Slide 1.

Hello everyone. That's a fourth laboratory work for algorithms and data structures. Our topic is sorting algorithms.

Slide 2.

Today we will analyze possible solutions of problem "Elephpotamus" from your previous homework, problem "Spy" and will have small overview of second test's tasks

Slide 3.

Problem 1444 - "Elephpotamus". You can go through the link to check full description. I will remind you the most important part

- Elephpotamuses' favorite dainty is elephant pumpkins. Having fed the animal with a pumpkin, Harry can direct it to any of the remaining pumpkins. In order to pass the exam, Harry must lead the elephpotamus so that it eats as many pumpkins as possible before it comes across its footprints.
- It is guaranteed that there are no two pumpkins at the same location and there is no straight line passing through all the pumpkins.
- The first number in this sequence must always be 1.

Slide 4.

It is geometric problem. Let's compare it with another geometric problem, which was discussed earlier. That problem was Median on the Plane. We can interpret this problem in following way. First of all, pumpkins are points, and in similar way as Median on the Plane, we can move center of coordinate plane to start point. Moving through all points without crossing of paths can be resolved, if all points will be sorted by angle, so we can go through them in order of angles.

Slide 5.

You can see example of such method in the image on slide. A is starting point; all other points are sorted by tangent to the line parallel to axis Ox and passing through point A. Then we start from smaller tangent to greater and go through all points in that order. But maybe this problem is still different from Median on the Plane? Let's check it.

Slide 6.

The first issue we can find in the description of problem. In this problem more than 2 points can lie on the same line. What does it mean for us? You can see image with example on the slide. Here points C, D and E lies on one line. From the start point A we goes to point B, because it has smallest angle. But where we should go next? It is unclear, because all three points D, C and E has the same angle related to line, moving through A. If we choose point C as next point, so we fail, because if we go to point D – it

means that we can't go to point E without crossing line from point C to point D. And the same, if we go to point E.

Slide 7.

This issue can be solved, if we will sort all points not only by angle, but also, we will use the second parameter, if angle is equal. The best possible parameter is distance from start point. It can be calculated in similar way as tangent or angle as hypotenuse in right triangle.

If we will sort by distance as second parameter in increasing order, then in example after point B we can choose closest point to start point among points D, C and E. It is point D. Next point is C, and then farthest point E. Issue has been solved.

Slide 8.

Second issue also follows from description of a problem. In Median on the Plane, we can choose the most convenient starting point for solution. But in this problem starting point is fixed. If we will still use tangent, we can have issues with points in the left half-plane from starting point.

Slide 9.

The solution for this problem is to choose different way for calculating angle. And the simplest way is to use atan2 function. This function is presented in standard libraries of many languages, but it isn't the same as arctangent inverse trigonometric function, because of return range from minus pi till pi. But it is very good return range for us, because it will present all 360 degrees around starting point.

Also, please note, that arguments for this function is coordinates related to the origin point of coordinate plane. You can check more about this function in related documentation.

Slide 10.

And finally, the third issue. Now we use for sorting 360 degrees around starting point. What if gap between some of the points is greater or equal to 180 degrees? Check the image on the slide. A is our starting point. All points lie in one half-plane from A. If we start from point B and goes to the left, then we reach point D. But here we have an issue – there is no points with gap less than 180 degrees. It is obvious, that if we want to connect point D and point E – we will cross path AB.

Slide 11.

Another similar situation, but in this case, points lie in second and fourth quarters. And here we have the same issue, because angle between points D and B is greater than 180 degrees, so it is impossible to draw line from B and D without crossing line AC.

Slide 12

Solution is following. We should detect such situation, when 2 points have a gap more than 180 degrees and start our path from point after this gap. In example, points D and E have such gap, so we should firstly go from A to E, then to B and finish in D.

Should be noted, that for calculation of gap we also can use function atan2.

Slide 13.

Other example can be solved in the same way as previous one. We should detect gap more than 180 degrees between point B and D, and start path from point B and finish in D.

Important note. It is impossible to have more than 1 gap of 180 degrees, because hole circle is 360 degrees. Only possible situation for this – when all points lie on the same line, but it is forbidden by description.

If there is no such issues and no points have such gap, you can start path from any point, with only exceptions for points on the same line.

All issues are solved.

Slide 14.

Problem 1322 - “Spy”. You can go through the link to check full description. I will remind you the most important part

- An input of the machine is a text line $S_1 = s_1s_2...s_N$. The machine constructs all cyclic permutations of this line, starting with shift to left. Then the resulted set S_1, S_2, ..., S_N is sorted lexicographically in the ascending order, and the lines are written out in this order in a column, one under another. Thus, an array N × N is obtained. One of the rows of this array contains the initial word. The number of this row and the last column of the array are the output of the machine.
- But as the information can certainly be deciphered (otherwise there is no sense in sending it), you have to invent a deciphering algorithm.

Slide 15.

