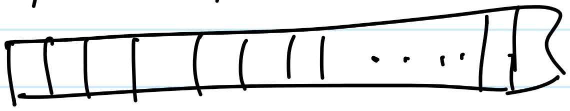
Fenwick Tree Concept

Monday, 18 January 2021 2:37 PM

Fenwick tree (or) Binary Indexed tree :-

$\rightarrow$ Long array -



a) get sum of 1st n elements.

We can get it by traversing each element

$\Longrightarrow$ O(N) time

b) update and get sum.

$\Longrightarrow$ for update O(1)

and for get sum again O(N)

Eg :-

7	8	4	9	3	2	7	9

1st Way :-

Brute force

O(N) for get sum.

O(N) for update and get sum.

O(N) for update and get sum.

2nd Way

Compute prefix sum.

prefix sum –

7	15	19	28	31	33	40	49

for this –

get sum – O(1)

update and getsum – O(N)

	Brute force	Prefix-Sum	Fenwick
getsum	O(n)	O(1)	O(logn)
update	O(1)	O(n)	O(logn)

Fenwick tree algorithm :–

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

⟶ In fenwick, indices start from 1.

		Values
8	– 1000	9
7	– 0111	7
6	– 0110	2
5	– 0101	3
4	– 0100	9
3	– 0011	4

4	0100	9
3	0011	4
2	0010	8
1	0001	7

→ Initially, all cells have values as '0'.

Now,
update (1,7) i.e., update index 1 with 7.

update (2,8)

⋮ ⋮ ⋮ ⋮

Update algorithm :-

→ We update the given index first.

i.e.,
~~For~~ update(2,8)

So, first we update index 2 in fenwick tree.

→ Now, we see lowest 1 in the binary representation of previous index. and add it to that index to get current index.

$\Longrightarrow$ 2 - 0010.

$\Longrightarrow$ to get next index -
find lowest 1 in binary form
i.e., 0010.
(for example, 5 - 0101 then lowest
1 is - 0001)

$\Longrightarrow$ add these two.

$$\begin{array}{r} 0010 \\ +\ 0010 \\ \hline 0100 \\ \hline \end{array}$$

$\Longrightarrow$ So, next index to update is 0100
i.e., 4.
So, we update 4th index with 8.

$\longrightarrow$ We do this till we reach end of array.

<u>Query (getSum) algorithm</u>:-

$\longrightarrow$ Suppose we are asked to getSum (1,5)

Initially -

sum = 0

$\longrightarrow$ We start at 5

sum + = fenwick[5]

$\longrightarrow$ Now, we subtract least 1 from the last index to get current index.

$$\begin{array}{r} 5 - 0110 \\ -\ 0010 \\ \hline 0100 \end{array}$$

= 4.

$\Longrightarrow$ sum + = fenwick[4]

$\longrightarrow$ Now, again do the same -

$$\begin{array}{r} 4 - 0100 \\ -\ 0100 \\ \hline 0000 \\ \hline \end{array}$$

$\longrightarrow$ End at 0 index because fenwick

Starts from index 1.

How to get the lowest 1 bit for a number :-

→ **In Java** –

Integer.lowestOneBit(i)

Update Code :-

```
update(i, delta)
{
    while(i<size)
    {
            a[i] = a[i] + delta;
            i = i + Integer.lowestOneBit(i);
    }
}
```

getSum Code :-

```
getSum(i)
{
        sum = 0
```

```
{       sum = 0
        while(i > 0)
        {
                sum = sum + a[i];
                i = i - Integer.lowestOneBit(i);
          }
        return sum;
 }
```

→ In other languages —

(i & -i)

gives lowest one bit number

__Limitations of fenwick :-__

→ If arrays are sparse i.e.,

we have,

val	index
5	1
8	10^3
13	10^4
11	5×10^4

In this case, we have to use

compression of array.

Program 1

Monday, January 18, 2021 4:34 PM

Problem Statement:

Malika taught a new fun time program practice for Engineering Students.
As a part of this she has given set of numbers, and asked the students
to find the sum of numbers between indices S1 and S2 (S1<=S2), inclusive.

Now it's your task to implement the Solution class:
- public Solution(int[] nums) : Initializes the object with the integer array nums .
- public long sum(int S1, int S2): Returns the sum of the subarray nums[S1, S2] (i.e., nums[S1] + nums[S1 + 1], ..., nums[S2]).

Input Format:

Line-1: An integer n, size of the array nums[] (set of numbers).
where 1 <= n <= 22000
Line-2: Two integers S1 and S2, index positions
where 0 <= S1 <= S2 < n

Output Format:

An integer, sum of integers between indices(s1, s2).

Sample Input-1:

```
8
2 6
```

Sample Output-1:

```
2864403
```

NOTE:

First 8 values of the input are:
115053, 59099, 681359, 526248, 123844, 612168, 920784, 591204

Code:

```
class Solution{
   static int arr[];
```

```
    static int n;
    static long fenwick[];
    public Solution(int nums[])
    {
        arr=nums;
        n=arr.length;
        fenwick=new long[n+1];
        for(int i=0;i<n;i++)
        {
            this.update(i+1,arr[i]);
        }
    }

    public static long query(int idx)
    {
        long sum=0;
        while(idx>0)
        {
            sum+=fenwick[idx];
            idx=idx-Integer.lowestOneBit(idx);
        }

        return sum;
    }

    public static void update(int idx,int delta)
    {
        while(idx<=n+1)
        {
            fenwick[idx]+=delta;
            idx=idx+Integer.lowestOneBit(idx);
        }
    }

    public long sum(int s1,int s2)
    {

        long res=query(s2+1)-query(s1);

        return res;
    }

}
```

Program 2

Monday, 18 January 2021 5:11 PM

P, q Matrix.

$$\begin{matrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Problem Statement:

A dangerous virus "ebola" is spreading across african countries.
Few people stand in a form of p*q grid, some positions in the grid are empty.

