Dynamic Programming: Example Dynamic Programming Problems

Kosie van der Merwe Michiel Baird

30 May 2009

Dynamic Programming

- Once you know what DP problems are (and we assume you all do) the main part in doing them well is:
  - Practice
  - Practice
  - Practice
- So bring on the sample problems!

Dynamic Programming

- Once you know what DP problems are (and we assume you all do) the main part in doing them well is:
  - Practice
  - Practice
  - Practice
- So bring on the sample problems!

Dynamic Programming

- Once you know what DP problems are (and we assume you all do) the main part in doing them well is:
  - Practice
  - Practice
  - Practice
- So bring on the sample problems!

Dynamic Programming

- Once you know what DP problems are (and we assume you all do) the main part in doing them well is:
  - Practice
  - Practice
  - Practice
- So bring on the sample problems!

Dynamic Programming

- Once you know what DP problems are (and we assume you all do) the main part in doing them well is:
  - Practice
  - Practice
  - Practice
- So bring on the sample problems!

Outline

1 Piggy bank

2 Chest of Drawers

3 Miners

4 Unicorn

Piggy bank

Piggy bank

- You are given a piggy bank filled with coins. You need to determine what the minimum amount of money is that could be in the piggy bank.
- You are given the the weight of the piggy bank.
- You are also given a set of N coins with weight and value.

Piggy bank-Strategy

- Brute force ?
- Efficiency $O(2^N)$
- This clearly isn't a feasible option
- So lets start with a this really simple...
- DP!

Piggy bank-Strategy

- Brute force ?
- Efficiency $O(2^N)$
- This clearly isn't a feasible option
- So lets start with a this really simple...
- DP!

Piggy bank - The plan

- How to solve this simple DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base states?
- Its fairly easy to see that if we pre-compute the minimum value for all weights less than W, the problem becomes a lot easier.
- That means the dimension is:
  - The weight of the piggy bank.

Piggy bank - The plan

- How to solve this simple DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base states?
- Its fairly easy to see that if we pre-compute the minimum value for all weights less than W, the problem becomes a lot easier.
- That means the dimension is:
  - The weight of the piggy bank.

Piggy bank - The plan

- How to solve this simple DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base states?
- Its fairly easy to see that if we pre-compute the minimum value for all weights less than W, the problem becomes a lot easier.
- That means the dimension is:
  - The weight of the piggy bank.

Piggy bank - The plan

- How to solve this simple DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base states?
- Its fairly easy to see that if we pre-compute the minimum value for all weights less than W, the problem becomes a lot easier.
- That means the dimension is:
  - The weight of the piggy bank.

Piggy bank - The plan

- How to solve this simple DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base states?
- Its fairly easy to see that if we pre-compute the minimum value for all weights less than W, the problem becomes a lot easier.
- That means the dimension is:
  - The weight of the piggy bank.

Piggy bank - The plan

- How to solve this simple DP:
    - What is the sub problem?
    - The problems dimensions?
    - Recurrence relation?
    - Base states?
- Its fairly easy to see that if we pre-compute the minimum value for all weights less than W, the problem becomes a lot easier.
- That means the dimension is:
    - The weight of the piggy bank.

Piggy bank - The plan

- How to solve this simple DP:
    - What is the sub problem?
    - The problems dimensions?
    - Recurrence relation?
    - Base states?
- Its fairly easy to see that if we pre-compute the minimum value for all weights less than W, the problem becomes a lot easier.
- That means the dimension is:
    - The weight of the piggy bank.

Piggy bank-Recurrence relation

- The state: DP_w
- The recurrence relationship becomes:
  - Over all coins:
  - $DP_w = min(DP_w, DP_{w-coinweight} + coinvalue)$
- Base cases: $DP_0 = 0$ and $DP_w = infinity$ for $1 <= w <= W$
- This gives us a effiency of $O(NW)$
- The final answer is just DP_W
- Questions?

Piggy bank-Recurrence relation

- The state: DP_w
- The recurrence relationship becomes:
  - Over all coins:
  - $DP_w = min(DP_w, DP_{w-coinweight} + coinvalue)$
- Base cases: $DP_0 = 0$ and $DP_w = infinity$ for $1 <= w <= W$
- This gives us a effiency of $O(NW)$
- The final answer is just DP_W
- Questions?

Piggy bank-Recurrence relation

- The state: DP_w
- The recurrence relationship becomes:
    - Over all coins:
    - $DP_w = min(DP_w, DP_{w-coinweight} + coinvalue)$
- Base cases: $DP_0 = 0$ and $DP_w = infinity$ for $1 <= w <= W$
- This gives us a effiency of $O(NW)$
- The final answer is just DP_W
- Questions?

