Enjeux et perspectives pour le logiciel libre

Roberto Di Cosmo

Professeur d'Informatique
Université Paris 7 Denis Diderot et INRIA

Congrès de SPECIF, Tours
14 Janvier 2009

Logiciel Libre: quelques définitions

Gratuit (anglais: free):
logiciel non payant (aujourd'hui)

Libre (anglais: free):
logiciel avec 4 droits

- Liberté d'utiliser le logiciel
- Liberté d'étudier les sources du logiciel et de l'adapter à ses besoins
- Liberté de distribuer des copies
- Liberté de distribuer les sources (même modifiées)

Il y a des obligations aussi, qui varient selon la licence: GPL/BSD/Mozilla/X, etc.

Les multiples facettes du LL pour la formation

das Wesen der Mathematik liegt gerade in ihrer Freiheit

Georg Cantor

Logiciel libre = accès au code, liberté de modifier et distribuer:

- comme outil, garantit égalité des chances: tous les étudiants ont accès à l'ensemble des outils, sans restrictions, sans besoin d'accords spécifiques (Firefox, OpenOffice, Gimp, Celestia ... R, Scilab, Ocaml ...)
- en Informatique, permet l'accès à une meilleure formation: pas de barrière à la connaissance, liberté d'innover *"comme si un élève ingénieur pouvait participer à la construction du pont d'Aquitaine"*

Le Logiciel Libre est incontournable pour l'enseignement

et encore plus pour l'enseignement de l'Informatique

Logiciel (Libre) et IT au 21ème siècle

Le logiciel évolue très rapidement, et il devient

plus pervasif

embarqué, jeux, telephones (TELEPHONES!), voitures, trains, avions, ...

plus critique

Systèmes à Logiciel Préponderant (e.g. avions): le logiciel compte pour 30% du cout total!

