Lesson plan

``Tudor Vianu`` National High School of Computer Science
Subject: Informatics
Class: 11th grade
Teacher: Corina Ciobanu
Chapter: Trees
Theme: Binary Indexed Trees (BITs)
Type of lesson: acquisition of new knowledge
Objective framework:
Application development using algorithms common for trees
Objective reference:
To apply algorithms common for trees
Operational objectives:
Students must be able to:
O1: define a binary indexed tree
O2: identify a binary indexed tree
O3: correctly use theoretical concepts learned
O4: acquire algorithms for different operations (both modification and query)
O5: correctly analyze each problem and to write the properly program/sub-program

Didactic strategies
Didactic principles
- The principle of participation and active learning
- The principle of ensuring the gradual progress of performance
- The principle of inverse connection

Learning methods
- Oral communication: conversation, explosion, problematization
- Action methods: exercise, learning through discovering

Instructional procedures
- Explanation in the communication stage
- Learning by doing, through problem solving
- Questioning by creating problem situations

- Revising conversation during the consolidation of knowledge stage
- Case study
- Guided observation and study

Interaction

Frontal and individual; teacher – students, students-students

Directing teaching forms

Directed by the teacher or independent

Means of teaching:

Blackboard, chalk, educational software

Bibliography:

Informatics – manual for 11th grade by Tudor Sorin;
Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest
Data Structures and Algorithm Analysis in C by M. Weiss

Evaluation methods:

Oral questions
A set of applications

Content analysis:

C1: Definition of AIB
C2: Practical application
Lesson plan:

1. *Preparing the lesson*
    - Creating and writing the didactic project
    - Preparing the set of questions
    - Preparing the set of applications
    - Preparing the homework
2. *Organization and preparation of the class*
    - Checking the attendance of students to class/doing the roll call
3. *Focusing the attention of the students*
    - Announcing the subject
    - Announcing the objectives
    - Announcing the way in which the activity will be developed
4. *Checking students' prior knowledge*

Preparing a set of questions in order to check and brush up on the theoretical knowledge of students

-table 1-

Question	Awaited answer
What is a tree?	It is a connex graph without cycles.
What is the number of edges in a tree with n nodes?	The number of edges in a tree with n nodes is n-1.
What types of trees have we studied?	Binary trees

Evaluation in this stage will be carried out by the teacher, and the main form of interaction will be teacher-students.

5. Presentation of new knowledge(using the educational soft BIT/AIB)

Being given G=(X,V) a non-oriented connex graph, where X is the set of nodes and U is the set of edges. A tree is such a graph which doesn't have cycles.
*Presenting binary indexed trees.

The basic starting problem is the following : Being given a set/string of numbers by length N, do the following operations efficiently:

- Add a valor x to an element from the string
- Return the sum of all the elements from an interval [x,y]

This data structure, binary indexed tree (we will be referring at it as `BIT`) can do both operations (modification and interrogation) in an O(log N) complexity.

The BIT will be stored as a vector by size N, in which the element on the i position will be equal to sum of elements from the interval [i-2k+1, i] from the initial string, where k represents the number of terminal zeros from binary representation of i.

For a better understanding of the mechanism, I will show next the positions (with their representation in 2 base) from 1 to 15 with the intervals whose sum they retain.

- 1 (0001): [1, 1]
- 2 (0010): [1, 2]
- 3 (0011): [3, 3]
- 4 (0100): [1, 4]
- 5 (0101): [5, 5]

- 6 (0110): [5, 6]
- 7 (0111): [7, 7]
- 8 (1000): [1, 8]
- 9 (1001): [9, 9]
- 10 (1010): [9, 10]
- 11 (1011): [11, 11]
- 12 (1100): [9, 12]
- 13 (1101): [13, 13]
- 14 (1110): [13, 14]
- 15 (1111): [15, 15]

Interrogation/Query

Assuming that we want to find out the sum of an interval [1, x]. We know that bit[x] retains the sum [x-2k+1, x], so, after we add bit[x] to the answer, minus x with 2k and replay the algorithm. More precisely if we want to calculate sum [1, 13], we will just add bit [1101], bit[1100] and bit[1000].

