CoCoALib - Slug #417

ex-Normaliz2-overflow.in too long & too much memory

04 Dec 2013 16:46 - John Abbott

Status:	Closed	**Start date:**	04 Dec 2013
Priority:	Normal	**Due date:**	
Assignee:	Christof Soeger	**% Done:**	100%
Category:	Various	**Estimated time:**	4.50 hours
Target version:	CoCoALib-0.99532	**Spent time:**	4.50 hours

Description

[in the CoCoALib examples directory]

Is it correct that ex-Normaliz2 running on the input ex-Normaliz2-overflow.in should require over 20Gbytes of memory?

If so, can it be replaced by a simpler example which triggers overflow, but which is not so resource hungry. I had to do an emergency reboot after trying it :-(

Related issues:

Related to CoCoALib - Feature #213: test-Normaliz1.C	**Closed**	**31 Jul 2012**
Related to CoCoALib - Slug #479: make check in examples/ directory is far too...	**Closed**	**17 Mar 2014**

History

#1 - 04 Dec 2013 23:15 - Christof Soeger

Short answer: Yes and Yes!

ex-Normaliz2-overflow.in is really large, the purpose was to give an example where overflow occurs. The smart CoCoALib cone detects the overflow and starts it with gmp again.
It is a single simplex with 10^9 points that have to be computed. And at the moment normaliz will first create all these vectors, which is too much for the memory. This is a weak point that we will attack in the future.

One solution could be to use a other computation type, SupportHyperplane instead of HilbertBasis should work. (Didn't tested it.)

I have attached a smaller example that also causes an overflow but should run without problems with gmp. But here the overflow could be avoided, so it might happen that future versions of normaliz do not cause an overflow.

#2 - 04 Dec 2013 23:20 - Christof Soeger

- Status changed from New to In Progress

- % Done changed from 0 to 20

File upload gives some error, so here the example:

```
6 6
6 7 2 1 4 4
8 4 9 8 4 4
1 9 6 8 0 9
3 6 3 8 9 5
0 1 7 1 9 8
2 6 7 5 9 3
integral_closure
```

#3 - 06 Dec 2013 21:53 - John Abbott

- Category set to Cleaning

- % Done changed from 20 to 30

I confirm that the smaller example you sent causes overflow on my 64-bit MacOS machine.

I tried adding SupportHyperplanes to end of the previous file, but the computation still occupied about 3Gbytes after only 30secs, and looked set to grow further [I have only 4Gbytes of RAM].

I have renamed the old overflow example ex-Normaliz2-big-and-slow.in :-)

#4 - 07 Dec 2013 12:44 - Christof Soeger

The CoCoA example reads only the matrix from the file, everything else is ignored. I added the last line to run it with "native" normaliz.
The decision what to compute is also in the ex-Normaliz2.C, there you have to replace HilbertSeries by SupportHyperplanes in line 62.

#5 - 07 Dec 2013 17:53 - John Abbott

Should I make a new example (ex-Normaliz3) which computes the support hyperplanes; then the "big" input can be renamed as ex-Normaliz3.in.

Or is there really no point in doing so?

Remember these are supposed to be clear examples of how to use features in CoCoALib; they are not intended to be thorough tests.

#6 - 09 Dec 2013 10:46 - Christof Soeger

Since it is just to demonstrate the behaviour when overflow occurs, I think it is okay to simply change it to SupportHyperplanes and maybe keep both example inputs.
In ex-Normaliz1.C all the possible computations are shown.

#7 - 02 Apr 2014 09:58 - Anna Maria Bigatti

- Project changed from CoCoA-5 to CoCoALib

- Category deleted (Cleaning)

#8 - 02 Apr 2014 10:04 - Anna Maria Bigatti

- Category set to Various

- Status changed from In Progress to Feedback

- Assignee set to Christof Soeger

- Target version set to CoCoALib-0.99532

- % Done changed from 30 to 90

I think this is now to be closed?

#9 - 02 Apr 2014 16:50 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

No problems after 4 months since the update (forgot to update redmine).
So closing!

#10 - 03 Apr 2014 11:55 - Anna Maria Bigatti

- Estimated time set to 4.50 h

#11 - 03 Apr 2014 17:30 - Winfried Bruns

With Normaliz 2.11 this will no longer be an issue of memory. But the computation time could still be too big.