Outline

1 Piggy bank

2 Chest of Drawers

3 Miners

4 Unicorn

Chest of Drawers

Chest of Drawers - Problem D of ACM World Finals Warmup 2008

- You are given a chest of drawers with N drawers, which can be locked.
- You want to have exactly S secured drawers.
- But not all locked drawers are secure.
- A locked drawer is secure:
  - If it's either the top drawer.
  - If the drawer directly above it is locked aswell.
- Constraints:
  - $0 \leq N \leq 65$
  - $0 \leq S \leq 65$
- You have to count how many ways you can locked N drawers to give you S secure drawers.

Chest of Drawers

Chest of Drawers - Problem D of ACM World Finals Warmup 2008

- You are given a chest of drawers with N drawers, which can be locked.
- You want to have exactly S secured drawers.
- But not all locked drawers are secure.
- A locked drawer is secure:
  - If it's either the top drawer.
  - If the drawer directly above it is locked aswell.
- Constraints:
  - $0 \leq N \leq 65$
  - $0 \leq S \leq 65$
- You have to count how many ways you can locked N drawers to give you S secure drawers.

Chest of Drawers

Chest of Drawers - Problem D of ACM World Finals Warmup 2008

- You are given a chest of drawers with N drawers, which can be locked.
- You want to have exactly S secured drawers.
- But not all locked drawers are secure.
- A locked drawer is secure:
  - If it's either the top drawer.
  - If the drawer directly above it is locked aswell.
- Constraints:
  - $0 \leq N \leq 65$
  - $0 \leq S \leq 65$
- You have to count how many ways you can locked N drawers to give you S secure drawers.

Chest of Drawers

Chest of Drawers - Problem D of ACM World Finals Warmup 2008

- You are given a chest of drawers with N drawers, which can be locked.
- You want to have exactly S secured drawers.
- But not all locked drawers are secure.
- A locked drawer is secure:
  - If it's either the top drawer.
  - If the drawer directly above it is locked aswell.
- Constraints:
  - $0 \leq N \leq 65$
  - $0 \leq S \leq 65$
- You have to count how many ways you can locked N drawers to give you S secure drawers.

Chest of Drawers

Chest of Drawers - Problem D of ACM World Finals Warmup 2008

- You are given a chest of drawers with N drawers, which can be locked.
- You want to have exactly S secured drawers.
- But not all locked drawers are secure.
- A locked drawer is secure:
    - If it's either the top drawer.
    - If the drawer directly above it is locked aswell.
- Constraints:
    - $0 \leq N \leq 65$
    - $0 \leq S \leq 65$
- You have to count how many ways you can locked N drawers to give you S secure drawers.

Chest of Drawers - Solution

- Brute forcing this problem is way too slow.
- The run time of the brute force solution is $O(N2^N)$
- Because there are 2^N possible locking combinations.
- And each combination takes $O(N)$ to check.
- You might be able speed up the single checks but the 2^N will continue to kill you.
- So we need another solution: DP

Chest of Drawers - Solution

- Brute forcing this problem is way too slow.
- The run time of the brute force solution is $O(N2^N)$
- Because there are 2^N possible locking combinations.
- And each combination takes $O(N)$ to check.
- You might be able speed up the single checks but the 2^N will continue to kill you.
- So we need another solution: DP

Chest of Drawers - Solution

- Brute forcing this problem is way too slow.
- The run time of the brute force solution is $O(N2^N)$
- Because there are 2^N possible locking combinations.
- And each combination takes $O(N)$ to check.
- You might be able speed up the single checks but the 2^N will continue to kill you.
- So we need another solution: DP

Chest of Drawers - Solution

- Brute forcing this problem is way too slow.
- The run time of the brute force solution is $O(N2^N)$
- Because there are 2^N possible locking combinations.
- And each combination takes $O(N)$ to check.
- You might be able speed up the single checks but the 2^N will continue to kill you.
- So we need another solution: DP

Chest of Drawers - Solution

- Brute forcing this problem is way too slow.
- The run time of the brute force solution is $O(N2^N)$
- Because there are 2^N possible locking combinations.
- And each combination takes $O(N)$ to check.
- You might be able speed up the single checks but the 2^N will continue to kill you.
- So we need another solution: DP

Chest of Drawers - Solution(continued)

- In the DP state we want to store the number of possibilities.
- The dimensions for the DP are:
    - Number of drawers.
    - Number of secure drawers.
    - Is last drawer locked or unlocked.

Chest of Drawers - Solution(continued)

- In the DP state we want to store the number of possibilities.
- The dimensions for the DP are:
    - Number of drawers.
    - Number of secure drawers.
    - Is last drawer locked or unlocked.

Chest of Drawers - Solution(continued)

- In the DP state we want to store the number of possibilities.
- The dimensions for the DP are:
  - Number of drawers.
  - Number of secure drawers.
  - Is last drawer locked or unlocked.

Chest of Drawers - Solution(continued)

- In the DP state we want to store the number of possibilities.
- The dimensions for the DP are:
  - Number of drawers.
  - Number of secure drawers.
  - Is last drawer locked or unlocked.

