Computational Geometry

Prof. Sandeep Sen
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Module No. # 07
Planer Point Locations
Lecture No. # 01

Point Location and Triangulation

So, we start with a new topic for today's lecture and that is point location and triangulation. In case, if you have any serious doubt about lower bounds that we did last time, you can ask me.

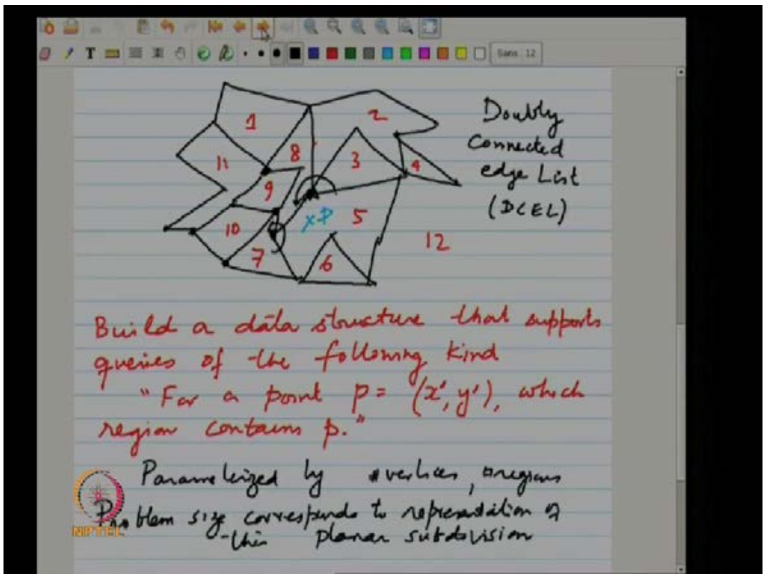
So, I have added couple of problems to the current problem sheet, not to say that so these are also due, but I would not say that they have to be done by whatever. So, I said before minor one. So, the last two problems are out of the discussion from the last couple of lectures. One of them is about lower bound for the two dimensional maximal problems. So, what we did in the last class was, n log h lower bound for the convex hull 2d convex hull.

So, I want you to come up with analogous construction for the 2d maximal problem and show that, it also has a lower bound of n log h. So, there is no direct reduction by the wave from convex hulls to maximal, sorry the convex hull problem to maximal problem. You cannot take that lower bound and claim that, this is the lower bound for maximal problems. So, these two problems, maximal and convex hull, sometimes people tend to think that they are closely related. Well, there is some relation, but you know overall in terms of the construction. Now, let us say the lower bound arguments. There is very little similarity, but if you were to define in a convex hull using what is called l 1 metrics.

So, convex maximal layer is something maximal. The 2d maximal is something like convex hull of a set of points in the l 1 metrics. So, l 1 metrics unlike what we have been doing is of l 2 metrics. That is the normal Euclidian norm. That is the distance between two points. You know is a difference in the coordinates squared and take this square root of that. That is why l 2 metrics. The l 1 metrics is simply the difference in the x and the y coordinates, and sum it up, so that is what is l 1 metrics. In many cases, l 1 metrics is much simpler to handle than l 2 metrics because you know there are so many possible ways to traverse the shortest distance between two points in the l 1 metrics because it is a miracle distance whether you go like this and this or you do zigzag. We have to parallel to the coordinate axis. That is why l 1 metrics is often easier to handle than l 2 metrics. So, even the construction of 2d maxima.

You know if you think about it conceptually, it is simple than the convex hull problem because you do not even need actually anything beyond comparison to do the maximal problems. Whereas, the convex hull, you do have to reserve to some linear inequalities ok. So, that is basically I have added that problems, so that you can make a little bit more about the construction that we did in the last couple of lectures. So, now we proceed to another couple of very fundamental problems and to mention, I have written here point location and triangulation. These two problems as such are not the same problem, but you know one helps the other kind of. So, essentially when we do point location and I just define it more formally. It is easier to do point location when you know how to triangulate it. So, let me first just define the problem of point location. So, point location in a very general. So, let us say, again confirm our discussion to two dimensions.

(Refer Slide Time: 04:09)



Build a data structure that supports queries of the following kind
"For a point p = (x', y'), which region contains p."

Parameterized by #vertices, #regions

Problem size corresponds to representation of the planar subdivision

So, you are given a planer map. Let us say planer sub-division. More specifically, let us say a straight line planer sub division you are not still prepared to handle curves. So, you can think about a map, essentially in a planer map where I can only drop piece wise linear, I guess.

Let take this, looks hopefully messy enough. So, this is the kind of map that you have given. So, the planer map, these are vertices and the edges joining vertices and then you can label them using regions. So, let me call these region 1, region 2, region 3, region 4 and of course, there is an outer region which is let us call it… So, I have labeled each of the regions. Closed region here and the problem is given such a map we want to build a data structure, so that supports queries of the following kind this for a point p equal to x-prime y-prime which region contains p.

