Next Generation Sequencing Technologies: Data Analysis and Applications

Shortest Common Superstring (SCS) Assembly

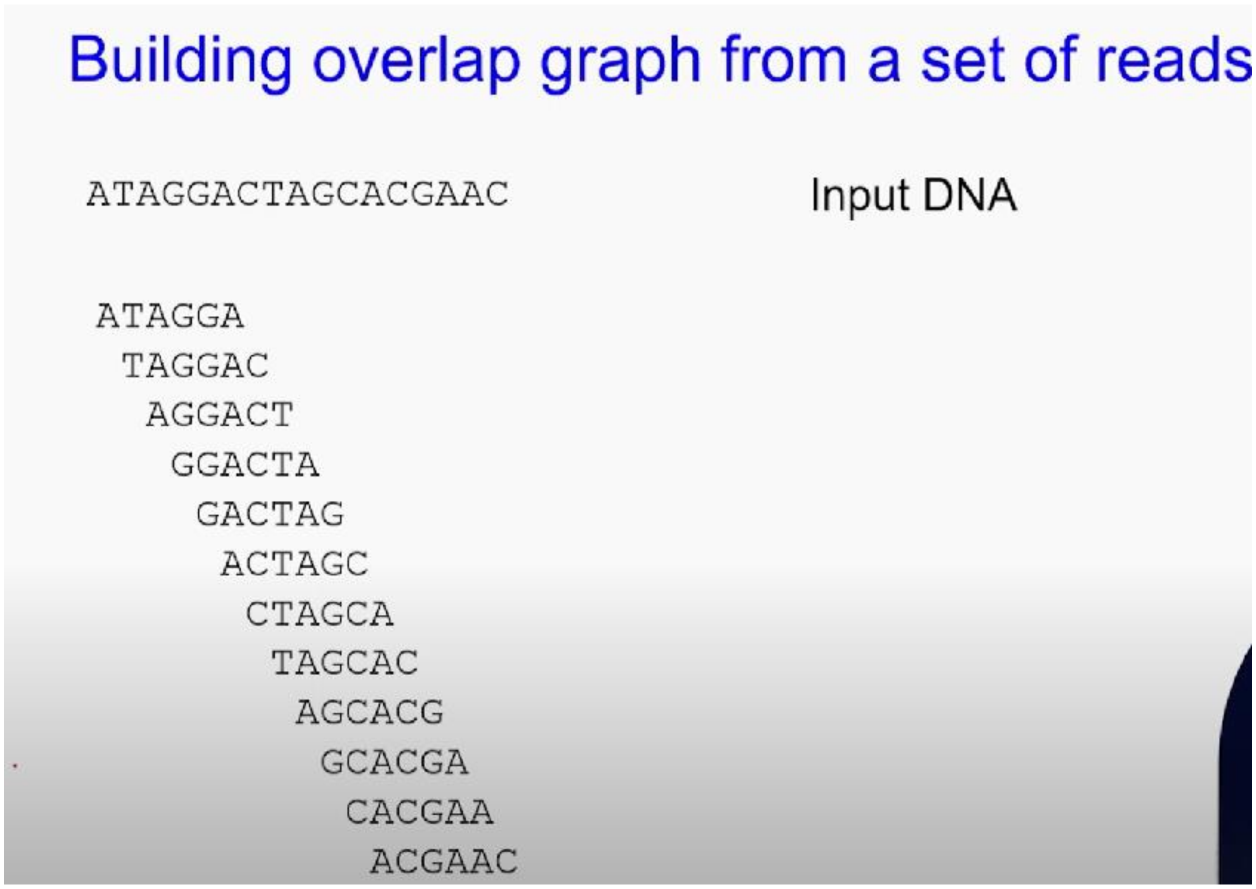
Dr. Riddhiman Dhar, Department of Biotechnology
Indian Institute of Technology, Kharagpur

Good day, everyone. Welcome to the course on Next Generation Sequencing Technologies, Data Analysis, and Applications. In the last class, we started talking about the assembly problem, and we discussed how challenging the assembly problem is. And then we started with one of the approaches for doing the assembly. So, it is called the shortest common supersizing approach. We actually mentioned how we start, right?

So, we start with an overlap graph, which is a directed graph with edge weights, and finally, what we saw is that finding the shortest common superstring from the overlap graph is an NP-hard problem, right? So, the question is: how are we going to solve that? So, in this class, we will talk about this shortest common superstring assembly and the solution for solving this reference genome from the overlap graph. So, we will talk about something called a greedy SCS approach.

So, this actually allows us to generate or derive the reference sequence from the overlap graph, ok? So, this is not an exact solution. This will give us an approximate solution that may not be optimal or the correct solution all the time, ok? So, these are the keywords for this class: super string and greedy approach, ok? So, just to briefly summarize what we have discussed,.

So, we have the shortest common superstring approach. We have built the overlap graph from a set of reads again with a very simple example, and you have seen that the graph can be very complex even with a very small number of reads. Now, once we have this weighted graph, we can then convert it into a cost graph, right? So, we just change the sign of the edge weights, and it is equivalent to the traveling salesman problem, and it is NP-hard. Now, the question is: how are we going to determine the SCS from the overlap graph?

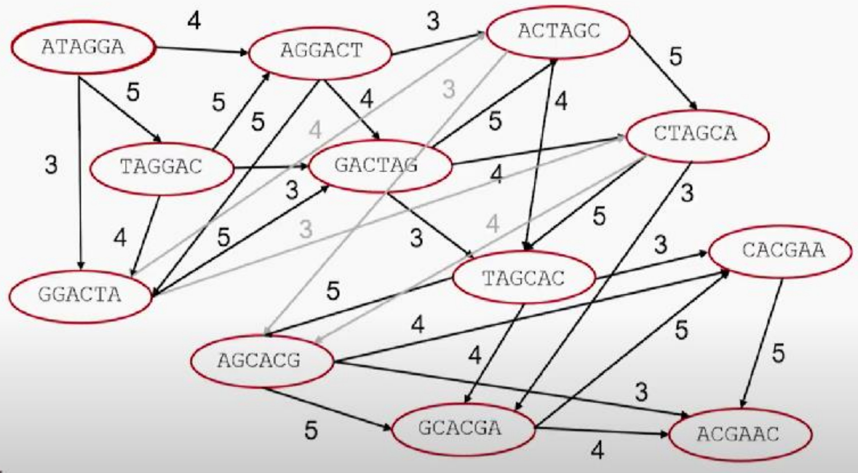


Building overlap graph from a set of reads

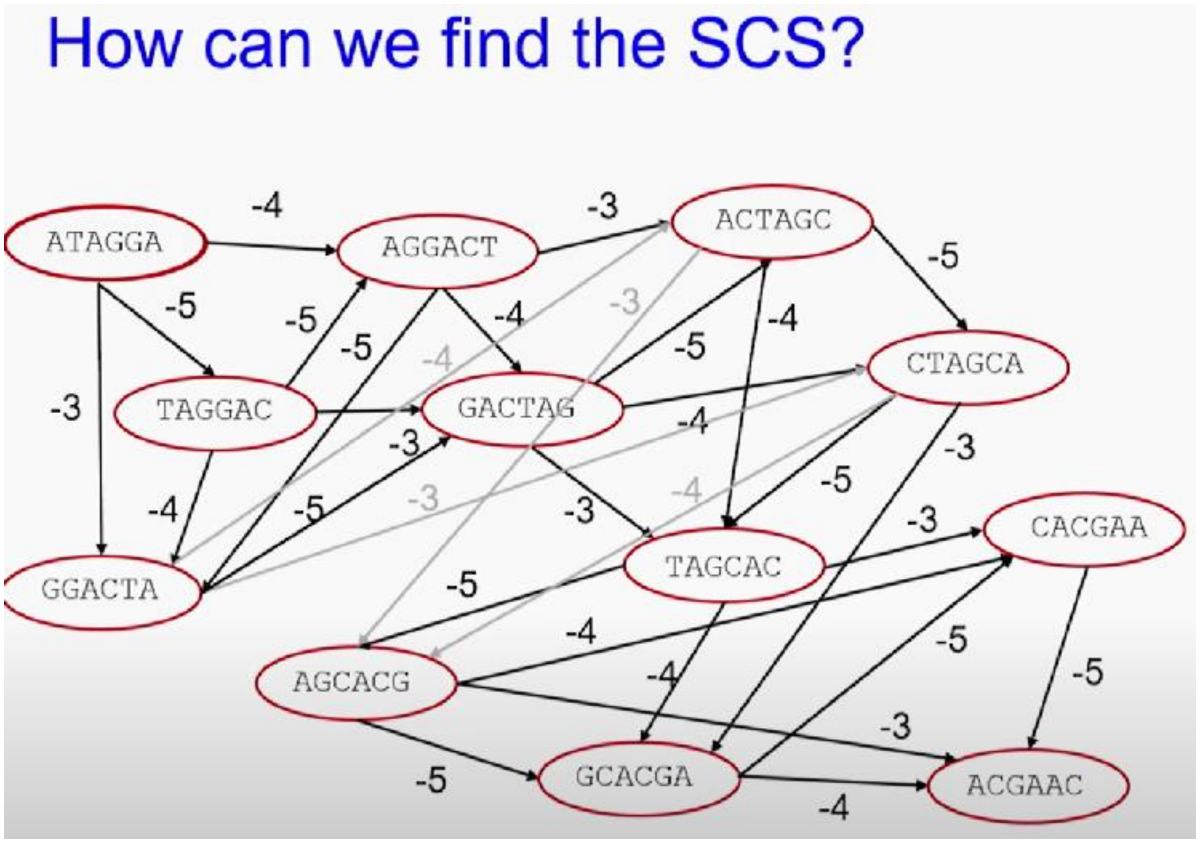
```
ATAGGACTAGCACGAAC                    Input DNA

 ATAGGA
  TAGGAC
   AGGACT
    GGACTA
     GACTAG
      ACTAGC
       CTAGCA
        TAGCAC
         AGCACG
          GCACGA
           CACGAA
            ACGAAC
```

Overlap graph with edge weights



Cut-off length for Overlap : 3



So, this SCS is likely to give us the reference sequence, ok? So, we take, or we just take, something called a greedy SCS approach, ok? So, what is a greedy approach, or what is a greedy algorithm? So, we must have heard about these greedy algorithms before, ok? So, these are approaches that help us solve a problem, and by taking the best choice that is available at hand, They are not going to give us the exact or ideal solution all the time or the best solution all the time, but they will give us a solution, ok?

They find a way to solve the problem by making the best choice that is available. So, what it means is that with a greedy SCS approach, we may not be able to find the shortest superstring, but maybe we will be close to the shortest superstring, ok? And we will obtain some superstrings, but they may not be the correct or best solution, ok? So, we can at least start generating some reference sequences, right? So, we can get close to the reference sequence, which may not be the correct one, but it might be close to the correct one, ok?

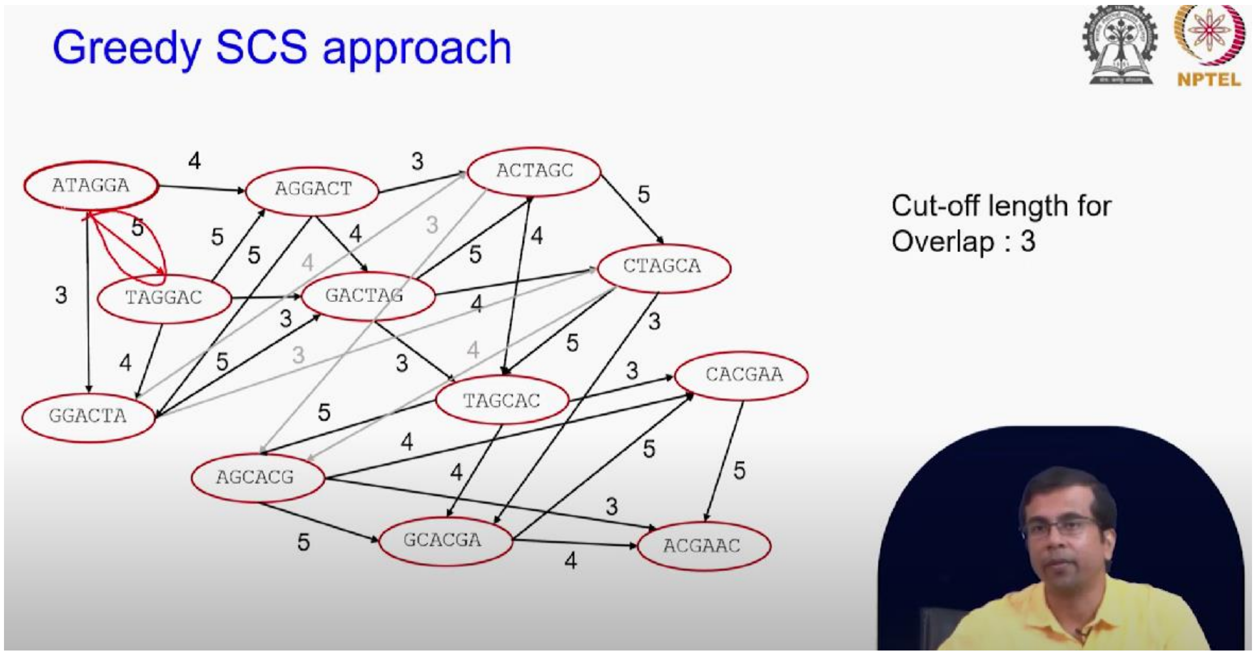
So, what are the steps here? So, the first step is to really choose the reads with the highest overlap.

