The following content is provided under a creative commons license your support will help mit opencourseware continue to offer high quality educational resources for free to make a donation or view additional materials from hundreds of mit courses visit mit opencourseware at ocw mit edu.
Start a brand new exciting topics dynamic programming.
So exciting actually i'm really excited because then i forgot mine is my favorite thing in the world in algorithms and it's going to be the next four lectures.
It's so exciting have lots of different facets.
It's a very general powerful design technique a little while to settle in we we like to inject it into now and double o six so in general our motivation is designing new algorithms.
Cinemax programming also called pp is a great way.
A great general powerful way to do this it's especially good and intended for optimization problems things like shortest path.
You want to find the best way to do something.
So i just pass you want to find the shortest path the minimum length path.
He want to minimize maximize something that's an optimization problem and typically go down to solve them involve dynamic programming of a broad statement think of dynamic programming as a kind of exhaustive search which is usually a bad thing to do because it leads to exponential time.
But if you do it in a clever way by then you typically get polymer polynomial time.
So one perspective is that dynamic programming is approximately careful brute force have an oxymoron.
Are we take the idea of brute force which is try all possibilities and you do it carefully and you'll get it polynomial time there a lot of problems were essentially the only known polynomial time algorithm is by dynamic programming doesn't always work.
There's some problems where we don't think there are.
When is possible dps a nicer general approach to it?
And we're going to be talking a lot about that and i program a has a lot of different.
There's a lot of different ways to think about it.
We'll look at a few today.
We're going to warm up today with some fairly easy problems that we already know how to solve namely computing fibonacci numbers pretty easy and computing shortest pads.
And then in the next three lectures were going to get some more interesting examples where that's pretty surprising that you can even solve the problem in polynomial time.
Probably the first burning question on your mind though is why is it called dynamic programming?
What does that even mean?
And i used to have this feel about well you know programming refers to the i think it's a british notion of the word where it's about optimization optimization in american english is something like programming in british english i know you want to set up the program the schedule for your trains or something i think is where programming comes from originally but i looked at the actual history.
Why is it called dynamic programming dynamic programming is invented by a guy named richard bellman you may have heard of bellman be in the bellman ford algorithm.
So this is actually the precursor to bellman ford and we're going to seem bellman ford come up naturally in the setting.

So here's here's what i caught about him says belmont explain that he invented the name dynamic programming to hide the fact that he was doing mathematical research.
He was working at this place called ranz and under a secretary of defense who had a pathological fear and hatred for the term research.
So he said it would be difficult to give a pejorative meaning to it and because it was something not even a congressman could have ject to basically it sounded cool.
So that's that's the origin of the name diner so why is it called that who knows?
I mean now you know but it's not it's a weird term just take it for what it is.
It may make sense some kind of sense.
So we are going to start.
With this example of how to compute fibonacci numbers and maybe before we actually start i'm going to give you a sneak peek what you can think of dynamic programming is and this this equation.
So to speak is going to change today's lecture and then we'll settle on a certain more accurate perspective.
The basic idea of dynamic programming is to take a problem split into some problems solve the some problems and reuse the solutions to your problems.
It's this like a lesson and recycling.
So we'll see that in fibonacci numbers.
So you remember fibonacci numbers?
Number of rabbits you have on day end if they reproduce.
We've mentioned before we're talking about avl trees.
I think so.
This is the usual i can think of it as a recursive definition or recurrence on fibonacci numbers.
The definition of what the ends fibonacci number is.
So let's suppose our goal and i'll grab my problem is compute the fibonacci number.
And i'm going to assume hear that that fits in a word and so basic arithmetic edition.
Whatever is constant.
I'm proper asian so how do we do it?
You'll know how to do it anyways but i'm going to give you the dynamic programming perspective on things.
So this will seem kind of obvious but it is a we're going to apply exactly the same principles of will apply over and over in dynamic programming but here it's in a very familiar.
Setting so we're going to start with the naive recursive algorithm.
And that is if you want to come get the food and fibonacci number you check whether you're in the base case.
I'm going to write it this way.
So f is just our return value you see why it's right at this way in a moment.
Then you return a half face kisses one.
Otherwise you recursively call fibonacci have in mind to add them together return that this is a correct algorithm.
Did a good algorithm know very bad exponential time?

