Slide 1.

Hello, dear students. That's the fifth laboratory work for algorithms and data structures. Our topic is data structures. For problems of this laboratory work you can use data structures implemented in standard libraries.

Slide 2.

Today we will analyze possible solutions of problem "Visits" from your previous homework, and problems "War games 2" and "Monobilliards"

Slide 3.

Problem 1726 - "Visits". You can go through the link to check full description. I will remind you the most important part

- The city is a rectangle with the sides parallel to the coordinate axes. All the streets stretch from east to west or from north to south through the whole city. The house of each member of the program committee is located strictly at the intersection of two orthogonal streets. When walking from one house to another, members of the program committee always choose the shortest way and walk only along the streets. All of them visit each other equally often.
- Output is the average distance, rounded down to an integer, that a member of the program committee walks from his house to the house of his colleague.

Slide 4.

We can imagine city from the problem as coordinate plane. Streets are lines parallel to axes Ox and Oy and can have only integer values. Because houses located only at the intersection of streets, they can be considered as points with integer coordinates. Output of this problem is average distance between each points.

Such geometry, compared to Euclidean geometry, is called taxicab geometry. Required in the problem distance between 2 points in such geometry is often called Manhattan distance.

For simplicity on the images, we will only check positive coordinates, but it isn't affecting the solution.

Slide 5.

First question for this problem is how to calculate distance between two points? Should be noted, that in the description, members of committee can only walk along the streets.

It means, that firstly member of committee should go to the required coordinate on the axis x and then to the required coordinate on the axis y, or vice versa.

So, to calculate distance between points A and B in such way, we should find difference between their x coordinates and their y coordinates and summarize them. Should be noted that with absolute values we don't care about which point A or B has greater coordinates.

The second question is how many pairs of points do we have? It is important because we need to find average distance between all pairs of points. So, we have n points, and from each of these points we can

go to other (n – 1) points. But in this way, we will calculate each path twice, in both direction. Final formula for number of pairs P is presented in the slide.

This formula means, that number of distance calculations between points depends on number of points quadratically, and algorithm will have quadratic complexity. By description of problem, it is possible to have up to fifth pow of ten houses, so such algorithm can be slow.

Slide 6.

Let's prepare final formula for calculation of average distance for this method. We should take each point and calculate distance only to points with greater index. In this way we skip calculation of the same distances two times. Finally, this sum should be divided on number of distances P.

Here should be noted, that calculation of path's parts parallel to axis Ox and axis Oy can be done independently. Let's create another approach, based on this fact.

Slide 7.

Let's prepare projection of all points on the axis Ox. Now we can sort all points by increasing of their x coordinate.

Let's check points 2 and 3 on the image. The question is how many times path between point 2 and 3 will be covered during calculating of distances?

Slide 8.

It is easy to note, that from every point to the left of this line to every point to the right of the line, each path will contain this distance. In this case it is paths between points 1 and 3, 1 and 4, 2 and 3, 2 and 4. So to calculate the number of times, when line between points 2 and 3 is covered, we should multiply number of points to the left of the line on number of points to the right of the line. In previous example, there are 4 paths covered this line.

There is the same number of paths from the right points to the left in reverse direction. But we can skip them because no need to calculate the same path twice.

Slide 9.

To calculate part of distances between all points, which are parallel to axis Ox, following formula can be used. Here we work with sorted coordinates of points. So, x_i and x_{i+1} are coordinates of neighboring points, and difference between them is line between points, mentioned in previous slide. It is obvious, that to the left from this line there are exactly i points, and (n – i) points to the right, so we can multiply number of paths, which are covered this line, on distance of the line. After summarizing them we get sum of all path's parts parallel to axis Ox. It is important, that all points are sorted by their x coordinate, so we can get neighboring points very fast.

The same approach can be used and for calculation parts of paths, which are parallel to axis Oy, but it requires sorting points by their y coordinate.

Slide 10.

Finally, we can get average distance between all points. For this purpose, we can summarize both sums: parts, which are parallel to axis Ox, and parts, which are parallel to axis Oy. Final formula is presented on the slide. Should be noted, that x_i and y_i on each step of formula can be coordinates of different points, because of independency in calculations of path's parts parallel to axis Ox and axis Oy.

From point of complexity for sorted arrays of x and y coordinates of points, this formula can be calculated for linear time.

Resulted complexity depends on sorting algorithm, so it can be said, that it is n*log n.

Slide 11.

Problem 1521 - "War Games 2". You can go through the link to check full description. I will remind you the most important part.

- In accordance with the scheme, the war games are divided into N phases; and N soldiers, successively numbered from 1 to N, are marching round a circle one after another. At each phase, a single soldier leaves the circle, while the others continue to march. At some phase, the circle is left by a soldier, who is marching K positions before the one, who left the circle at the previous phase. A soldier, whose number is K, leaves the circle at the first phase.
- You should output the numbers of soldiers as they leave the circle.

Important note. It is expected that complexity of student's solution will be not worse than O(n*log n). Quadratic complexity isn't acceptable.

