Concevoir, rédiger et corriger un sujet d'informatique

Jean-Christophe Filliâtre

27 mai 2023

Résumé

Ce petit document regroupe un certain nombre de suggestions pour l'élaboration et la correction d'un sujet d'informatique, qu'il s'agisse d'un examen ou d'un concours. Il n'y a là aucune prétention à des vérités absolues [1], mais uniquement le souhait de partager de nombreuses années d'expérience et d'éviter à d'autres les écueils dans lesquels je suis moi-même tombé.

On ne traite ici que d'un sujet d'examen ou de concours *sur table*, et non d'une épreuve orale ou sur machine.

1 Conception

Écrire un corrigé est la meilleure façon d'obtenir un sujet de qualité, et écrire le corrigé **en même temps** que le sujet est une encore meilleure idée. Si on rédige le sujet en LaTeX, il est très simple d'utiliser un environnement conditionnel pour les solutions et de placer celles-ci après chaque question. En particulier, on peut inclure dans ce corrigé les critères d'évaluation qui seront utilisés pendant la correction. S'il s'agit d'un examen plutôt que d'un concours, ces critères sont liés aux objectifs pédagogiques et on peut d'ailleurs construire le sujet autour de ces objectifs pédagogiques.

Au-delà de son caractère rassurant, la présence du corrigé permet d'évaluer la difficulté des questions et le temps qu'elles vont demander aux étudiants. En particulier, s'il y a des **questions de programmation**, on peut avantageusement compter le nombre de lignes de code qui sont attendues. Écrire du code sur le papier est relativement pénible, et limiter la quantité totale de code demandé est une bonne idée. *Suggestion* : diviser le nombre total de lignes de code du corrigé par le nombre de questions de programmation et s'assurer que le nombre obtenu n'est pas trop grand. L'expérience montre qu'une valeur autour de 7 est raisonnable.

Bien entendu, on peut de même estimer la longueur des démonstrations attendues ou encore le nombre d'éléments des schémas à dessiner. Plutôt que de donner un barème dans le sujet, ce qui est très contraignant pour la correction (voir section 3), on peut donner dans le sujet un temps de référence pour chaque exercice ou partie du sujet, construit sur la base de ces estimations. Dans le cadre d'un examen, cela rassure beaucoup les étudiants.

Il est fondamental d'éviter les **questions à tiroir**, c'est-à-dire des questions auxquelles il est nécessaire de répondre pour pouvoir poursuivre. C'est particulièrement vrai pour

1. Le monde est déjà trop plein d'injonctions.

les questions de programmation. Ainsi, il ne faut pas demander le code d'une fonction puis demander sa complexité, car on s'expose au risque d'une réponse inappropriée à la première question (un code trop naïf) mais d'une réponse tout à fait correcte à la seconde question (la complexité correcte de ce code naïf par un argument trivial, alors qu'on attendait une preuve plus difficile). Il en va de même pour la preuve de correction d'un morceau de code. Aussi, on suggère l'approche suivante :

— soit on demande un morceau de code avec une complexité exigée ou une propriété attendue ;
— soit on donne un morceau de code et on demande sa complexité ou la preuve de sa correction.

Ce n'est pas du tout limitant de procéder ainsi. On peut avantageusement marier ces deux types de questions dans un sujet.

Puisqu'on parle de donner un morceau de code dans un sujet, il n'est pas inutile de rappeler tous les problèmes que pose le **pseudo-code** par l'absence d'une sémantique claire. L'appel se fait-il par valeur ou par référence ? Telle opération est-elle en temps constant ? Telle structure de données est-elle mutable ou immuable ? etc. Pour cette raison, il est préférable de donner soit un algorithme décrit en français, soit un morceau de code écrit dans un langage de programmation connu. Il est d'ailleurs assez frappant de constater qu'un vrai morceau de code est rarement plus long que du pseudo-code.

2 Rédaction

Utiliser un **correcteur orthographique** n'est pas négociable. Et il ne faut pas hésiter à ajouter à son dictionnaire, globalement ou localement, tous les mots techniques et acronymes apparaissant dans le sujet, de sorte à éviter les faux positifs. S'il y a trop de signalements du correcteur orthographique, on n'y prête plus attention.