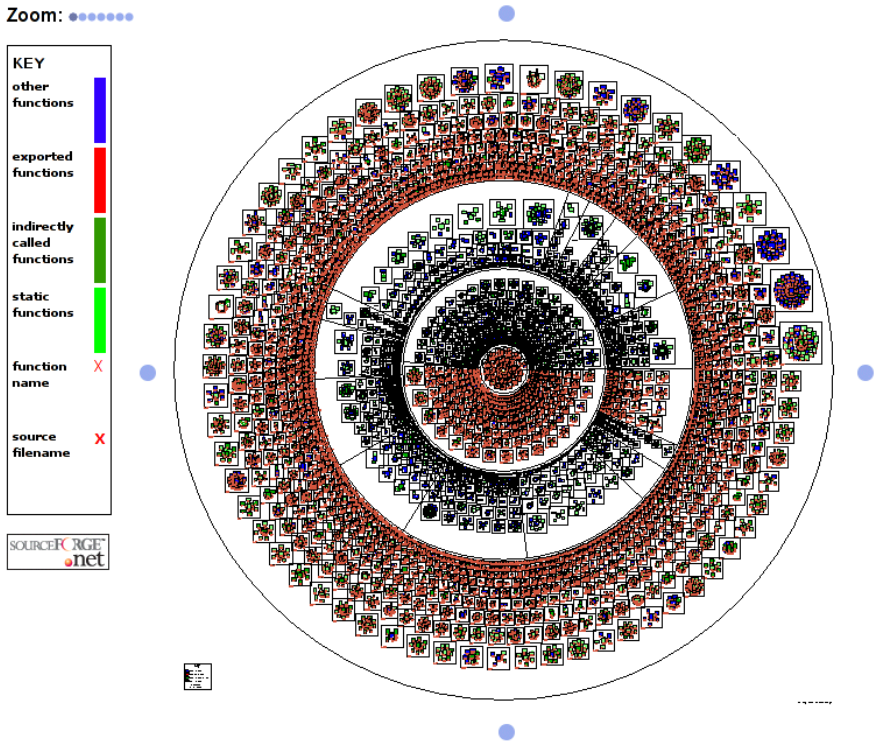
plus évolutif

le cycle en V est depassé; on doit *concevoir pour le changement*

plus complexe

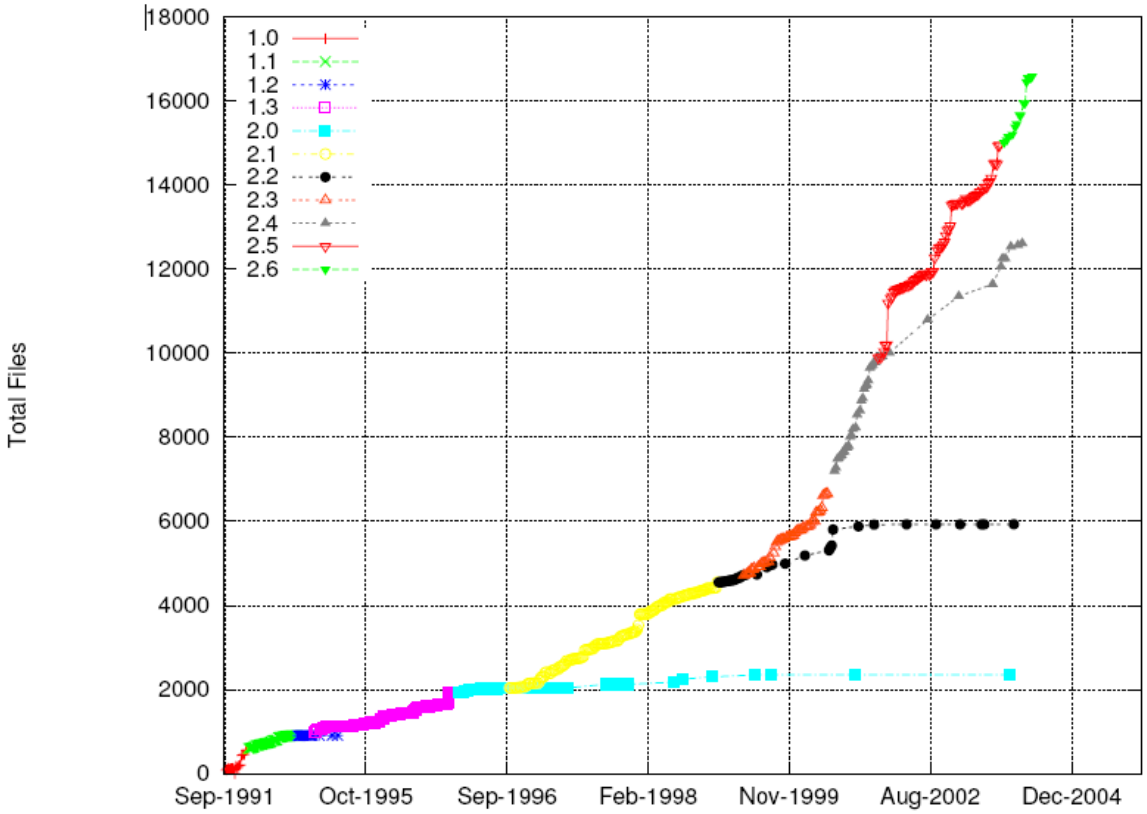
les composants sont plus complexes, et leur interrélations le sont encore plus

Des logiciels complexes. . .



