex} (n,x)$ et montrons-la vraie pour (n,x). D'après la définition de A, il reste deux cas à distinguer :

  - Si $x = 0$. Alors $A(n,x) = A(n-1,1)$. Or, $(n-1,1) <_{lex} (n,x)$ car $n-1 < n$. Donc d'après $H_{n-1,x}$, $A(n-1,1) > 1$. D'où $A(n,0) > 0$.
  - Sinon, $A(n,x) = A(n-1, A(n,x-1))$. Or, $n-1 < n$ donc $(n-1, A(n,x-1)) <_{lex} (n,x)$, donc par hypothèse de récurrence $A(n-1, A(n,x-1)) > A(n,x-1)$.

    Or, $(n,x-1) <_{lex} (n,x)$. Donc par hypothèse de récurrence, $A(n,x-1) > x-1$. Comme ce sont des entiers, on a $A(n,x-1) \geq x$.

    Bout à bout :

    $$A(n,x) = A(n-1, A(n,x-1)) > A(n,x-1) \geq x$$

    On obtient donc $A(n,x) > x$.

  D'où l'hérédité.

- **Conclusion** Par principe de récurrence bien fondée, la propriété est vraie pour toute paire (n,x).

3 Fusion de listes triées

Voici une façon d'écrire la fonction demandée :

```
let rec merge l0 l1 =
  match (l0 , l1) with
  | [] , [] -> []
  | [] , _  -> l1
  | _  , [] -> l0
  | t0::q0 , t1::q1 ->
      if t0 <= t1 then t0 :: (merge q0 l1)
      else t1 :: (merge l0 q1)
```

Cette fonction ni contient ni boucle ni appels d'autres fonctions : il ne reste qu'à prouver la terminaison de la suite d'appels récursifs. Pour cela, montrons que (`l0`, `l1`) est un variant selon le produit (bien fondé) des ordres structurels. À chaque appel récursif, une des deux listes perd sa tête (et décroit donc selon l'ordre structurel), et l'autre est inchangée. Il s'ensuit que la paire (`l0`, `l1`) décroit strictement selon un ordre bien fondé : c'est donc un variant, et la terminaison s'ensuit.

Remarque. On peut aussi faire cette preuve par récurrence sur la somme des longueurs de `l0` et `l1`

Correction partielle, informellement : si une des deux listes est vide puisque l'autre est triée elle est la fusion triée des deux et c'est bien ce que l'on renvoie. Si les deux listes sont non-vides, le minimum du tout est le minimum d'une des deux listes donc le minimum des têtes des listes (elles sont triées !). La fusion triée est donc la plus petit tête suivie de la fusion triée du reste, et c'est ce que l'on renvoie.

Arbres

Programme de colle

Semaine 21 (17 février 2025 – 22 février 2025)

Programme

Arbres

Généralités

- Vocabulaire : arbre, arité (d'un noeud), parent, enfant, ancêtre, descend, adelphe (frère/soeur). Profondeur, hauteur (longueur d'un chemin depuis la racine, donc $h(\perp) = 1$).
- Exemples divers : mobiles de Calder, arbre syntaxique d'une expression, *trie*, exploration exhaustive, ABR.
- Représentation récursive en OCaml avec liste des enfants.
- Dans le cas d'étiquettes en bijection avec $[0; 1; ...; n-1]$, représentation par tableau de parenté.

Arbres binaires

- Définition récursive. Lien avec l'arité.
- Représentation récursive.
- Arbres binaires particuliers : stricts, parfaits, complets, peignes.
- Propriétés combinatoires concernant le nombre de noeuds.
- Représentation d'un arbre binaire complet dans un tableau. Indice du parent/des enfants.
- Transformation LCRS.

Parcours

- Parcours en profondeur récursif.
- Ordre préfixe, infixe, postfixe.
- Parcours en profondeur itératif.

Et plus

Tout ce qui a été fait depuis le début de l'année est au programme. De plus, certains exercices peuvent tout à fait s'écarter du programme pour tester votre réactivité face à l'inconnu.

Questions de cours

Toute colle commencera par une de ces questions de cours, notée sur 10/20.

Une tolérance vis-à-vis de la syntaxe pourra être appliquée sur les questions d'implémentations si l'élève n'est pas sur machine.

