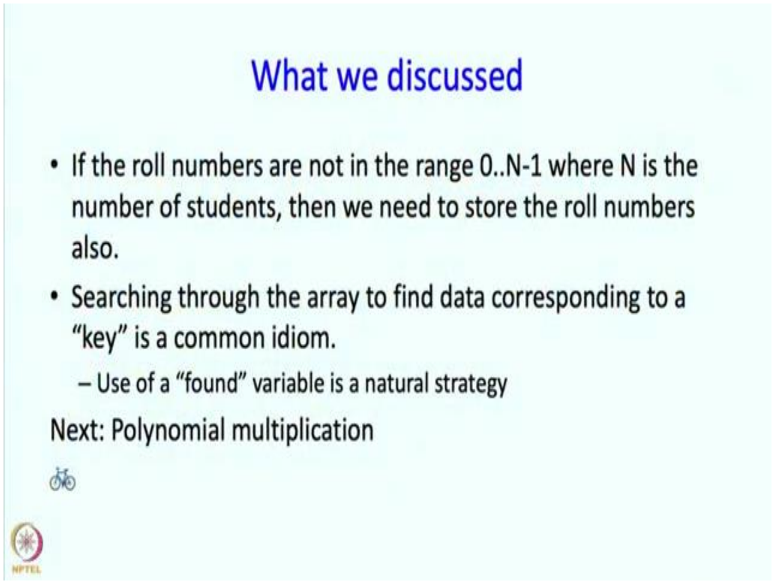
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering IIT Bombay
Lecture No. 15 Part-5
Array Part-1
Polynomial multiplication

(Refer slide time: 00:19)



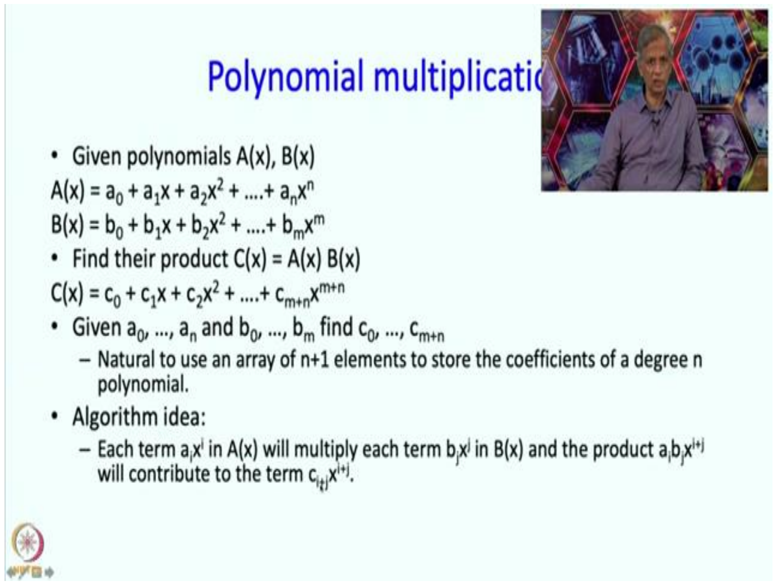
What we discussed

- If the roll numbers are not in the range 0..N-1 where N is the number of students, then we need to store the roll numbers also.
- Searching through the array to find data corresponding to a "key" is a common idiom.
  - Use of a "found" variable is a natural strategy

Next: Polynomial multiplication

Welcome back in the last segment we discussed a variation on the marks display program in which we needed to store the roll numbers as well as the marks. Now we are going to look at another interesting use of arrays namely for polynomials representing polynomials and operating upon them.

(Refer slide time: 00:39)



Polynomial multiplicatio

- Given polynomials A(x), B(x)

$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$

$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_mx^m$

- Find their product C(x) = A(x) B(x)

$C(x) = c_0 + c_1x + c_2x^2 + \dots + c_{m+n}x^{m+n}$

- Given $a_0, \dots, a_n$ and $b_0, \dots, b_m$ find $c_0, \dots, c_{m+n}$
  - Natural to use an array of n+1 elements to store the coefficients of a degree n polynomial.
- Algorithm idea:
  - Each term a_ix^i in A(x) will multiply each term b_jx^j in B(x) and the product $a_ib_jx^{i+j}$ will contribute to the term $c_{i+j}x^{i+j}$.