On suggère d'écrire des phrases **au présent uniquement** (« La partie 2 aborde... » plutôt que « La partie 2 abordera... »), dans un style relativement **neutre** (en évitant « nous » et « vous » notamment) et en évitant la voix passive. De même, les questions gagnent à être rédigées à l'infinitif plutôt qu'à l'impératif, en préférant « Donner la complexité de... » à « Donnez la complexité de... ».

Pour ce qui est de la rédaction de texte scientifique, on suggère très fortement la lecture de *Mathematical Writing* (Knuth, Larrabee, Roberts), que l'on trouve en ligne. En particulier, les tous premiers conseils sont fondamentaux : on sépare deux formules par au moins un mot, on ne commence pas une phrase par un symbole, etc.

En français, on dit qu'une fonction **renvoie** une valeur et non qu'une fonction retourne une valeur. C'est très bien expliqué dans *Vocabulaire international de l'informatique* (Association Française de Normalisation, 1975). Retourner est un anglicisme, mais surtout un contresens potentiel dans une phrase comme « la fonction retourne la liste... » ou pire encore « la fonction retourne le polyomino... ».

Lorsque du code est donné dans le sujet, on préférera une police à chasse fixe (`comme ceci`) et on numérotera les lignes pour permettre de faire facilement référence au code fourni. Si le sujet est rédigé en LaTeX, et que le code est formaté avec le paquet `lstlisting`, il est à noter que le paramétrage par défaut ne fait aucune de ces deux choses. On y remédie facilement de la manière suivante :

```
\lstset{%
  basicstyle=\ttfamily,%
  numberstyle=\tiny,%
  keywordstyle=\bfseries,%
  columns=fullflexible,%
}
```

Lorsque l'on demande de programmer une fonction, imposer son nom et ceux de ses paramètres formels aide à la compréhension (et aide également les correcteurs). Donner les types des paramètres et de la valeur renvoyée, le cas échéant, est également une bonne pratique.

Mise en page. Il convient de bien identifier les différentes questions (et sous-questions, le cas échéant) et de bien les séparer visuellement du texte situé entre les questions. On peut utiliser pour cela de l'espace vertical, mais on peut également entamer tout paragraphe situé entre deux questions par un titre (par exemple en utilisant `\paragraph` en LaTeX).

Les éléments fournis relativement gros (algorithme, code, table, exemple, etc.) gagnent à être présentés dans des **figures**. On prêtera soin au placement de ces figures (en haut de page ou sur une page dédiée, plutôt qu'au milieu du texte, est un bon principe), notamment au regard des questions qui y font référence. En particulier, si la figure est sur la même page, ou la page d'en face, que la question, on évite ainsi les manipulations désagréables du sujet. Les visionneurs de PDF proposent typiquement un *mode dual* qui affiche côte à côte la page paire à gauche et la page impaire à droite, ce qui permet de vérifier facilement la mise en page.

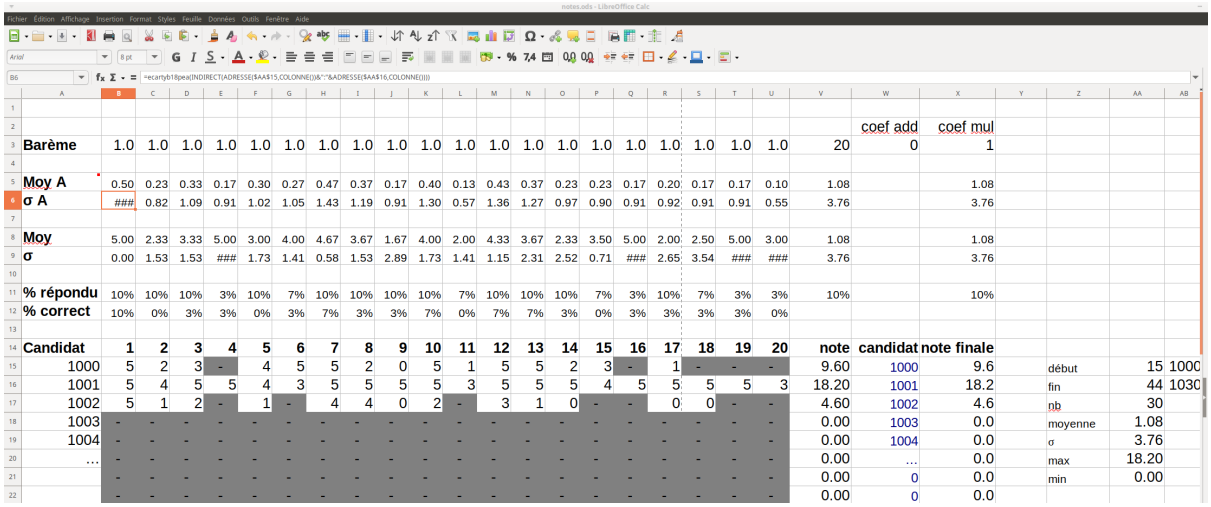
Mieux encore, on peut déporter certaines figures en fin de sujet, dans une **annexe détachable**, c'est-à-dire une ou plusieurs pages que l'on pourra garder sous les yeux quelle que soit la question traitée.