How do we know it's exponential time other than from experience what we can write the running time as a recurrence?
The event represents the time to commute the inside of an option number.
How can i write the recurrence?
It's throwback to the early lectures divide and conquer.
Harrisburg yeah 7 1 + 7 2 constant i don't know how many you have by now and then we take constant.
I'm otherwise we do council member of additions comparisons and return all these operations take on some time.
So that's a recurrence.
How do we solve this recurrence?
Well one way is to see this is fibonacci recurrence the same thing.
This is plus whatever this is at least the number and if you know fibonacci stuff that's about the golden ratio to the nth power.
Too bad.
We had a similar recurrence in avl trees.
And so another way to solve it say oh well that's at least two times given 2 as it's going to be monotone.
The bigger it is the more work you have to do because davidians thing you have to do the first thing so we can just reduce 7 137 2 that will give us a lower bound and now these two term now this is sort of an easy thing.
You see that you're multiplying by two each time.
You're subtracting two from anytime.
How many times can i subtract 2 from 10 and over 2 times before i get down to a constant.
And so this is equal to 2 to the and over to i mean time some constant which is what you get in the base case.
So i guess i should say.
This thing is stated that ss at least that big and the right constant is a fee.
The base the exponent okay so that's a bad.
I'll go then we all know it's bad for them.
But i'm going to give you a general approach for making bad over this like this good and that channel approach is called memorization.
Here and this is a technique of dynamic programming.
I call this memorized dynamic programming algorithm.
So i said i'm using memo in the nuts.
That is simple.
Whenever we compute a fibonacci number we put it in a dictionary if it's and then we need to compute the answer bonacci number we check is it already in the dictionary that we already solve this problem?
If so return that answer otherwise computer.
I'll see the transformation is very simple.
Okay these two lines are identical to these two lines.

So you can see how the transformation works in general you can do this with any recursive algorithm.
Memorization transformation on that algorithm.
Just we initially make an empty dictionary thug memo.
And before we actually do the computation we say well check weather this this version of the fibonacci problem competing at the van is already in our dictionary so that key is already in the dictionary.
We return the corresponding value in the dictionary.
So and then once we computed the answer but i don't remember if we bother to do this if this didn't apply then we store it in the memo table.
So we store we say well if you ever need to compete at the van again here it is and then we return that value.
I said this is a general procedure.
Complete any recursive algorithm?
With no side effects i guess technically and turns out this makes the algorithm efficient now there's a lot of ways to see why it's efficient.
In general maybe it's helpful to think about the recursion for you.
So if you want to compute a fan in the old al gore then we compute + 10 1 2 completely separately to compute f n 1.
We compute f n 2 + f n 3 to computer from my stupid computer is 3 and 10 4 and so on you can see why that's exponential and end cuz we're only decrementing and buy one or two each time.
But then you observe hey these fn 3 is are the same.
I should really only have to compute the once.
And that's what we're doing here.
The first time you call f n 3 you do work but once it's done and you could over to this other recursive call this will just get cut off.
There's no tree here here.
We might have some recursive calling here.
We won't because it's already in the memo table.
Pan factus rtm happens if + 2 this whole this whole tree disappears because if in 2 is already been done.
Concerts clear white improve things so you in fact you can you can argue that.
This call will be free because you already did the work in here.
But i want to give you a very particular way of thinking about why this is efficient.
Which is towing so you could write down a recurrence for the running time here but in some sense recurrences aren't quite the right way of thinking about this because recursion is kind of a rare thing if you're calling fibonacci some value k you're only going to make recursive called the first time you call fibonacci okay because henceforth it's been the you put in the memo table you will not reverse so you can think of there being two versions of calling fibonacci okay there's the first time which is the non memorize version that does recursion doesn't work.
And then every time henceforth your you're doing menopause cause if not you can those cost constant.