Chest of Drawers - Solution(continued)

- In the DP state we want to store the number of possibilities.
- The dimensions for the DP are:
  - Number of drawers.
  - Number of secure drawers.
  - Is last drawer locked or unlocked.

Chest of Drawers - Solution(continued)

- Now for the recurrence relationships.
- Let's label our state: $DP_{n,s,l}$
- Where l is whether the last drawer is locked or not: 0 - unlocked, 1 - locked.
- Then we have the following recurrence relationships:
  - $DP_{n,s,1} = DP_{n-1,s,0} + (DP_{n-1,s-1,1}$ if $s > 0$ else 0)
  - $DP_{n,s,0} = DP_{n-1,s,0} + DP_{n-1,s,1}$
- Where all states equal 0 except $DP_{1,1,1} = DP_{1,0,0} = 1$
- Then the answer we are looking for is: $DP_{N,S,0} + DP_{N,S,1}$

Chest of Drawers - Solution(continued)

- Now for the recurrence relationships.
- Let's label our state: $DP_{n,s,l}$
- Where l is whether the last drawer is locked or not: 0 - unlocked, 1 - locked.
- Then we have the following recurrence relationships:
  - $DP_{n,s,1} = DP_{n-1,s,0} + (DP_{n-1,s-1,1} \text{ if } s > 0 \text{ else } 0)$
  - $DP_{n,s,0} = DP_{n-1,s,0} + DP_{n-1,s,1}$
- Where all states equal 0 except $DP_{1,1,1} = DP_{1,0,0} = 1$
- Then the answer we are looking for is: $DP_{N,S,0} + DP_{N,S,1}$

Chest of Drawers - Solution(continued)

- Now for the recurrence relationships.
- Let's label our state: $DP_{n,s,l}$
- Where l is whether the last drawer is locked or not: 0 - unlocked, 1 - locked.
- Then we have the following recurrence relationships:
  - $DP_{n,s,1} = DP_{n-1,s,0} + (DP_{n-1,s-1,1} \text{ if } s > 0 \text{ else } 0)$
  - $DP_{n,s,0} = DP_{n-1,s,0} + DP_{n-1,s,1}$
- Where all states equal 0 except $DP_{1,1,1} = DP_{1,0,0} = 1$
- Then the answer we are looking for is: $DP_{N,S,0} + DP_{N,S,1}$

Chest of Drawers - Solution(continued)

- Now for the recurrence relationships.
- Let's label our state: $DP_{n,s,l}$
- Where l is whether the last drawer is locked or not: 0 - unlocked, 1 - locked.
- Then we have the following recurrence relationships:
  - $DP_{n,s,1} = DP_{n-1,s,0} + (DP_{n-1,s-1,1} \text{ if } s > 0 \text{ else } 0)$
  - $DP_{n,s,0} = DP_{n-1,s,0} + DP_{n-1,s,1}$
- Where all states equal 0 except $DP_{1,1,1} = DP_{1,0,0} = 1$
- Then the answer we are looking for is: $DP_{N,S,0} + DP_{N,S,1}$

Chest of Drawers - Solution(continued)

- Now for the recurrence relationships.
- Let's label our state: $DP_{n,s,l}$
- Where l is whether the last drawer is locked or not: 0 - unlocked, 1 - locked.
- Then we have the following recurrence relationships:
  - $DP_{n,s,1} = DP_{n-1,s,0} + (DP_{n-1,s-1,1}$ if $s > 0$ else 0)
  - $DP_{n,s,0} = DP_{n-1,s,0} + DP_{n-1,s,1}$
- Where all states equal 0 except $DP_{1,1,1} = DP_{1,0,0} = 1$
- Then the answer we are looking for is: $DP_{N,S,0} + DP_{N,S,1}$

Chest of Drawers - Solution(continued)

- Now for the recurrence relationships.
- Let's label our state: $DP_{n,s,l}$
- Where l is whether the last drawer is locked or not: 0 - unlocked, 1 - locked.
- Then we have the following recurrence relationships:
  - $DP_{n,s,1} = DP_{n-1,s,0} + (DP_{n-1,s-1,1}$ if $s > 0$ else 0)
  - $DP_{n,s,0} = DP_{n-1,s,0} + DP_{n-1,s,1}$
- Where all states equal 0 except $DP_{1,1,1} = DP_{1,0,0} = 1$
- Then the answer we are looking for is: $DP_{N,S,0} + DP_{N,S,1}$

Chest of Drawers - Solution(continued)

- Now for the recurrence relationships.
- Let's label our state: $DP_{n,s,l}$
- Where l is whether the last drawer is locked or not: 0 - unlocked, 1 - locked.
- Then we have the following recurrence relationships:
  - $DP_{n,s,1} = DP_{n-1,s,0} + (DP_{n-1,s-1,1} \text{ if } s > 0 \text{ else } 0)$
  - $DP_{n,s,0} = DP_{n-1,s,0} + DP_{n-1,s,1}$
- Where all states equal 0 except $DP_{1,1,1} = DP_{1,0,0} = 1$
- Then the answer we are looking for is: $DP_{N,S,0} + DP_{N,S,1}$

Chest of Drawers - Discussion

- The DP solution is $O(NS)$ running time and uses $O(NS)$ memory.
- You can trivially bring down the memory usage to $O(S)$.