Enfin, si le sujet est destiné à être distribué avant le début de l'épreuve, penser à laisser une dernière page paire vide.

3 Correction

Une façon très efficace de corriger consiste à noter chaque question sur un même nombre de points, par exemple 5 points, en nombre entier de points uniquement c'est-à-dire une note dans $\{0, 1, 2, 3, 4, 5\}$. Le barème se décompose alors en deux éléments.

- **Avant de corriger**, on détermine la répartition des points pour chaque question. Il y a là de nombreuses approches, que l'on peut varier selon les questions.
  - Le barème peut être additif : 2 points pour cet élément attendu de la réponse et 3 points pour cet autre élément.
  - Le barème peut être soustractif : -1 pour telle erreur, -2 pour telle erreur, etc. Ceci est particulièrement adapté aux questions de programmation, pour lesquelles on peut se donner un barème général (voir ci-dessous).
  - Une question très facile peut être notée de façon binaire, *i.e.*, avec 0 ou 5.
- **Après avoir corrigé**, on fixe le poids de chaque question. En particulier, on peut le faire en mode examen (quelles sont les compétences attendues) ou en mode



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1																												
2																							coef add	coef mul				
3	**Barème**	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	20	0	1				
4																												
5	**Moy A**	0.50	0.23	0.33	0.17	0.30	0.27	0.47	0.37	0.17	0.40	0.13	0.43	0.37	0.23	0.23	0.17	0.20	0.17	0.17	0.10	1.08		1.08				
6	**σ A**	###	0.82	1.09	0.91	1.02	1.05	1.43	1.19	0.91	1.30	0.57	1.36	1.27	0.97	0.90	0.91	0.92	0.91	0.91	0.55	3.76		3.76				
7																												
8	**Moy**	5.00	2.33	3.33	5.00	3.00	4.00	4.67	3.67	1.67	4.00	2.00	4.33	3.67	2.33	3.50	5.00	2.00	2.50	5.00	3.00	1.08		1.08				
9	**σ**	0.00	1.53	1.53	###	1.73	1.41	0.58	1.53	2.89	1.73	1.41	1.15	2.31	2.52	0.71	###	2.65	3.54	###	###	3.76		3.76				
10																												
11	**% répondu**	10%	10%	10%	3%	10%	7%	10%	10%	10%	10%	7%	10%	10%	10%	7%	3%	10%	7%	3%	3%	10%		10%				
12	**% correct**	10%	0%	3%	3%	0%	3%	7%	3%	3%	7%	0%	7%	7%	3%	0%	3%	3%	3%	3%	0%							
13																												
14	**Candidat**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**note**	**candidat**	**note finale**				
15	1000	5	2	3	-	4	5	5	2	0	5	1	5	5	2	3	-	1	-	-	-	9.60	1000	9.6		début	15	1000
16	1001	5	4	5	5	4	3	5	5	5	5	3	5	5	5	4	5	5	5	5	3	18.20	1001	18.2		fin	44	1030
17	1002	5	1	2	-	1	-	4	4	0	2	-	3	1	0	-	-	0	0	-	-	4.60	1002	4.6		nb	30	
18	1003	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.00	1003	0.0		moyenne	1.08	
19	1004	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.00	1004	0.0		σ	3.76	
20	...	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.00	...	0.0		max	18.20	
21		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.00	0	0.0		min	0.00	
22		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.00	0	0.0				

FIGURE 1 – Feuille de correction.

concours (favoriser un grand écart type notamment). Jouer sur les coefficients *a posteriori* permet également d'atteindre facilement une moyenne cible.

