CS506: Midsem

Due on Tuesday, March 01, 2022

Siddharth Charan 190101085

Question 1 : Design an algorithm to compute the product of two numbers

Introduction:
Let,we have 2 numbers a and b. As, we are dealing with external memory algorithm. They are very large . Otherwise, we can easily multiply them. They are store continuously in external memory. So, this is basically an array like structure, where at each position we have digits stored. Wher first element of array is most significant bit. So, now we will try to multiply these numbers in I/O efficient manner. Let us assume that numbers are stored in arrays A and B. We will store ans in another array C.

Main observation/ (Idea)
We can use basic carry method to multiply numbers part by part and it is very efficient in context of I/O.

Algorithm:
if(A.size()+B.size()==0) return 0;
for i = 1 to A.size()
carry = 0
for j = 1 to n C [i+j-1] = carry + a[i]*b[j]
carry = C[i+j-1]/10
C[i+j-1] = C[i+j-1] mod 10
return C

I/O Analysis:
if M = digits in number1 and N = digits in number 2.
As, we need linear complexity in I/O terms to store both the arrays. We will need O(M/B)*O(N/B) I/O to solve the problem . Which means we will have O(M*N)/(B^2) I/O complexoty.

Question 2 : Assume that you are given a number n in binary. Design an algorithm to compute n mod 3.

Introduction:
Number is given in the format of bits. And, number is very large and stored in continuous fashion in external memory. Again, similar to 1st question we can think that as terms of array. Let, us assume that array A represents the number. We need to calculate the mod with 3. main problem is that we can't store whole number in memory to calculate mod directly.

Key Idea:
Here, Key idea is that we need not to process whole number at once, we can process it one by one. As, mod have distribution property, we will calculate mod bit by bit and process in a way, that it gives us final answer.

For example ans = 101(5) mod 3 = 2:
We can do it bit by bit in below way:
ans = 0
1st bit => ans = (ans* 2+ 1(bit)) mod 3 = 1
2nd bit => ans = (ans* 2+ 0(nit)) mod 3 = 2
3rd bit => ans = (ans * 2+ 1(bit)) mod 3 = 2

Algorithm:
ans = 0
for i = 0 to A.size()-1
ans = (ans * 2 + num[i]) mod 3
return ans

I/O Analysis:
We will process B bits at one time. So, algorithm will take linear time. If N is input size then, it will take O(N/B) time.

Question 3 : Assume that you are given a number n is binary. Design an algorithm to compute n mod 13.

It is very similar to question as, I was able to find general algorithm.

Introduction:

Number is given in the format of bits. And, number is very large and stored in continuous fashion in external memory. Again, similar to 1st question we can think that as terms of array. Let, us assume that array A represents the number. We need to calculate the mod with 13. main problem is that we can't store whole number in memory to calculate mod directly.

Key Idea:

Here, Key idea is that we need not to process whole number at once, we can process it one by one. As, mod have distribution property, we will calculate mod bit by bit and process in a way, that it gives us final answer.

For example ans = 101(5) mod 13 = 5:
We can do it bit by bit in below way:
ans = 0
1st bit => ans = (ans* 2+ 1(bit)) mod 13 = 1
2nd bit => ans = (ans* 2+ 0(nit)) mod 13 = 2
3rd bit => ans = (ans * 2+ 1(bit)) mod 13 = 5

Algorithm:

```
ans = 0
for i = 0 to A.size()-1
ans = (ans * 2 + A[i]) mod 13
endfor
return ans
```

I/O Analysis:

We will process B bits at one time. So, algorithm will take linear time. If N is input size then, it will take O(N/B) time.

Question 4 : Assume that you are given two sorted arrays A and B of integers. Compute the median of the A and B combined. You may assume that the elements are distinct.

Introduction:
Let, we have 2 arrays A and B. We want to calculate median for total array A+B.
Main problem is that we can't simply calculate median for arrays separately and use those answers directly. Because, when we merge them, we can have completely different array, with no relation to separate medians.

But, if we observe clearly, We can use recursion to solve our problem.

Main observation(Idea) :
If x is median for 1st array and y is for 2nd. Then, we can observe that, we will need to search median in right part of smaller median and left part of larger median. As, median is middle element. So, that way we will have another problem of size half of original problem and recursively we can solve the problem.

Algorithm :
Solutin(A,B):
Calculate the medians x and y for both arrays.
Search left part of array with larger median and right part of array with smaller median.(return Solution(A',B')), A' and B'updated arrays.

Repeat the above process until size of both the subarrays becomes 2.
return (max(A[0], B[0]) + min(A[1], B[1]))/2

Complexity Analysis:
We will need to do O(N/B) I/O for storing the array in main memory. And as, we are dividing problem in n/2 each time, we, will need to do it for only log(N) times.
So, we can solve the problem in O((N/B)(log_2 N)) time.

Question 5 : Consider n water tanks in R 2. Assume that each tank has a height and associated list of neighboring tanks. If tank A is a neighbour of tank B and height of A is greater than height of B, water from A will be eventually drained to B. If there are multiple neighbors at lower height, water flow into all those neighbors. Design an algorithm to find all the tanks that can contain water in the steady state.

Introduction:
We assume that all tanks have some water in them. And, each has some neighbours and we know that, if it has a neighbour with lower height, then water will eventually from this tank to neighbouring tank. But, height of neighbour will not change. Now, we will try to answer how can we solve this problem in I/O efficient manner. Similar problems were solved in class too.

Key Idea
Key idea is that we can denote all those tanksas DAG and if we observe carefully, the value of each vertex(finally water will be there or not, depends only on it's neighbour). Hence it is a local function and in class we saw how to handle them. We will use time forwarded analysis.
Here local function :
f(v) = 1, if for i =1 to numbers of neighbours, h(v)<h(neighbours)
f(v) = 0, otherwise.
f(v) =1 denotes water will be there evantually.
We can start with any node, as if any neighbour with lower height is there, it's height will be 0. **But, only catch is that we will not require to update values in list every time.**

Algorithm
Initialize an array F[0..n − 1] = 1 and a store all neighbours in list L(neighbours height).
for i = 0 to n − 1 do
Compute f(vi) from h(L(i)) and the extracted f-values; set F[i] using f(vi).
If(h(v[i])<h[A[i]] F(A[i])=0
endfor

I/O Analysis:
If we were updating the list containing details about, neighbours then we would requir. It would take standard time for time forawarded analysis.
But, here we will require O(V/B) time.