Again, for simplicity we may assume that p does not lie on a line segment itself. So, that we just do not consider that possibility. So, you know p. So, given this coordinate pair x-prime and y-prime, I want to return the label 5 because 5 is the region where p lies in. This is an extremely natural problem. If you have any kind of data representation of a map or some kind of geographical regions, this is some internal representation and I am saying that you are allowed to build data structure on that. Then, you know suppose a collecting some statistics, some demographic on something you know rainfall crop production whatever. Then, you want to add, you want to say that here in this region indicated by the point, this represents a population of one million or whatever. Here is this point. You tell me this, whatever this point which state does this point lie on, this city lies in. So, these are all extremely natural problems settings.

So, we need to somehow answer this query and we want to answer this query as quickly as possible. That is why I am saying that we can do some pre processing where you build a data structure there. Given the point, I should be able to quickly tell you where quickly in most cases would imply something logarithmic time; otherwise you know brute force you can if you have a list of all these regions you can try to individually brute force you know exhaustively. Look at you now does one contain does p line 1. Does p line 2, does p line 3 and so on and so forth that would be you know very expensive. You do not want to do that because these kinds of queries will come in very often. So, this is essentially the problem of point location in planer sub divisions, so desirable.

So, it will be parameterized by many things. You know it could be number of vertices number of regions etcetera because this is a planer maps because all these are linearly related. So, a problem size would be actually strictly speaking the problem size should be or they should be a representation of this map. So, problem size corresponds to representation of this planer sub division. Well, if you say that about n vertices, you know that they are free about n regions. They are free about n edges, but what we are really looking for in strictly speaking the problem size is the representation of this planer sub division. So, I am not sure if you have ever really given too much thought to how to represent a planer map.

So, each region is let us say, the way have drawn it essentially the simple polygon not necessarily a convex polygon each region is a closed area and this is a simple polygon. Now, if you just represent each region separately, what is the information that you lose out. It will be sufficient to answer planer point location, but you are going to lose out some information. What is that? The adjacent information is completely lost. You only know or let us say to find the adjacency regions. You have to really struggle hard right. So, if I want to find out what are the edges and regions, then may be again have to be exhaustibly go through this entire data set to find out which regions share edges with the region 1.

So, just the representation of the planer sub division itself requires failure sophisticated. Well, sophisticated sense that you know should be a data structure where retrieval should be easy. That is you know if I want what are the adjacent regions of one, I should be able to do sort of list them out. If there are ten adjacent regions, let us say I want to do about an order 10 of (( )) in a linear amount of time I should be able to pick them out. This could be an independent problem, in the sense that what is good representation of planer sub-division? Where these operations are supported because of that? So, today I am not going to talk too much about this how to represent maybe I should let you think about it, but the data structure that is most commonly used in literature is called a doubly connected edges or DCEL for short.

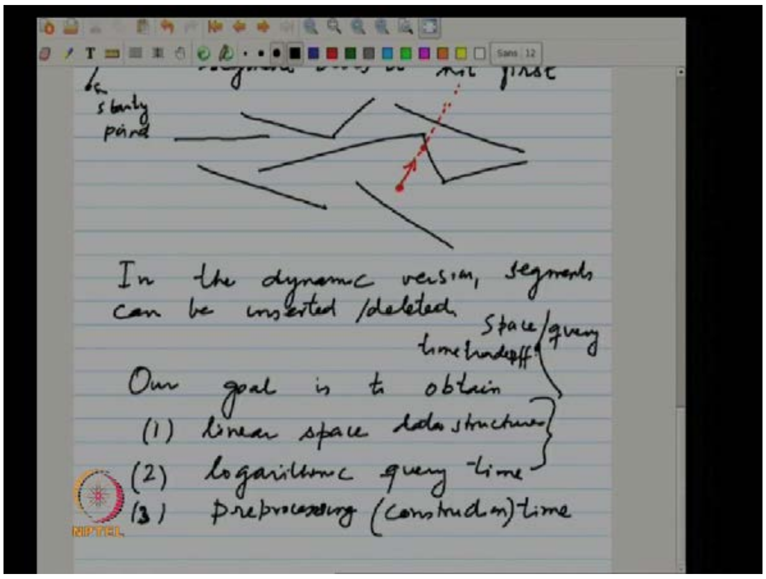
Now, if you have written any programs related to penalty testing, then you will be familiar with this. So, I am not sure, how much of planer graphs or penalty testing you have. We have seen before, if you have that, essentially it is a planer graph represent. How do you represent a planer graph? How do you solve problems with penalty testing? So, you would like to do the testing. So, one idea that is often used is that you actually store the oriented edges around a given vertex. So, let us say we are talking about this vertex. So, I have, I should, I can do this following. I can at least store these edges in some anti-clock wise order around this vertex and then, from this vertex to the neighboring vertex, I should be able to go by storing information with this vertex. There also, I have the same thing. I have something stored in counter clock wise fashion. So, that is why it is called doubly connected edges list ok.