So let us say we are given 2 polynomials A(x) and B(x) this is what A(x) might look like $a_0+a_1x+a_2x^2+a_nx^n$. And this what B(x) might look like $b_0+b_1x+b_2x^2+b_mx^m$. In general m may not be equal to n we would like to find the product C(x) equals A(x) times B(x), so you know the product is going to look like this. There is going to be a constant term $c_0+c_1x+c_2x^2+a_{m+n}x^{m+n}$.

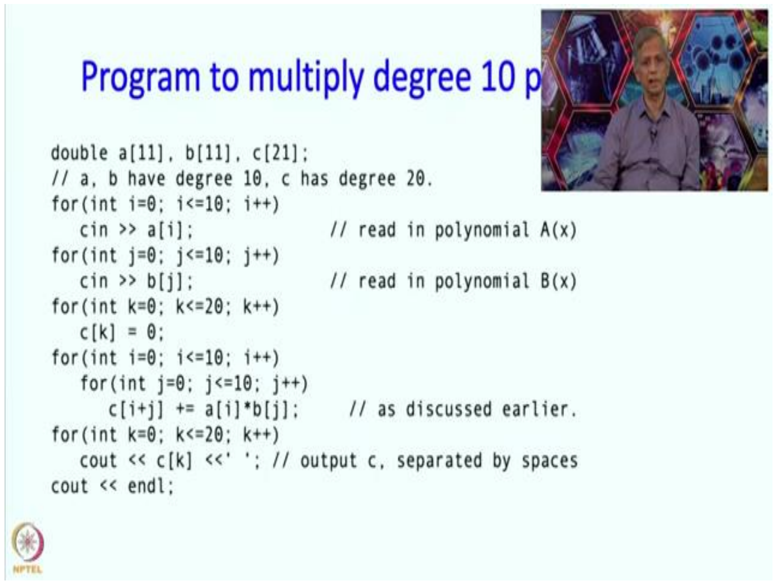
So there are going to be m plus n plus 1 coefficients, So basically what is the problem? As input we are given this numbers a0 a1 all the way till an so this numbers are given to us. You are also given this numbers and we are supposed to find these numbers. So we are given n plus 1 numbers this m plus 1 numbers this and we are supposed to find m plus n plus numbers which are this over here. So the question is-how to we do that? And first where do we store this numbers? So by now the answer should of this you should store this numbers in an array so, it is a natural to use an array of n plus 1 elements to store the coefficients of a degree n polynomial.

In fact we should store a_i in array element with index i. So now we are supposed to generate the c_i, so how do we do that? So here is sort of the simple idea, so when we do this multiplication every term over here is going to multiply every term over here so in particular let us say a_ix^i is a term in A(x) now that will multiply some term b_jx^j in B(x).

And if you just look at the product of this two terms that will be aibj times x to the power i plus j what will be the effect of this term? So this is going to contribute to the coefficients of x to the power i plus j, and what is that coefficients? That coefficients is c sub i plus j. So basically the idea over is that we form every such possible product and then we look at well, if this first the coefficient of x to the i if this was the coefficient of x to the j.

Then the product that we calculate should contribute to the coefficient of i plus j, so in fact we should add the product into the current value of c i plus j. So that is exactly whatever algorithm is going to be. So let us say for simplicity we have two polynomials both of degree 10 and we want to multiply those.

(Refer slide time: 04:20)



Program to multiply degree 10 p

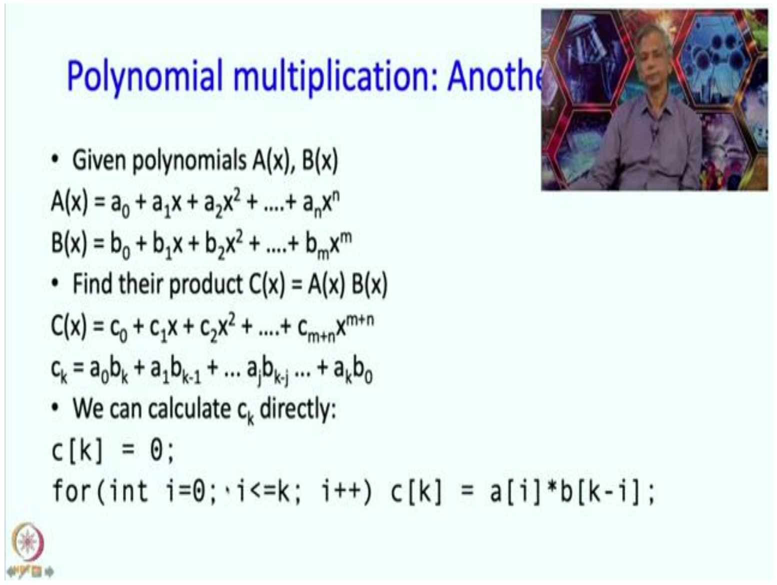
```
double a[11], b[11], c[21];
// a, b have degree 10, c has degree 20.
for(int i=0; i<=10; i++)
   cin >> a[i];              // read in polynomial A(x)
for(int j=0; j<=10; j++)
   cin >> b[j];              // read in polynomial B(x)
for(int k=0; k<=20; k++)
   c[k] = 0;
for(int i=0; i<=10; i++)
   for(int j=0; j<=10; j++)
      c[i+j] += a[i]*b[j];     // as discussed earlier.
for(int k=0; k<=20; k++)
   cout << c[k] <<' '; // output c, separated by spaces
cout << endl;
```