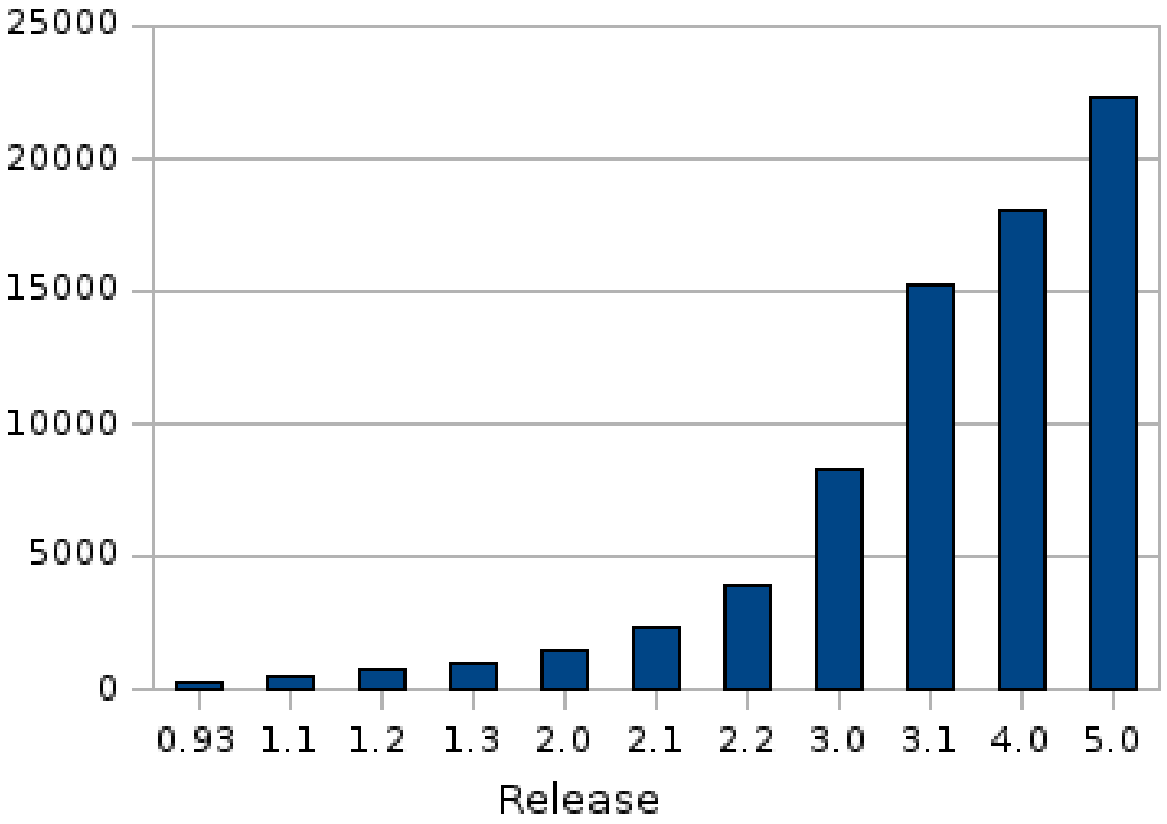

```
linux-2.6.16.20 > sloccount .
...
Total Physical Source Lines of Code (SLOC) = 4,827,227
Data generated using David A. Wheeler's 'SLOCCount'.
```

