Reduce Report

RunJie Ma

4 September 2018

1 Introduction on The Reduce Work

During the first experiment work, we are mainly based on the pdf of lecture 2 and finish the work of translate the whole pseudo-code to the reduce code. I will mainly focus on the difficulties when I try to solve the problem. At the end of the report, we will give another example using the polynomials in the Hea79.

1.1 Aim

The aim of this exercise is to implement various g.c.d. algorithms in Reduce. You are writing in ordinary Reduce (not the "symbolic mode described in Chapter 17 of [HS18]. You should implement functions for g.c.d. of polynomials in several variables, using, recursively, these algorithms as defined in Lecture 2. Note that these algorithms (slide 11 etc.) fulfill the role of "Some Pseudo Euclid" in the Gauss algorithm on slide 8. To do g.c.d. of polynomials in several variables (say x, y, z) you let them be in Z[z][y][x] and use the Gauss algorithm in R[x], where R = Z[z][y] etc. A lot of this infrastructure will be common to all implementations, and I expect you to cut and paste it from one file to the next.

1.2 Achievement

I achieve to write 5 GCD programs in three variables based on the thought of the Gauss Algorithm.

2 Basic Function

This part is to introduce three basic function that we are used among these different GCD function.

2.1 myCont

We do the work basically finish two functions, the coefficient list return all the coefficients of any polynomials in terms of a list and the other one get the gcd of the list giving above.

2.1.1 Coefficient List

```
CoefficientList(Poly, Var)
{
    while(deg(poly, var) != 0)
    {
        cof := coefficient of term with the highest degree;
        res := res . cof;
        poly := poly - cof*(var^deg(poly));
    }
    if(isempty(res)) return 0;
    else return reverse res;
}
```

2.1.2 GCDCoef

```
GCDCoef(List)
{
    bool flag := false, res:= 1;
    if(isEmpty(List))return 1;

    while(isNotEmpty(List))
    {
        if(First List == 0) List := rest List;
        else if(flag == false)
        {
            flag := true;
            res := First List;
            List := rest List;
        }
        else
        {
            res := GCD(res, First List);
            List := rest List;
        }
    }
    return res;
}
```

After implementing these two functions, we can easily see that the myCont function isto call the CoefficientList function first and then use the GCDCoef function to get the answer.

2.2 Primitive Part

The PP function return the pp form of a function. The only thing that we should pay attention to is that we should deal with the 0 on the divisor.

```
PP(poly)
{
    if(poly == 0) return 0;
    return poly/myCont(poly);
}
```

2.3 Divide Polynomial

This Function is used in the Hearn Algorithm which aims at simplifying a polynomial with a list and then insert into it. Attention: judging the first list == 0 is not so necessary but if not there is always some unnecessary situations happening when we test.

```
DividePoly(poly, list)
{
    if(poly == 0) return 0;
    while(isNotEmpty(list))
    {
        if(first xs == 0) list := rest list;
        else if(mod(poly, first list) == 0)
        {
            poly := poly/first list;
            list := rest list;
        }
        else list := rest list;
    }
    return poly;
}
```

Actually, there is another thing to be announced that is the mod function. The polynomial and the content function can be two different situations. So we should call different functions to deal with them. We will omit it here.

2.4 NumericsGCD

As we know the gaussGCD can get the GCD of any two polynomials or content. But sometimes we are willing to use some simpler situations like the GCD of two number.

```
NumericGCD(a,b)
{
    if a == 0 return b;
    else if b == 0 return a;
    else return NumericGCD(b, a mod b);
}
```

We should also pay attention to the situation that a ¡ 0 or b ¡ 0, or even a or b == 0.

3 Main Part

3.1 The common factor

This part is about the main part of our function implementation.

First our main goal is to implement a GaussGCD function which can get the GCD of the polynomial in Z[x][y][z]. We will quote the pseudo-code in the pdf.

