Slide 1.

Hello everyone. That's a first laboratory work for algorithms and data structure. Topic of our first classes is introduction to algorithms. We will analyze a couple of problems related to general concept of algorithms, approaches to solving problems and algorithms complexity.

Slide 2.

Today we will analyze 2 problems with names "Hyperjump" and "Troubleduons".

Slide 3.

Problem 1296 - "Hyperjump". You can go through the link to check full description. I will remind you the most important part

- The sequence of integers p_i represents field intensities at different moments in time.
- If the alpha-phase begins at moment i and ends at moment j, then the value of gravity potential accumulated will be equal to the sum of sequence elements at places from i-th to j-th inclusive.
- The only line of output contains the largest possible value of the gravity potential that can be accumulated by a hyperspacecraft during the alpha phase.

Slide 4.

First, we need to understand, what is the goal of this problem and what should be our final result.

Gravy-potential in this problem is sum of some subsequence. That's mean, we have to find subsequence with greatest sum and calculate this sum.

It is important to note, that subsequence can go through negative value. Let's check it in the first sample.

For example, our subsequence starts with value p3 and equal to 59. Then we can add value of p4, which equal to 26, total sum now is 85. Next intensity is -53. Our sum will become less. Should we stop now?

No, we can continue, because p6 is 58 and after adding p5 and p6 to total sum, it becomes greater.

In the slide you can see subsequence marked with red color. It is subsequence with greatest sum for this sample, which is equal to 187. Let's create algorithm, which can automatically find this sum.

We will skip second sample, because it is mostly focused on fact, that sum less than zero has no sense – we always can return zero as maximum.

Slide 5.

The first idea to solve any problem – what if we can just check all possibilities? For this problem we need to calculate sum of each subsequence and find maximum sum between them.

It means that we should find sum of any possible combination of sequential values. For example, from p1 to p2, from p1 to p3, from p1 to p4, in the same way for p2, p3 and so on.

In this way we can fill the whole table of potentials for any subsequence. After this it is easy to find the maximum value. Obviously, this table will have (n squared divided on 2) values.

It is very simple solution. But what are disadvantages of it? As was mentioned earlier, the table will have n squared values. It means, that we need to loop over all elements for each n and during each step loop over all values of m. This algorithm will have quadratic time complexity. For some algorithms it can be a good complexity. But Hyperjump problem has very strong time limitations and total amount of values in sequence can reach 60000. What if we can find a faster solution?

Slide 6.

Second approach is based on Divide and conquer paradigm.

For this paradigm we need to split our problem on several smaller problems, solve them separately and then merge results. You will have more closer meeting with this paradigm during learning of sorting algorithms.

It is obvious, then we can split our starting sequence on 2 subsequences in the middle; call the same function in recursion for these subsequences and so on.

But the main question is how to merge results of subsequences.

Slide 7.

Let's firstly check any arbitrary sequence, after merge of two smaller subsequences. Let's call subsequence with values from a1 to a4 as "left" subsequence and with values from a5 to a8 as "right" subsequence.

During merge we need answer on a question: where maximum subsequence of merged sequence a1-a8 can be? There are 3 possibilities.

In first case, maximum subsequence lies fully in the left part. For example, it is sum of a1 and a2. Second case is similar, but for the right part. In this cases we can use values of maximum subsequences, which were calculated on previous step.

Slide 8.

The third case is more complicated. Maximum subsequence can lie in both parts at the same time, crossing the border of dividing sequence in 2 subsequences.

How to find it? On first step we can start from center and go to the left. We will sequentially summarize values and find maximum from the left of the center without skipping any value. Then we will repeat it for the right direction. After we can sum both parts, in such way we will find maximum subsequence, which lies on the border of divided parts.

Slide 9.

Now we can formulate our algorithm. For one-element sequences we can always return value of this element.

For merged “sequences” we should check all 3 cases and choose maximum between them. Let’s check some examples based on sample. For sequence from p6 to p7 we will check maximum, which we get from recursion call for the left part pf sequence (it is called left max). This value is 58. Then check the same for right max. It is 97. Subsequence, which lies in both parts and have maximum sum, we will call “middle”. In example with red color marked part of “middle” subsequences, which is used during moving from the center to the left, and with blue color – part, which is used during moving from center to the right. For this sequence maximum is “middle” sum.

On the next step of merging, we can see that value of “left” and “middle” sums are equal. We can choose any of them. Please, note, that for subsequence with negative sum we should return value 0.