Un croissance superlinéaire



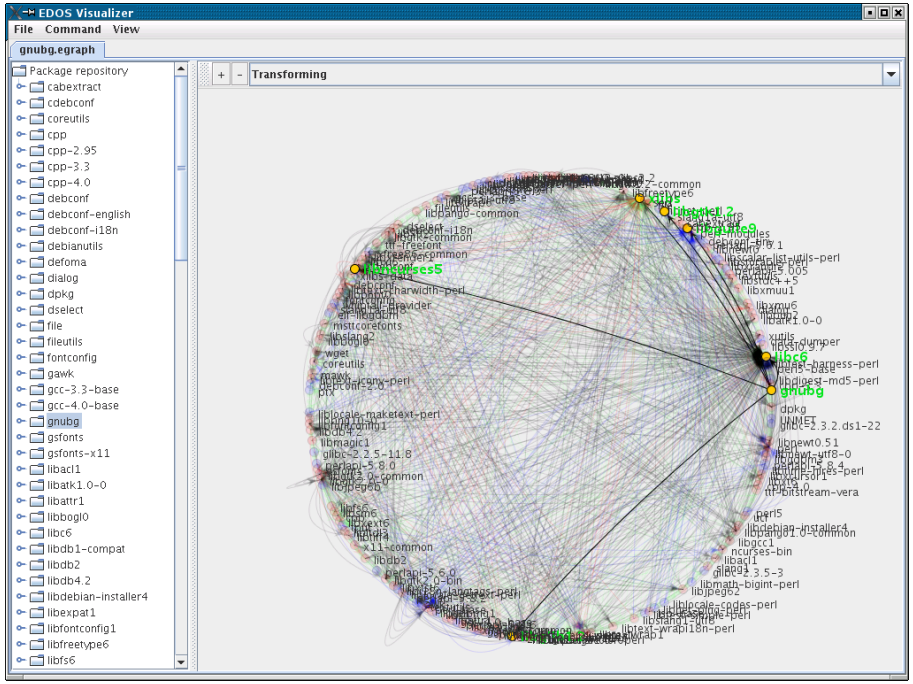
Nombre de fichiers dans le noyau Linux

Un croissance superlinéaire



Nombre de paquets dans Debian

Des interdépendances complexes. . .

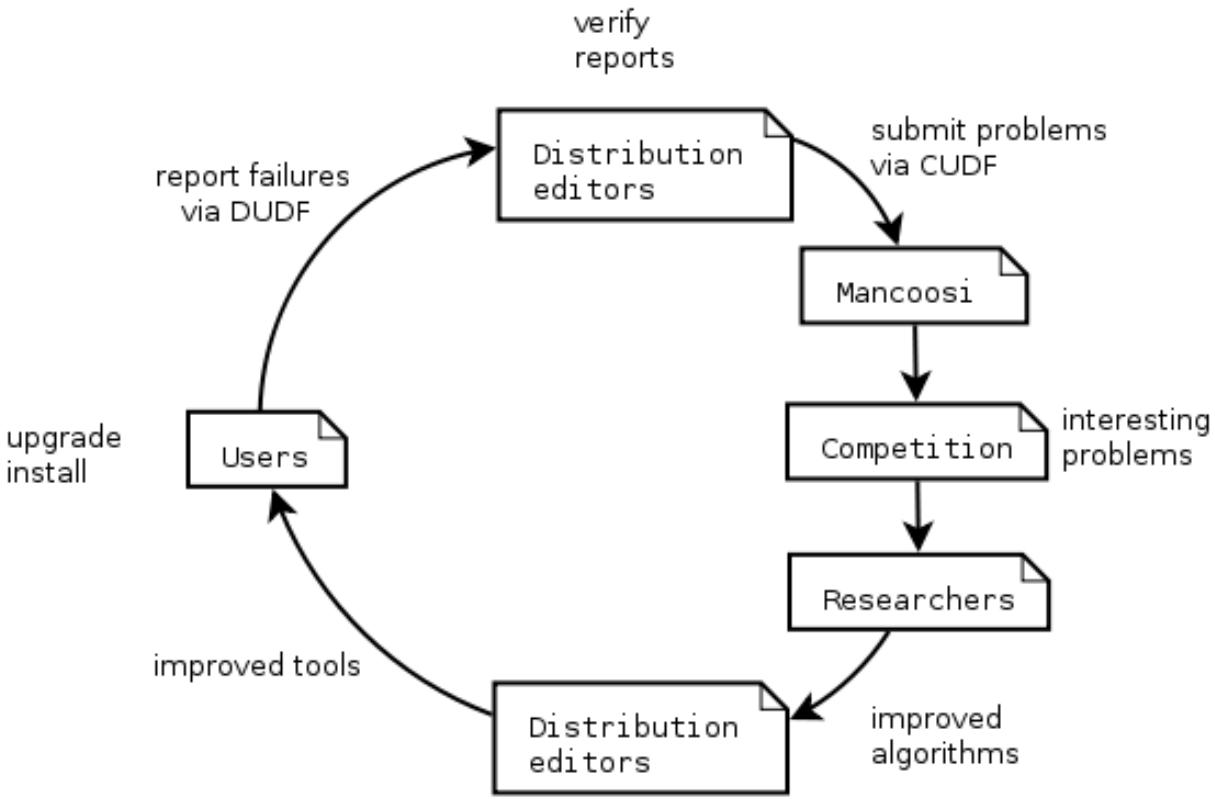



```
Package: gnubg
Version: 0.14.3+20060923-4
Depends: gnubg-data,
ttf-bitstream-vera, libartsc0
(>= 1.5.0-1), ..., libgl1-mesa-glx
| libgl1, ...
Conflicts: ...
```