- Les questions de cours des semaines précédentes.
- Soit A un arbre binaire. Montrer que l'étage p de A contient au plus 2^p noeuds.
- Montrer qu'un arbre binaire strict à f feuilles et n_i noeuds internes vérifie $f = n_i + 1$.
- Soit A est un arbre binaire strict. Montrer que si A est parfait, alors toutes ses feuilles sont à même profondeur.
- Appliquer à la main la transformation LCRS ou sa réciproque sur un ou des arbres choisis par le colleur.

1 Étage p d'un arbre binaire

Soit A un arbre binaire. Nommons e_p le nombre de noeuds à profondeur p dans A. Montrons par récurrence sur $p \in \mathbb{N}$ que $e_p \leq 2^p$.

- Initialisation : Il y a au plus 1 noeud de profondeur 0 : la racine.
- Hérédité : Supposons la propriété vraie au rang p, montrons-la vraie au rang $p+1$.

  Chaque noeud de profondeur $p+1$ est enfant d'un parent de profondeur p. Chacun de ces parents a au plus 2 enfants. Il s'ensuit que $r_{p+1} \leq 2r_p$.

  Donc par H.R., $r_{p+1} \leq 2^{p+1}$, et cette borne est atteinte.
- Conclusion : Nous avons prouvé que pour tout $p \in \mathbb{N}$, $r_p \leq 2^p$.

Remarque. Il est important de faire la preuve par récurrence une fois l'arbre A introduit : on fait une preuve par récurrence sur la profondeur dans un A fixé (quelconque, mais fixé).

2 $f = n_i + 1$ dans un arbre binaire strict

Montrons la propriété vraie par induction structurelle sur un arbre binaire strict A.

- Initialisation : Si A est une feuille, c'est immédiat.
- Hérédité : Si A est de la forme $N(\_, g, d)$ avec g et d (arbres binaires stricts) vérifiant la propriété, montrons la propriété vraie pour A. On note $n_i(g)$ le nombre de noeuds internes de g et $f(g)$ son nombre de feuilles (de même pour A et d).

  Les noeuds internes de A sont partitionnés en : sa racine, les noeuds internes de g, et ceux de d. Donc $n_i(A) = 1 + n_i(g) + n_i(d)$.

  Similairement, $f(A) = f(g) + f(d)$ (la racine n'est pas une feuille).

  En appliquant l'hypothèse d'induction on obtient : $f(A) = (n_i(g)+1) + (n_i(d)+1) = 1 + n_i(A)$.

  D'où l'hérédité.
- Conclusion : On a prouvé par induction structurelle que pour tout arbre binaire strict, $f = n_i + 1$.

3 Binaire strict parfait implique feuille même profondeur

Soit A un arbre binaire strict. Prouvons l'implication « si A est parfait, alors toutes les feuilles de A sont à même profondeur » par induction structurelle sur A.

- Initialisation : Si A est une feuille, c'est immédiat.
- Hérédité : Si A est un arbre binaire strict de la forme $N(\_, g, d)$ avec g et d (arbres binaires stricts) vérifiant la propriété, montrons la propriété vraie pour A. Supposons donc A parfait.

  Les feuilles de A sont celles de g et de d. Or, comme A est parfait, g et d le sont aussi : donc d'après l'hypothèse d'induction, les feuilles de g sont à même profondeur $h(g)$ dans g donc à même profondeur $h(g)+1$ dans A . De même dans d.

  Mais comme A est parfait, $h(g) = h(d) = h(A) - 1$. D'où l'hérédité.
- Conclusion : On a prouvé par induction structurelle que pour tout arbre binaire strict, s'il est parfait alors ses feuilles sont à même profondeur.

4 LCRS

Cf Cours et TD.

Arbres

Programme de colle

Semaine 22 (10 mars 2025 – 15 mars 2025)

Programme

Arbres

Parcours

- Parcours en profondeur récursif.
- Ordre préfixe, infixe, postfixe.
- Parcours en profondeur itératif.
- Parcours en largeur.