From the doubly connected edges list, you can recover this adjacent information very quickly, basically linear time, linear time proportional to the number of edges, adjacent regions and so on so forth. So, this representation is quite common and useful in many of these applications. So, fine. So, we have some way of storing these doubly connected edge list and if you do this anyway, this oriented storage, it will still be linear in the number of regions and vertices and edges. So, we were not blowing off the space too much. So, that is the basic representation of the planer graph or a planer sub division. Now, what we want? Yes.

No, so that again. So, if I want the adjacent regions of a given region, how do you find that how quickly can you find that? So, what you are suggesting is that you store the dual structure of this right. Yeah that could be, that would be another. So, what you are suggesting is essentially the same as storing the dual structure of this. So, dual of the planer graph will give you the adjacent information true. So, what this drably connected edge list is doing is, it actually compressing all that information in one graph. The primal and the dual information is together.

So, drably connected edge list is actually the dual and the graph both sort of combined together. So, I am not saying it is a very difficult problem, but you know it requires some thought. There is some natural or some useful data structures for that. Now, what we want for this problem is a more specialized data structure that will support the planer point location, not just the adjacent information, but planer point location. Now, there is another equivalent problem which is very closely related and that is the problem of ray shooting.

(Refer Slide Time: 16:01)



In the dynamic version, segments can be inserted/deleted.

Space/query time tradeoff

Our goal is to obtain
(1) linear space data structure
(2) logarithmic query time
(3) preprocessing (construction) time

So, what do you mean by ray shooting? So, in ray shooting, again you are given a set of edges. Well, for simplicity let us say non-intercepting. Well, in the case of planer graph, the set of edges are non-intercepting except at the end points right. So, ray shooting did not necessarily deal with non-intercepting edges. It may also involve in intercepting edges for just that is a simpler version. So, I am only dealing, I am only defining that. So, given a set of edges, you can do the following. Build a data structure to support queries of the following type. So, what is a ray? A ray is basically, you know something like this. This is some kind of a starting point and this is some vector, some direction. So, that is your ray. So, if you shoot a ray, which segment does it hit fast?

So, suppose we have these kinds of line segments. They said they are non-intercepting something that of a here is the ray I am shooting. So, of course, you know the ray is an infinite ray if I extended it right. Now, I am interested in which is the segment that intercept it first which happens to this one. So, it can be thought about like a visibility problem. So, if it is a light ray and these segments are let us say opaque that it does not let light to pass through which is the visible. I mean, so which edge the light rays strikes first. That is all.

So, if you are dealing with the problems like in graphics where you are solving some kind of, we have some kind of illumination model. You are solving this ray shooting problem millions of times actually because you are shooting some light ray. You are essentially simulating a lighting model right. You shooting a ray and there are some fast hardware which does this ray shooting for you, but actually you know if you go into the details of what that hardware does, it does a very brute force computations. So, we will look at much more sophisticated all these things, but anyway I am just giving motivation for this ray shooting.

So, you shoot a ray and then, you strike something. If you have a surface model whether how it reflects, whether it defuses reflection or it is some other kind reflections. Then, keep following that light ray and there essentially you are looking at if the surface is opaque. It is going to be reflected back if it is not opaque, it may have some refractions. So, it all depends on the model you have and it is very complicated and involves huge number of computation, but each primitive is a ray shooting primitive that is one immediate application of that. So, why have I defined the ray shooting problem here if you go back to the point location problem? So, the point location problem and the ray shooting problem are closely related in the following way. So, what I can do is, I can for the point location, I may want the answer in the following way. So, from point p, I can shoot a vertical way. So, all my points are vertical now, but the starting point of the tray will be the coordinate x y. No, it is a vector and a starting point.

Yeah. It is just a vector or a direction of the starting point right. So, from this given point p, we shoot a vertical ray and it is going to hit some segment right. Now, of course, you may object saying that what if the point was here. Then, it sort of does not hit anything and goes to infinity. Now, again the simple solution of that is enclosed everything within a box, so that nothing escapes. So, now depending on where it hits, which segments it hits, that again you can with that you can neatly identify the region where the point lies in right because of point is going to hit it from below. So, there will be unique. So, whatever segment lies in each segment, I am going to store the information which is the region above and which is the region below, so I know exactly. Then, that will also help me answer the same query.