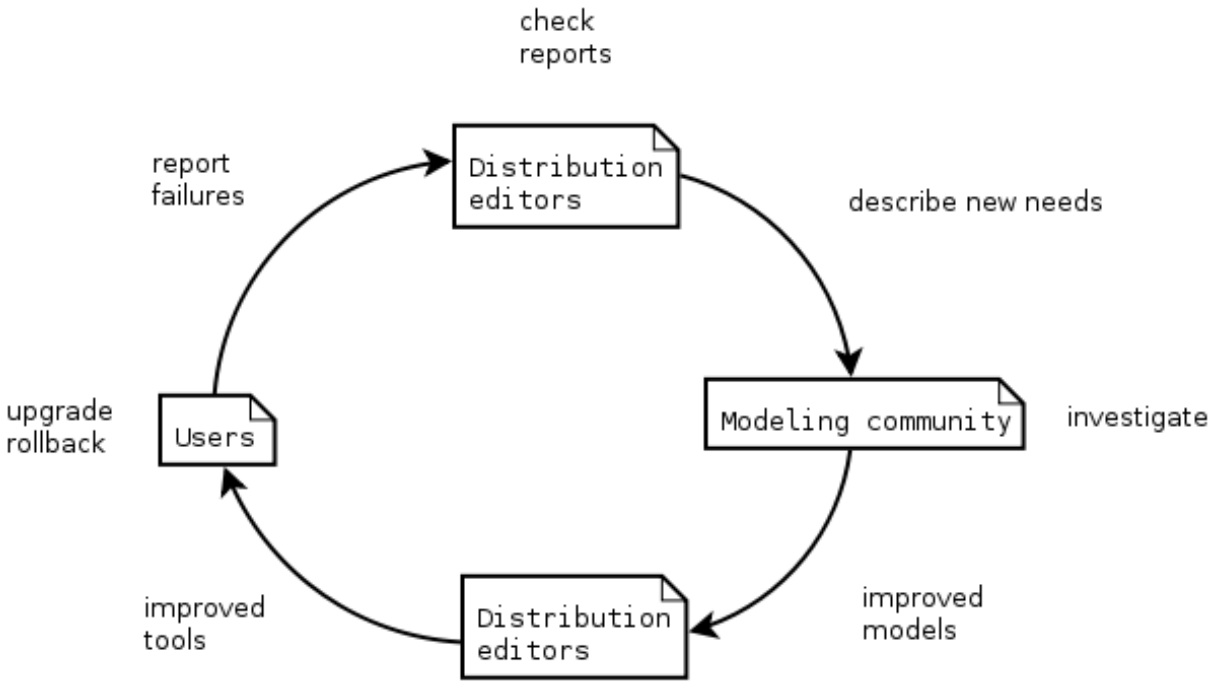
Cela change *tous les jours*! Comment s'y retrouver?

Ces problèmes sont au coeur des projets EDOS, et MANCOOSI.