Chest of Drawers - Discussion

- The DP solution is $O(NS)$ running time and uses $O(NS)$ memory.
- You can trivially bring down the memory usage to $O(S)$.

Outline

1 Piggy bank

2 Chest of Drawers

3 Miners

4 Unicorn

Miners

Miners - IOI 2007 Day 2

- There are 2 coals mines and you are given a sequence of N food shipments.
- You have to send each shipment in its entirety to one mine or another.
- There are 3 types of food shipments: Meat('M'), Fish('F'), Bread('B')
- Miners only produce coal after food arrives, but they also like variety.
- They consider the current shipment plus the last 2:
    - If 3 types of food, then they produce 3 units of coal.
    - If 2 types of food, then they produce 2 units of coal.
    - If 1 types of food, then they produce 1 units of coal.
- Find the maximum amount of coal you can produce.
- Constraints: $1 \leq N \leq 100000$

Miners

Miners - IOI 2007 Day 2

- There are 2 coals mines and you are given a sequence of N food shipments.
- You have to send each shipment in its entirety to one mine or another.
- There are 3 types of food shipments: Meat('M'), Fish('F'), Bread('B')
- Miners only produce coal after food arrives, but they also like variety.
- They consider the current shipment plus the last 2:
  - If 3 types of food, then they produce 3 units of coal.
  - If 2 types of food, then they produce 2 units of coal.
  - If 1 types of food, then they produce 1 units of coal.
- Find the maximum amount of coal you can produce.
- Constraints: $1 \leq N \leq 100000$

Miners

Miners - IOI 2007 Day 2

- There are 2 coals mines and you are given a sequence of N food shipments.
- You have to send each shipment in its entirety to one mine or another.
- There are 3 types of food shipments: Meat('M'), Fish('F'), Bread('B')
- Miners only produce coal after food arrives, but they also like variety.
- They consider the current shipment plus the last 2:
  - If 3 types of food, then they produce 3 units of coal.
  - If 2 types of food, then they produce 2 units of coal.
  - If 1 types of food, then they produce 1 units of coal.
- Find the maximum amount of coal you can produce.
- Constraints: $1 \leq N \leq 100000$

Miners

Miners - IOI 2007 Day 2

- There are 2 coals mines and you are given a sequence of N food shipments.
- You have to send each shipment in its entirety to one mine or another.
- There are 3 types of food shipments: Meat('M'), Fish('F'), Bread('B')
- Miners only produce coal after food arrives, but they also like variety.
- They consider the current shipment plus the last 2:
  - If 3 types of food, then they produce 3 units of coal.
  - If 2 types of food, then they produce 2 units of coal.
  - If 1 types of food, then they produce 1 units of coal.
- Find the maximum amount of coal you can produce.
- Constraints: $1 \leq N \leq 100000$

Miners

Miners - IOI 2007 Day 2

- There are 2 coals mines and you are given a sequence of N food shipments.
- You have to send each shipment in its entirety to one mine or another.
- There are 3 types of food shipments: Meat('M'), Fish('F'), Bread('B')
- Miners only produce coal after food arrives, but they also like variety.
- They consider the current shipment plus the last 2:
    - If 3 types of food, then they produce 3 units of coal.
    - If 2 types of food, then they produce 2 units of coal.
    - If 1 types of food, then they produce 1 units of coal.
- Find the maximum amount of coal you can produce.
- Constraints: $1 \leq N \leq 100000$

Miners - Solution

- You can either send a shipment to one mine to one mine or another, so you have 2^N possiblities.
- If you brute force this problem you will get a running time of $O(N2^N)$, which is far too slow even if you get rid of the N factor.
- So let's try DP.

Miners - Solution

- You can either send a shipment to one mine to one mine or another, so you have 2^N possiblities.
- If you brute force this problem you will get a running time of $O(N2^N)$, which is far too slow even if you get rid of the N factor.
- So let's try DP.

Miners - Solution

- You can either send a shipment to one mine to one mine or another, so you have 2^N possiblities.
- If you brute force this problem you will get a running time of $O(N2^N)$, which is far too slow even if you get rid of the N factor.
- So let's try DP.

Miners - Solution(Continued)

- At each point in the DP you'll want to store the biggest amount of coal you can get.
- The dimensions:
  - The number of deliveries so far.
  - Which shipments have gone to the first mine.
  - Which shipments have gone to the second mine.
- Note: that you only have to store the last two shipments recieved by each mine and the you just have a special value to indicate that no shipments have been recieved.
- Trick: When you add a shipment to a mine that hasn't recieved any shipment before, just double the shipment(but count once), this will simply coding yet give the same result.