Feuille Excel de correction. Une suggestion de feuille de correction est proposée à l'adresse `https://www.lri.fr/~filliatr/pub/crc/`. Un aperçu en est donné figure 1. On suppose ici 20 questions (colonnes B à U), mais cela s'adapte trivialement à un nombre différent de questions. Les copies correspondent aux lignes 15 et suivantes, avec une copie par ligne. Une question non traitée est indiquée par le texte - et un fond gris. Une formatage conditionnel signale en rouge toute note saisie qui ne serait pas comprise entre 0 et 5. La ligne correspondant à la dernière copie est renseignée en AA16.

Pour chaque question, on dispose de sa moyenne et de son écart type, en prenant en compte uniquement les copies ayant traité la question (lignes 8 et 9) ou bien au contraire toutes les copies (lignes 5 et 6). Ces informations permettent notamment l'établissement des poids, qui sont renseignés ligne 3 une fois la correction terminée. La note (colonne V) est alors calculée par

$$note = \frac{4}{\sum q_i} \sum_i q_i \cdot c_i$$

où q_i est le poids donné à la question i et c_i sa note. On applique alors à cette note un coefficient multiplicatif (X3, par défaut 1) et un coefficient additif (W3, par défaut 0), puis on arrondit au dixième et on majore à 20 pour obtenir la note finale (colonne X). Bien entendu, d'autres formules sont envisageables pour l'obtention de la note finale.

Le numéro de la copie est repris en colonne W, afin que l'on puisse facilement copier-coller les deux colonnes W et X dans un système de report de notes (qui peut être une autre feuille Excel).

Correction à plusieurs. Si les copies sont partagées entre plusieurs correcteurs, et que l'on tient à éviter des harmonisations *a posteriori*, on peut adopter l'algorithme suivant :

1. On définit un barème, *i.e.*, la répartition des 5 points pour chaque question.

2. Chacun corrige un certain nombre de copies, idéalement une trentaine ou plus. Pendant cette période, la barème est précisé / rectifié si besoin.
3. On compare les moyennes par question (ligne 8). Lorsque la différence est importante entre deux correcteurs, on cherche à déterminer où la correction a divergé, le cas échéant. Si besoin, tel ou tel correcteur repasse sur certaines questions. C'est un peu pénible, mais nécessaire.

   Note : Il se peut que les copies ne soient pas équitablement distribuées entre les correcteurs. Pour en avoir une idée, on peut considérer une question très simple et dont la notation n'est pas discutable, et comparer les moyennes des différents correcteurs sur cette question.
4. Enfin, on corrige toutes les copies restantes, idéalement sans retoucher au barème.

Cette méthode a fait ses preuves. Si le volume de copies est suffisamment important (par exemple 250 copies par correcteur), **aucune harmonisation** n'est nécessaire.

Barème générique pour les questions de programmation

Voici quelques suggestions pour un barème soustractif pour les questions de programmation (sur la base de 5 points par question) :

— on ne sanctionne pas les petites erreurs de syntaxe, voire pas les petites erreurs de typage non plus
— code inintelligible par le correcteur $\rightarrow -5$
— pas la complexité attendue $\rightarrow -2$
— oubli d'appel récursif $\rightarrow -2$
— mauvais appel récursif induisant une boucle infinie $\rightarrow -2$
— oubli d'un cas de base dans une fonction récursive $\rightarrow -2$
— non respect du prototype fourni $\rightarrow -1$
— formule utilisée dans le code (par ex. 2^n au lieu du bon opérateur ou de la bonne fonction pour la puissance) $\rightarrow -1$
— non traitement d'un cas simple $\rightarrow -1$
— erreur de borne, décalage, etc. $\rightarrow -1$
— accès en dehors des bornes d'un tableau $\rightarrow -1$
— oubli de la valeur de retour $\rightarrow -1$
— oubli d'un argument dans une fonction $\rightarrow -1$

Remerciements. Je tiens à remercier Sylvie Boldo et Thibaut Balabonski pour leurs remarques et suggestions.