So clearly, a will have 11 coefficient so will reserve an array with 11 elements for it and array of 11 elements for b and c will have 21 coefficient so will use an array of 21 elements for c. So let us see how do we read them reading is straight forward so we are going to read A first starting with a0 going all the way till a10, so we so when I say read in the polynomial A(x) well the polynomial is determined by the coefficient. So I really mean reading the coefficients and of course you know by now that reading in the coefficients is done simply using a for loop, so by the way for loops are sort of the ideal things to use with arrays because the index can be the control variable can simply be the index, next we are going in the polynomial B in a similar manner and now we are going to generate our polynomial C. So first we have to set all the coefficients in it to 0 so the coefficients range from 0 through 20 so we are going to set all those coefficients to 0. And next we are going to form the product of every term in the A polynomial with every term in the B polynomial.

So, i is going to be the index which is going to range over the A polynomial the terms in the A polynomial and j is going to be the index which ranges over the terms in the B polynomial and if we form the product ai times bj. We said earlier it should affect c of i plus j. So that is exactly that has what is we are going to do so we are going to add ai times bj to c i plus j, remember that this plus equal to the short form for saying $c_{i+j}=c_{i+j}+a_ib_j$.

So that is it the entire multiplication after that we just have to print out the result. Now I am going to just describe another polynomial multiplication algorithm and this algorithm uses sort of an interesting manipulation of the indices.

(Refer slide time: 06:54)



Polynomial multiplication: Anothe

- Given polynomials A(x), B(x)

$A(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$

$B(x) = b_0 + b_1x + b_2x^2 + ... + b_mx^m$

- Find their product C(x) = A(x) B(x)

$C(x) = c_0 + c_1x + c_2x^2 + ... + c_{m+n}x^{m+n}$

$c_k = a_0b_k + a_1b_{k-1} + ... a_jb_{k-j} ... + a_kb_0$

- We can calculate c_k directly:

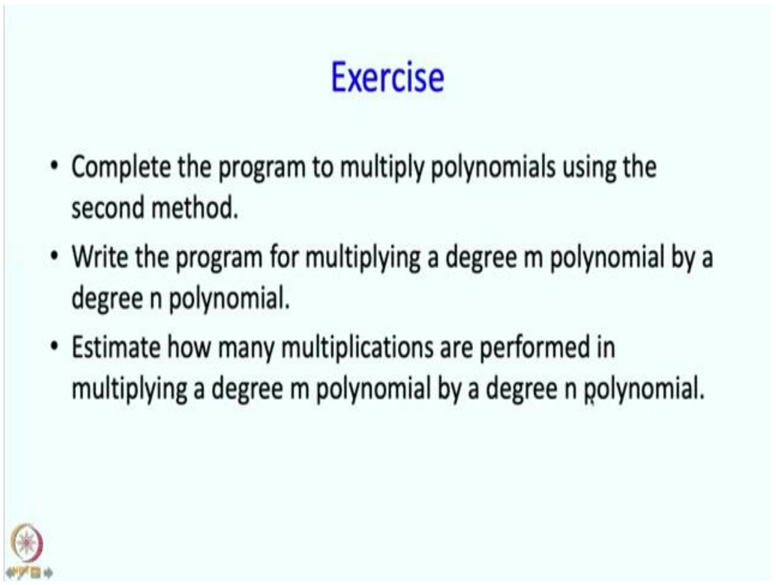
```
c[k] = 0;
for(int i=0; i<=k; i++) c[k] = a[i]*b[k-i];
```

So the problem is really the same we are given polynomials we want to find the product and this time I am going to note explicitly what is Ck going to equal. So Ck is going to get the product the value of a0 bk will contribute to Ck then a1 bk minus 1 will also contribute to Ck and so on until ak be 0. So what now we are going to do is we are going to compute c0 first then c1 then c2 then all the way till c m plus n. So, earlier we sort of did the multiplication in a convenient order and we updated Ck's now we are going to say no let us construct Ck's, let us construct the complete Ck's and on c0 then the complete c1 complete c2 and so on. And for that we need this formula and as you can see if we want to implement this formula we have to access the elements of a and b in an interesting order.