Algorithm (Gauss's algorithm)

```
1:  procedure GAUSS(a, b)                          ▷ The g.c.d. of a, b ∈ R[x]
2:      if b = 0 then
3:          return a
4:      end if
5:      if a = 0 then
6:          return b
7:      end if
8:      c_a ← cont(a)                              ▷ g.c.d. in R needed
9:      c_b ← cont(b)                              ▷ g.c.d. in R needed
10:     c_g ← gcd(c_a, c_b)                        ▷ g.c.d. in R needed
11:     p_g ← Some PseudoEuclid(a, b)
12:     g ← c_g * pp(p_g)                          ▷ g.c.d. in R needed
13:     return g
14: end procedure
```

So the difference between each method is about the how to achieve the pseudoEuclid function.

Algorithm (Primitive Pseudo-Euclid's algorithm)

```
1:  procedure PPEUCLID(a, b)                       ▷ The primitive g.c.d. of
    a, b ∈ R[x]
2:      if b = 0 then
3:          return a
4:      end if
5:      r ← pp(prem(a, b))                         ▷ These pp are the only difference
6:      while r ≠ 0 do                             ▷ We have the answer if r is 0
7:          a ← b
8:          b ← r
9:          r ← pp(prem(a, b))                     ▷ These pp are the only difference
10:     end while
11:     return b                                   ▷ The gcd is b, up to a factor in R
12: end procedure
```

Algorithm (SR-Euclid's algorithm)

```
1: procedure SREUCLID(f, g) ▷ Almost the g.c.d. of a, b ∈ R[x]
2:     if deg(f) < deg(g) then
3:         a_0 ← pp(g); a_1 ← pp(f);
4:     else
5:         a_0 ← pp(f); a_1 ← pp(g);
6:     end if
7:     δ_0 ← deg(a_0) − deg(a_1);
8:     β_2 ← (−1)^{δ_0+1}; ψ_2 ← −1; i ← 1;
9:     while a_i ≠ 0 do
10:        a_{i+1} = prem(a_{i−1}, a_i)/β_{i+1};
11:        δ_i ← deg(a_i) − deg(a_{i+1}); i ← i + 1;
12:        ψ_{i+1} ← (− lc(a_{i−1}))^{δ_{i−2}} ψ_i^{1−δ_{i−2}};
13:        β_{i+1} ← − lc(a_{i−1}) ψ_{i+1}^{δ_{i−1}};
14:    end while
15:    return pp(a_{i−1})        ▷ The gcd is this, up to a factor in R
```

Algorithm (Hearn-Euclid's algorithm)

```
1: procedure HEUCLID(a, b)  ▷ Almost the g.c.d. of a, b ∈ R[x]
2:     if b = 0 then
3:         return a
4:     end if
5:     r ← prem(a, b)
6:     l ← {lc(a), lc(b), lc(r)}              ▷ All leading coefficients
7:     while r ≠ 0 do                  ▷ We have the answer if r is 0
8:         a ← b; b ← r
9:         r ← prem(a, b)
10:        while any element of l divides r do cancel it
11:        end while
12:        l ← l ∪ {lc(r)}
13:    end while
14:    return b                    ▷ The gcd is b, up to a factor in R
15: end procedure
```

What we actual do is that we translate these pseudo-code into the reduce code. There are some many problems we meet when we translate them. I will point out some.

3.1.1 Assigning problem:

when we load the code from a file, it would be so strange the reduce do some assignment that we do not expect to. For example, sometimes the reduce let all the content equal to zero, which is ridiculous. We can not let it happen again, but when it happen, there are always some ridiculous assignment but it is different each time.

3.1.2 Function problem:

One terrible thing is that we always waste much time on debugging one piece of code, but when we restart the reduce terminal, the bug has gone away. I do not why it happened, but I guess both these two problems is caused by the memory leakage.

3.2 JHD Improvement

Actually the best way to improve the list is to use the factorize function. However it is not allowed to use. And it is not hard to find out that achieve this function seem to be much more difficult than the original function. So we give up this way.