Next, it is presented a code of interrogation function and in the figure 1 we can observe the `division` of the intervals in a BIT.

```
int query(int poz){
        int answer = 0;

        for (; poz; poz -= zero(poz))
                answer += bit[poz];

        return answer;
}
```

Zero(poz) is a function which returns 2^number of terminal zeros in base 2 of poz (position)

Once we can calculate the sums on intervals like [1,x] we can move on immediately at intervals like [x,y] because the sum(x,y) = sum(1,y) – sum(1,x-1).

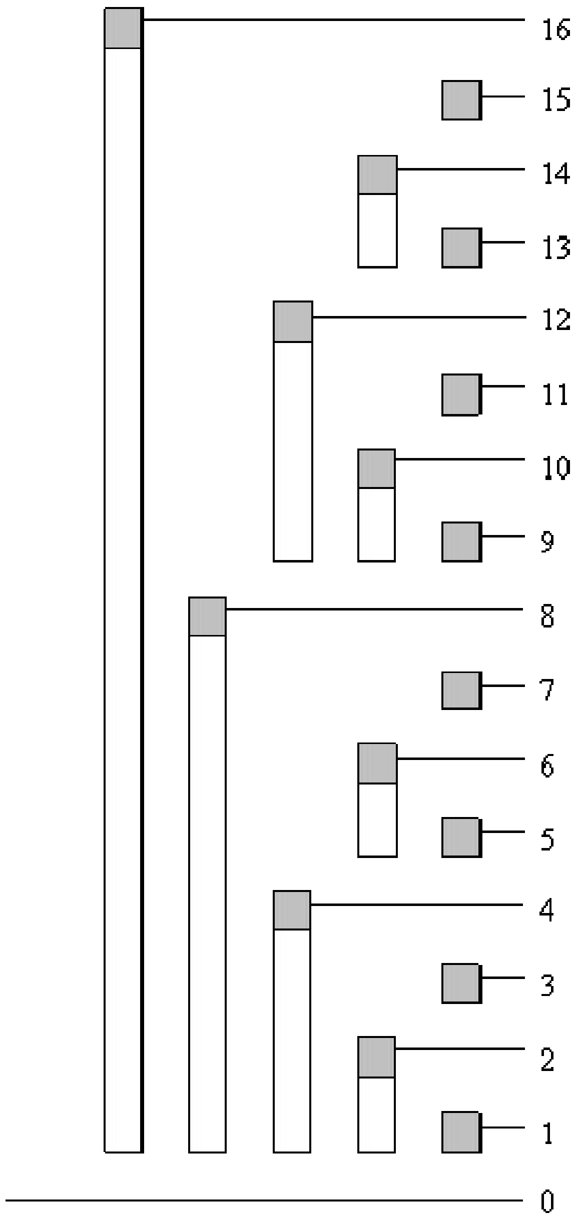

Figure 1 (www.topcoder.com/tc)

Modification

Modification is executed almost in the same way as the interrogation. When we want to modify the value on the position x we have to modify all y positions of which intervals include x, too. For example, if we want to modify the value on element 9, we will modify, in order (see figure 1 too): bit[9], bit[10], bit[12], bit[16].

To identify the next position which can be modified we will add zero (position)to the current position:

```
void update(int poz, int val){
        for (; poz <= n; poz += zero(poz))
                aib[poz] += val;
}
```

Calculating the zero() function
In order to keep the O(log N) complexity, we need to find an efficient method (O(1)) to find out the number of terminal zeros in base 2 of a number, and also 2^its power. For this, we will use bits operations, &, | and ^ , and we can see below a recap of how they work.

x	y	x & y	x | y	x ^ y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Let’s take for example a random number and assume that it has 3 terminal zeros in base 2. xxxx1000
If we decrease 1 from that number it will look like that: xxxx0111
If we apply the opperator ^ between those two numbers, the x-es from front will become 0 (because x^x=0): 00001111

Finally, applying the opperator & between the original number and the resulting one, we will get exactly the desired result.

In order to ensure the feedback and evaluation of the performance the following problems are given:

1. The analysis of complexity on given algorithms
2. Determination of zero function