Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES
(COMMON TO CSE & IT)

UNIT No. 1:APPLICATIONS OF LIST
POLYNOMIAL MANIPULATIONS

Version: 1.XX

Representing a polynomial using a linked list:

A polynomial can be represented in an array or in a linked list by simply storing the coefficient and exponent of each term. However, for any polynomial operation , such as addition or multiplication of polynomials , you will find that the linked list representation is more easier to deal with.

First of all note that in a polynomial all the terms may not be present, especially if it is going to be a very high order polynomial.

Consider, $5x^{12} + 2x^9 + 4x^7 + 6x^5 + x^2 + 12\ x$. Now this 12th order polynomial does not have all the 13 terms (including the constant term). It would be very easy to represent the polynomial using a linked list structure, where each node can hold information pertaining to a single term of the polynomial.

Each node will need to store the variable x, the exponent and the coefficient for each term. It often does not matter whether the polynomial is in x or y. This information may not be very crucial for the intended operations on the polynomial.

Thus we need to define a node structure to hold two integers , viz. exp and coff. Compare this representation with storing the same polynomial using an array structure. In the array we have to have keep a slot for each exponent of x, thus if we have a polynomial of order 50 but containing just 6 terms, then a large number of entries will be zero in the array. You will also see that it would be also easy to manipulate a pair of polynomials if they are represented using linked lists.

Addition of two polynomials:

Consider addition of the following polynomials

$5\ x^{12} + 2\ x^9 + 4x^7 + 6x^6 + x^3$

$7\ x^8 + 2\ x^7 + 8x^6 + 6x^4 + 2x^2 + 3\ x + 40$

The resulting polynomial is going to be

$5\ x^{12} + 2\ x^9 + 7\ x^8 + 6\ x^7 + 14x^6 + 6x^4 + x^3 + 2x^2 + 3\ x + 40$

Now notice how the addition was carried out. Let us say the result of addition is going to be stored in a third list. We started with the highest power in any polynomial. If there was no item having the same exponent , we simply appended the term to the new list, and continued with the process.

Wherever we found that the exponents were matching, we simply added the coefficients and then stored the term in the new list. If one list gets exhausted earlier and the other list still contains some lower order terms, then simply append the remaining terms to the new list.

Now we are in a position to write our algorithm for adding two polynomials.

Let phead1 , phead2 and phead3 represent the pointers of the three lists under consideration.

Let each node contain two integers exp and coff .

Let us assume that the two linked lists already contain relevant data about the two polynomials.

Also assume that we have got a function append to insert a new node at the end of the given list.

```
p1 = phead1;
p2 = phead2;
```

Let us call malloc to create a new node p3 to build the third list

```
p3 = phead3;
 /* now traverse the lists till one list gets exhausted */
while ((p1 != NULL) || (p2 != NULL))
/ * if the exponent of p1 is higher than that of p2 then the next term in final list is going to be the node of p1* /
 while (p1 ->exp > p2 -> exp )
{
 p3 -> exp = p1 -> exp;
p3 -> coff = p1 -> coff ;
 append (p3, phead3);
 /* now move to the next term in list 1*/
 p1 = p1 -> next; }
 / * if p2 exponent turns out to be higher then make p3 same as p2 and append to final list*/
while (p1 ->exp < p2 -> exp )
 { p3 -> exp = p2 -> exp;
 p3 -> coff = p2 -> coff ;
 append (p3, phead3);
 p2 = p2 -> next; }
```

```
 /* now consider the possibility that both exponents are same , then we must add the coefficients
to get the term for the final list */
while (p1 ->exp = p2 -> exp )
{
 p3-> exp = p1-> exp;
 p3->coff = p1->coff + p2-> coff ;
 append (p3, phead3) ;
 p1 = p1->next ;
p2 = p2->next ;
}
 }
 /* now consider the possibility that list2 gets exhausted , and there are terms remaining only in
list1. So all those terms have to be appended to end of list3. However, you do not have to do it
term by term, as p1 is already pointing to remaining terms, so simply append the pointer p1 to
phead3 */
if ( p1 != NULL)
append (p1, phead3) ;
else append (p2, phead3);
```

Now, you can implement the algorithm in C, and maybe make it more efficient.