The algorithm mentioned in the problem’s description is known as Burrows-Wheeler transform. Our task is to apply inverse transformation for it. Let’s use example from Timus for further observations.

Slide 16.

Let’s start from naive algorithm and try to restore the full table.

As we can find in description of the problem, input line is last column of the table, so we can safely add it to the table.

Slide 17.

For next step we should make several observations. First of all, as we know from description, all lines of this table is cyclic permutations. It means, that on each i-th place of the string, all characters will be present exactly same times as in original string. So, each column of the table contains all characters of original string.

Second observation is that how this table was made. Last step of Burrows-Wheeler transform is sorting of table rows. Strings are sorted lexicographically. It means, that order of string depends on first character in this strings, then from second and so on.

But if we know, that in last column we have all characters of original string, first column has the same characters, and all lines is sorted firstly by first column, it means, that if sort last column, so we get exactly first column of the table.

Slide 18.

Now we have first and last columns. It is obvious, that pairs from first and last columns characters are presented somewhere in the table as consequent characters, because of cyclic permutations. More than that – for some permutations, this pairs are first 2 characters of string. So, let's add last column as first and shift previous column to be the second.

Because we have starting 2 characters for all permutations in each row of the table, so now we can sort first two columns, and we get exactly first and second column of original table.

Slide 19.

On the next step we can continue the same reasoning for tuple of 3 characters. Output of this step is table, where first three columns are filled.

Slide 20.

After this we can repeat previous step, until all columns of table will be filled.

Slide 21.

Finally, we fill all cells of the table. The last step is finding resulted line. From description we have that number 3 is number of original line in table of transform. So, third line is the answer!

Slide 22.

For this algorithm we need to fill the table, which size is square of N. But time complexity is even more important. We need to sort rows of the table N times, but also each time size of rows grows from 1 till N. Time limit for this problem is 0.25 seconds, so we definitely need faster solution. What if there isn't necessary to restore full table?

Slide 23.

Let's try to create optimized version of this algorithm, based on vector of inverse transform.

For this method we need to make more observations. In naive algorithm on each step, we make the same 2 actions – add last column as first and then sort first columns. Should be noted, that after moving last column in place of first, each i-th row now starts exactly with i-th character of last column. Based on this we can easily said that the line, which was i-th row in the table before this actions, becomes exactly j-th row after sorting, where j is position of i-th character of last column in sorted last column or in the first column. You can see visual explanation of this process on the slide.

More than that, pairs of i and j will be always the same, because on each step we add last column and then sort them. Also, should be noted, that such sorting is stable, if character was higher in last column, it also will be higher in first row, because column inserted on previous step already have lexicographically order.

Slide 24.

Let's find mentioned pairs of i and j. For this purpose, we need to numerate all characters of last column and then stably sorted to get first column. In resulted middle table on the slide, j is number of row, where character was originally presented in last column. Now we can see, for example, that from row 1 character 'r' goes to row 10, or that character 'a' from row 8 goes to row 4.

We can write all such pairs, as it done in table T.

Slide 25.

Based on table T now we can calculate vector of inverse transform. By the task we know that target line is 3rd line in the table. As was mentioned earlier, on each step we add to i-th row exactly i-th character of last column and then change position of this row in table by rules from table T. It is obvious, that rules of these table are cyclic. That's mean, that we can create such vector, which will explain, what path through the table make our target row. If we now, which position this string have on each step of algorithm, so we now what character was added on each step of the algorithm to this string.

Let's find the path through the rules of table T, which will bring us to 3rd row. By rule in third line of table T we know that 3rd row is always moved from 7th row, by rule in seventh line of table T we know, that 7th row is always moved from 11th row and so on. We will stop when our vector returns to the 3rd row.

In the naive algorithm we collect all characters in rows backwards. For optimized algorithm we can easily goes through this vector backwards, so we will get the target string starting from the first character.

Slide 26.

Let's check example of using this vector of transform in context of how it can be programming. First of all, we need stably sorted pair of last column characters and their position in last column, result of this sort is left part of the table. That find initial row of our line, it is third line.

We will go through the vector in reverse order relatively to naive algorithm. So, we see, that third line was moved after sort from 7th row. Let's check, which character should be added to 7th row. It is character 'a'. 7th row on previous step was moved from 11th row. Let's check which character was added before this sort to 11th row. It is character 'b'. 11th row was moved during sort from 4th row and so on. You can see, that we go exactly in reverse order to naive algorithm, but it is also possible to go in the same direction, and restore string starting from the last character.

Slide 27.

Finally, we restore original string. This optimized algorithm is preferred for implementation. But it can be optimized more for even better time complexity.

Slide 28.

There are two main options for optimizations.

First of all, standard string types can be slow for this task. And it is better to work with arrays of type char, which is primitive type, for any purpose.

Secondly, sorting in this problem can be realized for linear time with counting sort.

Slide 29.

Counting sort is an algorithm, which sorts elements of array without direct comparison of elements. This algorithm has several restrictions on sortable values and space complexity because it requires extra memory.