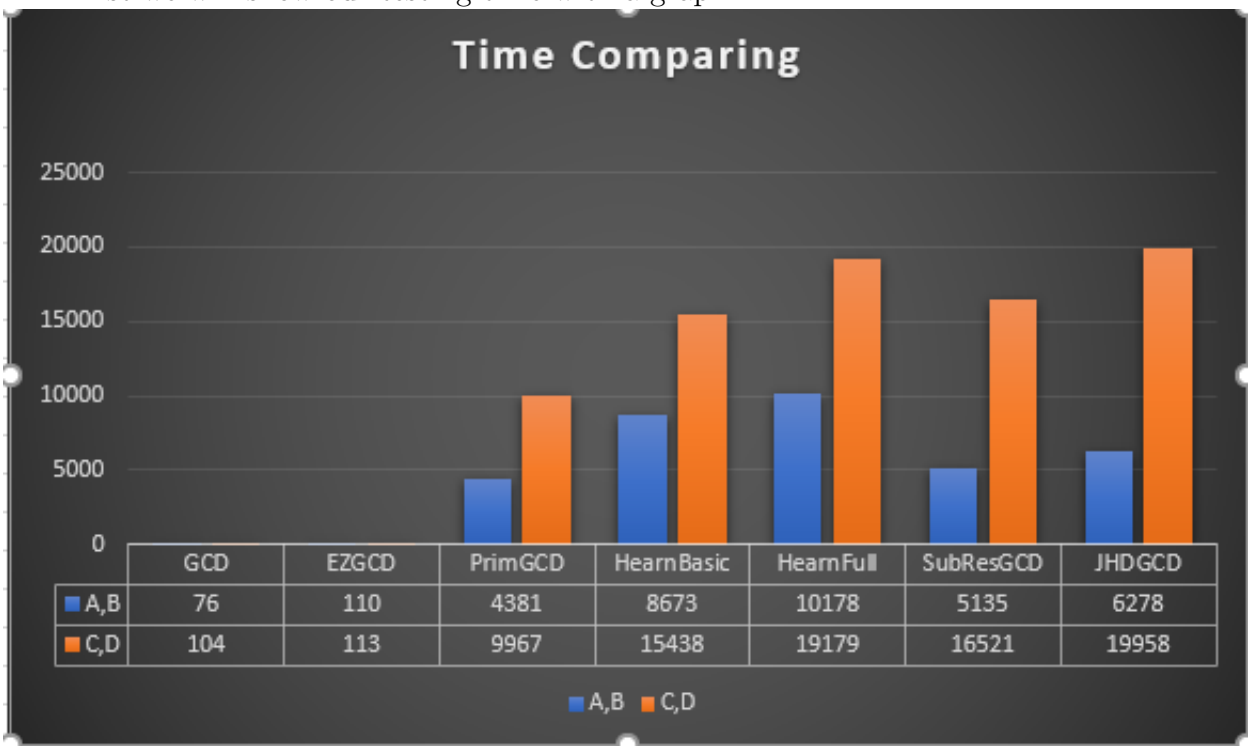
But as we can see in the pdf, the bad situation always happens when the bigger one in the list has been the divisor first. And it is better to store a relatively small one in the list. So here comes the idea.

First, when we want to insert one element into the list, we can detect if any element of the list can divide the element. If any can , do it. If not, we will continue the next element. When we finish the dividing process, we can insert it by the degree. It means we would like to store the polynomials of high degree first implemented by using another temp list to store the elements which has higher degrees than the one we want to insert. It may be better if the elements in the list is ordered by the degree from high to low.

Then it would be my final version of the JHD.

4 Research Question

First we will show our testing time with a graph



The time above is measured by the mini second and we run the GCD function for 1000 times to get the answer.

4.1 Q1:

Compared with the built-in function named GCD, we can see that the built-in function is much more faster than ours. I figure out two main parts which could low the speed.

4.1.1 The Built-in System

The built-in function could be achieved by writing Lisp which is much more efficient than what we think. However, we do not what the deg function actually does. I think there are some redundant work that deg function does so that we pay too much on it.

4.1.2 Algorithm

The next part is about that the algorithm we use is not a high-efficient one. Also these two polynomials maybe seem not be too friendly to the GCD functions. Maybe using other testing examples would be better especially the JHDGCD function. However, one more possible reason is that the built-in algorithm can deal with different situation in different method so that it can be efficient.