So, instead of trying to find the most optimal path, what greedy approach will do? It will first start with the reads that show the highest overlap, ok? So, if you remember the graph, we will again go back to the graph and we will actually illustrate this whole process, ok, and we will understand that better. So, instead of trying to find the best path, what will it do? It will just choose the reads with the highest overlap in the graph, right? If it is 5, you will just choose any two randomly from the graph, and then it will merge them, right, and then look for the reads with the next highest overlap, ok?

So, in this way, it will slowly build the reference sequence, or the contigs, or the fragments or parts of the reference sequence, and it will continue until all reads are used. So, it has considered all reads and found this string at the end, or if there are reads that are not showing any overlap between each other, ok. So, that is the idea that the greedy approach will take, ok? So, let us now try this on this graph that we have already built, ok? So, we have these reads as nodes, and we have the edge weights that are showing the overlaps.

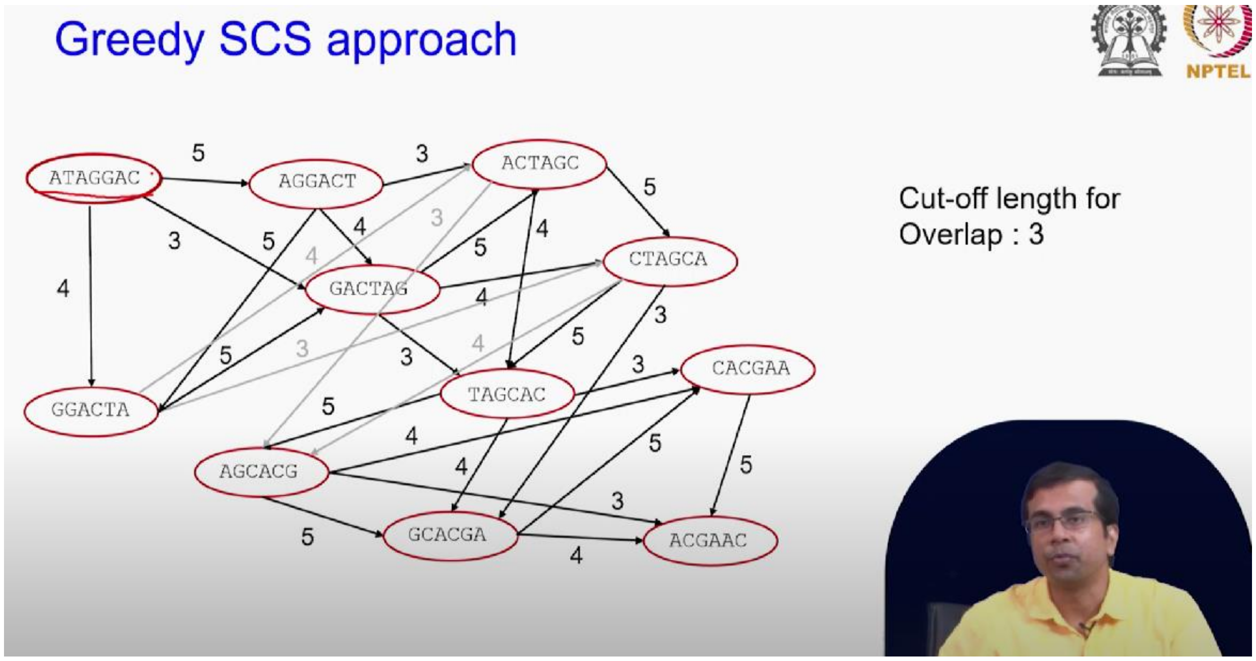
What you see in this graph is that the highest overlap we have is 5, ok? That is the highest value in this graph. So, what greedy approach will do? It will choose any 2 nodes that show overlap of 5 bases, ok? So, at random, you can choose any of these. So, let us say we chose this one.

I am highlighting this here in red, ok? So, these 2 reads show overlap of 5 bases, ok, and the greedy approach will merge them, ok, and subsequently will also update the connections, ok. So, again, we have to recalculate the overlaps between the other nodes or other reads with the newly generated merge sequence. So, this is generated after this merging process, right? This read was generated after the merging process, ok?



Greedy SCS approach

Cut-off length for Overlap : 3

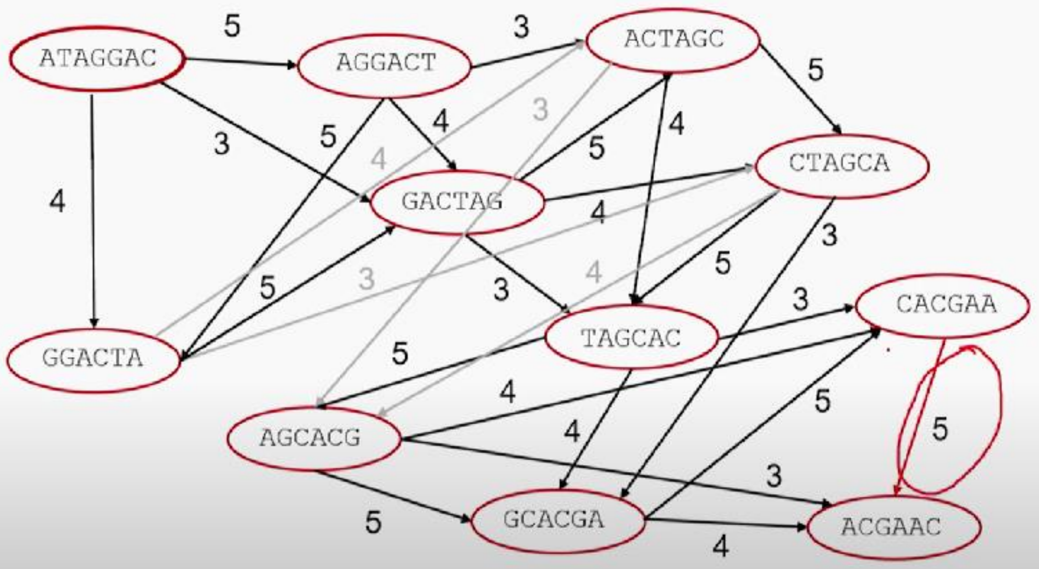


Greedy SCS approach

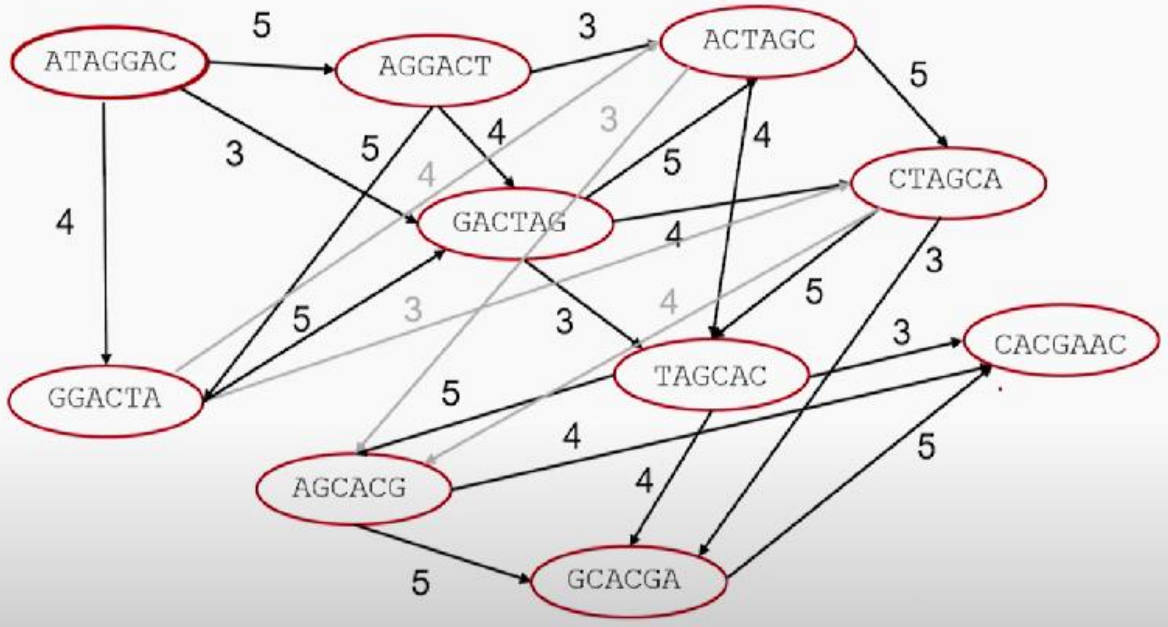
Cut-off length for Overlap : 3

So, once we have updated the connection, the greedy approach will then look for the next set of reads, right? So, the next set of reads also shows the highest overlap among all. Again, the number is still 5, right? Because you see a lot of reads showing this 5 base pair overlap, ok, and again, you can choose anything at random because it is a greedy approach. It will choose anything random, and maybe it chooses this one, ok? So, here is the five-base overlap between these two reads.