So, I can solve the planer point location also using ray shooting queries. This problem also has very interesting extensions to high dimensions ray shooting high dimensions. Another interesting extension of ray shooting is that when the environment is not static, but you know the segments can be (( )). So, will not again talk about these problem, but you know just so that you aware of these versions. So, in the dynamic version segments can be inserted or deleted, so my data structure should be able to support insertions and deletions. Let us come back to the original problem. How do we do point location in planer sub divisions? You know we are very ambitious, you know I said that.

So, our goal is to obtain, let us say one linear space data structures and say logarithmic query time. There is another parameter that is involved and that is another parameter of important so pre-processing time, the construction time. This is for the static thing, for dynamic thing. Of course, you know there is no pre-processing as such its update time, but anyway we will only talk about the static problem. So, of these three more or less, actually I have written it in the order of importance.

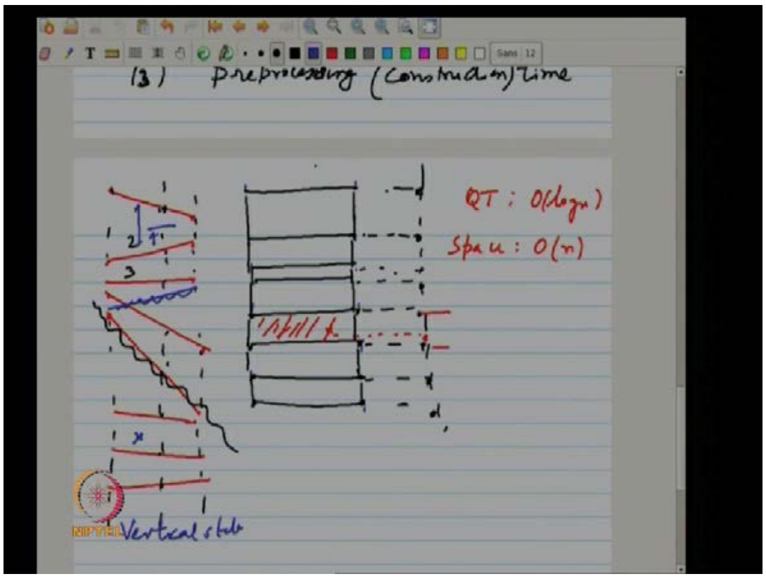
Why am I saying this space is more important than query time? Well, some people you know may have some issues with that, but let me say in most cases people do consider space to be more of a bottle neck because it is a data structure that you have to store. It occupies some space where ever and we are dealing with large very large input size or very large scenes. So, even if it is like say twenty times n or something where n is the problem size, it is twenty times n means if I can save it down to five times, then that is a huge improvement because I am saving so much of space. After all space is something that we have to leave with. Whereas query time is also very important if I want fast answers, but it is not some times as crucial as space because if we run out of space, you do not have space. There is nothing that you can do acquired time instead of log n time. Maybe, it is log square n times. You are still willing to live with it. If it is often, it is noticed that there is a space query time period of that is a very common trade.

A space query time trade off is very common for these kind of query problems. If you allow more space, you know it appears that you can get faster query times. If you allow less space than query is more. So, one end of the spectrum is that you do not build a data structure at all. This I say you know given this planer sub division and do not do anything and brute force I find out and that will be in a linear time because I have to look in to every simple polygon to see the point is contained within that. So, that is one end of the spectrum, but I am more linear ambitious. We want to keep linear space at least asymptotically linear and get a fast query time. Pre-processing time is the least important because you incur it only once. So, even if it is, let us say n square times you incur it only once and forget about it, but yes if you can get something that is fast, we should prefer that ok.

Sir, in dynamic setting, we try to ensure that say it something that inserted there our pre-processing time is only constant. Let me repeat the question. You are saying that in the dynamic settings, no. So, if you have ever looked at any dynamic problem including a simplest dynamic problem that we look at the dynamic decision, so everyone was done a course in algorithms. You will know the dynamic decision which is basically either your avail trees or red black trees. Whatever these are dynamic decisions. So, when you talk about the dynamic version of the problem, you do not want to reconstruct everything from scratch because that does not make any sense. So, whenever you are talking about dynamic versions, you are talking about only update times. What is the time for insertion? What is the time for deletion? Once we have that, you do not need any. You do not need to even talk about pre-processing time because you can build everything incrementally. So, it is an update time that is the crucial parameter.

So, what would be our first sort of attempt for the planer sub division point location problem? It is a two dimensional problem. If it were a one dimensional problem, I will give an example. It is a very special case. Suppose my planer sub division was like this, well sorry. So, this is a very special case where all the line segments are parallel to each other. Also, this plan has exactly the same interval or let me say if you are uncomfortable with that, I can actually say these are all rectangles. Suppose, my planer sub-division was like this, then how would you attack the point location problem?