Algorithmes et spécifications pour les mises à jour



Mises à jour transactionnelles

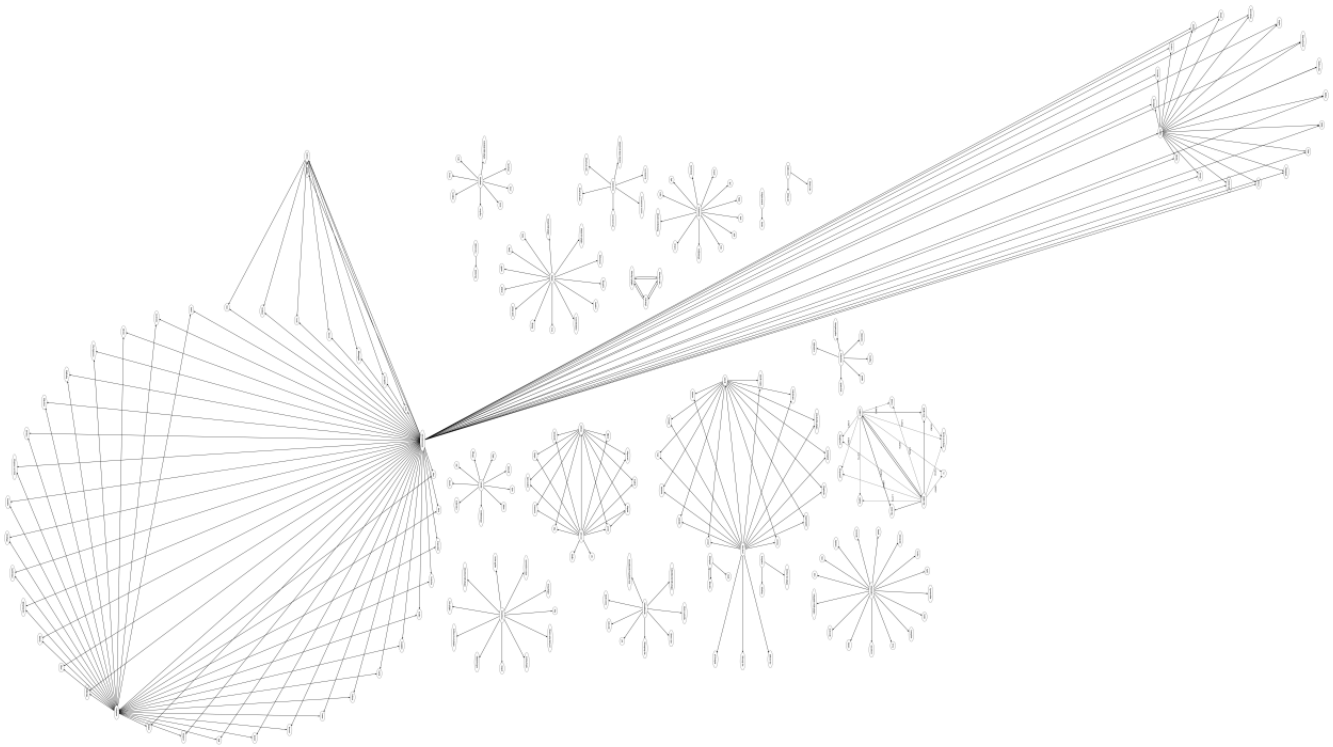


Visualisation de systèmes complexes, naïf

`kde-amusements` et `kde-toys` dans Debian 3.1 (aka *sarge*, 2005) se superposent à `kde-games` et `kde-edu`.Il ne sont plus là aujourd'hui (2009).
Il y a seulement 244 composants en jeu: on devrait pouvoir voir le problème...

Visualisation de systèmes complexes, 2ème essai

Voilà le graphe des *strong dominators* pour le sousystème KDE:



Plus d'info sur `http://www.mancoosi.org`

Le Logiciel Libre devient nécéssaire

Bien sur, notre arme première contre la complexité reste *l'abstraction*:

- open standards, application frameworks, librairies, patterns, MDA, MDE etc. fournissent des mechanismes d'abstraction

MAIS

les standards évoluent, les librairies contiennent des erreurs, les modèles peuvent être incomplets, les besoins peuvent changer. . . *vite, et souvent!*

L'accès au code source, et la possibilité de le surveiller, modifier, faire évoluer et distribuer devient *un beson primaire* de l'industrie du logiciel.

DONC

Le logiciel libre devient une necéssité *technique* et on doit former des *bons* professionnels du Logiciel Libre

L'Informatique a changé, il faut adapter l'enseignement

- généraliser l'enseignement *pour tous* déjà au lycée (voir `http://www.epi.asso.fr/revue/docu/d0912a.htm`) à ne pas confondre avec B2I et C2I: voir à ce sujet l'article du NYT du 21/12/2009 `http://bits.blogs.nytimes.com/2009/12/21/computer-science-education-its-not-shop-class/`)
- faire évoluer l'enseignement de l'Informatique dans le supérieur
  - créer des Masters spécialisés sur le Logiciel Libre, avec un curriculum adapté
  - plus généralement, repenser les cours traditionnels pour profiter du Logiciel Libre

  Voir le travail commencé pour un Curriculum Open Source sur `http://oscurr.v2.cs.unibo.it/`

IT education for the 21st century

Un bon ingénieur a une vie demandante...