Miners - Solution(Continued)

- At each point in the DP you'll want to store the biggest amount of coal you can get.
- The dimensions:
  - The number of deliveries so far.
  - Which shipments have gone to the first mine.
  - Which shipments have gone to the second mine.
- Note: that you only have to store the last two shipments recieved by each mine and the you just have a special value to indicate that no shipments have been recieved.
- Trick: When you add a shipment to a mine that hasn't recieved any shipment before, just double the shipment(but count once), this will simply coding yet give the same result.

Miners - Solution(Continued)

- At each point in the DP you'll want to store the biggest amount of coal you can get.
- The dimensions:
  - The number of deliveries so far.
  - Which shipments have gone to the first mine.
  - Which shipments have gone to the second mine.
- Note: that you only have to store the last two shipments recieved by each mine and the you just have a special value to indicate that no shipments have been recieved.
- Trick: When you add a shipment to a mine that hasn't recieved any shipment before, just double the shipment(but count once), this will simply coding yet give the same result.

Miners - Solution(Continued)

- At each point in the DP you'll want to store the biggest amount of coal you can get.
- The dimensions:
  - The number of deliveries so far.
  - Which shipments have gone to the first mine.
  - Which shipments have gone to the second mine.
- Note: that you only have to store the last two shipments recieved by each mine and the you just have a special value to indicate that no shipments have been recieved.
- Trick: When you add a shipment to a mine that hasn't recieved any shipment before, just double the shipment(but count once), this will simply coding yet give the same result.

Miners - Solution(Continued)

- At each point in the DP you'll want to store the biggest amount of coal you can get.
- The dimensions:
    - The number of deliveries so far.
    - Which shipments have gone to the first mine.
    - Which shipments have gone to the second mine.
- Note: that you only have to store the last two shipments recieved by each mine and the you just have a special value to indicate that no shipments have been recieved.
- Trick: When you add a shipment to a mine that hasn't recieved any shipment before, just double the shipment(but count once), this will simply coding yet give the same result.

Miners - Solution(Continued)

- At each point in the DP you'll want to store the biggest amount of coal you can get.
- The dimensions:
  - The number of deliveries so far.
  - Which shipments have gone to the first mine.
  - Which shipments have gone to the second mine.
- Note: that you only have to store the last two shipments recieved by each mine and the you just have a special value to indicate that no shipments have been recieved.
- Trick: When you add a shipment to a mine that hasn't recieved any shipment before, just double the shipment(but count once), this will simply coding yet give the same result.

Miners - Solution(Continued)

- At each point in the DP you'll want to store the biggest amount of coal you can get.
- The dimensions:
  - The number of deliveries so far.
  - Which shipments have gone to the first mine.
  - Which shipments have gone to the second mine.
- Note: that you only have to store the last two shipments recieved by each mine and the you just have a special value to indicate that no shipments have been recieved.
- Trick: When you add a shipment to a mine that hasn't recieved any shipment before, just double the shipment(but count once), this will simply coding yet give the same result.

Miners - Solution(Continued)

- Let's name our DP state: DP_{n,sr_1,sr_2}
- Let's label our shipments: $shipment_i$ where $1 \leq i \leq N$
- Let's describe the shipments to each mine using a tuple:
    - Where () signifies no shipments recieved.
    - All other tuples are of the form (s_1, s_2), where s_1 is the last shipment recieved and s_2 the second last.
    - Define $(s_1, s_2) + s_3 = (s_3, s_1)$ and $() + s_3 = (s_3, s_3)$
- All the states in the DP start out as zero.
- Except for $DP_{1,(shipment_1,shipment_1),()} = DP_{1,(),(shipment_1,shipment_1)} = 1$
- $value(sr, s)$ returns the amount of coal recieved, after recieving sr and now recieving s.

Miners - Solution(Continued)

- Now for the recurrence relations:
  - $DP_{n,sr_1+shipment_n,sr_2} = max(DP_{n,sr_1+shipment_n,sr_2}, DP_{n-1,sr_1,sr_2} + value(sr_1, shipment_n))$
  - $DP_{n,sr_1,sr_2+shipment_n} = max(DP_{n,sr_1,sr_2+shipment_n}, DP_{n-1,sr_1,sr_2} + value(sr_2, shipment_n))$
  - If $DP_{n-1,sr_1,sr_2} = 0$ then you ignore this state, as it's unreachable.
- Then the answer is $max(DP_{N,a,b})$ for all valid tuples a and b.

Miners - Solution(Continued)

- Now for the recurrence relations:
  - $DP_{n,sr_1+shipment_n,sr_2} = max(DP_{n,sr_1+shipment_n,sr_2}, DP_{n-1,sr_1,sr_2} + value(sr_1, shipment_n))$
  - $DP_{n,sr_1,sr_2+shipment_n} = max(DP_{n,sr_1,sr_2+shipment_n}, DP_{n-1,sr_1,sr_2} + value(sr_2, shipment_n))$
  - If $DP_{n-1,sr_1,sr_2} = 0$ then you ignore this state, as it's unreachable.
- Then the answer is $max(DP_{N,a,b})$ for all valid tuples a and b.