So the elements of a have to be accessed starting with 0 1 all the way till k which is simple enough but the elements of b have to be accessed starting at k then going down k-1 k-j coming back down to 0. So that sort of the unusual part of this program. So we are going to do this by setting ck equal to 0 first and then this loop can written down in this manner so ck, so i is going to range from 0 to k so this is I, the index of a is i.

So i is going to be 0 first, 1 all the way till k. So that is what this, loop range is telling us so ai then b of k-i, so indeed if you see the coefficient over here is k minus this coefficient this is k minus this coefficient and so on. so that is what we end up writing so this will generate Ck for any fixed k so all that remains is to just put this loop and we are done.

(Refer slide time: 09:19)



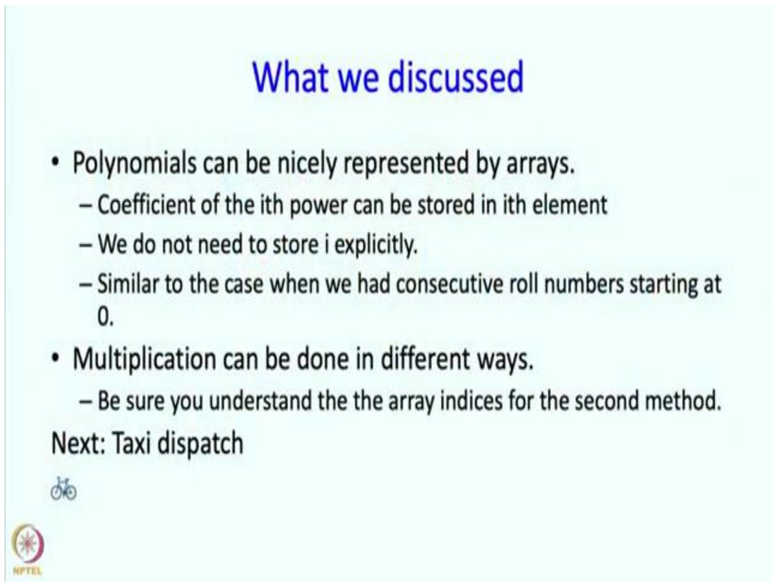
Exercise

- Complete the program to multiply polynomials using the second method.
- Write the program for multiplying a degree m polynomial by a degree n polynomial.
- Estimate how many multiplications are performed in multiplying a degree m polynomial by a degree n polynomial.

So as an exercise I will let you complete the program to multiply polynomials using the second method and then I would also suggest you to change both programs, so that instead multiplying a degree 10 polynomial by a degree 10 polynomial you write it for a general m you can put m and n as constants. But make them different and write them as use the idea define them as const int m=10, const int m=15 and then write in terms of m and n. Then I would also like you to go over the loops and estimate, how many multiplications are performed in multiplying a degree m polynomial by a degree n polynomial, and you should see roughly there are about m times n multiplications that get performed.

So this is just to get a sense of how much is this algorithm is going to take let us say if we are multiplying really huge polynomials alright, so what did we discuss?

(Refer slide time: 10:30)



What we discussed

- Polynomials can be nicely represented by arrays.
  - Coefficient of the ith power can be stored in ith element
  - We do not need to store i explicitly.
  - Similar to the case when we had consecutive roll numbers starting at 0.
- Multiplication can be done in different ways.
  - Be sure you understand the the array indices for the second method.

Next: Taxi dispatch

So, we said that polynomials can be nicely represented by arrays nicely in the sense that the coefficient of the ith power can be stored in the ith element and we do not need to store i explicitly just the position tells us the position of coefficient in our memory tells us what i it refers to and this actually very similar to the case when we have consecutive roll numbers starting at 0.

Then we also saw that multiplication can be done in different ways, and especially the second way is slightly trickier. And I would request you to go over that maybe print out the indices that get accessed in the loop, if you are not sure but anyway so there are two ways and please also look at the exercise just that were given.

So, in the next segment we are going to discuss a somewhat longish problem called taxi dispatch which uses the array in another interesting manner but will take a quick break.