Greedy SCS approach

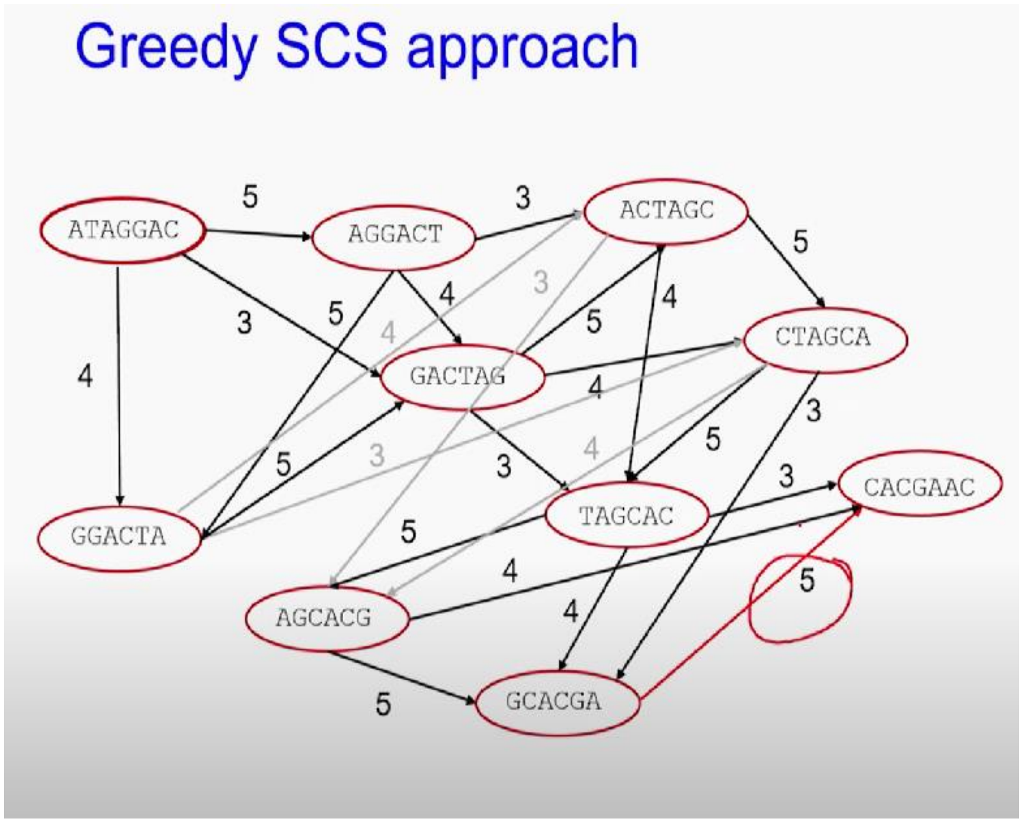


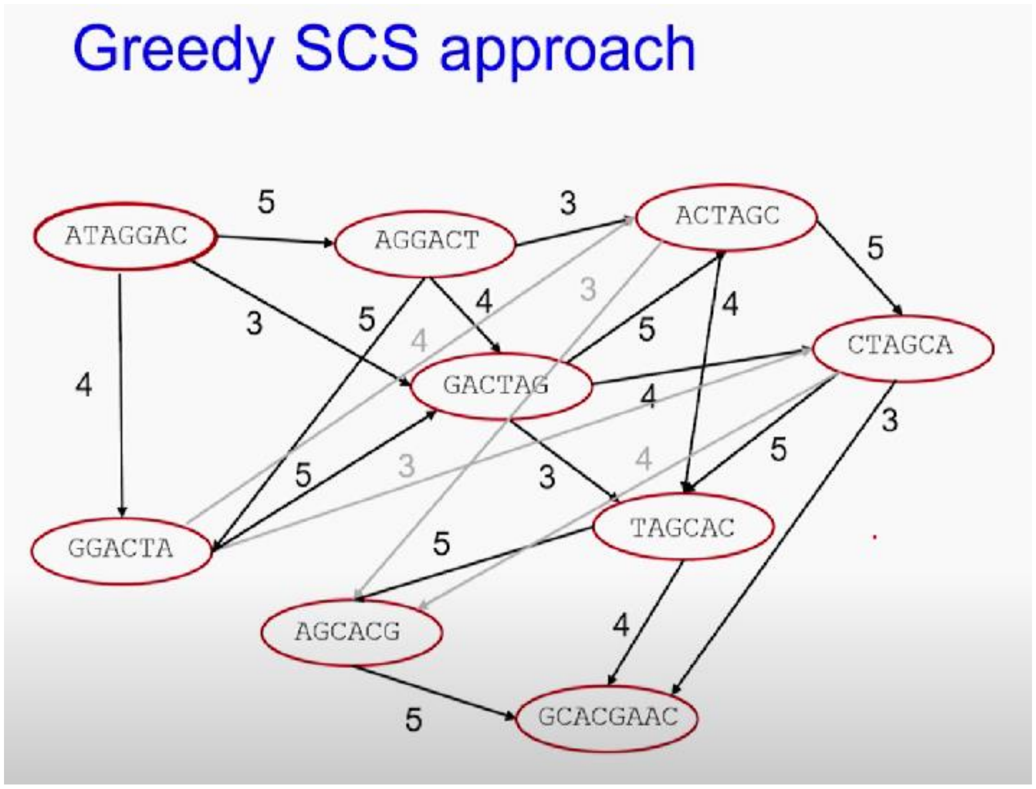
Greedy SCS approach



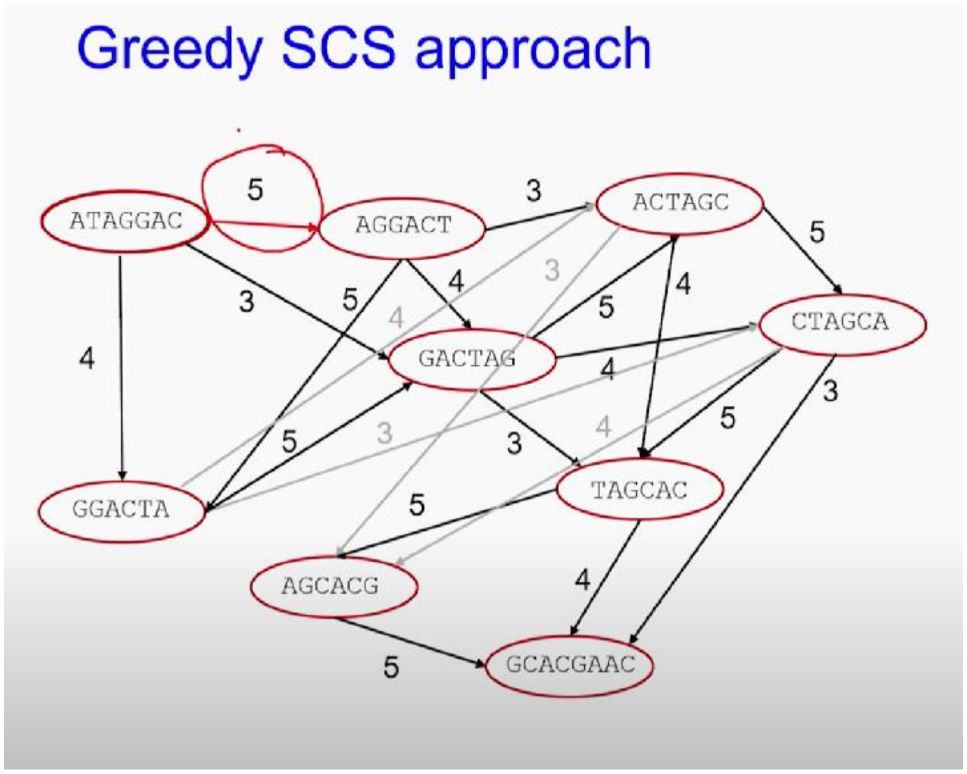
So, the step is the same. You merge and update the connections, ok? So, what I will suggest is that you try this yourself, ok? Unless you try, you will not get a feeling for the whole thing, ok? So, this is something that is very important.

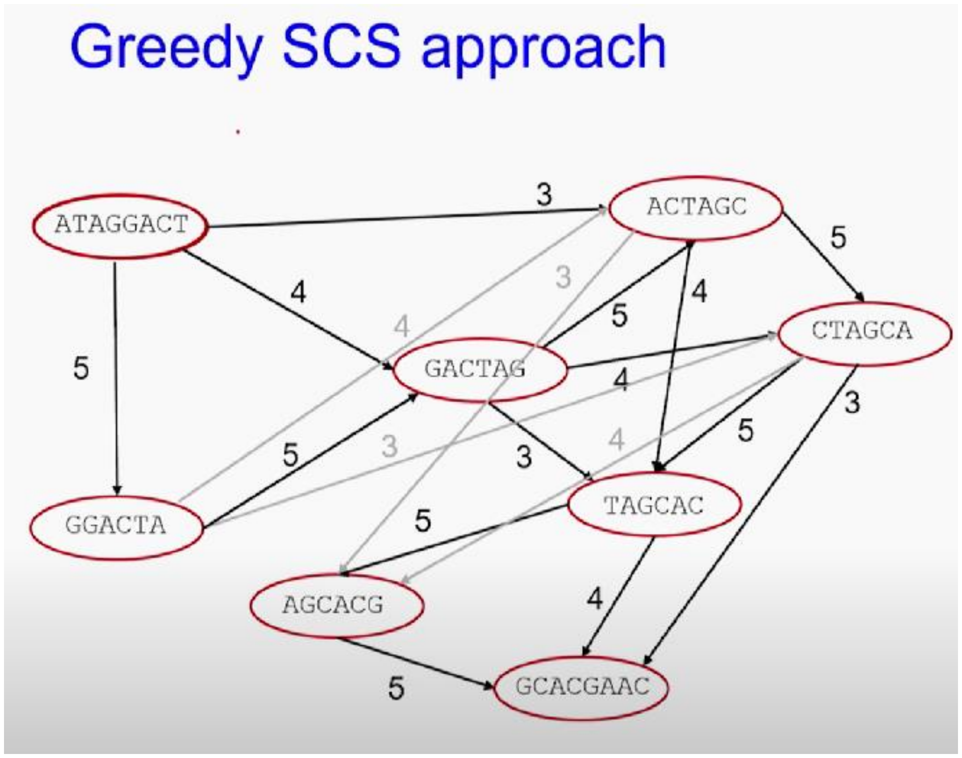
You try it with pen and paper, and this will help you understand this process much better, ok? So, then you continue. We can take anything, right, anywhere, because this greedy approach will choose randomly. So, let us say it chooses this one, ok, this connection. So, it has to merge this and again update the connections, ok?





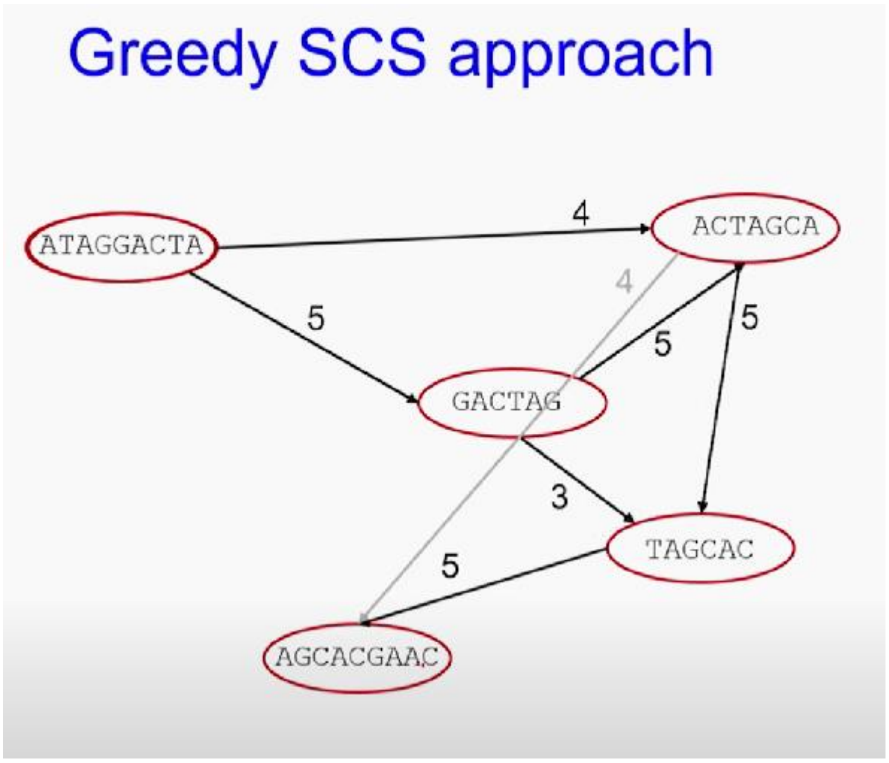
The next step is the same thing we do, right? So, again, we have chosen here, and we still have like five overlaps, right? So, we can go on, and this will be updated, and the connections will be updated. So, you can see, you can go on, right this, right. So, again, we can choose this one here on the left; it will be updated, ok?



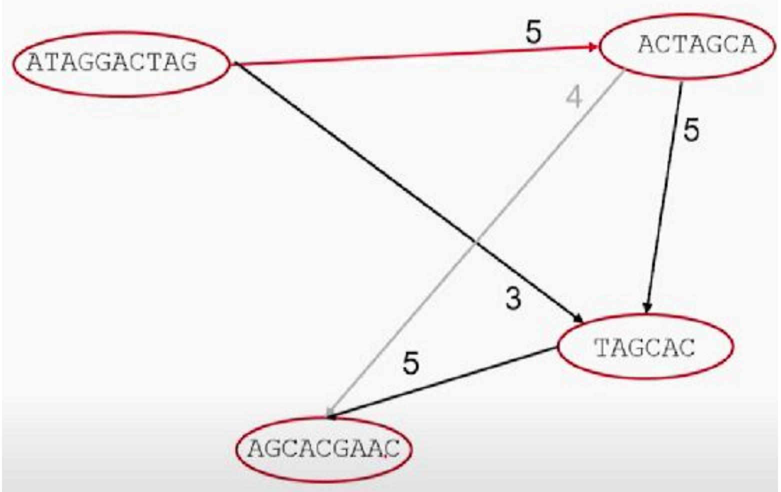


Then we have this one highlighted in red again, and then this will be updated. And as you can notice, we are now simplifying the graph, right? We are getting fragments of the reference sequence, and we are also simplifying the graph. So, there is no traversal involved, right? So, the greedy SCS approach is just merging these reads based on their overlap, ok, and we are getting something close to the reference sequence.

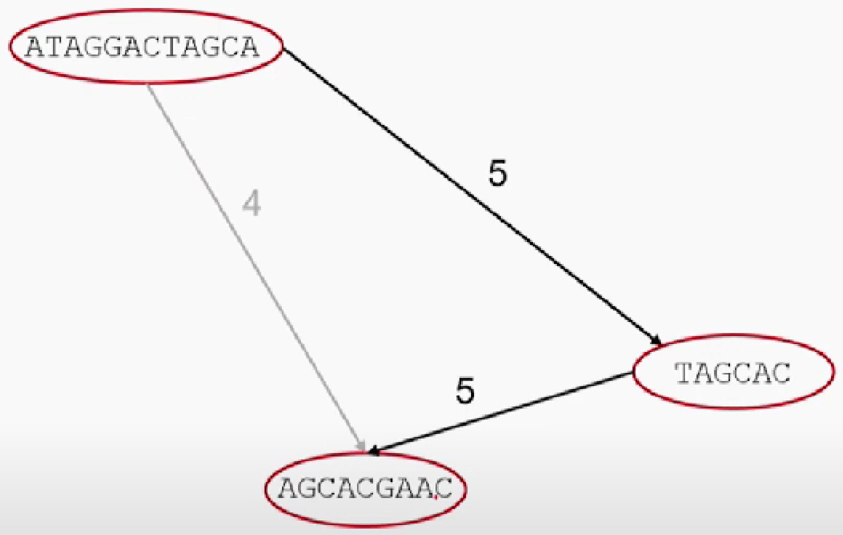
So, in and let us see if we get the right or correct reference sequence, ok? So, this will continue, and at each step we simplify the graph, and then finally, we are left with these three nodes, as you can see. So, the greedy approach will simply take this one and this one, right? So, this one, this path will be taken by this greedy approach, and finally, what we will get is the reference sequence, ok? We will end with only one string, and this is our reference sequence.