Miners - Solution(Continued)

- Now for the recurrence relations:
  - $DP_{n,sr_1+shipment_n,sr_2} = max(DP_{n,sr_1+shipment_n,sr_2}, DP_{n-1,sr_1,sr_2} + value(sr_1, shipment_n))$
  - $DP_{n,sr_1,sr_2+shipment_n} = max(DP_{n,sr_1,sr_2+shipment_n}, DP_{n-1,sr_1,sr_2} + value(sr_2, shipment_n))$
  - If $DP_{n-1,sr_1,sr_2} = 0$ then you ignore this state, as it's unreachable.
- Then the answer is $max(DP_{N,a,b})$ for all valid tuples a and b.

Miners - Solution(Continued)

- Now for the recurrence relations:
  - $DP_{n,sr_1+shipment_n,sr_2} = max(DP_{n,sr_1+shipment_n,sr_2}, DP_{n-1,sr_1,sr_2} + value(sr_1, shipment_n))$
  - $DP_{n,sr_1,sr_2+shipment_n} = max(DP_{n,sr_1,sr_2+shipment_n}, DP_{n-1,sr_1,sr_2} + value(sr_2, shipment_n))$
  - If $DP_{n-1,sr_1,sr_2} = 0$ then you ignore this state, as it's unreachable.
- Then the answer is $max(DP_{N,a,b})$ for all valid tuples a and b.

Miners - Solution(Continued)

- Analysis:
  - The DP solution has a running time $O(100N)$, because for each mine you have can have $3 \times 3 + 1 = 10$ possible shipments recieved states.
  - Similarly the memory usage is $O(100N)$, but can easily be brought down to $O(100 \times 2) = O(1)$ and indeed has to be in order to score full marks.

Miners - Solution(Continued)

- Analysis:
  - The DP solution has a running time $O(100N)$, because for each mine you have can have $3 \times 3 + 1 = 10$ possible shipments recieved states.
  - Similarly the memory usage is $O(100N)$, but can easily be brought down to $O(100 \times 2) = O(1)$ and indeed has to be in order to score full marks.

Miners - Solution(Continued)

- Analysis:
  - The DP solution has a running time $O(100N)$, because for each mine you have can have $3 \times 3 + 1 = 10$ possible shipments recieved states.
  - Similarly the memory usage is $O(100N)$, but can easily be brought down to $O(100 \times 2) = O(1)$ and indeed has to be in order to score full marks.

Miners - Solution(Continued)

- Implementation tips:
  - You can represent the shipment recieved states by using bitmasks and then using bitwise operators to implement addition.
  - The easiest scheme for this I could come up with is using the first two bits to represent the last shipment recieved and the third and forth to represent the second last shipment recieved.
  - Then you can use 0 to represent the case where no shipments have been recieved.
  - If you check that a DP state isn't 0 before using it you don't have to worry about nonsensical shipment states like 0010_2 ruining your DP, as they will always be ignored and thus stay 0.

Miners - Solution(Continued)

- Implementation tips:
  - You can represent the shipment recieved states by using bitmasks and then using bitwise operators to implement addition.
  - The easiest scheme for this I could come up with is using the first two bits to represent the last shipment recieved and the third and forth to represent the second last shipment recieved.
  - Then you can use 0 to represent the case where no shipments have been recieved.
  - If you check that a DP state isn't 0 before using it you don't have to worry about nonsensical shipment states like 0010_2 ruining your DP, as they will always be ignored and thus stay 0.

Miners - Solution(Continued)

- Implementation tips:
  - You can represent the shipment recieved states by using bitmasks and then using bitwise operators to implement addition.
  - The easiest scheme for this I could come up with is using the first two bits to represent the last shipment recieved and the third and forth to represent the second last shipment recieved.
  - Then you can use 0 to represent the case where no shipments have been recieved.
  - If you check that a DP state isn't 0 before using it you don't have to worry about nonsensical shipment states like 0010_2 ruining your DP, as they will always be ignored and thus stay 0.

Outline

1 Piggy bank

2 Chest of Drawers

3 Miners

4 Unicorn

Unicorn

Unicorn - TCO 09 Round 1 Question 3

- The unicorn is an exotic chess piece similar to the knight. The difference is that while the knight moves two cells in one direction and one in the other direction, the unicorn moves more than two cells in one direction and more than one in the other direction.
- You are given an $R \times C$ chess board with numbers on the squares.
- You are also given a sequence of numbers.
- Find the number of ways you can use The Unicorn to produce the given sequence on the board.

Unicorn-Constraints

- R and C will be between 1 and 300, inclusive.
- The numbers on the board will be between 1 and 30
- The sequence will have a length of between 1 and 50.