(Refer Slide Time: 28:19)



Which line do you want to project on? On the vertical line I see. So, all what you are saying essentially is that actually right. So, once project is on this vertical line and whatever point I am given, suppose it is some point here, that point is also projected here. So, it now comes down to a one dimensional problem on the line. So, if I know that this red point lies between this and this, it must basically be in this rectangle. So, although this problem is drawn as rectangles, it turns out actually like it turns out to be a one dimensional problem, you know exaggerated. I mean exaggerated one dimensional problem. Actually, the one dimensional problem I just made it triangles to make it look like two dimensional problem. So, fine. So, that is the time query.

What is the space etc for this log n? So, query time is order log_n, essentially binary search and space is order n linear. Pre-processing time is what if a given these rectangles in a sorted manner order n. If not, you sort it order n_log_n, either n or n_log_n depending on whether it is given in sorted order or not right. So, we have achieved our goal for the simple case. So, now we just make it slightly more complicated. So, now I have again, I have these line segments spanning the same x interval, but these are not necessarily parallel to each other, known are the parallel to the x axis. Now, they could be tilted. So, again we can project it and solve it the way that you have done before. Both sides.

Can you give me a conceptually simpler approach? I mean idea is there, but let me warn you about one thing. Whenever you are handling two dimensional problems, you cannot treat the two dimensional, two dimensions completely independently. There is a very strong correlation between the x and y. So, I solve it for x and solve it for y and somehow, I combine it. This is a common fallacy. It almost never works. So, can you give me something, some conceptually simpler thing that is related to the previous solution because if I can solve this, I can also solve the previous problem? So, what do you do with the previous problem? Essentially, what do you do? What is the technique? Technique is that we use binary search right. So, can you do binary search literally on this? Yes or no. How?

Oh god! You know do not talk what projecting and point, you will be in trouble. You simply see all that you need. See these red segments. Although, they are tilted, but you can see they are ordered in some, they are actually ordered right. This one, if this is the top segment, the second from the top, the third one from the top. How do you? How can you order this? No. You take any vertical line, look at the intersections of these and they are completely ordered and that also the ordering of the line segment.

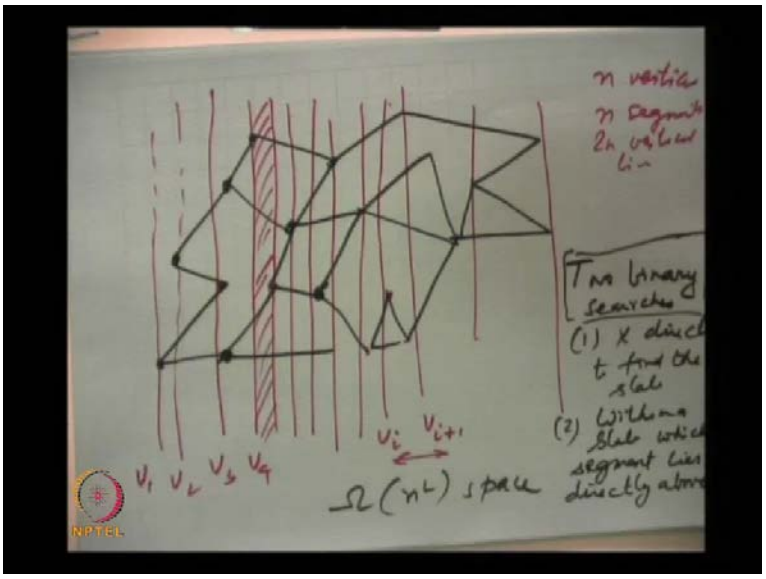
Yes. So, once you have ordered them, now you directly store the segments in kind of an array or searched you are whatever. Suppose this is the middle segment. I want to do point location for something like this. What do I do? The first thing I do a compare with the middle segment. Is it above or below that is all. So, I can do binary search, conceptually binary search directly on this. Therefore, I could also do it on the previous problem. I do not need to even think about projections and things like that. That is why I move from here to here because I realize that just straight forward projection is not going to work conceptually. You cannot consider the x and the y coordinates separately fine. So, this is slightly more non-trivial from the previous one, but it is still relatively simple, but it is not a one dimensional problem. It is somewhere between one and two problem right. I am not talking about factors and something like that. I mean do not confuse with that, but conceptually this is not quite a one dimensional problem by projecting on to one axis.

I would not be able to solve this problem. So, I had to resort to a binary search when I am actually using the information about whatever the slopes or the actual segments, full segments whether it is above or below. So, the above below relation is not as simple as the above below relation in the previous problem. So, we are getting somewhere, now I claim that this thing that I have done where I have taken a vertical slab. So, what is this? Basically, it is a vertical slab and these segments span the vertical slab. There are no intersections in between and that is the reason why you could order them. If we had a situation where you know there were some segments that were going like this, this solution is not going to work. So, that is why I said we are not even dealing with intersections, but there could be situations where you should deal with intersections. Otherwise, we are not dealing with intersections. So, this is not even allowed. This is not even allowed.