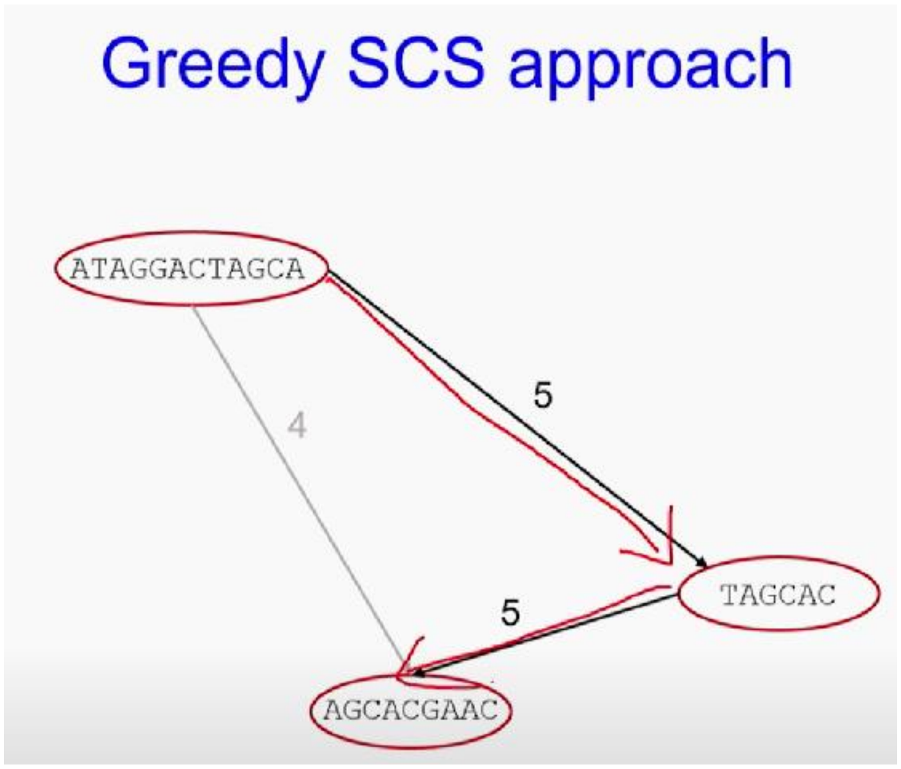


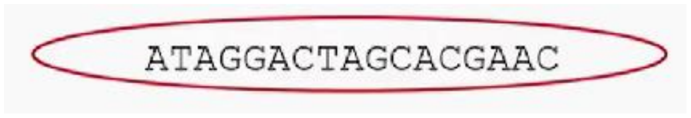
Greedy SCS approach



Greedy SCS approach

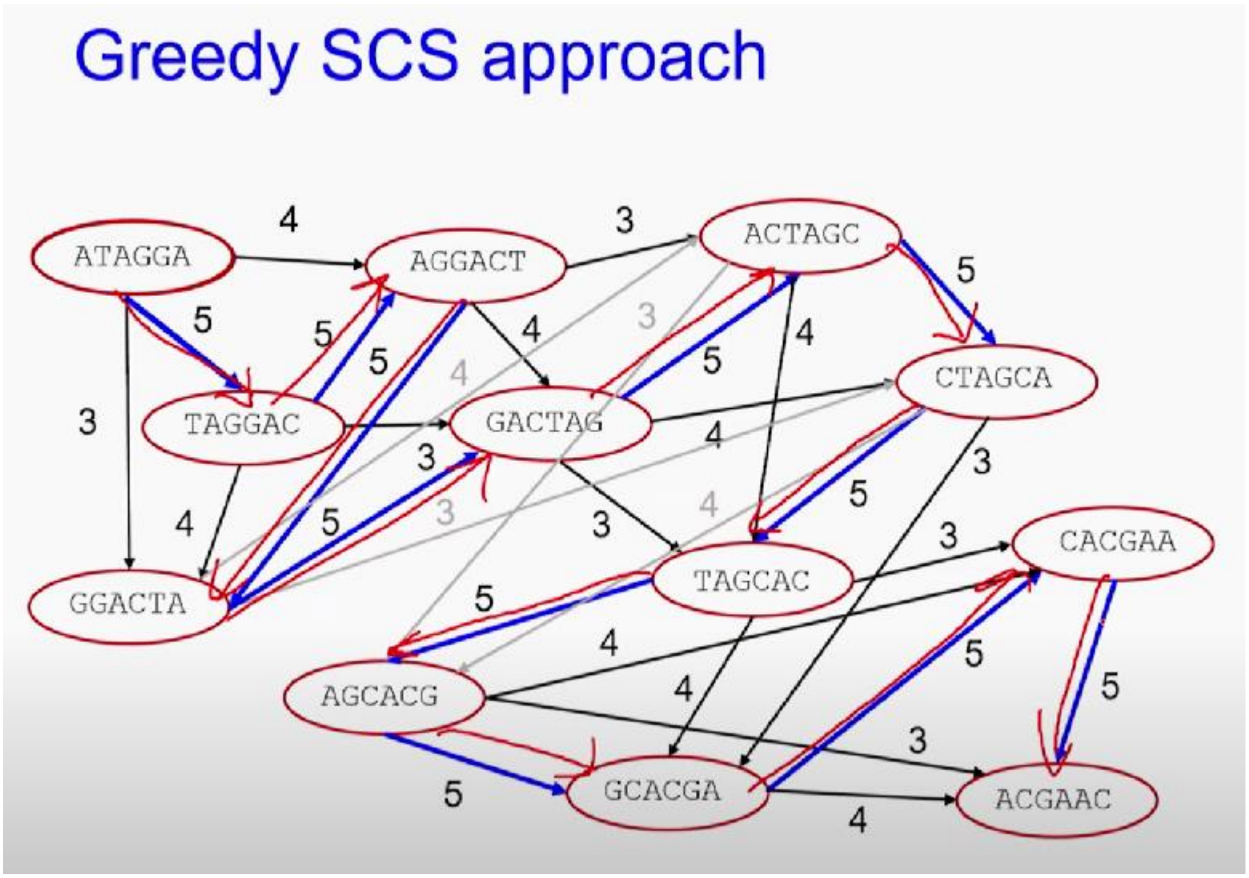



Greedy SCS approach



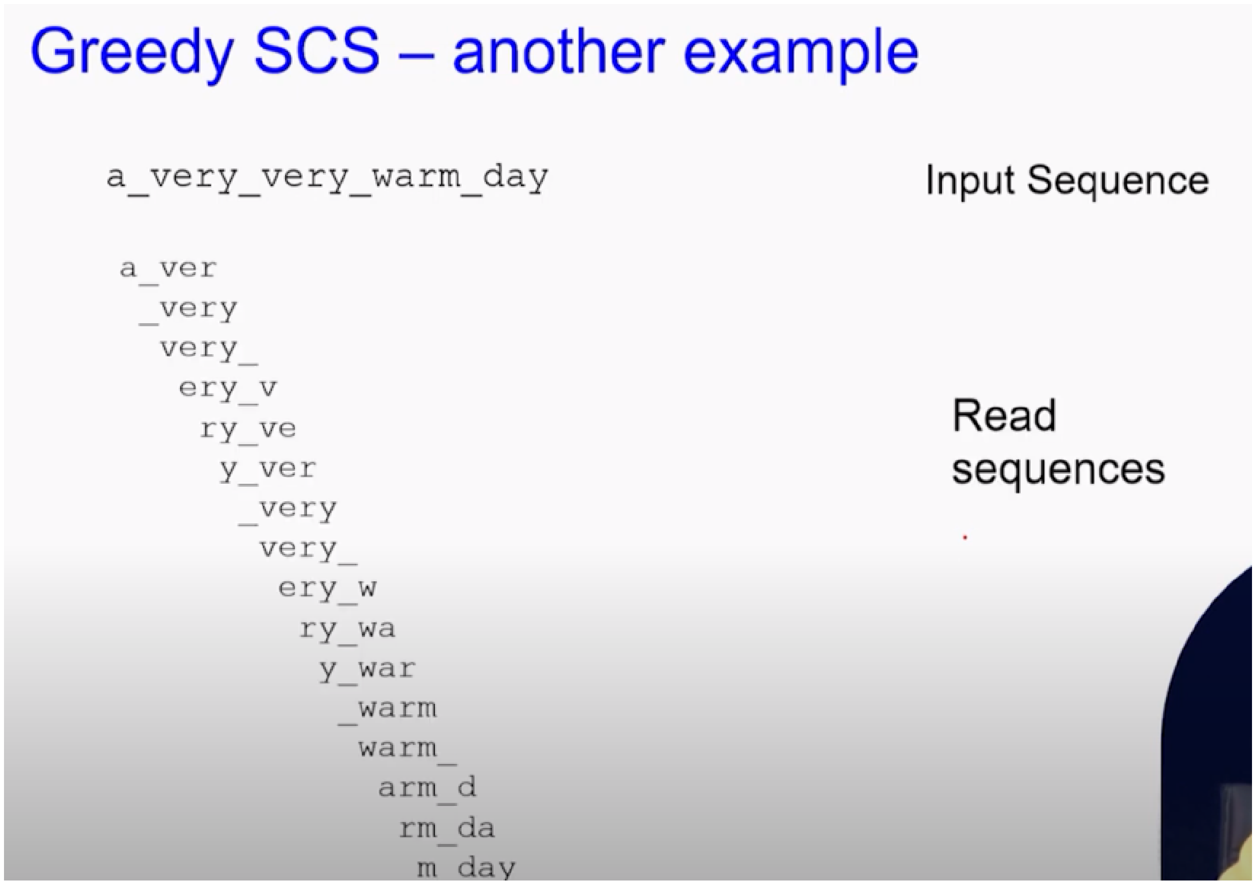
And if you compare with the starting sequence, you will see we got the correct reference sequence in this approach, ok? So, this means this greedy approach can actually give us a solution, ok? In this example, this is the correct solution, but the question is, will it always give us the correct solution? So, for that, we will take a few more examples, and we will try to understand whether the greedy SCS approach is giving us the correct solution or not. So, we can represent this greedy SCS approach.

Instead of this merging thing, we can also think of this as traversing through the graph, right? So, you can think of this path, right? We have this here, we can traverse here, we are coming here, and, ok. So, you can see this highlighted here in this blue color in this graph, ok and then you are going through this, through this path, this path, ok. So, you will get the same results, ok?



If you go along these paths, you will get the same results, and you will generate the reference sequence, ok? So, instead of merging and etcetera, you can represent this as paths through the overlap graph. You can also represent it as simple strings instead of this graph representation. We will also see in one example how we can do that, ok? So, the question is, would the greedy SCS approach give us the optimal solution in all cases?

So, let us take other examples and see if we get the correct solution, ok? So, here is a very simple example instead of this DNA example, and I have just taken a very wordy kind of example, right? So, where you have something called a very warm day, ok and we can generate, let us say, read sequences. These are again examples of certain lengths, right? So, let us say we have reads of length 5, ok, and we can generate the overlap graph with the cut-off length for overlap of three bases or three letters, ok, in this case, ok.



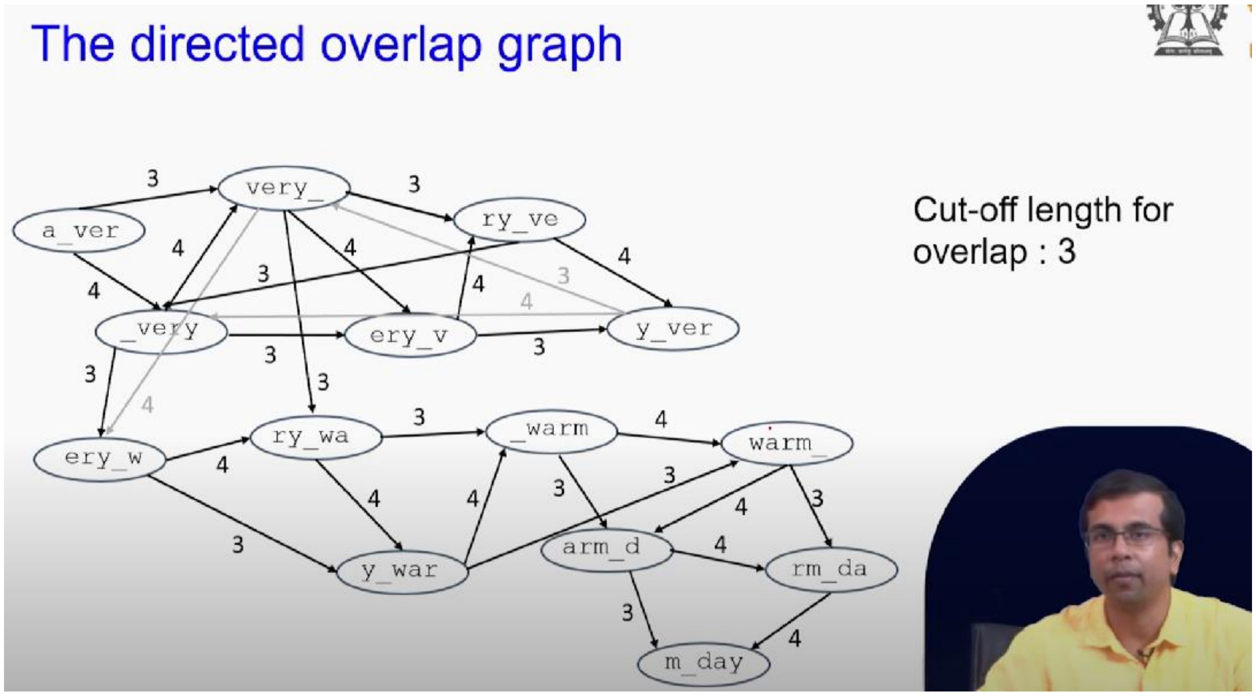
Greedy SCS – another example

```
a_very_very_warm_day
```

Input Sequence

```
 a_ver
  _very
   very_
    ery_v
     ry_ve
      y_ver
       _very
        very_
         ery_w
          ry_wa
           y_war
            _warm
             warm_
              arm_d
               rm_da
                m_day
```