Sortable data should have limited range of different elements, for example, integers from 1 to 100, letters of some alphabet. Otherwise, it is not effective to use this algorithm.

Slide 30.

Let’s check an example of how this algorithm works. We have array of integers, each of them is greater than 0, but lower than 9, so totally 8 possible values.

It is necessary to have another array, which is called count. This array is filled with zeros and has size equal to number of all possible different values.

Slide 31.

Algorithm starts with iterating over array. First element is 5. Let’s increment element of count array with index 5. It means, that currently only one element of that value was counted.

Second element is 4. Now one element with value 5 and one element with value 4 were counted.

Slide 32.

Third element is 5. Now one element with value 4 and two elements with value 5 were counted.

In the bottom part of the slide, you can see filled count array.

Slide 33.

Second part of this algorithm is iterating over count array. Let's sort array in increasing order. So, elements with values 1, 2, 3, and 4 were counted exactly one time, and we can add one element of each value to array.

Slide 34.

Now we at index 5. We have counted two elements of this value. Let's add to sorted array two elements of value 5 and so on, until iteration of count array has been finished.

Slide 35.

Complexity of this algorithm is O(n + k), where k is size of count array. Should be noted, that for complex objects, counting sort is not stable. But it can be handled with introducing list of values for each element of count array.

Let's sort array from the example of the problem. Each letter of given string is marked with index, which order should be preserved.

Slide 36.

In the same way we will add ordinal indexes to the lists of count array. Please, note, that for value R we add 2 elements with ordinal indexes 1 and 4.

Slide 37.

After all elements are counted, we can restore array in sorted order. But now we don't just add letter A to the array but put letter A from position 3 in original array, then from position 6 and so on. Array is sorted in stable way.

Instead of ordinal indexes it can be any other value, which order should be preserved.

These optimizations can help you to pass required time limitations.

Slide 38.

Mandatory task. You should implement source code for problem 1322 "Spy", 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 39.

You will have 1 problem for homework. It is problem 1726 "Visits". You should solve it by yourself. Please, note, that report for this problem should contain explanation, why your algorithm is correct.

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

Slide 40.

Let's briefly check the tasks of the second test.

The first task was related to complexity of sorting algorithms.

Quicksort algorithm depends on choice of pivot (partitioning method) and input data.

Quicksort's complexity: Θ(n log$n$) for best case and degrades till Θ(n^2) in worst case.

Slide 41.

Comparison sorting algorithms can't be faster than O(n log$n$) for average-case.

This is a consequence of the limited information available through comparisons alone.

O(n log$n$) can be achieved by such algorithms as merge sort or heap sort.

Slide 42.

Insertion sort complexity is O(n^2) in worst case.

Quicksort can degrade till O(n^2) in worst case.

Heap sort for average and worst case have O(n log$n$) .

Slide 43.

Task 2 is related to different data structures. Let's check some possible examples of this task.

For this question we can see that elements were returned in exactly reversed order. It is principle Last In – First Out. Data structure, which realized it, is stack.

Slide 44.

For this question elements were returned in sorted order from the least to greatest. Queue, stack, and linked list don't change relative position of elements. So, answer is priority queue.

Slide 45.

The last question is slightly different, because here we have examination of internal memory layout of data structure. For stack, queue and array we can expect storing relative order of elements. Also, should be noted, that top element is greatest element. Other elements looks like levels of heap.

The answer is heap.

Slide 46.

Task 3 is related to binary search trees. Let's check some possible examples of this task.

All these trees are obviously binary search trees

Balanced binary tree is a binary tree in which the height of the left and right subtree of any node differ by not more than 1

So, answer is middle one, because left and right trees have subtrees with greater difference in height.

Slide 47.

For this question, balanced binary search tree has 31 elements. It means that tree has 5 full levels. So, we can have maximum one comparison on each level and totally 5 comparisons or log 32 base on 2.

Slide 48.

For this question, balanced binary search tree has 20 elements. It means that tree has 4 full levels and 1 unfinished. So, we can have maximum one comparison on each level and totally 5 comparisons.

Slide 49.

Answers to questions of task 4 can be found in lecture presentations

Short versions of answers:

Select and get operations are faster for sorted array. They take constant time, while insert and remove are faster for binary search tree.

Examples of linear data types are linked list, queue, deque, stack, array. Examples of non-linear data types are set, map, priority queue. Interfaces for this data types are explained in related lectures.

There is 2 different methods for three-way partitioning in quicksort algorithm.

Main advantage of Dijkstra method is if array consists of repeated elements only, no swaps are performed. Disadvantage is additional swap for each element that is greater than p (compared to normal partitioning).

Advantages of Bentley-McIlroy method: if array consists of repeated elements only, no swaps are performed. Also repeated elements are swapped exactly twice: once to place them near array bound and once to return to the middle. Disadvantage is that additional operation is required to put equal elements into the middle.

Slide 50.

Thank you for attention