So, in any vertical slab, we have where every segment spans the entire interval. If something stop short like this one, then also you would have a problem because the point if it is here, then this is whatever you know the vertical visibility thing. If the point is here, then it hits this one. So, we do not have situation like this. We have only like this vertical slab and everything going cross. That is the situation that we can handle. Now, I claim that if you can handle this, we have some handle on the original problem. What is the original problem? You know I drew this figure, so can you relate it?

(( )) rectangular slab (( )) within that there is no other vertices are. So, let me just blow up this. I mean that is the idea and this also very simple to achieve.

(Refer Slide Time: 37:16)



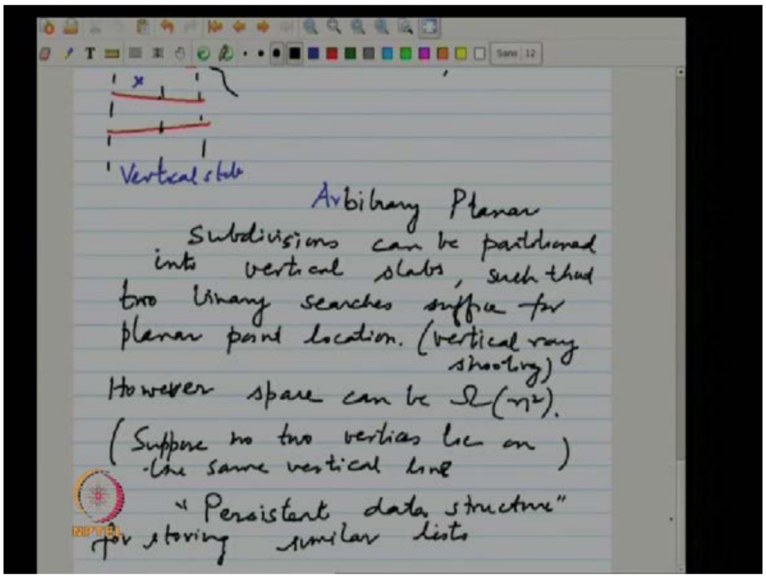
So, if I copy this figure, may be this is not exact, but see it comes like this. So, let me kind of reface what was being suggested. If I draw vertical lines through each of these end points, so let me not draw through everything because that will be very complicated. See every point I am drawing a vertical line, I am not drawing all of these. You know I have left out some of them etcetera now and you look at two consecutive vertical lines. Let us say, I look at this region. So, by definition there cannot be any vertex in this region because I have drawn a vertical line through every vertex.

So, there cannot be any new segment beginning or ending within. So, I am calling these as vertical slabs. So, these are sorted. So, say let us call it as V1 V2 V3 V4. So, essentially Vi and Vi plus 1 that is the vertical slab I am referring to. So, any vertical slab will be free of any. There would not be any vertex, basically in this region by definition, by construction. Therefore, you just focus on any vertical slab that is precisely the problem that we have just solved alright. So, how many vertical lines we have? Let us say we have n vertices. Well, if we have n vertices, we have n lines right. So, if we have n segments, then you have 2n vertices and 2n vertical lines right. So, anyway it is some linear number of vertical slabs right. Now, how would you now solve the planer point location problem?

So, two binary searches suffice right two binary searches. One x directs to find the slab and number two, within a slab which segment lies directly above right. So, this is important two binary searches suffice great. So, we have achieved to our one goal of order log_n query time. How about space? Space could be a problem and already someone is sounding ominous. This is n square right. Why it is ‘n’ Square? You have n slabs and you can certainly have a construction or a planer map where you will have, let us say about n over 2 slabs contain about n over 2 segments each. If it is not obvious to you know you should go back and do it.

So, I am claiming that there can be cases where n over 2 slabs. Can each slab can contain a n over 2 segments? So, therefore, you know if you are storing these as separate data structure for binary search, you are stuck with omega n square space. So, let me just record what we just observed. So, arbitrary planer sub-divisions can be partitioned into vertical slabs such that two binary searches suffice for planer point location. In fact, we are solving the ray shooting problems, the vertical ray shooting problem. However, space can be omega n square and certainly, would not be more than order n square because there n slabs and each slabs can at most n segments. So, this is the bad news right.

(Refer Slide Time: 42:32)



Arbitrary Planar Subdivisions can be partitioned into vertical slabs, such that two binary searches suffice for planar point location. (vertical ray shooting)

However space can be $\Omega(n^2)$.