Slide 12

Let's start from obvious approach. We will take a cyclic linked list of soldiers, and we will go around it, stopping after each k transitions between nodes. When next k-th element is found, it can be removed from the list.

You can see this approach on the slide. For example, at the start we have 6 soldiers. We go through first, second and third soldiers, and stop at fourth soldier. For simplicity, mark removed soldier as 0.

On the next step we should go through next 4 elements, which are not removed. In this case these elements are fifth, sixth, first and second. Second element should be removed and so on.

The main restriction for this algorithm is its complexity. At the end we should find all n elements, and each operation of finding requires to go through k elements. So, complexity is n-th power of k.

Slide 13.

Should be noted, that main idea of this problem is searching of k-th order statistic, but at each step k can be different.

So, we need data structure, which can have fast update operation and have fast search of elements.

Data structure, which meets these requirements, is segment tree. Let's check, how it can be used in this problem.

Slide 14.

The first step is to build this tree. The following rules should be applied. Soldiers will be presented as leaves of this tree. Each leaf presents one soldier. So, its value equal to one.

For simplicity of example, let the array of soldiers has size of closest power of 2, and assume, that extra soldiers are already removed. Please, note, it isn't necessary step for segment tree.

It means, that number of leaves is equal to closest greater power of 2, let it be the h-th power of two. It also means, that height of the tree is h + 1.

Let's use previous example with 6 soldiers. In this example, we have 8 leaves in the last level of tree and height of the tree is 4.

Slide 15.

Second important thing, that all parent nodes are the sum of values of its children's nodes. Also, this number can be interpreted as number of soldiers in subtree of this node. In this way we can create segment tree for sum operation.

Slide 16.

Unused elements of tree can be filled with zeros as already removed soldiers or entire subtrees of them.

Also, important to mark each leaf node with sequential number of soldier's original position, which is presented by this leaf. The leftmost leaf is first soldier, its sibling is second and so on. It will be used for outputting number of soldier and doesn't necessarily match with numeration for k.

Please, note, that position of first soldier in the circle will be mentioned as start. Also, later if it isn't mentioned specifically ordered number of soldier is his current position (not original) from the start.

Slide 17.

Finally, we've got a segment tree. Now it is required to define algorithm for getting k-th element. In our example we should each time get next fourth soldier.

Let's start from the root of the tree. Where is the required soldier present: in the left subtree or in the right? Firstly, check left subtree.

Slide 18.

Root node of the left subtree has value 4. It means, that this subtree contains soldiers from the first to fourth.

We need to find 4-th element starting from the first soldier. Because the value of this node is less or equal then searched value, it means, that 4-th soldier is presented in this subtree and we can continue.

Slide 19.

Let's check, which subtree we should choose to continue searching. From the element with value 4 we can try to go to the left subtree. Root of this subtree is equal to 2. It means, that this subtree contains soldiers from first to second. Because searched value 4 is greater, than value of this node, we can understand, that 4-th soldier presented in another subtree. We have only one possible candidate for this – it is a right subtree.

Here important rule should be noted. If we go to the right subtree, we should subtract value of root of the left subtree from the searched value before continue searching. What does this operation mean?

Each subtree of segment tree doesn't know about any other subtree, but before we go to the right subtree we know, that current subtree contains elements from first to fourth, and left subtree contains elements from first to second. It means that right subtree contains elements from third to fourth. So, after we go to the right subtree, it is easier to interpret it as subtree, which contains elements from first to second but with changed searched element. For this purpose, we decrease searched element on exact value of root of the left subtree, and we can work with right subtree as it has elements from first to second. In this way we will work only with specific segment of the whole tree.

So, in this case new searched value is 4 – 2 = 2.

Slide 20.

On the next step we again will check left subtree. Value of its root is 1 and it is less, than searched element, which now is equal to 2. So, we should go to the right subtree, but root of right subtree is leaf of the whole segment tree.

Required soldier is found, so we can take associated sequential number of this soldier's original position, which is equal to 4.

Slide 21.

After we found the soldier and get his number, it is important to recalculate the tree.

We should clean him from the tree. Let's apply zero to value of his leaf node in the tree.

Slide 22.

Now we need recalculate rest part of the tree. For this purpose, we should decrease each node in our path to this leaf by 1. So, we should decrement parent of this leaf node, its parent and so on, by one.

After these actions, all parent nodes in the tree again contains sum of children's value, and segment tree is ready to find next element.

Please, note, that it is possible to combine searching and decreasing all nodes on the path of the search.

Slide 23.

And here we have a problem. Which node is next k-th element, but from the start of elements?

It can be calculated by formula presented on the slide, where k-found is position of last found element, currently it is 4 in explained example. So it is decreased by one, because this soldier is removed,

increased by k, because we need next k-th soldier, and we need take a remainder from division on number of rest elements (here it is presented as value of root), because all soldiers are in a circle.

For explained example next k is equal to 2.

Slide 24.

Let's find a couple more soldiers in the tree.