Unicorn-Strategy

- A brute force solution is clearly out of the question.
- Upper bound at $O(LR^2C^2)$
- Where L is the length of the word
- So as the spirit of this presentation goes...
- DP!

Unicorn - The plan

- How to solve this DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base cases?
- With a bit of thinking you'll see that we just have to build up the sequence starting from the first number in the sequence which has a possibility of one.
- The dimensions you need are:
  - Number of rows.
  - Number of columns.
  - Number of letters.

Unicorn - The plan

- How to solve this DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base cases?
- With a bit of thinking you'll see that we just have to build up the sequence starting from the first number in the sequence which has a possibility of one.
- The dimensions you need are:
  - Number of rows.
  - Number of columns.
  - Number of letters.

Unicorn - The plan

- How to solve this DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base cases?
- With a bit of thinking you'll see that we just have to build up the sequence starting from the first number in the sequence which has a possibility of one.
- The dimensions you need are:
  - Number of rows.
  - Number of columns.
  - Number of letters.

Unicorn - The plan

- How to solve this DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base cases?
- With a bit of thinking you'll see that we just have to build up the sequence starting from the first number in the sequence which has a possibility of one.
- The dimensions you need are:
  - Number of rows.
  - Number of columns.
  - Number of letters.

Unicorn - The plan

- How to solve this DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base cases?
- With a bit of thinking you'll see that we just have to build up the sequence starting from the first number in the sequence which has a possibility of one.
- The dimensions you need are:
  - Number of rows.
  - Number of columns.
  - Number of letters.

Unicorn - The plan

- How to solve this DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base cases?
- With a bit of thinking you'll see that we just have to build up the sequence starting from the first number in the sequence which has a possibility of one.
- The dimensions you need are:
  - Number of rows.
  - Number of columns.
  - Number of letters.

Unicorn - The plan

- How to solve this DP:
  - What is the sub problem?
  - The problems dimensions?
  - Recurrence relation?
  - Base cases?
- With a bit of thinking you'll see that we just have to build up the sequence starting from the first number in the sequence which has a possibility of one.
- The dimensions you need are:
  - Number of rows.
  - Number of columns.
  - Number of letters.

Unicorn-Recurrence relation

- Now for the recurrence relationships.
- Let's label our state: $DP_{l,r,c}$
- Then we have the following recurrence relationships:
    - $DP_{l,r,c} = \sum_{x=0}^{R} \sum_{y=0}^{C} (DP_{l-1,x,y})$ if x, y can be reached from r, c over all x, y and $board[r][c] == sequence[l]$
- Base cases: $DP_{0,r,c} =$ if $board[r][c] == sequence[0] = 1$ else $= 0$
- The final answer we are looking for is: $\sum_{x=0}^{R} \sum_{y=0}^{C} (DP_{L,x,y})$ over all x, y

Unicorn-Recurrence relation

- Now for the recurrence relationships.
- Let's label our state: $DP_{l,r,c}$
- Then we have the following recurrence relationships:
  - $DP_{l,r,c} = \sum_{x=0}^{R} \sum_{y=0}^{C} (DP_{l-1,x,y})$ if x, y can be reached from r, c over all x, y and $board[r][c] == sequence[l]$
- Base cases: $DP_{0,r,c} =$ if $board[r][c] == sequence[0] = 1$ else $= 0$
- The final answer we are looking for is: $\sum_{x=0}^{R} \sum_{y=0}^{C} (DP_{L,x,y})$ over all x, y

Unicorn-Recurrence relation

- Now for the recurrence relationships.
- Let's label our state: $DP_{l,r,c}$
- Then we have the following recurrence relationships:
  - $DP_{l,r,c} = \sum_{x=0}^{R}\sum_{y=0}^{C}(DP_{l-1,x,y})$
    if x, y can be reached from r, c over all x, y and $board[r][c] == sequence[l]$
- Base cases: $DP_{0,r,c} =$ if $board[r][c] == sequence[0] = 1$ else $= 0$
- The final answer we are looking for is: $\sum_{x=0}^{R}\sum_{y=0}^{C}(DP_{L,x,y})$

  over all x, y

Unicorn - A problem presents itself..

- Space?
  - Storing the values in 3 dimensions means space efficiency is $O(LRC)$ which is too big
- Time?
  - Searching trough the values that can be visited takes $O(RC)$
  - This causes the entire efficiency to go to $O(LR^2C^2)$
  - Which is just as bad as what we started with.
- We need to fix this...

Unicorn - A problem presents itself..

- Space?
  - Storing the values in 3 dimensions means space efficiency is $O(LRC)$ which is too big
- Time?
  - Searching trough the values that can be visited takes $O(RC)$
  - This causes the entire efficiency to go to $O(LR^2C^2)$
  - Which is just as bad as what we started with.
- We need to fix this...

Unicorn - A problem presents itself..

- Space?
  - Storing the values in 3 dimensions means space efficiency is $O(LRC)$ which is too big
- Time?
  - Searching trough the values that can be visited takes $O(RC)$
  - This causes the entire efficiency to go to $O(LR^2C^2)$
  - Which is just as bad as what we started with.
- We need to fix this...