(Suppose no two vertices lie on the same vertical line)

"Persistent data structure" for storing similar lists

Now, let me just mention something in the passing. It is an idea that I would not perceive right now, but it is an idea that is so natural that you should think little bit about it. You feel that you will be tempted to figure out what really happens to (( )). So, when you look at these two vertical slabs that is we are talking about, these n square bottle neck right. Now, these two vertical slabs can have many segments, but on the other hand, if you look at it closely, you know this figure. You make one more assumption which I never stated, but again it is implicit in many things that I have been doing. Suppose no 2. It is a non-singularity condition no 2 vertices lie on the same vertical line. It is in a assumption that in a past also, I have made which simplifies the description.

I am not saying it is absolutely necessary for the algorithm. It is that and also some, there will be some special cases in the algorithm where you have to struggle little bit more. If this condition is not satisfied and unlike, well I mean you can do the same thing right. You can do the same trick about rotation and make sure that after all, it is a planer sub division. It is not the vertical. It is not the vertical ray shooting that we are solving, actually solving the planer sub division point location problem. So, if you rotate the map, the point continues to be in the same region. Just the vertical direction changes, but that does not matter because we are actually interested in the region it lies in. So, that kind of rotation will fix this problem again. So, the reason again I am saying this is you know each, so that you know if you do not assume make this assumption. Then, what happens between two consecutive vertical slabs? You can see something, some segment is ending and some segments are beginning.

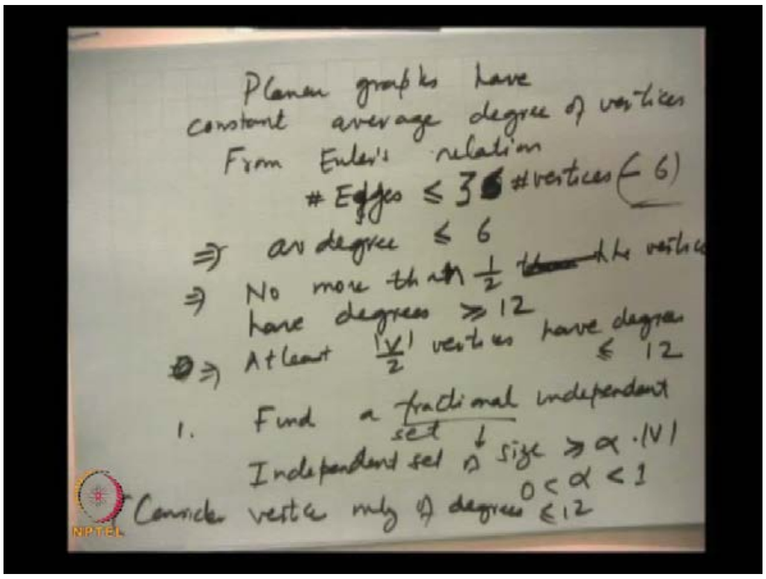
So, there are not too many segments that are beginning. They can like two segments are beginning, but you know one segment is ending. So, not too much is happening when you go from one vertical slab to the next vertical slab. So, this relation that when you go from one vertical slab to other vertical slab, the amount of change that actually happen. See, if you see when we say two binary searches, this I stored these vertical lines in some data structure, so that I can find the slab that is only order n, my space is only order n. When I do the x axis binary search, the problem is y axis, y direction right. If I store these data structures separately, I pay the prize for the omega n square because each of the data structure can take linear amount of time because they have about order n line segments.

However, what we have just noticed is that between two consecutive vertical slabs, they may share a lot of line segments, but the new changes are very small. The numbers of line segments that appear here and do not appear here etcetera, it only differs by some constant amount, you know two or three segments. So, this similarity of two lists or let us say two data structures that can be exploit it in more complicated high level data structure where too less, too consecutive less have lots of similarities. I do not know if you heard this word. There is something is called persistence data structures. Persistent for storing what are called similar lists.

So, these lists are similar in the sense, the two consecutive lists have most things in common. Only a few things change. Of course, between the left most and the right most slab, there is a lot of changes, but then this can be stored in a way, in a fashion where you can exploit these common information. The whole thing can be combined is n list, can be combined into one single data structure where the overall space is still going to be linear. It will still support your binary searches in order log_n time. So, it is a very interesting development in any of the data structures that happened fairly about 25 to 30 years back. This is first done by, I think in the paper by I think by (( )). I will give you some, but we may actually even in this course we may discuss it later on. So, this is such.

It is not just the contest of planer point location, but any data structures where you are trying to solve, store similar items, similar lists. This kind, these common structures can be, you know you can avoid storing. The common repeated instances of the common structure. So, this is the much bigger field of whatever research or whatever, but realistically these data structures basically are so complicated. You know you think 10 times before you want to implement it right. So, I will not pursue this line of thought, but I just mention it because you may have occurred to you independently or you may come across the situation. What I am saying is, actually there is a lot of work in the area of (( )) structures. So, we will not do this, but we will try to see some you know follow some completely different line of attack. So, I will only give you some idea today and then, we will take it in the next class. So, the idea is the following. So, let just use this paper and it is a very clever idea.