I'm southern memorized calls costco sometime so we can think of them as basically free.
That when you call fibonacci of n 2 because that's a memorize call.
It's you really doesn't pay anything for it.
And we already paying thompson time to do addition and whatever so you don't have to worry about the time is whenever corrosion here.
And then what we care about is that the number of non memorize calls.
Which is the first time you call fibonacci okay?
And then no they don't even necessary.
We are going to call fibonacci one some point.
We're going to call fibonacci have to at some point and the original call is fibonacci have in all of those things will be called at some point.
It's pretty easy to see but it's chiklis certainly at most of this whenever call fibonacci been plus one to computer.
So it most and calls it will be exactly on calls that are not memorize those ones we have to pay for.
How much do i have to pay?
Well if you don't count the recursion which is what this recurrence does persian then the total amount of work done here is constant.
Hey so i will say is the non recursive work.
Her call is constant.
And therefore i claim that the running time is constant linear.
Carson would be pretty amazing.
This is actually not the best algorithm to decide the best dog in the fog computing and fibonacci number uses long and operation so you can do better.
But if you want to see that you should text except for 6.
Okay we're just going to get to linear today which is a lot better than exponential.
So why linear because there's and non member wise calls and each of them cost constant?
So the product of those two numbers.
This is an important idea and so important i am going to write it down again.
And it's slightly more general framework in general.
And dynamic programming i didn't say why it's called memorization.
You have this memo pad where you write down all your scratch work.
That's this memo dictionary and to memorize is to write down on your memo pad.
I didn't make it up another crazy term.
It means remember.
And then you remembered all the solutions that you've done and then you reuse those solutions family solutions are not really a solution to the problem that i care about.
The problem i care about is computing the nth fibonacci number to get there.
I had to compute other fibonacci numbers cuz i had a recursive formulation.
This is not always the way to solve the problem.
But usually when you're solving something you can split it in two parts into some problems with.
They're not always of the same flavor is the original goal problem but there's some kind of related parts and this is the the big challenge in designing a dynamic program is to figure out what are the

some problems but they always the first thing i want to know about a dynamic program is what are the sub problems?
Somehow they are designed to help solve your actual problem.
And the idea of memorization is once you solve a sub problem write down the answer if you ever need to solve that same some problem again you reuse the answer.
That is the core idea and son distance dynamic programming is essentially recursion memorization.
And so in this case these are the same problems fibonacci one through not even the one we care about is fibonacci then but to get there we saw all these other stuff problems.
In all cases if this is the situation so for any dynamic program the running time is going to eat equal to the number of different some problems.
You might have to solve or that you do song.
Pi times the amount of time you spend per sub problem.
In this situation we had and some problems and for each of them we spent constant time and when i measure the time for some problem which in the fibonacci case i claim is constant.
I ignore recursive calls.
That's the key.
We don't have to solve recurrence without any programming.
Count the persians obviously don't count memorized recursions.
The reason is i only need to count them.
Once after the first time i do it it's free.
So i count how many different some problems do i need to do these are they going to be the expensive recursions?
Why do work i do some amount of work but i don't count the recursions cuz otherwise i'd be double counting only one account if some problem once and that it's all so simple idea in general dynamic programming a super simple idea.
It's nothing fancy.
There is one extra check for going to pull out but that's the idea.
Alright let me tell you.
Another perspective this is the one maybe most commonly taught.
Esther think of that.
I'm not a particular fan of it.
I really like them was a she and i think it's a simple idea.
And as long as you remember this formula here it's really easy to work with.
How about some people like to think of it this way?
And so we can pick whichever way you find most intuitive instead of thinking of recursive algorithm which in some sense starts at the top of the big have what you want to solve and works its way down.
You can do the reverse you can start at the bottom and work your way up and this is probably how you normally think about computing fibonacci numbers or how you learned it before.
I'm going to write it in a slightly funny way.
The point i want to make is that the transformation i'm doing from the naive recursive algorithm to