Read sequences

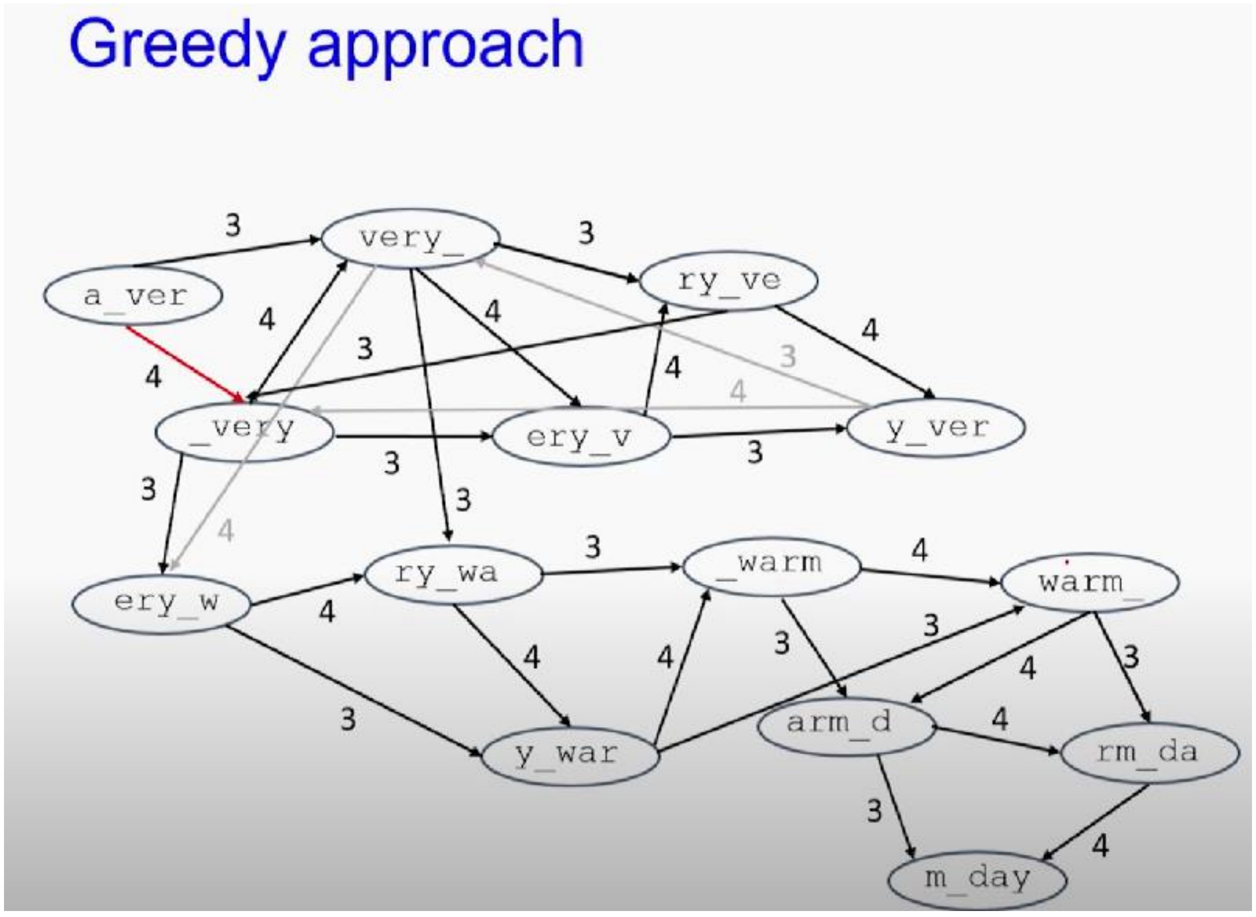


The directed overlap graph

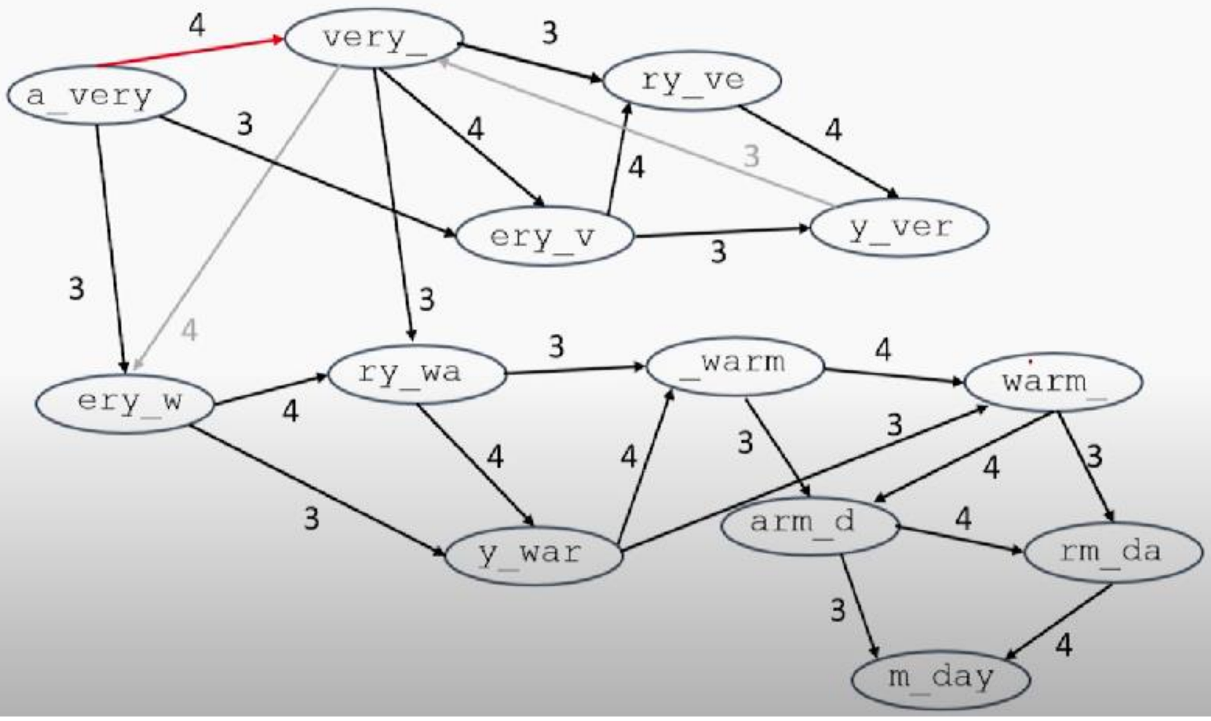
Cut-off length for overlap : 3

So, again, you can generate this, you can take any other example, and you can try to generate this kind of overlap graph, and you can try to understand the properties of the SCS approach and the greedy SCS approach, especially. So, again, we have this graph with the edge weights, OK, and the directions. So, what we will do in greedy SCS is that we will again identify the nodes that show the highest overlap and merge them, ok? So, this is what we are going to do, right? You can see you have this first one here, okay, in red.

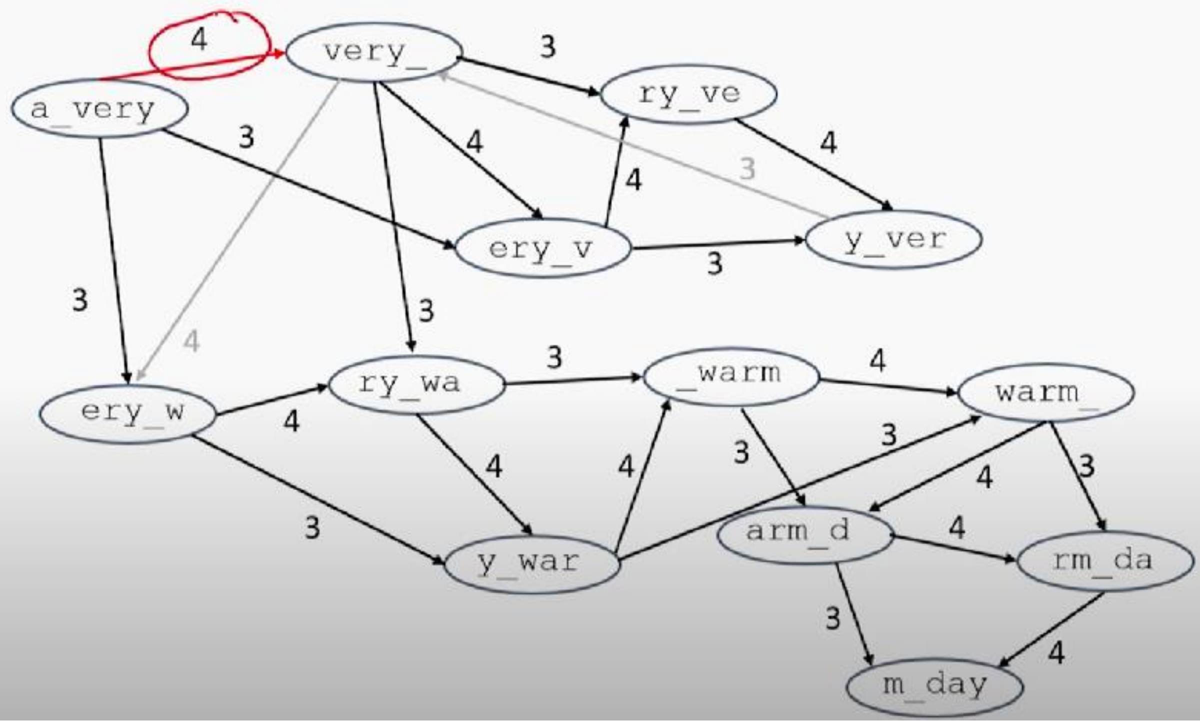
So, the greedy approach will merge them, update the weights, and take the next one, ok? So, here is the next one. Again, it will look at the overlap length, ok, and it will update, ok. So, it can choose anything out of all these possibilities, right? So, in the next step, it can choose this one, maybe, ok or maybe this one, right?