The grid is represented with three values 0,1, 2.
Where
- 0 indicates an empty position,
- 1 indiactes a healthy person , or
- 2 indiactes an infected person.

Every minute, any healthy person who is 4-directionally adjacent to an infected person becomes infected.

Your task is to find out the minimum amount of time in minutes that the virus takes to spread among all the people in that grid.

If this is impossible, return -1.

NOTE:
4-directions are Up, Down, Left, Right.

Input Format:

Line-1: Two integers P and Q, size of the grid.
Next P lines: contains Q space separated integers, either 0, 1, or 2.

Output Format:

An integer, the minimum amount of time in minutes

Sample Input-1:
----
3 3

2 1 1
1 1 0
0 1 1

Sample Output-1:

4

Explanation-1:

There is an infected person at position (0, 0).
In the first minute: people in (0, 1) and (1, 0) positions are infected.
In the second minute: people in (0, 2) and (1, 1) positions are infected.
In the third minute: person in (1, 2) position is infected.
In the fourth minute: person in (2, 2) position is infected.

Sample Input-2:
----
3 3
2 1 1
0 1 1
1 0 1

Sample Output-2:

-1

Explanation-2:

The healthy person in the bottom left corner (row 2, column 0) is never infected,
because infection only happens 4-directionally.

Sample Input-3:
----
1 2
0 2

Sample Output-3:

0

Explanation-3:

Since there is already no healthy person at minute 0, the answer is just 0.

Code:

```
import java.util.*;
```

```
class EbolaBFS
{

    public static int BFS(int mat[][], int p, int q, Queue<Integer> start)
    {
        int i,j;
        int noOfMin=0;
        boolean flag=false;
        Queue<Integer> queue=start;
        int size=queue.size();
        while(!queue.isEmpty())
        {
            flag=false;
            size=queue.size();
            while(size-->0)
            {
                int front=queue.poll();
                i=front/q;
                j=front%q;
                if(i-1>=0 && mat[i-1][j]==1)
                {
                    mat[i-1][j]=2;
                    queue.add((q*(i-1)+j));
                    flag=true;
                }
                if(i+1<p && mat[i+1][j]==1)
                {
                    mat[i+1][j]=2;
                    queue.add((q*(i+1)+j));
                    flag=true;
                }
                if(j-1>=0 && mat[i][j-1]==1)
                {
                    mat[i][j-1]=2;
                    queue.add((q*i+j-1));
                    flag=true;
                }
                if(j+1<q && mat[i][j+1]==1)
                {
                    mat[i][j+1]=2;
                    queue.add((q*i+j+1));
                    flag=true;
                }
            }
            if(flag==true)
            {
                noOfMin++;
            }
        }

        for(i=0;i<p;i++)
        {
            for(j=0;j<q;j++)
            {
                if(mat[i][j]==1)
                {
                    return -1;
                }
            }
        }

        return noOfMin;

    }

    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int p=sc.nextInt();
        int q=sc.nextInt();
        int mat[][]=new int[p][q];
        Queue<Integer> start=new LinkedList<Integer>();
        for(int i=0;i<p;i++)
        {
            for(int j=0;j<q;j++)
            {
                mat[i][j]=sc.nextInt();
                if(mat[i][j]==2)
                {
                    start.add(i*q+j);
                }
            }
        }

        int res=BFS(mat,p,q,start);
        System.out.println(res);

    }
}
```

Monday, 18 January 2021 7:05 PM

1	2	3
1	1	2

$$\begin{tikzcd}1 \arrow[r, bend right, leftrightarrow] & 2\end{tikzcd}$$

min (1, 1)
= $\textcircled{1}$

1	2
1	3
2	5
3	5
2	6
3	7

1 : [ ]
2 : [1]
3 : [1]
4 : [ ]
5 : [2, 3]
6 : [2]
7 : [3]

1	2	3	4	5	6	7
1	2	2	1	3	3	3

1	2
2	3
4	1

1	2	3	4
2	3	4	1

$$\begin{tikzcd} & & & \textcircled{4} \\ & & \textcircled{1} \arrow[ur] & \\ & \textcircled{2} \arrow[ur] & & \\ \textcircled{3} \arrow[ur] & & & \end{tikzcd}$$

Problem Statement:

EA Sports, developed a video game.
They designed a game in such a way that, there are L number of levels from 1 to L.
There are D number of dependencies where each dependency[m] = [Xm, Ym],
represents a prerequisite relationship, that is, in order to play level-Ym,
you must have completed the level-Xm .

In one day you can complete any number of levels as long as you have completed
all the prerequisites levels in the game.

You cannot play a level-Ym which has some prerequisite level-Xm on same day.

Write a method to return the minimum number of days to complete all the levels
in the game. If there is no way to complete all the levels, return -1.

Input Format:

Line-1: An integer L, number of levels.
Line-2: An integer D, number of dependencies.
Next D lines: Two space separated integers, Xm and Ym.

Output Format:

An integer, the minimum number of days to complete all the levels in the game.

Sample Input-1:

```
3
2
1 3
2 3
```

Sample Output-1:

```
2
```

Explanation-1:

On the first day, levels 1 and 2 are completed.
On the second day, level 3 is completed.

Sample Input-2:

```
3
3
1 2
2 3
3 1
```

Sample Output-2:

```
-1
```

Explanation-2:

No level can be completed because they depend on each other.