(Refer Slide Time: 50: 12)



Planar graphs have constant average degree of vertices

From Euler's relation

$\# \text{Edges} \leq 3 \, \# \text{vertices} (- 6)$

$\Rightarrow$ av degree ≤ 6

$\Rightarrow$ No more than $\frac{1}{2}$ ~~the~~ the vertices have degrees ≥ 12

$\Rightarrow$ At least $\frac{|V|}{2}$ vertices have degree ≤ 12

1. Find a fractional independent set

Independent set of size $\geq \alpha \cdot |V|$ $0 < \alpha < 1$

Consider vertex only of degree ≤ 12

So, any planer graph. So, we will actually use some nice structural properties of the planer graphs. So, the planer's have division in the planer graphs. Now, planer graphs have constant average degree of vertices. Are you aware of this? Essentially, from the Euler's relation, it follows that number of edges, sorry edges is less than or equal to roughly about 3 times, no 6 times, no 3 times right. Yeah 6 times is a number of vertices. I mean gn minus t is a number of edges, but if you sum the. Yeah. So, that is what I was confused. So, 3 times, sorry 3 times the number of vertices is minus something. Minus some quantity.

So, certainly I can write it less than or equal to 3 times number of vertices right. So, minus whatever 6 or something that you know, let us forget about this. This thing which implies that average degree is about less than 6. So, which implies that no more than half the vertices, no more than half the vertices have degrees greater than equal to 12 right or in other words though right. So, at least over 2 vertices have degrees less than equal to 12 the same thing right. This is the property that we can exploit in the following way. What you do is, you find what is called a fractional independent set. What is the fractional independent set? It is a independent set that you know I want to identify.

So, fraction basically means let us say some independent size. Independent set of size greater than equal to alpha times number of vertices where alpha is some fraction between 0 and 1, say large fraction of the vertices. Can you argue that a planer graph has a large sized which is a constant fraction, the vertices independent set just from this observation. So, an independent set means that 2 vertices do not share an edge and fraction basically means it is some fraction of the total number of vertices. So, I am saying alpha is some fraction between 0 and 1. It can be very small. Let us say, point 1 or whatever.

So, I can argue that a planer graph has a large sized independent set which is what I am calling the fraction between independent sets just from the observation you made before. So, suppose I do a greedy direction of independent set. I take 1 vertex. How many vertex I choose? All the vertices. Let us say of degree less than 12. So, consider vertices only of degrees less than equal to 12. Rest other vertices do not even consider. Now, I will confine my independent set to only this subset which has degrees less than equal to 12. Now, let us run a greedy algorithm on that. I pick a word is the independent set. How many of such vertices do I rule out? Yeah right. So, essentially then you can argue that you have a large I mean constant fractions of these vertices will be in the independent sets.

So, therefore, this set itself is a constant fraction of the original graph right. That is why if you have a large size, independent set; it is a constant fraction of number of vertices. Agree with me? So, I heard a noise. So, I am only building independent set from vertices of the degrees less than equal to 12 and I run a greedy algorithm on those vertices. So, I choose a vertex. Then, I cannot choose any of the neighbors. So, if I choose 1 vertex or degree less than or equal to 12, how many vertices are less than or equal to 12? Do I have to leave out another 12? Right. That is it.

So, I am roughly choosing about 1 out of 12 vertices. So, which is a constant fraction of the number of vertices or degree less than or equal to 12 and the number of vertices less than or equal to degree 12 is half the total number of vertices. So, what I have eventually in the independent set of size alpha n alpha times alpha times number of vertices where alpha is some constant. May be a very small constant something like that, pretty good calculation. So, depending how you do the counting you can get a better and better number, but right now I am not going to, I am not looking at only asymptotic. I am only using the fraction is a constant fraction. So, I will just do a little bit hand waving and then we will do it later on.

So, you take these vertices and you remove those vertices. When you remove those vertices before even that, so what we do is, we actually triangulate this planers are division. Now, when I take out this, see this thing about when you get this Euler equation for this. This bound is actually is for a triangulated graph. This is for a triangulate because that is the maximum number of edges that even add without crossing. Now, when I remove a vertex of degree 12, I can remove a constant fraction of these vertices. So, I have actually taken out some vertices from the original sub-division and I am going to re-triangulate it. I am going to over lay this division on the original sub-division.

So, if I do my planer point location on the sparser graph, I can refine it. Then, complete may search this is the kind of recursive structure that we will build and because I am removing constant fraction on the vertices, my level of recursion will be about log n because picking up this constant degree vertices, the refinement itself we will take only constant times. This will give us an overall order log n kind of search time and every time we are removing a constant fraction of vertices, the space is a geometric sum. So, the space is also linear. So, this is a very you know one minute description of the thing and we will complete it next time.