the memorized algorithm to the bottom is completely automated.
I'm not thinking i'm just doing is easy.
This card is exactly the same as this code.
And is that code except i replace the end by k just cuz i need a couple of different and valleys here or i want to enter eight over and values.
And then there's this stuff around that code which is just formulaic.
I thought goes into this for loop but that's it.
And the stars exactly the same thing as the memo is algorithm.
Maybe takes a little bit of thinking to realize if you unroll all the recursion that's happening here and just write it out.
Sequentially this is exactly what's happening.
Cuz this code does exactly the same editions exactly the same computations as this the only difference is how you get there here.
We using a loop here we using recursion but the same things happen in the same order.
Really no difference between the cut this cuz probably going to be more efficient and practice because you don't make function calls so much.
In fact i was not a function call to look up into a table using a hash table to be simple.
But of course you can use an array but they're both constant.
I'm alright so is it clear with this is doing i think so.
I think i made a little so we have to compute.
F12 f and which one is that?
And now we compute it exactly how we used to accept now instead of her cursing.
I know that when i'm computing the case of an option number because they're laughing number.
I know that i've already computed the previous two.
Why because i'm doing them in increasing order nothing fancy and then i can just do this and this solutions will just be waiting there if they weren't and get a key here.
So i know that there's a bug but in fact i won't get a key or i will have always computer these things already.
That i stored in my table then i iterate eventually i saw the all the some problems f13 fn and the one i cared about was the antoine queso straightforward i'm do this cuz i don't really want to have to go through this transformation for every single problem.
We do or doing any fibonacci cuz it's super easy to write the code out explicitly but you can do it for all of the dynamic programs that we cover in the next four lectures.
Okay.
I'm going to give you now the general case.
This was the special fibonacci version in general the bottom up does exactly the same competition as the memorize version.
And what we're doing is actually a topological sort.
Some problem.
Dependency dag.
So in this case the dependency tag is very simple in order to compute an fn 2i can do if i know those i can compute then there is fn 3 which is necessary to complete this one and see what this

bag looks like drana conveniently.
So all the edges go left to right.
So this is a topological order from left to right and so i just need to do f1 f2 fn in order.
Hey usually it's totally obvious what order to solve some problems in but in general what we what you should have in mind is that we are doing a topological sort here.
We just did it cuz it's so easy and usually it's so easy.
It's just a for loop nothing fancy.
All right.
Singing arrow let's do something a little more interesting shelly i one thing you can do from this bottom up perspective is you can save space.
Storage space in the oven i don't usually worry about space in his class but it matters in reality.
So here we're building a table size end.
But in fact we really only need to remember the last two values.
So you can just store the last two values in the each time.
You make a new one.
Delete the oldest.
So by thinking a little bit here you realize you only need constant space still linear time in space.
And that's it.
That's often the case from the bottom up perspective.
You see what you really need to store what you need to keep track of.
Alright i guess another nice thing about this perspective is the running time is totally obvious or this is clearly constant.
I'm so this is clearly linear time.
Whereas in this memo you have to think about when's it going to be memorize when's it's not i still like this perspective because with this rule just multiply number so problems by time for some problem you get the answer but it's a little less obvious that anna been cold like this.
So choose.
However you like to think about it.
We move onto shortest paths.
So i'm again as usual thinking about single source shortest paths.
So you want to compute the shortest path weight from s to be for all the hey i'd like to write this.
Initially as a naive recursive algorithm which i can then memorize which i can then bought them up if i i just made that up.
So how can i write this as a naive recursive algorithm?
So obvious?
Hey bud.
First i'm going to tell you how just as an oracle tells you here's what you should do.
But then we're going to think about go back step back.
But actually i mean it's up to you.
I can tell you the answer and we can figure out how we got there or we can just figure out the answer.
Preferences figure it out.

No divine inspirational ass so let me give you a tool.
Tool is guessing.
Cat sound silly but it's a very powerful tool.
General idea is suppose you don't know something but you'd like to know it.
Satellite lena what's the answer this question?
I don't know and i really want to question.
How am i going to answer the question?
Yes okay it's head tried and tested method for solving any problem.
I'm laboring the point here at concept is don't just try any guess try them all.
Cast of i said that simple tryall guesses this is central to the programming.
I know it sounds obvious.
But if i want to fix my equation here.
Dynamic programming is roughly recursion + memorization.
There should really be plus guessing.
Memorization which is obvious guessing which is obvious are the central concept 2 dynamic programming trying to make it sound easy because usually people have trouble try all the guests that something a computer can do great.
This is the brute force part.
Okay but we're going to do it carefully.
Knock knock that carefully.
I mean we're just trying all the gases take the best one.
That's kind of important that we can choose one to be called best.
That's why dynamic programming is good for optimization problems.
When i max my something to my something you try them all and then you can forget about all of them and just reduce it down to one thing which is the best one or a best one.
Okay so now i want you to try to apply this principle to shortest paths now.
I'm going to draw a picture which may help.
Have a source s where's somewhere text me we like to find the shortest a shortest path from s to be at suppose.
I want to know what the shortest path is supposed to this was it?
Have an idea what do you do?
Where you can go.
Got it so i can look at all the places i can go from ass and then look at the shortest path from there to be so we can call this a prime.
So here's the idea.
There's some hypothetical shortest path.
I don't know where it goes first.
So i will guess where it goes first.
I know the first edge must be one of the outgoing i just format so i don't know which one try them all.
Very simple idea then for me to those if somehow i can compute the shortest path from there to me.