Firstly, check left and right subtrees of the root. Left subtree's root value is 3. It is greater than 2, so our element in this subtree. Then check roots of its subtrees. Root of left subtree has value 2, which is equal to searched element, so element in this subtree. And finally, we can find, that element is in the right leaf. Original position of this soldier is second.

Don't forget to recalculate all nodes on this path.

Next value of k is 1 by the formula.

Slide 25.

As usual, check left and right subtrees of the root. Left subtree's root value is 2. It is greater than 1, so our element in this subtree. Then check roots of its subtrees. Root of left subtree has value 1, which is equal to searched element, so element in this subtree. And finally, we can find, that element is in the left leaf. It is a first soldier.

Don't forget to recalculate all nodes on this path.

Next value of k is again 1 by the formula.

Slide 26.

Check left and right subtrees of the root. Left subtree's root value is 1. It is equal to 1, so our element in this subtree. Then check roots of its subtrees. Root of left subtree has value 0, which is less than searched element, so element in the right subtree. No need to decrease searched element because left subtree is totally empty.

And finally, we can find, that element is in the left leaf. It is a third soldier.

Don't forget to recalculate all nodes on this path.

Next value of k is 2 by the formula. And so on. Next soldier is sixth, and final soldier is fifth.

Slide 27.

Algorithm will stop, when value of root node becomes zero, so there is no more soldiers in the circle. Complexity of searching of each element is logarithmic.

Great advantage of this data structure is that it can be build based on array (like heap), which make it very fast data structure for access elements. Disadvantage is obvious – segment tree takes a lot of extra memory for unnecessary and removed nodes.

Slide 28.

Possible alternative approach is to use balanced binary search tree. Each node contains number of elements in its subtree and state of soldier, associated with this node. It means, that each node contains a soldier, so it is required much less memory than segment tree.

Searching can be applied in the similar way. Should be noted, that implementation of this data structure is harder than segment tree.

Slide 29.

Problem 1494 - “Monobilliards”. You can go through the link to check full description. I will remind you the most important part

- The rules of monobilliards are very simple. One has to pocket successively the balls with sequential numbers into the only pocket. During the game inspector several times came up to the table and took out from the table's pocket the last of the pocketed balls. In the end it turned out that Chichikov had pocketed all the balls and the inspector had taken out and inspected them. The owner understood that this was his chance because the inspector had to remember the order in which he had taken out the balls.
- Output the word "Cheater" if Chichikov could not pocket all the N balls in the right order, otherwise output "Not a proof".

Slide 30.

First of all, to solve this problem we should understand how pocket works. The player can pocket the ball inside it. Each next pocketed ball is placed on top of the previous pocketed ball.

When inspector takes a ball from pocket, he always takes the topmost ball.

It means, that pocket is working with principle LIFO or Last In – First Out, the same way as data structure stack.

Let’s simulate the pocket during the game. Should be noted, that inspector can take balls from the pocket in random moments of game. The main problem of this simulation is insufficient data because we know, in what order balls were taken by inspector from the pocket, but don’t know, in what order they were pocketed. That’s why we can detect situations, when player cheated, but otherwise we can’t say, was he fair or not.

For simulation we should analyze state of pocket in all different cases when inspector takes a ball from the pocket.

Slide 31.

Let inspector takes a ball with number m in random moment of game, when some number of balls is already inside the pocket.

First case is when top of stack is equal to m. It means, that inspector takes exactly expected ball from the pocket. So, we can simply remove it from the top of the stack and continue.

Slide 32.

Second case is more complex. What if ball, which is taken by inspector, is greater than top of the stack?

It means, that several balls were pocketed after previous action of inspector. To simulate this, we should add balls to stack in increasing order till reach m. But only balls, which are greater, then previously inserted can be add to stack, because otherwise these balls were already inserted in the stack. So, value of max added ball to stack should be stored and controlled. In the example on the slide, this value is mentioned as k.

After this actions element, which is equal to m, can be removed from the stack.

Slide 33.

And case three, when top of stack is greater than m. It means that during simulation we get that some ball is already pocketed, but inspector took a ball with less number. It isn't possible, if player pockets balls in correct order. So, player is cheater in this case.

Slide 34.

To finalize the solution of this problem. If after simulation of all inspectors' actions, we don't find that player is a cheater – it isn't a proof.

Also, please, note, that at the start ball with number 1 can be added to stack for simplicity, because inspector always takes ball from pocket only after first ball was pocketed by player.

Slide 35.

Mandatory task. You should implement source code for problem 1521 "War Games 2", upload it to Timus system and pass all tests. After this you should prepare a report and send it via email. Please, use template document for your report and carefully set correct subject for the mail.

Slide 36.

You will have 2 problems for homework. First problem is 1494 "Monobilliards", which was explained earlier. Second problem is 1067 "Disk Tree". You should solve it by yourself. Please, note, that report for this problem should contain explanation of used data structures.

Homework is optional, but successful completion of this problems can give you extra points for better grade.

Slide 37.

Thank you for attention