- concevoir des systèmes rééls qui entrent en production
- comprendre des logiciels complexes,
  au moins ce qu'il faut pour pouvoir les adapter
- construire des systèmes complexes en réutilisant des composants existants
- collaborer avec d'autres dévéloppeurs, souvent opiniâtres

Et pourtant, on continue à le former en enseignant comme il y a 20 ans: un algorithme à la fois, un programme monlithique pour chaque projet, chaque étudiant en isolation.

Le Logiciel Libre peut nous aider à changer cette situation

Un ancien président de l'ACM le dit fort bien

Patterson, D. A. Computer science education in the 21st century. Commun. ACM 49, 3 (Mar. 2006), 27-30.

"Course I Would Love To Take #1: *Join the Open Source movement*... it would be inspiring for students working on real production software. Even companies that don't use open source software may benefit from students who can *do more than just write programs from scratch*." ...
"The recruiting pitch is to join CS and *learn in part by contributing immediately to the real world*.
To help *learn a large system*, writing documentation for portions of open source code could be an assignment. Documentation is important yet rare in the classroom and the open source movement."

Un example: enseigner l'algorithmique autrement

Revisons un cours introductif traditionnel à la programmation dynamique (en anglais)
Longest Common Subsequence (LCS)

> *given two sequences* $X = (x_1, x_2, \ldots x_n)$ *and* $Y = (y_1, y_2, \ldots y_m)$, *we whish to find a maximum length common subsequence of X and Y.*

For example, for X = BDCABA and Y = ABCBDAB, the sequence BCBA is such a common subsequence (BDAB is another one).
How do we find one?
Sould we enumerate all subsequences of X and Y, then find the common ones and pick a longest one?
Hey, that would require exponential time!

The algorithmic insight, 1

We remark that the LCS problem has an *optimal substructure* property:
for $X = (x_1, x_2, \ldots x_n)$ and $Y = (y_1, y_2, \ldots y_m)$, and $Z = z_1, \ldots, z_k$ an LCS

- if $x_n = y_m$ then $z_k = x_n = y_m$ and Z_{k-1} is an LCS of X_{n-1} and Y_{m-1}
- if $x_n \neq y_m$ then $z_k \neq x_n$ implies Z is an LCS of X_{n-1} and Y
- if $x_n \neq y_m$ then $z_k \neq y_m$ implies Z is an LCS of X and Y_{m-1}

The algorithmic insight, 2

So we can fill an n by m table $c[i,j]$ containing the length of the LCS of X_i and Y_j

$$c[i,j] = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & x_i > 0, y_j > 0, x_i = y_j \\ max(c[i,j-1], c[i-1,j]) & x_i > 0, y_j > 0, x_i \neq y_j \end{cases}$$

The algorithmic insight, 3

This can be done bottom up with the simple code that follows

```
for i = 1 to n do c[i,0] = 0
for j = 1 to m do c[0,j] = 0
for i = 1 to n do
  for j = 1 to m do
    if x[i]=y[j] then c[i,j] = c[i-1,j-1] +1
                 else c[i,j] = max(c[i,j-1], c[i-1,j])
```

Notice that:

- we can actually recover an LCS from the matrix c
- the algorithm runs in $O(mn)$ time
- the algorithm requires $O(mn)$ space

The algorithmic insight, 4

Many lecturers conclude "this is how the diff program works!"

really?

- Is $O(nm)$ an acceptable space and time complexity, *in practice*?
- Is diff *really* building an n by m array of *text lines*?
- Is diff *really* comparing *text lines*?

Is *the student* asking himself these fundamental questions?
With proprietary software, you would never know. . . . . .
With free software, things change radically!