Unicorn - A problem presents itself..

- Space?
  - Storing the values in 3 dimensions means space efficiency is $O(LRC)$ which is too big
- Time?
  - Searching trough the values that can be visited takes $O(RC)$
  - This causes the entire efficiency to go to $O(LR^2C^2)$
  - Which is just as bad as what we started with.
- We need to fix this...

Unicorn - A problem presents itself..

- Space?
  - Storing the values in 3 dimensions means space efficiency is $O(LRC)$ which is too big
- Time?
  - Searching trough the values that can be visited takes $O(RC)$
  - This causes the entire efficiency to go to $O(LR^2C^2)$
  - Which is just as bad as what we started with.
- We need to fix this...

Unicorn - A problem presents itself..

- Space?
  - Storing the values in 3 dimensions means space efficiency is $O(LRC)$ which is too big
- Time?
  - Searching trough the values that can be visited takes $O(RC)$
  - This causes the entire efficiency to go to $O(LR^2C^2)$
  - Which is just as bad as what we started with.
- We need to fix this...

Unicorn - A bit of thinking

- Idea...
  - Space: This is easily solved by realising that we only need to store the last two states.
  - This reduces our space down to $O(RC)$
  - Solving the time issue however you need to realise that there are more places on the board that you can visit than there are places you can't.

Unicorn - A bit of thinking

- Idea...
  - Space: This is easily solved by realising that we only need to store the last two states.
  - This reduces our space down to $O(RC)$
  - Solving the time issue however you need to realise that there are more places on the board that you can visit than there are places you can't.

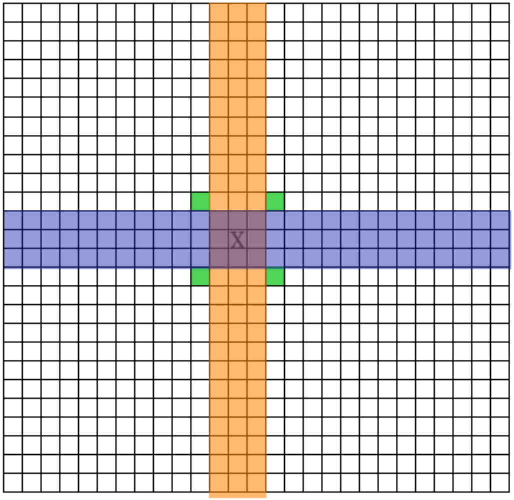
Unicorn - A bit of thinking

- Idea...
  - Space: This is easily solved by realising that we only need to store the last two states.
  - This reduces our space down to $O(RC)$
  - Solving the time issue however you need to realise that there are more places on the board that you can visit than there are places you can't.

Unicorn - A bit of thinking

- Idea...
  - Space: This is easily solved by realising that we only need to store the last two states.
  - This reduces our space down to $O(RC)$
  - Solving the time issue however you need to realise that there are more places on the board that you can visit than there are places you can't.

The board



Unicorn - Reducing time

- By calculating the total possibilities for the entire grid initially for each state as well as the counts for the rows and the columns. We can just remove the places it can get from, from the total.
- Now finding the possibilities can be done in constant time
- We now have a complexity of $O(LRC)$ which with the constraints would work.
- Questions?

Unicorn - Reducing time

- By calculating the total possibilities for the entire grid initially for each state as well as the counts for the rows and the columns. We can just remove the places it can get from, from the total.
- Now finding the possibilities can be done in constant time
- We now have a complexity of $O(LRC)$ which with the constraints would work.
- Questions?

Unicorn - Reducing time

- By calculating the total possibilities for the entire grid initially for each state as well as the counts for the rows and the columns. We can just remove the places it can get from, from the total.
- Now finding the possibilities can be done in constant time
- We now have a complexity of $O(LRC)$ which with the constraints would work.
- Questions?

Unicorn - Reducing time

- By calculating the total possibilities for the entire grid initially for each state as well as the counts for the rows and the columns. We can just remove the places it can get from, from the total.
- Now finding the possibilities can be done in constant time
- We now have a complexity of $O(LRC)$ which with the constraints would work.
- Questions?

Conclusion

- This lecture isn’t suppose to make you a master at DP.
- But rather give you some new tools in your arsenal for solving DP problems.
- The key to doing well in DP problems is still: Practice

Conclusion

- This lecture isn't suppose to make you a master at DP.
- But rather give you some new tools in your arsenal for solving DP problems.
- The key to doing well in DP problems is still: Practice

Conclusion

- This lecture isn't suppose to make you a master at DP.
- But rather give you some new tools in your arsenal for solving DP problems.
- The key to doing well in DP problems is still: Practice

Conclusion

- This lecture isn't suppose to make you a master at DP.
- But rather give you some new tools in your arsenal for solving DP problems.
- The key to doing well in DP problems is still: Practice