Just do that and take the best choice for what that first edge was.
So this would be me guess first edge approach.
That's a very good idea.
Not quite the one i want it because unfortunately that changes s answer this would work.
It would just be slightly less efficient.
If i'm solving single source shortest paths by guessing the last edge instead of the first day there really a cool one.
If i was doing this side essential to be solving single target shortest path which we talked about.
Before so i'm going to draw the same picture.
I want to get the v.
I'm going to guess the last edge call it uv i know it's one of the incoming edges to be.
Unless equals b then there's a special case as long as this passes linked at least one.
There's some last edge what is it?
I don't know guess guess all the possible incoming edges to v and then recursively compute the shortest path from s to you and then add on the edge b peso what is the shortest path?
It's delta of s u which is looks the same as another some problem that i want to solve their visa problems here i care about so that's good.
I take that i add on the weight of the edge uv and that should hopefully give me delta a mess be well if i was lucky and i guess the right choice of you in reality.
I'm not lucky so i have to minimize.
Overall edges uv so this is the word minimizing over the choice of uv is already given here.
So i take the minimum overall edges of the shortest path from sd you plus the weight of the suv patrick give me the shortest path cuz this gave me the shortest path from s to you then i added on the edge i need to get there and wherever the shortest path is it has some and use the sun last ngv.
There's got to be some first of you that's the right one.
That's the that's the good guess that we're hoping for.
We don't know what the good guesses.
So we just try them all but whatever it is this will be the weight of that pass going to take the best path from s to you because sometimes the shortest has the shortest structure.
So this part will be dealt.
So that's you this part is obviously wfuv.
So this will give the right answer.
Hopefully certainly going to i mean this is the analog of the naive recursive algorithm for fibonacci so it's not going to be efficient.
If i mean this is an algorithm right you could say this is a recursive call.
I treat this as recursive call.
Instead of just a definition then this is a recursive algorithm.
How good or bad is this recursive algorithm?
Terrible target very bad.
Definitely going to be exponential.
Without memorization but we know we know how to make alkanes better.

We memorize i think you know how to write this as a man wise algorithm to define the function delta of sv first check is s v in the memo table.
If so return that value.
Otherwise do this computation with this is a recursive call and then stored in the memory table.
Okay i don't think i need to write that down.
So just like the memorize code over there just there's not to argue instead of want.
In fact ss and changing so i only need to store with v t is that a good algorithm?
I claim my position makes everything faster.
Is that a fast algorithm?
That's why i guess i guess.
Yes how many people think?
Yes it's a good over there.
Better feeling better.
Can't be worse.
How many people think it's a bad out in them.
Including the s boats good all right.
It's not such a key.
Let me dry you a graph.
Something like that.
So we wanted to come in delta s v i think of these guys names a&b so we compute delta s v to compute that we need to know delta of s a & s be at those are the two ways.
Actually we just need one.
Only one incoming edge to be so it's delta of ass sorry i should have put a better bass case here to delta equals 0.
Okay the best and plus the edge and there's some shortest path to i want to compute the trespass from me.
I don't know there's two ways to get to be one of them is delta s be sorry sms came from s the other way is delta of s v just your problem yeah she trying to figure out.
Thought you might say.
Oh it's okay because we're going to memorize or answer is belfast me and then we can reuse it here except we haven't finished computing delta s v we can only put it in the memo table once we're done.
So at this one s call happens remember tables not been set and we're going to do the same thing over and over and over again.
This is an infinite algorithm.
Oops not so hot.
So it's going to be internet.
Time on grass recycles if your dad's are you sick with graphs?
It actually runs envy pussy time.
This is the case in this situation we can use this formula.
The time is equal to the number of star problems x the time purse problem.
So i guess we have to think about that a little bit.