Slide 10.

The most interesting case is on the next step. During merging subsequences to sequence from p6 to p10, “left” sum is greater than “middle” and “right” sums. So, we should return value of the “left” sum, because it is subsequence with greatest value for sequence from p6 to p10. In the same way we can do the last step and get final result of 187.

Usually, such algorithms have complexity of n*logn, which is much faster than quadratic complexity. But this algorithm is hard enough for implementation. Maybe we can find a solution with linear complexity?

Slide 11.

Let’s check the third approach – Kadane’s algorithm.

Starting from p1 we can calculate sum of all subsequent elements. On each step we will compare this sum with previously saved maximum and update it, if required.

For each step we will always have a maximum possible sum for processed subsequence. We have no need to check other subsequences, which are starting not from p1, they are always less with one important exception. Sum can become negative.

In this case we should replace current sum to zero, negative part makes the sum less, so we can drop previous subsequence and start it again from zero. Maximum value of previous subsequence is already saved, and we can compare new sum with it.

Let’s check it on the sample. Starting with element p1. It is equal to 31 and currently it is maximum sum. Next element p2 is -41. Current sum becomes negative, so we should replace it with zero. Previous subsequence either have maximum sum or doesn’t part of this sum. Maximum isn’t updated. On the next step we have element 59, current sum is 59. Maximum value is 31, so we should replace it with new maximum 59 and so on.

Slide 12

The full calculation you can see on the slide.

It can be said that on each step i current sum is the largest sum of subsequence, which ends in element i, when maximum sum is the largest sum anywhere between p1 and pi. For example, on the row of p8 in column sum you can see the largest sum, which is starting from p3 and ending in p8, while column max has largest sum anywhere between p1 and p8 (in our case it is a sum between p3 and p7).

Because on each step we solve problem for smaller sequence than original, this algorithm is simple example of dynamic programming.

This is the best and fastest solution for this problem. It is required for you to implement it later in source code and prepare a report. Don't forget to estimate complexity of this algorithm and write it to report.

Slide 13.

Next problem is 1155 - "Troubleduons". You can go through the link to check full description. I will remind you the most important part

- Experimental set consists of eight cameras, situated in the vertices of a cube. Cameras are named as A, B, C, …, H. It is possible to generate or annihilate two troubleduons in neighboring cameras. You should automate the process of removing troubleduons.

It is allowed, that this problem has no solution. In this case program should print IMPOSSIBLE to output.

Slide 14.

Very important question is can we detect all situations, when this problem has no solution to optimize our program?

The first idea is that amount of all troubleduons should be even, otherwise we can't remove them all in pair. But what about more strong condition?

Let's check the vertices. We can split them into two sets: ACFH and BGEG. Note, that you can't create or annihilate troubleduons in 2 vertices of the same set at the same time. It means, that for generation or annihilation we always must choose vertices from two different sets. If amount of troubleduons for different sets isn't equal to each other, so the problem will have no solution. Otherwise, it has solution.

Simplest algorithm is just taking one set of vertices (for example ACFH) and annihilate all troubleduons from its vertices, using troubleduons from adjacent vertices of another set.

If number of troubleduons in our set becomes zero – it means that problem successfully solved. Let's check one case that should be handled.

Slide 15.

Look at first picture. Here we have one troubleduon in vertices E and G and 2 troubleduons in vertex A. If we will try to annihilate troubleduon in vertex G – there is no troubleduons in adjacent vertices. To solve this case, we should use operation of moving troubleduons between vertices of the same set. Let's move troubleduon from vertex G to vertex E, where it can be easily annihilated with troubleduon of vertex A.

For this purpose, we create a new pair of troubleduons between vertex E and vertex, which is adjacent to vertex G. For example, it will be vertex F.

On the next step we can annihilate troubleduons between vertices F and G. Now on image 3 vertex G has no troubleduons, we moved it to vertex E.

Using such technique and simple algorithm you can solve this problem.

Slide 16.

You will have 2 problems for homework. First problem is 1155 “Troubleduons”, which was explained earlier. Second problem is 1005 “Stone pile”. You should solve it by yourself. Please, note, that for this problem is very important to explain in the report, why your solution is suitable.

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

Slide 17.

Current task for the rest of today’s classes. You should implement source code for problem 1296 “Hyperjump” using the third approach from this presentation, 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 18.

Thank you for attention