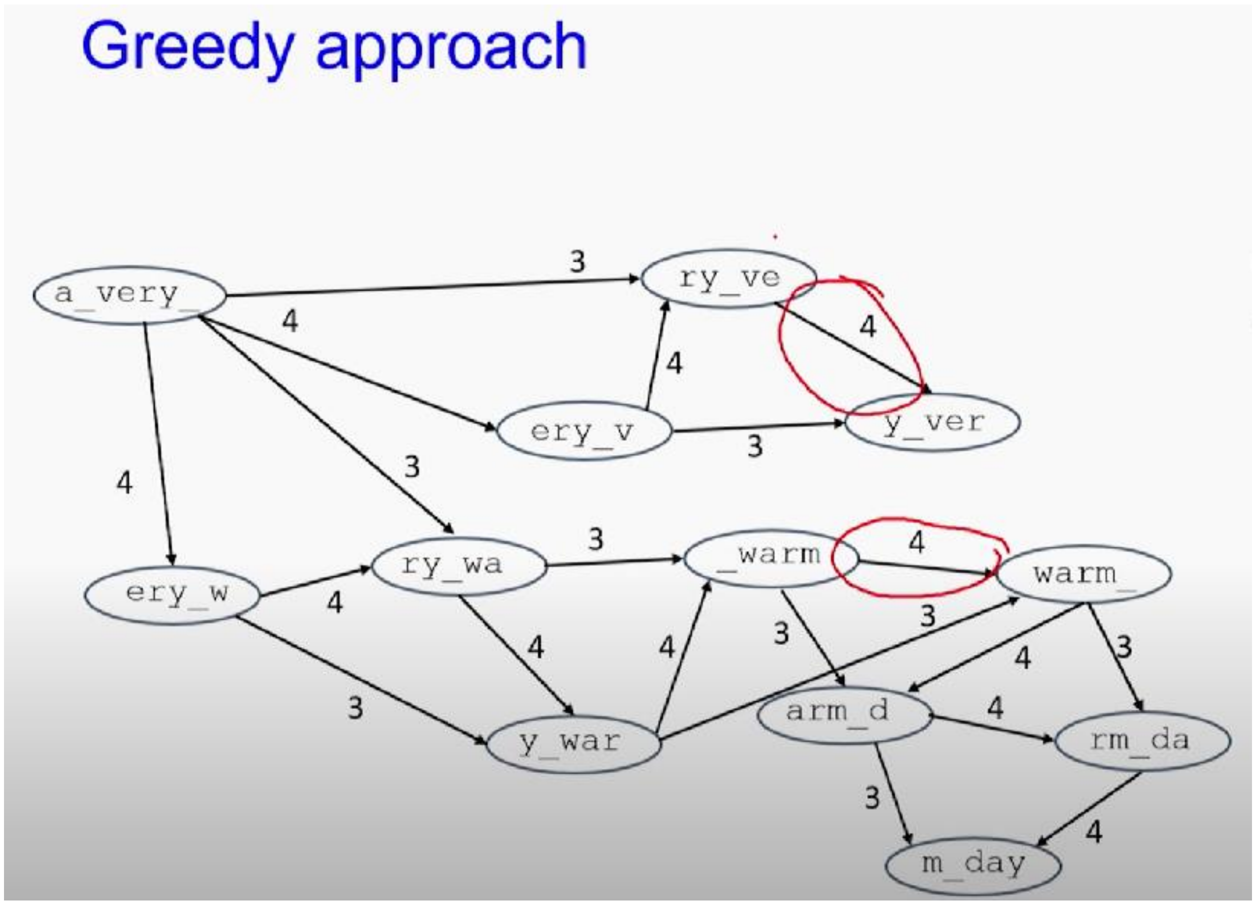


Greedy approach



Greedy approach

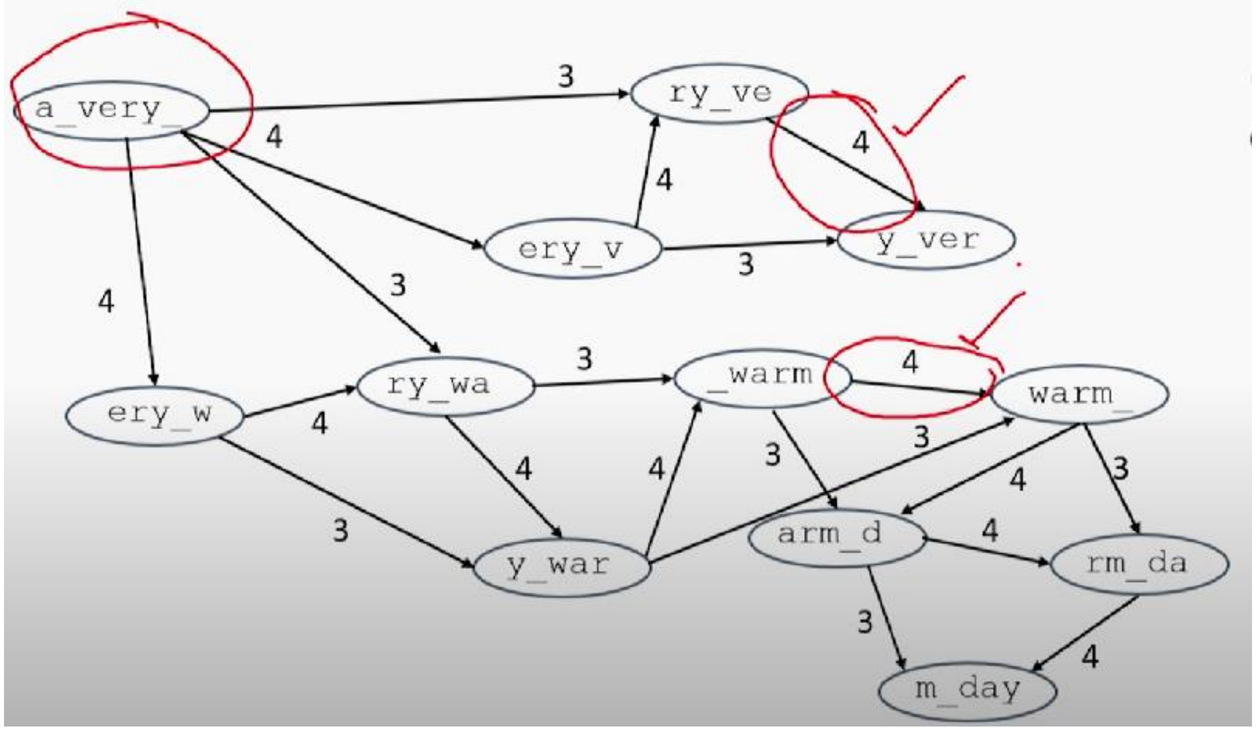




So, it is not necessary that it will have to go in one order, right? So, it will not just choose everything around this first node here, ok? So, it will go around in other ways as well, ok? It can choose it. So, what it means also now is that if you run the greedy approach multiple times, you may not get the same steps, ok?

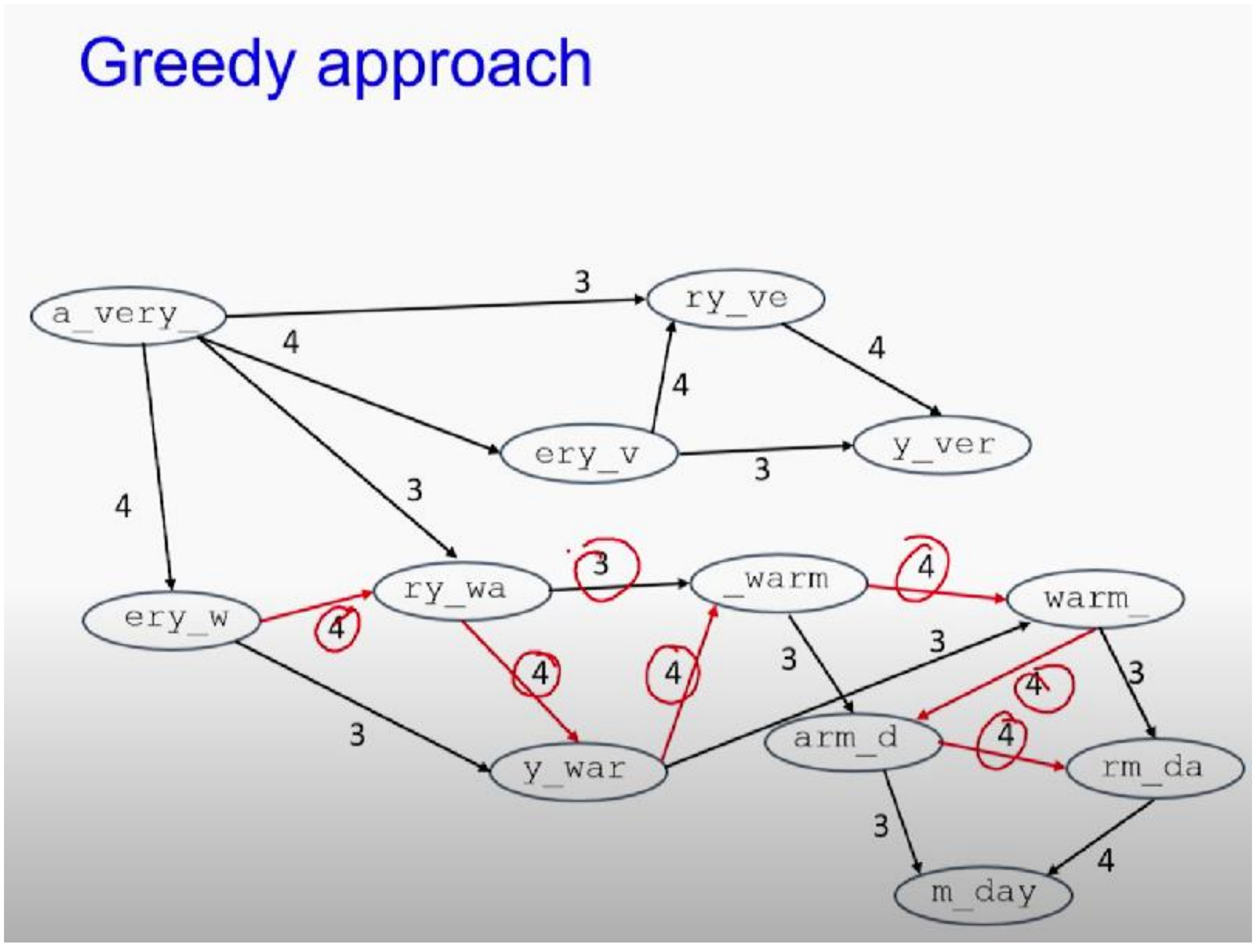
So, it might go in different ways, right? Because it can choose these overlaps, these nodes, or these overlaps at random depending on the value only. So, in one run, it might choose this one; in another run, it might choose this one. So, the steps might be different, and we can then ask whether the outcome is also different, right? If we ran this greedy approach multiple times, would we get the same outcome all the time?

Greedy approach



So, let us now continue with this part here. It is actually pretty simple, right? So, as you can see, you have this path again, and one thing you probably notice is that this path goes through the highest overlap values, right? So, if you see here, the overlap value is 4, and it is also 4, right? So, here is 4, right? So, because a greedy approach will choose the highest overlaps only, it will not choose the ones that are not the highest values in that path, right?

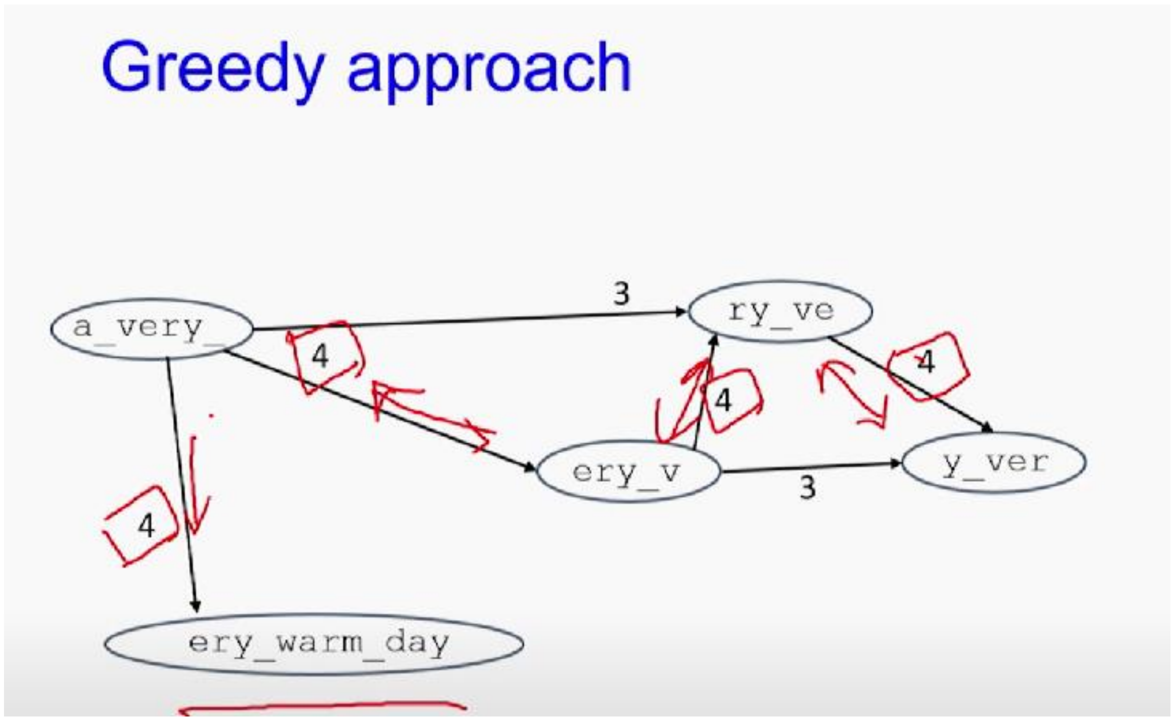
So, it will not choose the three paths that carry the weight of three, ok because it will always get to take the highest overlap, ok. So, for this part, it is actually quite simple. It will simply end up in this part where it will have this ery_warm_day. So, let us say a greedy approach took this path, right? It will; it has merged this segment and generated this kind of part of the reference sequence, ok?



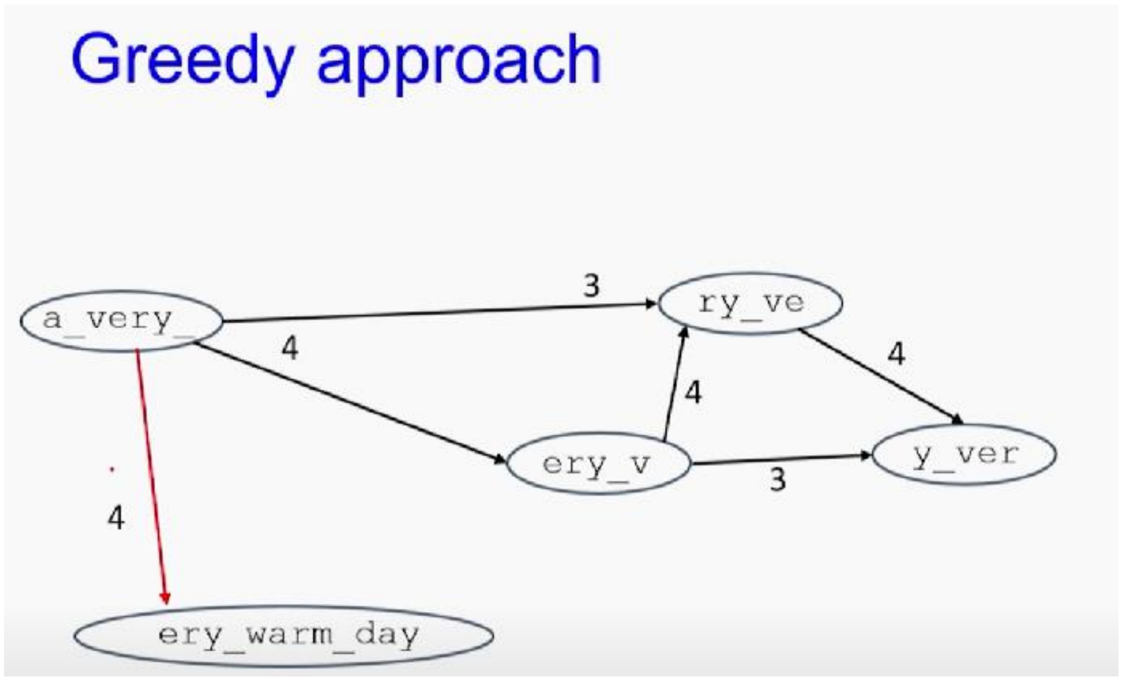
Greedy approach

So, what you see now again has these two approaches, right? So, you can take again these values, maximum values 4, and you have these four values in different places, ok? So, at four different places, you have these values, ok, and again, a greedy approach can choose anything, ok. So, let us say it chooses this path, right? So, first it merges these two, ok, then these two maybe, then these two, ok.