Where's my code?
Here's my code number of some problems is v is be different some problems that i'm using here.
I'm always raising some problems out the form delta s something but something could be any of the v vertices how much time do i spend per sub problem?
It's a little tricky.
The number of incoming edges to me.
So time for sub problem.
Delta msp is the in degree?
The number of incoming i just to be so i just depends on peace so i can't just take a straight for a product here.
But this is really saying is you should sum up overall subproblems of the time personal problem.
So total time and some of robin me in degree v and we know this is number mattress that's really some in degree + 1 + 3 + 1 so this is hey i'm shaking one again.
Okay now we already knew and i were the four shortest paths in bags and it ran if he plus he time so it's another way to do the same thing.
If you think about it long enough this algorithm memorize is essentially doing a depth first search to do a topological sort to run one round of bellman ford.
So we've had topological sort plus one round of this is kind of it all rolled into one.
This should look kind of like the bellman ford relaxation step or short as bad as relaxation step it is this man is really doing the same thing.
So it's really the same time but we come at it from a different perspective.
Okay but i claim i can use the same approach to solve shortest paths in general grass even when they have cycles how am i going to do that?
Bag seem fine what was the lesson learned here lesson learned?
Is that some problem?
Tendencies should be a cyclic.
Otherwise we get an infinite algorithm for memorization to work.
This is what you need all you need.
Hey we almost seen this already because i said that to do bottom up algorithm.
You do a topological sort of the sun problem.
Dependency dag i already said it should i forgot i didn't tell you yet.
So for that to work it better be a cichlid for dp to work for memorization to work.
It better be a cyclic if you're a cyclic then this is the running time.
So that's all general so somehow i need to take a cyclic graph and make it a cyclic.
He's actually done this already in recitation.
So if i have a graph?
Let's just take a very simple cyclic graph.
Hey one thing i can do is explode it into multiple layers who did the sun quiz 2 and various forms.
It's like the only cool thing you want to make us for his mouth from harder require that you reduce your graph to take copies of the graph.
I'm going to do it in a particular way.
Which is the think of this axis as time or however you want and make all of the edges go from

each layer to the next layer.
And it should be a familiar technique is every time i go down to the next layer.
This makes any graph a cyclic done.
What in the world is does this mean what is it doing?
Does it mean double rainbow?
All right so i used to be my favorite.
Here's what it means delta sub k s v i got to define this first.
This is a new kind of some problem.
Which is what is the shortest?
What's the weight of the shortest?
Destiny path but uses at most k address so i wanted to be sure to sometimes a total weight but i also wanted to ask you i just total so this is going to be zero sense if you look at so here is his ass and always going to make ask this and then this is going to be v in the zero situation is going to be v in the one situation be so if i look at this fee i look at the trespass from s2v.
That is delta sub zero sub zero patriots path from here to here is the there's no way to get there until reggie shortest path from here to hear.
That's this the best way to get there with that most 1 inch trespassed from here to hear some vertical itis to i guess cheating a little bit then this is the best way to get from s to be using at most two edges and then you get a recurrence.
Does the man of raw last stages?
So just copying that recurrence but realizing that the ps2 u part uses one fewer edge and then i use the suv that's our new recurrence by adding this k parameter.
I've made this recurrence on some problems a cyclic unfortunately have increase the number of subproblems number of some problems.
Now is b squared?
Technically three times be minus one cuz i really and what i care about but my goal.
Is delta savvy 1 sv because my bellman ford analysis.
I know that i only care about simplepass as a like that must be mad if i'm assuming here.
No negative weight cycles said that earlier you soon that then this is what i care about.
So k ranges from 0 to be minus one.
So there be choices for cat their v choices for v so number so problems is b squared how much time do i spend for?
Some problem was same as before the in degree?