Polynomial Manipulation – Merge, Traversal

The linked list data structure supports the following operations.

We assume that the location of the first node is given by the special vairable called head. x

Insert(x, a) : Insert x next to element a in the list. x

Remove(x): Remove element x from the list. x

Find(x) : Find and return, if present, the location of element x. x

Print() : Print the contents of the list.

Using C style pointers,

The following is the pseudocode for the operations.

We assume that we have a structure such as:

```
Struct node {
int value;
struct node *next;
}
```

With the above definition of a structure, let us implement the above operations.

Given a linked list via the head pointer, let us first implement the operations Find and Print.

Algorithm

```
Find(x)
begin temphead = head;
while (temphead != NULL)
do
if temphead ->data == x
then return temphead;
temphead = temphead ->next;
end-while return NULL;
end
Algorithm Print()
begin temphead = head;
while (temphead != NULL)
Print(temphead ->data);
temphead = temphead ->next;
end-while
end
```

The insertion and removal of items from a linked list requires that we have a pointer to the place of insertion/deletion. To insert, where do we insert? Several options possible x insert at the beginning of the list

Addition of Polynomials:

To add two polynomials we need to scan them once. If we find terms with the same exponent in the two polynomials, then we add the coefficients; otherwise, we copy the term of larger exponent into the sum and go on. When we reach at the end of one of the polynomial, then remaining part of the other is copied into the sum.

To add two polynomials follow the following steps:

• Read two polynomials.

• Add them.

• Display the resultant polynomial.

Addition of Polynomials:

```
void add()
 { poly *ptr1, *ptr2, *newnode;
 ptr1=list1; ptr2=list2;
while(ptr1!=NULL && ptr2!= NULL)
{
 newnode=malloc(sizeof(struct poly));
 if(ptr1->power==ptr2->power)
 { newnode->coeff = ptr1->coeff + ptr2->coeff;
newnode->power=ptr1->power;
newnode->next=NULL;
list3=create(list3,newnode);
ptr1=ptr->next; ptr2=ptr2->next;
}
else
{
 if(ptr1->power > ptr2->power)
{
newnode->coeff = ptr1->coeff;
 newnode->power=ptr1->power;
newnode->next=NULL;
```

```
list3=create(list3,newnode);
 ptr1=ptr1->next;
 }
else
 {
newnode->coeff = ptr2->coeff;
newnode->power=ptr2->power;
newnode->next=NULL;
 list3=create(list3,newnode);
 ptr2=ptr2->next;
 }
 }
 }
```

FOR SUBTRACTION OF POLYNOMIALS :

add this statement in the above program

newnode->coeff = ptr1->coeff - ptr2->coeff

MULTIPLICATION OF POLYNOMIALS:

Multiplication of two polynomials however requires manipulation of each node such that the exponents are added up and the coefficients are multiplied. After each term of first polynomial is operated upon with each term of the second polynomial, then the result has to be added up by comparing the exponents and adding the coefficients for similar exponents and including terms as such with dissimilar exponents in the result

```
 void Mul()
 { poly *ptr1, *ptr2, *newnode;
 ptr1=list1; ptr2=list2;
 if(ptr1 == NULL && ptr2 == NULL)
return;
```

```
if(ptr1 == NULL) // I polynomial does not exist
 list3 = list2;
else if(ptr2 ==NULL) // II polynomial does not exist
 list3 =list1;
 else // Both polynomial exist
{
 if(ptr1!=NULL && ptr2!= NULL)
{ while(ptr1!=NULL)
 {
 newnode=malloc(sizeof(struct poly));
while(ptr2!=NULL)
 {
newnode->coeff = ptr1->coeff * ptr2 ->coeff;
newnode->power=ptr1->power + ptr2->power;
list3=create(list3,newnode);
ptr2=ptr2->next;
}
```