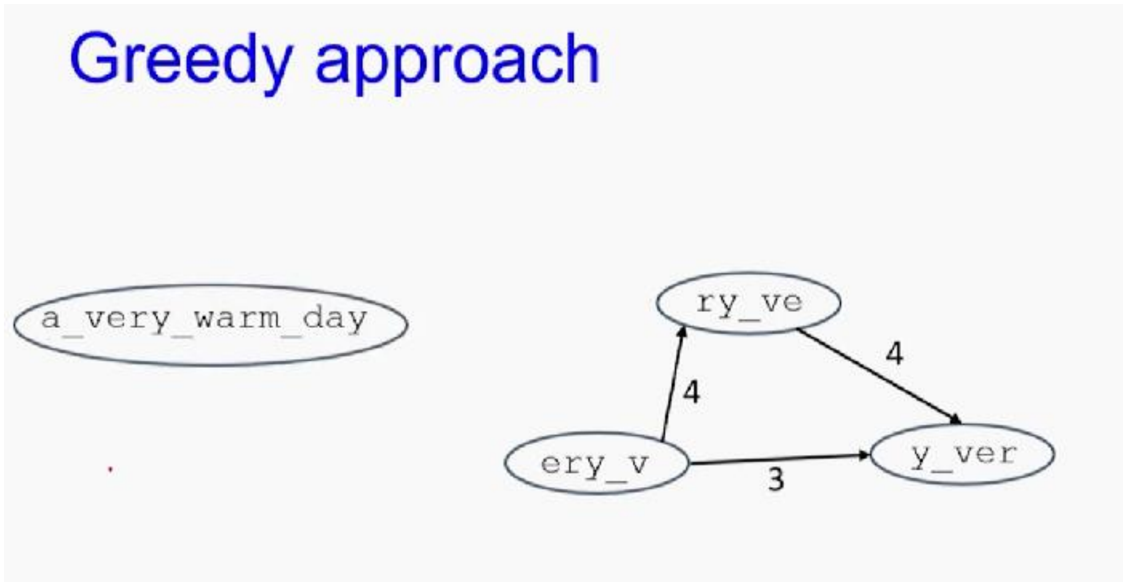
So, it takes this path, merges them, and then chooses this one. So, in that case, what you will get is a very, very warm day. That is the correct reference sequence. Imagine that, by chance, because it has this option, it chooses this one first. Instead of these other three, it chooses this one first.



So, what will it generate? Let us say it chooses this one. So, what it will generate is this one, a very warm day, and the moment it generates this part of the reference string, you see, like for the other graph, there is no overlap with that fragment, ok? Based on the properties that we have discussed—the suffix and prefix—this part does not have overlap, right? So, what you end up with are two different or two disconnected fragments. But if you notice here, ok, this actually contains all reads as substrings, ok.

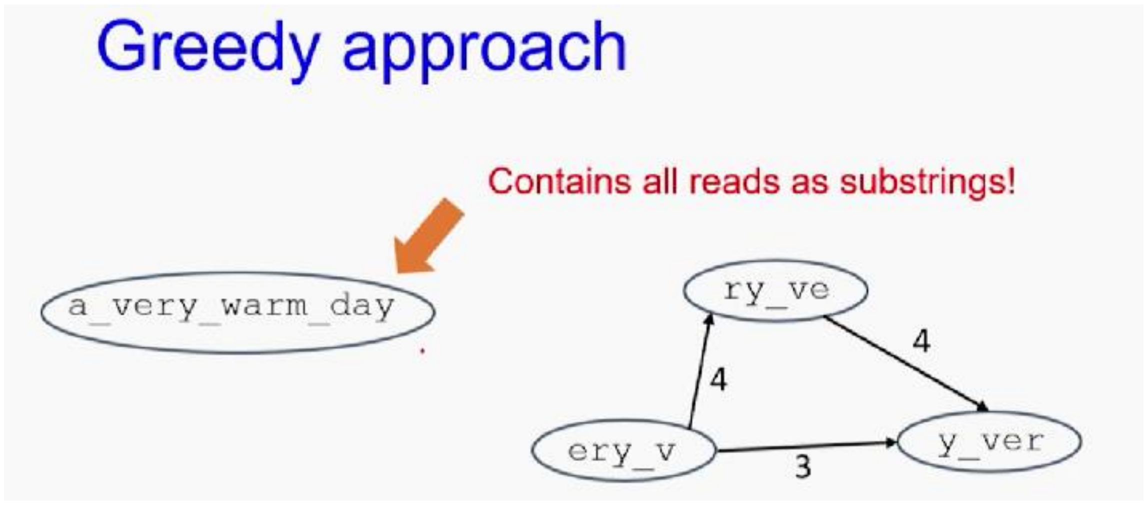


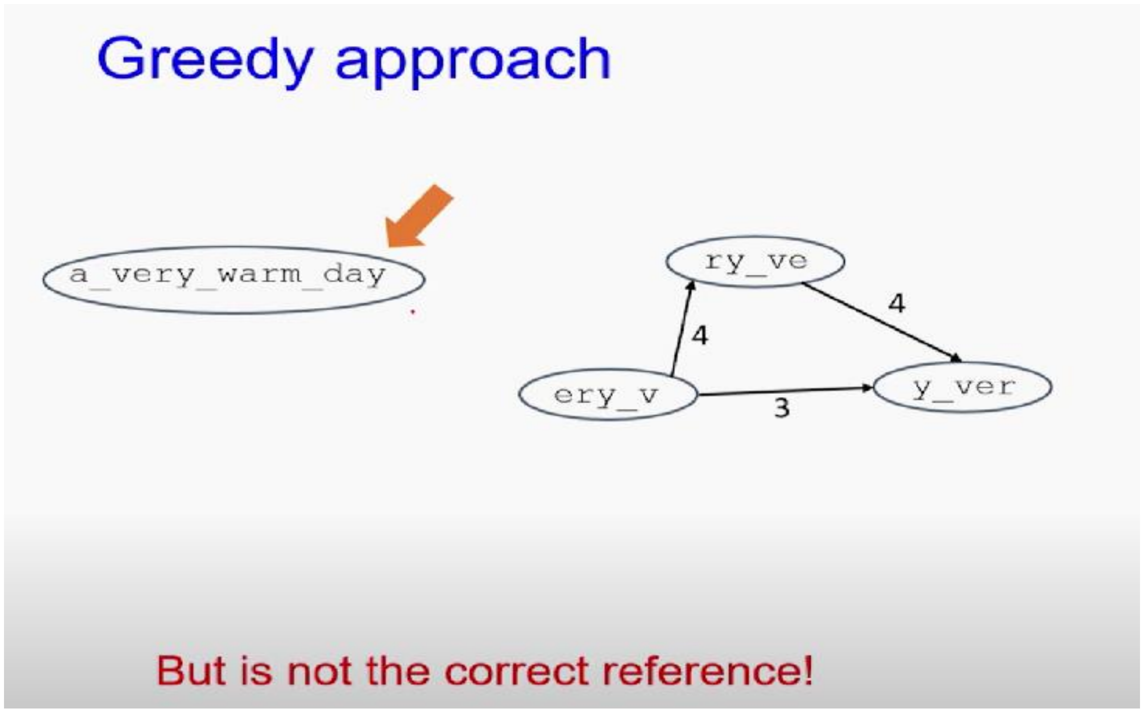
Greedy approach



Greedy approach

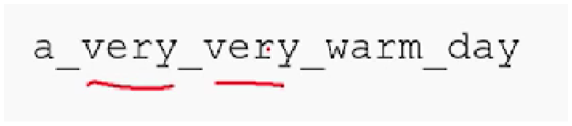
Here, you see that this one is giving all reads as substrings, right? So, we have this very, very warm day, right? So, it contains all the substrings, all the reads in this string, right, as substrings, ok? But then this is not the correct reference, right? So, the correct reference sequence was a very, very warm day, ok?


Greedy approach


Greedy approach

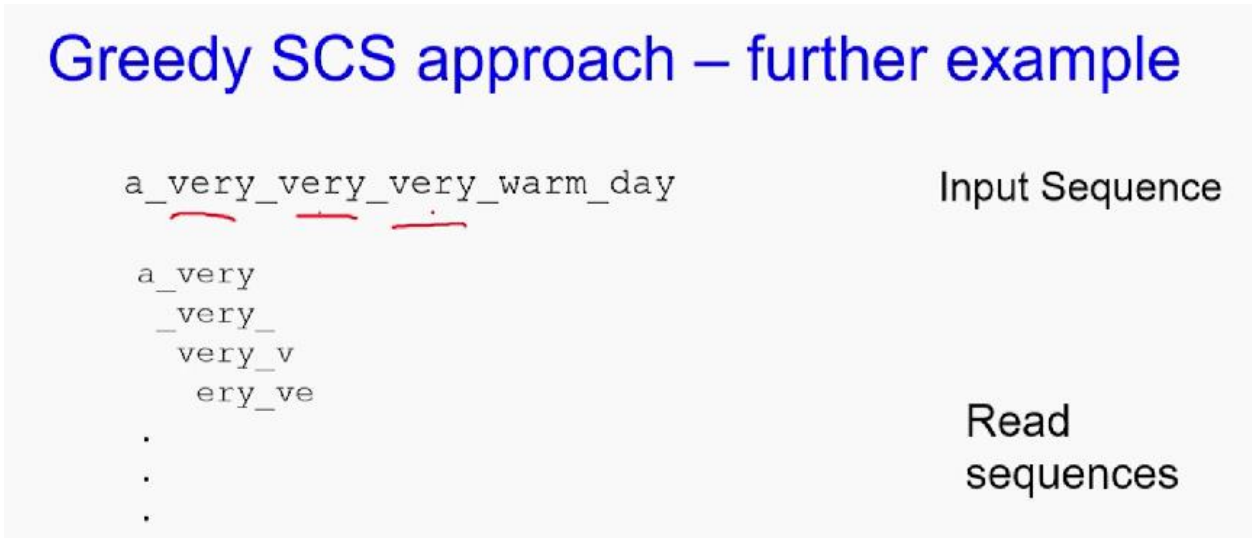
But is not the correct reference!

So, this means what you see is that the greedy approach is not giving you the correct reference sequence. And in this case, because of the choices, it gave you only a part of this, right? So, it gave you two different disconnected graphs and then only part of the correct reference sequence, ok? So, why did this happen? So, this is a question, right? So, why did we see this kind of scenario? And going back, you can probably now understand that this repeats here, ok?



a_very_very_warm_day

So, one word is repeated wisely, ok? And because of these repetitions, they might actually confuse that approach with the greedy approach, ok? You have these repeats, and this might lead to this kind of non-optimal assembly, or we get only part of the assembly, ok? So, we will illustrate that with one more example, and that will actually contain more repeats, and you will now understand the process and why that happens, ok? So, now that we have expanded this repeat region, it is a very, very warm day, right?



Greedy SCS approach – further example

a_very_very_very_warm_day

Input Sequence

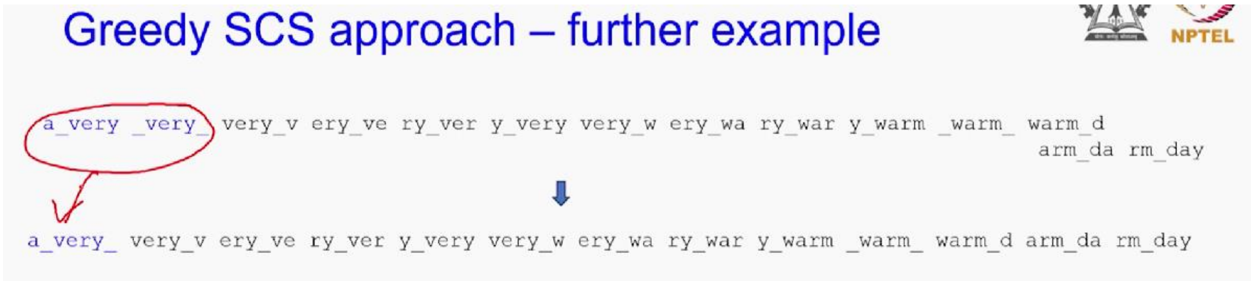
```
a_very
 _very_
  very_v
   ery_ve
.
.
.
```

Read sequences

So, we have had this repeat three times now, ok? And, of course, we can generate these rich sequences. I am not going to generate them; there will be quite a few, right? But you know how to generate this as long as we know the length, ok? So, the length is also increased here, right? So, the length of reads is 6. So, we have 6 letters for reads, and we can generate all possible reads that we will get from these input sequences, right?