Arbres binaires de recherche

- ABR : définition, exemples.
- Un arbre binaire est un ABR ssi son parcours en profondeur infixe est trié.
- Recherche, insertion, suppression dans un ABR (par remontée du max gauche / min droit) : fonctionnement, exemples, complexité.
- Tri par ABR. Pire des cas $O(n^2)$, cas moyen $O(n \log n)$. *La définition de cas moyen n'a pas été manipulée et n'est pas maitrisée; ici le but est d'illustrer l'impact de l'ordre d'insertion.*
- Arbre Rouge-Noir : définition, exemples.
- Caractère équilibré d'un ARN.
- Insertion dans un ARN. *La suppression dans un ARN est admise.*

Et plus

Tout ce qui a été fait depuis le début de l'année est au programme. De plus, certains exercices peuvent tout à fait s'écarter du programme pour tester votre réactivité face à l'inconnu.

Questions de cours

Toute colle commencera par une de ces questions de cours. Traiter sa question de cours convenablement est nécessaire et suffisant pour avoir au moins 10/20 .

Une tolérance vis-à-vis de la syntaxe pourra être appliquée sur les questions d'implémentations si l'élève n'est pas sur machine.

- Les questions de cours des semaines précédentes.
- Un des sens de l'équivalence « un arbre binaire est un ABR ssi son parcours en profondeur infixe le trie ».
- Insertions successives dans un ARN choisi par le colleur.
- Dans un arbre bicolore, montrer que si (2) : « tous les chemins de la racine à un $\perp$ ont autant de noeuds Noirs », alors (2′) « pour tout noeud, tous les chemins de ce noeud à un $\perp$ contiennt autant de noeuds Noirs ».
- En admettant qu'un ARN A a au moins $2^{bh(A)} - 1$ noeuds internes, prouver que la hauteur d'un ARN est un $O(\log_2 n)$.

1 Un ABR est trié par dfs infixe

Notation : $infixe(A)$ est la liste des noeuds dans l'ordre d'un dfs infixe, et @ est la concaténation.

On montre par induction structurelle sur A un ABR que $infixe(A)$ est trié.

- Initialisation : Si A est vide, c'est immédiat.
- Hérédité : Si $A = N(g, x, d)$ avec g et d vérifiant l'implication, montrons que $infixe(g, x, d)$ est trié. On a :

$$infixe(A) \;=\; \underbrace{infixe(g)}_{\text{trié par H.I.}} \;@\; x \;@\; \underbrace{infixe(d)}_{\text{trié par H.I.}}$$

  Par H.I, les deux sous-parcours infixes sont triés par ordre croissant. Or, $\max S(g) < x < \min S(d)$; donc x est bien placé. Il s'ensuit que le tout est trié, d'où l'hérédité.

- Conclusion : Le parcours infixe d'un ABR est trié.

2 Un arbre binaire trié par dfs infixe est ABR

On procède par induction structurelle sur A un arbre binaire.

- Initialisation : Si A est vide, c'est immédiat.
- Hérédité : Si $A = N(g, x, d)$ est un arbre binaire avec g et d vérifiant l'implication réciproque, montrons que A est un ABR. Comme $infixe(A)$ est trié, ses sous-suites $infixe(g)$ et $infixe(d)$ le sont aussi : donc par H.I., g et d sont des ABR.

  De plus comme $infixe(A)$ est trié, $\max S(g) = \max infixe(g) < x < \min infixe(d) = \min S(d)$. Donc A est un ABR, d'où l'hérédité.

- Conclusion : Un arbre binaire trié par parcours infixe est un ABR.

3 Insertions dans un ARN

Cf la méthode du cours. La méthode du TP étoilé, qui permet de parfois réparer un R-R sans le faire remonter jusqu'à la racine, n'est pas attendue.

4 $(2) \implies (2')$

J'écris ici la preuve que vous m'avez proposée en classe, qui est différente de celle faite dans le cours.

Soit A un arbre binaire bicolorié dont tous les chemins de la racine à $\perp$ contiennent le même nombre bh de noeuds Noirs, et x un noeud de A. Montrons que tous les chemins de x à $\perp$ contiennent tous autant de noeuds Noirs.

Comme x est un noeud de A, en remontant ses parentés on obtient $racine(A), ..., x$ un (unique) chemin vertical de la racine à x; on note $bd(x)$ (« **b**lack **d**epth ») son nombre de noeuds Noirs sans compter x.

Ainsi, tout chemin $x, ..., \perp$ correspond à un chemin $racine(A), ..., x, ..., \perp$. Or, tous ces chemins depuis la racine ont bh noeuds noirs, donc tous les $x, ..., \perp$ ont le même nombre $bh - bd(x)$ noeuds Noirs.