A look at diff internals

```
apt-get source diffutils
cd diffutils-2.8.1/src
less analyze.c
...
/* The basic algorithm is described in:
"An O(ND) Difference Algorithm and its Variations", Eugene Myers,
Algorithmica Vol. 1 No. 2, 1986, pp. 251-266;
see especially section 4.2, which describes the variation used below.
Unless the --minimal option is specified, this code uses the TOO_EXPENSI
heuristic, by Paul Eggert, to limit the cost to O(N**1.5 log N)
at the price of producing suboptimal output for large inputs with
many differences.

The basic algorithm was independently discovered as described in:
"Algorithms for Approximate String Matching", E. Ukkonen,
Information and Control Vol. 64, 1985, pp. 100-118. */
```

A look at diff internals, 2

```
less io.c
...
/* Lines are put into equivalence classes of lines that match in lines_
Each equivalence class is represented by one of these structures,
but only while the classes are being computed.
Afterward, each class is represented by a number. */
struct equivclass
{
lin next; /* Next item in this bucket. */
hash_value hash; /* Hash of lines in this class. */
char const *line; /* A line that fits this class. */
size_t length; /* That line's length, not counting its newline. */
};

/* Hash-table: array of buckets, each being a chain of equivalence
classes. */
static lin *buckets;
```

A look at diff internals, 3

```
less analyze.c
...
/* Discard lines from one file that have no matches in the other file.

A line which is discarded will not be considered by the actual
comparison algorithm; it will be as if that line were not in the file.
The file's 'realindexes' table maps virtual line numbers
(which don't count the discarded lines) into real line numbers;
this is how the actual comparison algorithm produces results
that are comprehensible when the discarded lines are counted.

When we discard a line, we also mark it as a deletion or insertion
so that it will be printed in the output. */

static void
discard_confusing_lines (struct file_data filevec


)
```

Free software makes a difference

By looking at the *free source code* of a real-world, industry-strength implementation of the diff algorithm, our students have learned :

- a real-world program is much more than just *one* algorithm
  - optimize the common case (the $O(DN)$)
  - use hashing where appropriate (line equivalence classes)
  - reduce the size of the problem (remove lines that are not common)
- follow references to *freely accessible* research papers
- documentation, and comments, are essential to understand the code

Les défis à réléver

- organiser un CV Open Source:
  - utiliser le Logiciel Libre dans les anciens cours, en identifiant les spécificités *techniques* et*non techniques*;
  - écrire des livres de texte;
  - costruire l'infrastructure pour coordonner les efforts entre industriels, communautés et universités (GSC est très loin du compte)
- faire reconnaître les efforts: enseigner un cours avec du Logiciel Libre *peut* être *bien plus difficile* qu'un cours traditionnel, il *doit* être reconnu en conséquence

Les défis à réléver, bis

Passage à l'échelle:

- des bons professionnels IT sont recherchés partout
- les academiques compétents en Informatique sont sollicités de partout (biologie, nanosciences, mathematiques, physique, économie, etc. . .); pourtant, nous sommes une ressource *rare*, même si renouvelable

Tragedy of the commons in IT research and higher education

If everybody harvests, and nobody makes investments, there will be little left to free ride

Conclusions et recommandations

En résumé:

- il faut plus de professionnels compétents en Logiciel Libre
- ils ont besoin de connaissances *specifiques*
- on doit repenser le curriculum, pour former des bons professionnels en LL *grâce* aux cours, et plus *malgrés* les cours;
- les nouveaux cours vont demander un gros investissement;
- les *bons* enseignants/chercheurs in IT sont *une ressource rare*.

Recommandations

Les pouvoirs publiques et l'industrie doivent soutenir, reconnaître, financer et recompenser nos efforts pour une modernisation des cursus avec du Logiciel Libre, et investir *maintenant* dans l'enseignement et la recherche en Informatique.

Nous devons nous organiser pour que ces besoins soient reconnus et pris en compte rapidement.

Références

1. Patterson, D. A. Computer science education in the 21st century. Commun. ACM 49, 3 (Mar. 2006), 27-30.
2. Computer Science Education is Not Shop Class (enseigner l'Informatique ne se reduit pas au B2I et C2I) `http://bits.blogs.nytimes.com/2009/12/21/computer-science-education-its-not-shop-class/`
3. Ebauche de travail sur un curriculum Open Source: `http://oscurr.v2.cs.unibo.it/`