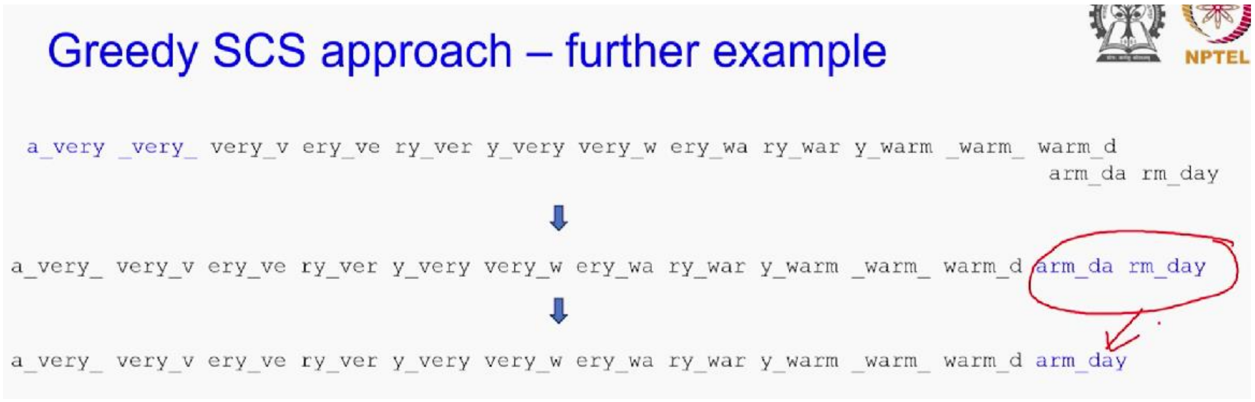
So, once we have these read sequences, what we do now? So, these are the read sequences I have written one after another. We will illustrate two approaches, or two ways of doing this. One is just merging them from here, not looking at the overlap graph at all, and then subsequently we will look at the overlap graph. So, what this will do is again remember the approaches. So, the greedy approach will simply calculate the overlaps, and then we will merge, ok?

So, it will merge any two that show very high overlap, ok? So, if these are highlighted here, the highest overlap would be 5, right? So, the reads are of length 6. So, the highest overlap you can see would be 5 letters, ok? You cannot have six overlaps, 6 letters overlap because then the two reads are identical. So, what does this greedy approach do? Then it will start merging them, ok?



Greedy SCS approach – further example

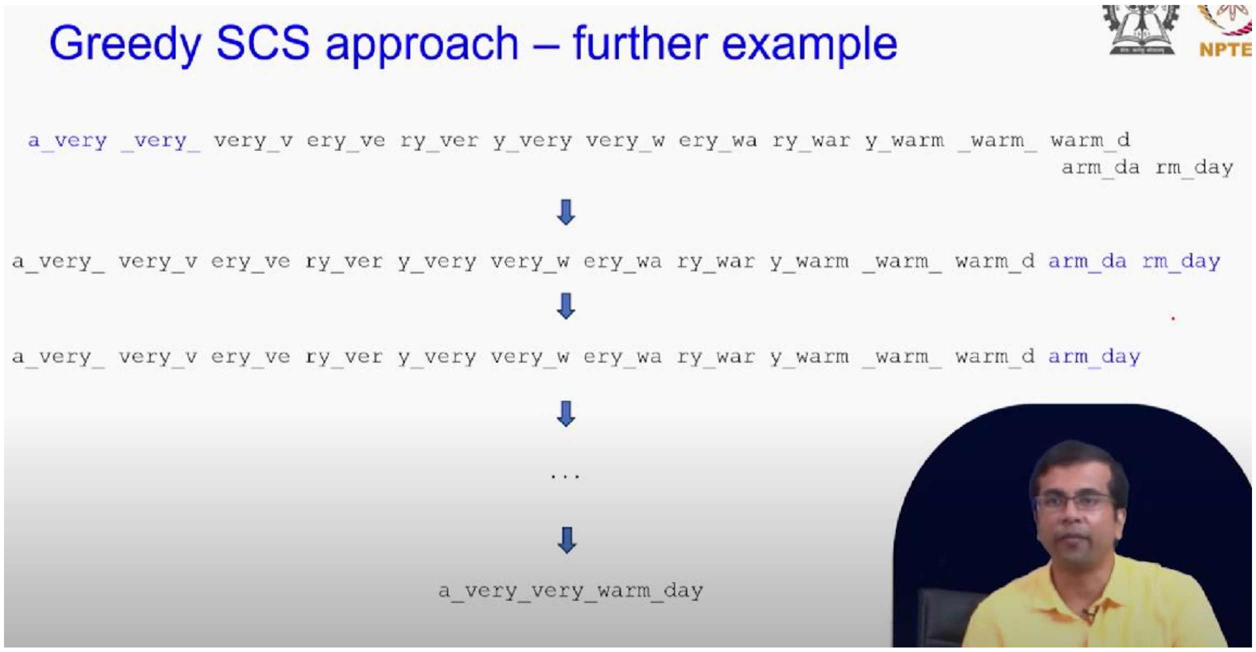
```
a_very  _very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d
                                                                               arm_da rm_day
                                          ⇓
a_very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d arm_da rm_day
```



Greedy SCS approach – further example

```
a_very  _very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d
                                                                               arm_da rm_day
                                          ⇓
a_very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d arm_da rm_day
                                          ⇓
a_very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d arm_day
```

So, it merges these two into this one here, ok? And then, again, it will continue this process, and let us say here that it then merges these two here, ok? And then we get this one, ok? And you can think of, right, you can imagine that this process actually would go on. I am not going to illustrate all the steps, but this process will go on, and in the end, you might get this one: a very, very warm day.



Greedy SCS approach – further example

```
a_very _very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d
                                                                          arm_da rm_day

a_very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d arm_da rm_day

a_very_ very_v ery_ve ry_ver y_very very_w ery_wa ry_war y_warm _warm_ warm_d arm_day

...

a_very_very_warm_day
```

So, you probably noticed something, right? So, again, we have missed one way, all right. So, this is the reference string that contains all the reads, right? So, if you check all the reads, these are contained in this refined sequence, right? So, this is the super string that will serve as a reference sequence without any problem, but this is not the correct reference sequence as we have just seen, ok? We started with a very, very warm day and we had repeats, right?

The very word was repeated three times, but at the end, when we assemble these reads, we get this very warm day with one very word missing from the reference string, ok? So, then you probably understand now, ok? So, when you are doing this assembly with this approach, the repeats are the culprits here, right? So, they are causing these problems, and they are not allowing the correct assembly. We will now illustrate this with the graph, this directed overlap graph for the same example.

As you can see, this is quite a complex graph. Again, try yourself; only then will you understand how we can add these edges. Again, different colors do not mean anything; they are just colored in different ways for better visualization because there are so many edges going from one node to another. So, again, we follow the same principles that we have just discussed, ok? So, we have the directions determined by the suffix and prefix where the overlap is, right, which will determine which would be the source node and which would be the sink node, ok?

So, this is again determined by the overlap. And the weight—the edge weights that we see—are determined by the length of overlap. Again, the maximum length is 5 here; we cannot have 6 because the reads are of length 6. So, once we have this overlap graph, we can then take the greedy approach in this graph, and we can find solutions that will give us the reference string. Again, as I have mentioned, right, so because this is a greedy approach, every time you run it, you might get different results, ok, because of the choices that you can make, ok. So, maybe you can merge; in one case, you might start with this merging; in another case, you might start with this one; and then in another case, you might start with this one, because these are all showing values of 5.

So, in principle, you can start with any of those nodes and merge them. Now, the thing is, right, if you start with one, you might get different results compared to if you start with another one, ok? So, all these decisions will actually impact the final outcome, the final reference sequence that you generate. So, this is something you realize again: with a very small string as a reference sequence, we are getting these different solutions or different ways to generate the reference sequence because of the greedy approach, right? So, we have different choices whereas, in reality, right, so where you have these billions or millions of reads, there would be even more choices, right?

So, you might get tons of these different outcomes if you run the greedy approach multiple times, ok, and none of them might be the right solution, ok. So, with this in mind, we can actually try to solve this and see what we get from this outcome, ok? So, let us say we take this path, right? So, what we have done is right, and then we take this path here, ok?

We take this path, this one, and then this one, right? Here is one path now. So, I am highlighting these paths in blue. You can see them, right?

And you can take this path. Here is the other one, right? So, we are taking this path with the highest overlap as you can see, right, and you have this final solution, ok. So, as you traverse along this path, you will see you will get an outcome or the reference string, which will give you a very warm day again with one very missing. So, it is kind of equivalent to what we have just seen, ok? So, it is finding the path with the maximum weight, right, and/or the maximum total score, ok, and then traveling along these edges exactly once. As you can see again, as I have told you, you can have different solutions instead of traveling through this one first. Instead of traversing through this node first, you might traverse through this one first.

So, you might say, OK, this is 5 value. So, let us try this one out, ok? So, again, you can have these different ways of traversing through different nodes, and that will give you different sets of solutions. And again, at the end, depending on these choices, the final outcome will be different, right? So, I think with these examples you now realize, right, that the greedy SCS approach will give you a solution, but that might not be the correct solution, right? So, it might not be the correct reference sequence, but you get some idea or you get close to the correct reference sequence. You also understand that the repeats cause problems or hinder the assembly process, and this is something we need to think about, and we will actually talk about this when we discuss the other approaches as well.

So, what we get from this example when you traverse through this graph, which I illustrated in blue, is that if we traverse that path, we will get this very warm day, right? Again, one very word will be missing, ok? Because of these repeats, we will see that the assembly process is in progress, ok? We are not getting the right reference sequence. Now, you might say, Okay, why are you considering the repeats because this is actually closer to the actual scenario?

So, the human genome, or if you consider mammalian genomes, contain a large percentage of repeats, ok? And then that means the assembly process becomes really challenging when you have this kind of read, small reads, and large repeats—right, a large number of repeats across the genome, ok? So, this is something that we need to keep in mind, and these examples actually give us a feel for the real world, right? So, what are the drawbacks of the greedy SCS approach after having discussed this? So, the first drawback, as we have seen, is that a greedy approach may not give optimal solutions.

You can take different examples, and we will see that at the end we get an optimal solution. A greedy approach will give you a completely different solution in many cases. You can try out with different strings, right? Not just letters, ATGC, etc., but you can try out with different strings, right? Because these can apply to strings, right? So, this is the shortest common superspring approach. So, you can try with different strings and take this greedy approach and see if you get the right superspring or not the superspring that you started with, ok? So, again, one of the things you probably understand is that the SCS may not be the reference sequence at all, right?

When you have these repeat sequences, you might not, and the SCS, the shortest common superspring, might not be the correct reference sequence because then anything that kind of omits the reference sequence omits the repeat sequences, right, and instead of three repeat sequences, let us say one or two repeat sequences. Those will be the shortest ones, right? So, they will be shorter than the actual reference sequence, and SCS will say, Ok, this is the shortest superspring that I could find and they contain all the read data as substrings. So, again, when you think about this SCS approach, this is something you also need to think about, right, whether SCS is the right idea. So, whether finding superspring is the best idea, especially when you have the repeat sequences in the genomes, ok.

And on top of that, what we have seen with the approach may not give an optimal solution, especially when you have these repeat sequences present in the genome. And one of the points that we have not discussed at all is the impact of sequencing errors. In this part, you probably realize, right, that when you are working with billions of reads, we will have sequencing errors, ok? This is something we touched upon when talking about RNA sequencing as well. So, when you have these reads that are not perfect, right, so in the graph you can probably now imagine, right, you will have certain base changes to something because of a sequencing error that is not matching with the reference sequence, and then the overlap will change also, right.

So, the overlap value will change, which means the structure of the graph is going to change, ok? So, there are two; there could be two impacts of sequencing error, right? So, you might not find the right overlap. So, you might miss these right overlaps, or you might get spurious overlaps, right? So, if you have read data that contains sequencing errors, you might say that you might miss these right overlaps, so correct ones that were actual overlaps in the genome, but because of the sequencing error, you are not seeing this overlap, right?

So, there is no match between the prefix and the suffix, and one mismatch is there. So, it will not be included in the overlap graph, or in addition, you can generate these spurious overlaps, which means you are making the whole structure even more complex, right? So, you are adding more nodes with reads that are actually not found; these are not actual sequences from the genome, but they are generated because of the variation, right, because of the sequencing error that is present, ok? So, we have not talked about the sequencing error, right? So, this is something we again need to address, which we will do when you talk about other metrics. So, one of the things we probably realize after all this discussion is that the SCS approach can get us close, right?

This is one of the earliest approaches, because it has been applied in computer science. So, we could take this idea and apply it to reads and generate a sequence that is close to the reference sequence. It may not be exactly the same sequence, but it is close to the reference sequence. But with this large number of repeat sequences, we are not going to get the actual sequence, and we might end up with something that looks very different from the actual reference sequence. So, the question is, how can we address the issue of repeat sequences? So, this is a big problem we have seen, right?

If you have repeat sequences, this is something that is going to hinder assembly. And this is something we will discuss when we discuss the other methods, and perhaps there might be some ways to address this. One of the points we will discuss and realize is that this is where the longer reads come in. So, that is where the long read sequencing is really valuable, ok? When you are assembling genomes with a large number of repeat sequences, the long reads actually help you assemble them correctly.

We will discuss that in the next class with some examples, and you will understand those issues better. So, here are the references for this class. To summarize, we have discussed the greedy-SCS approach, and we have talked about or seen some examples of how we go about this greedy approach using the overlap graph, or maybe just as strings, and this gives us a way to determine the superstring. So, when you determine the superstring it will contain all the reads that are given, but it might not be the shortest one or it may not be the most optimal or the correct one. We have seen in some examples that these are different from the actual reference sequence. We have also seen that it repeats in non-optimal assemblies, and we need to think about some ways to address this issue. Thank you very much.