5 Caractère équilibré d'un ARN

Notations : n le nombre de noeuds, n_i le nombre de noeuds internes

On va montrer que $h(A) \leq 2\log_2(n_i + 1)$. Comme il y a au moins une feuille et par croissance de $\log_2$, on obtient $h(A) = O(\log_2(n))$.

D'après la propriété (1) des ARN (tout noeud Rouge a un parent Noir), dans tout chemin de la racine à $\perp$ au moins la moitié des noeuds sont Noirs. Donc :

$$bh(A) \geq \frac{h}{2}$$

Or d'après le lemme, $2^{bh(A)} - 1 \leq n_i$, donc $2^{\frac{h}{2}} \leq n_i + 1$. On en déduit le résultat par croissance de $\log_2$.

Révisions

Programme de colle

Semaine 23 (17 mars 2025 – 22 mars 2025)

Programme

Tout depuis le début de l'année.

Et plus

Tout ce qui a été fait depuis le début de l'année est au programme. De plus, certains exercices peuvent tout à fait s'écarter du programme pour tester votre réactivité face à l'inconnu.

Questions de cours

Toute colle commencera par une de ces questions de cours. Traiter sa question de cours convenablement est nécessaire et suffisant pour avoir au moins 10/20 .

Une tolérance vis-à-vis de la syntaxe pourra être appliquée sur les questions d'implémentations si l'élève n'est pas sur machine.

- Les questions de cours des semaines précédentes.
- Résoudre une équation de complexité de la forme $T(n) = rT(n/c) + f(n)$ (avec r et n entiers tels que $r > 0$ et $c > 1$), dans le cas où $f(n)$ est un polynome. *Le théorème maitre est hors programme et ne constitue pas une preuve acceptée.*

Diviser pour régner et programmation dynamique

Programme de colle

Semaine 24 (24 mars 2025 – 29 mars 2025)

Programme

Diviser pour régner

- Théorème borne inférieure des tris par comparaison.
- Exemple : tri fusion. Implémentation en C et OCaml. Analyse de complexité.
- Résolution d'équations de complexité de la forme $T(n) = rT(n/c) + f(n)$ (avec r et n entiers tels que $r > 0$ et $c > 1$).
- Principe général de la méthode diviser pour régner.
- Exemple : algorithme de Karatsuba. *L'idée de découper un entier en partie haute et partie basse est exigible. Toutefois, l'astuce de calcul qui permet de se réduire à 3 multiplications au lieu de 4 n'est pas exigible.* Analyse de complexité.
- Exemple : tri rapide. Partitionnement (drapeau hollandais). Implémentation en C. Analyse de la complexité dans le pire des cas. Astuce de choix du pivot pour (probablement) éviter le pire des cas.

Rencontre au milieu

- Exemple pour SubsetSum.

Programmation dynamique

- Récursivité et redondance des appels en cas de chevauchement de sous-problèmes. Effet sur la complexité temporelle.
- Exemple sur les pyramides.
- Méthode de haut en bas / mémoïsation.
- Exemple sur les pyramides. Analyse de complexité temporelle *et* spatiale.
- Méthode de bas en haut.
- Exemple sur les pyramides.

L'optimisation de la mémoire parfois permise par la méthode de bas en haut n'a pas encore été vue. La reconstruction d'une solution optimale non plus.

Impératif en OCaml

- Les tableaux OCaml ainsi que les boucles ont été introduites. Les références et les enregistrements mutables n'ont pas encore été vus.

Et plus

Tout ce qui a été fait depuis le début de l'année est au programme. De plus, certains exercices peuvent tout à fait s'écarter du programme pour tester votre réactivité face à l'inconnu.

Questions de cours

Toute colle commencera par une de ces questions de cours, notée sur 10/20.

Une tolérance vis-à-vis de la syntaxe pourra être appliquée sur les questions d'implémentations si l'élève n'est pas sur machine.

- Implémentation du tri fusion : toute partie du code (division/fusion/tri) choisie par le colleur, en C ou en OCaml.
- Résolution du problème du drapeau hollandais en pseudo-code ou en C.
- Pseudo-code d'une solution mémoïsée en programmation dynamique (de haut en bas). Explication de pourquoi (et quand) c'est plus efficace qu'une solution récursive naïve.