4.2 Q2:

Between the five method, the trends seem to be a little weird. The PrimGCD works really well then the SubResGCD, and finally the Hearn's method. Between the three ways of the Hearn's methods, we do some optimization from the HearnBasic to the HearnFull, and finally to the JHDGCD.

The actual computing method is quite different between the Prim, Hearn and the SubRes. In multiple testing situaions, the prim always works the best. The Hearn works the worse. But I still think there are some examples to make different functions work the best.

4.3 Paper Polynomials

Here we would like to show the results of using the polynomials in the paper Hea79.

4.4 First

We will use the polynomial below.

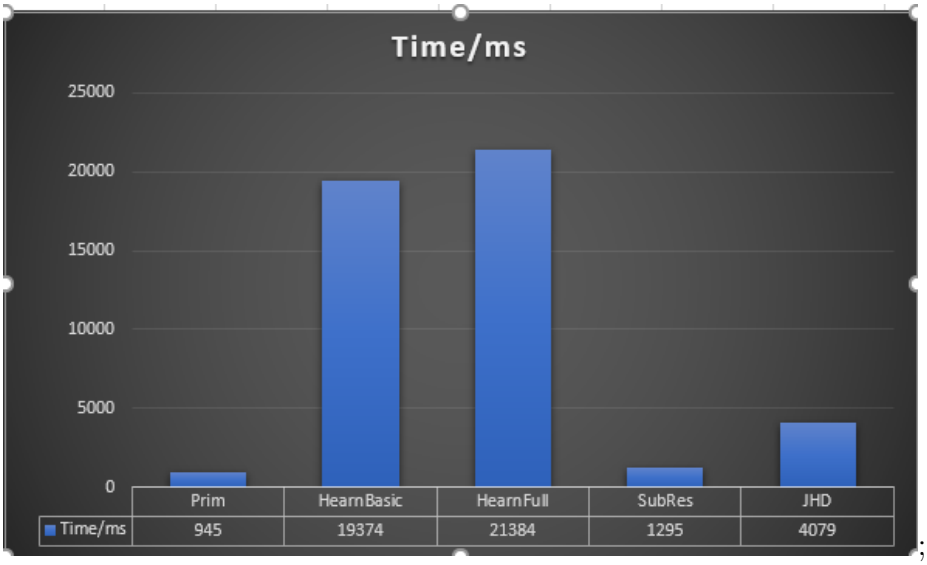
```
8: f;
```

$$x^2 + 2\,x\,y + x\,z + x + y^2 + y\,z + y - z - 2$$

```
9: g;
```

$$-3\,x^3\,y - 3\,x^2\,y^2 - 3\,x^2\,y + x\,y^2 - x + y^3 + y^2 - y - 1$$

The result is shown as below.(running 50 times)



;

4.5 Second

We will use the polynomial below.

```
9: h;
```

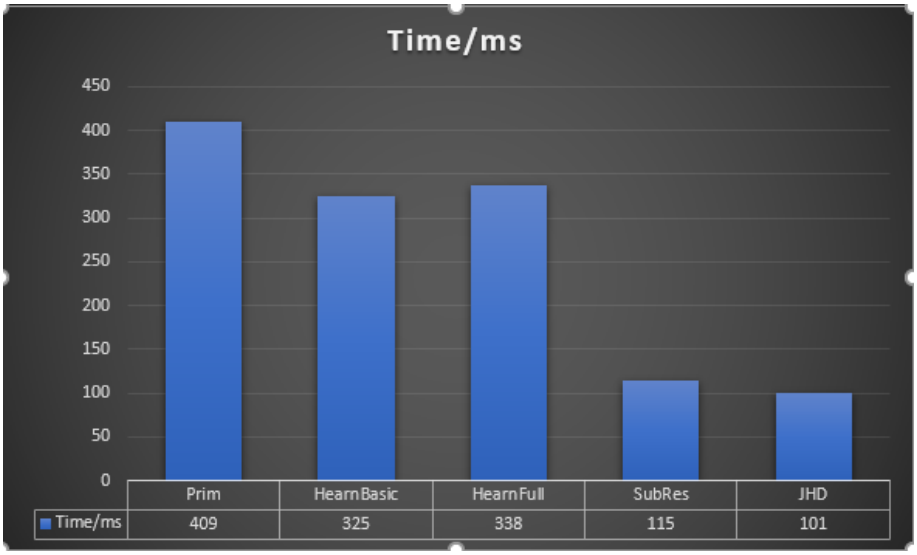
$$x^2 + y^2 + z^2 + 1$$

```
Time: 15 ms

10: j;
```

$$x^4 + 2\,x^2\,y^2 + 2\,x^2\,z^2 - x^2 + y^4 + 2\,y^2\,z^2 - y^2 + z^4 - z^2 - 2$$

The result is shown as below.(running 50 times)



4.6 Third

We will use the polynomial below.

```
2: l;
```

$$x^2 y^2 + x^2 + y^2 + z^2 + 1$$

```
3: m;
```

$$x^4 y^4 + 2 x^4 y^3 + 2 x^4 y^2 + 2 x^4 y + x^4 + 2 x^3 y^4 + 2 x^3 y^3 z + 6 x^3 y^3 + 2 x^3 y^2 z + 6 x^3 y^2 + 2 x^3 y z + 6 x^3 y + 2 x^3 z + 4 x^3 + 2 x^2 y^4 + 2 x^2 y^3 z + 6 x^2 y^3 + 2 x^2 y^2 z^2 + 4 x^2 y^2 z + 7 x^2 y^2 + 2 x^2 y z^2 + 2 x^2 y z + 6 x^2 y + 2 x^2 z^2 + 4 x^2 z + 5 x^2 + 2 x y^4 + 2 x y^3 z + 6 x y^3 + 2 x y^2 z^2 + 2 x y^2 z + 6 x y^2 + 2 x y z^2 + 6 x y z + 2 x z^2 + 4 x z + 2 x + y^4 + 2 y^3 z + 4 y^3 + 2 y^2 z^2 + 4 y^2 z + 5 y^2 + 2 y z^2 + 4 y z + 2 y z + 4 y + z^4 + 4 z^3 + 5 z^2 + 4 z + 4$$

The result is shown as below.(running 50 times)

```
2: l;
```

$$x^2 y^2 + x^2 + y^2 + z^2 + 1$$

```
3: m;
```

$$x^4 y^4 + 2 x^4 y^3 + 2 x^4 y^2 + 2 x^4 y + x^4 + 2 x^3 y^4 + 2 x^3 y^3 z + 6 x^3 y^3 + 2 x^3 y^2 z + 6 x^3 y^2 + 2 x^3 y z + 6 x^3 y + 2 x^3 z + 4 x^3$$
$$+ 2 x^2 y^4 + 2 x^2 y^3 z + 6 x^2 y^3 + 2 x^2 y^2 z^2 + 4 x^2 y^2 z + 7 x^2 y^2 + 2 x^2 y z^2 + 2 x^2 y z + 6 x^2 y + 2 x^2 z^2 + 4 x^2 z + 5 x^2 +$$
$$2 x y^4 + 2 x y^3 z + 6 x y^3 + 2 x y^2 z^2 + 2 x y^2 z + 6 x y^2 + 2 x y z^3 + 6 x y z^2 + 2 x y z + 6 x y + 2 x z^3 + 4 x z^2 + 2 x z + 4 x + y^4 +$$
$$2 y^3 z + 4 y^3 + 2 y^2 z^2 + 4 y^2 z + 5 y^2 + 2 y z^3 + 4 y z^2 + 2 y z + 4 y + z^4 + 4 z^3 + 5 z^2 + 4 z + 4$$

4.7 Some common thing about them

We should know that the Testing multiple times can lead to a serious memory leakage in reduce. And another essential thing is that when we are calling the Hearn's function, the sequential finding could be a big problem on time costing. Using the order of the partial does not make it better.
The first two polynomials is really in a good design.

5 conclusion

Here I will state some feelings about finishing the project. I think the main difficulty is there is so less information about the reduce so that it would be really tough to finish the project. The algorithm seems to be really fun and the hearn's method is really funny.