CPSC 490 – Problem Solving in Computer Science

Lecture 7: Bitmask DP, Matrix Exponentiation

Jason Chiu and Raunak Kumar
Based on slides by Paul Liu (2014)

2017/01/20

University of British Columbia

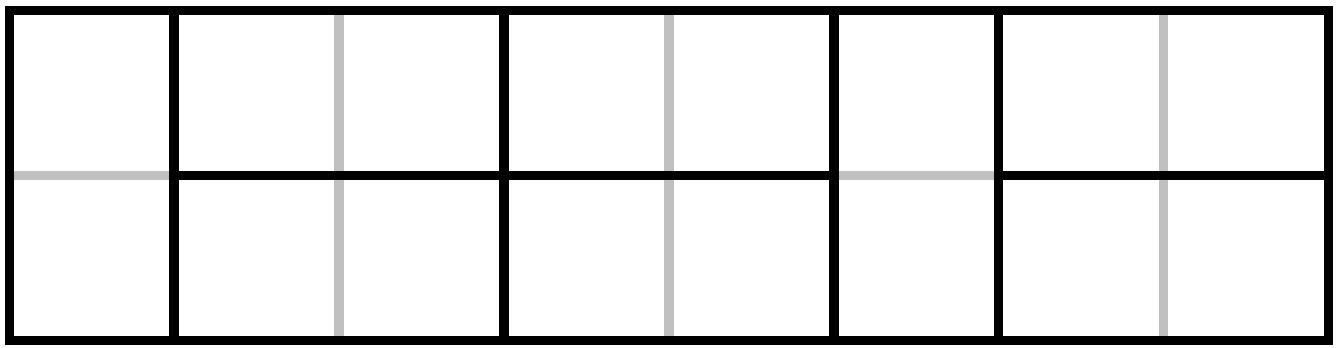
Announcements

Presentations

- List of suggested topics are posted on the website
- You can choose a topic not on a list – just talk to us first.
- Presentation choices due **Monday, January 30th**
- Presentation dates are **February 10th, 13th, 15th 17th**

Remember This Problem?

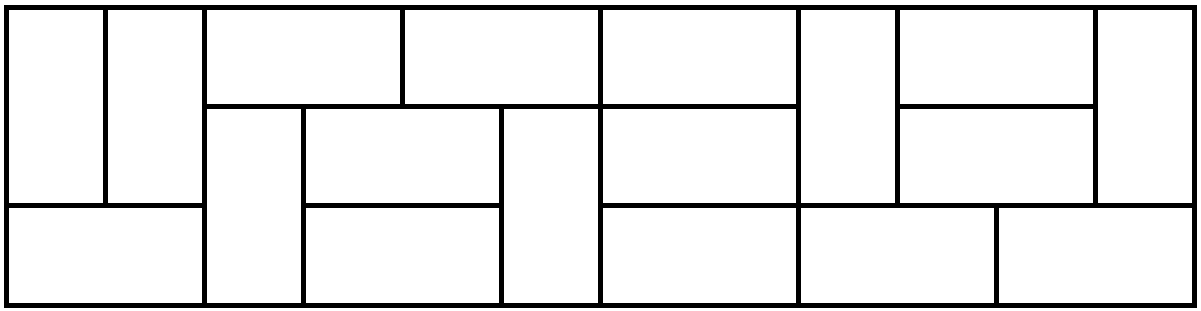
How many ways are there to tile a $2 \times n$ grid with 2×1 tiles?
Constraint: $n \leq 1,000,000$



Solution: $f(0) = f(1) = 1, f(n) = f(n-1) + f(n-2)$

Problem 1 – A Harder Problem

How many ways are there to tile a $3 \times n$ grid with 2×1 tiles?
Constraint: $n \leq 1,000,000$



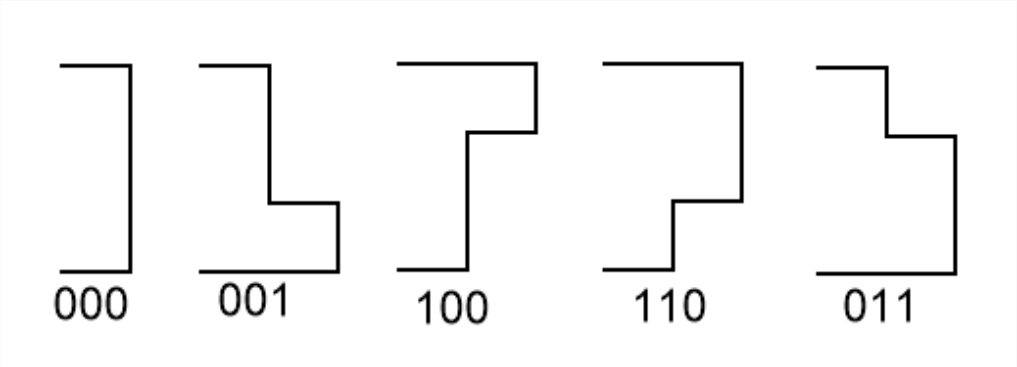
Notice that $f(n) =$ number of ways to fill $3 \times n$ grid does NOT work!

A New Kind of DP State

$f(n) =$ number of ways to fill $3 \times n$ grid does NOT work because after filling a single column, the right most column may not be full!

We need more information: configuration of the right most column!

Represent right most column by a bitmask: 1 = filled, 0 = not filled.



Review of Bitmask and Bit Operations

A bitmask represents a set: i-th bit is 1 if the i-th element is in set.

In C++ and Java, you have bitwise operations on integers:

`^, ~, <<, >>, &, |`

What bitwise operations correspond to these set operations?

- Union of two sets x and y:
- Intersection:
- Symmetric difference:
- A singleton set i:
- Checking if $i \in x$:
- The element in x with the smallest label:

More bit-twiddling hacks:
`https://graphics.stanford.edu/~seander/bithacks.html`

Review of Bitmask and Bit Operations

A bitmask represents a set: i-th bit is 1 if the i-th element is in set.

In C++ and Java, you have bitwise operations on integers:

`^, ~, <<, >>, &, |`

What bitwise operations correspond to these set operations?

- Union of two sets x and y: `x | y`
- Intersection:
- Symmetric difference:
- A singleton set i:
- Checking if $i \in x$:
- The element in x with the smallest label:

More bit-twiddling hacks:
`https://graphics.stanford.edu/~seander/bithacks.html`

Review of Bitmask and Bit Operations

A bitmask represents a set: i-th bit is 1 if the i-th element is in set.

In C++ and Java, you have bitwise operations on integers:

`^`, `~`, `<<`, `>>`, `&`, `|`

What bitwise operations correspond to these set operations?

- Union of two sets x and y: `x | y`
- Intersection: `x & y`
- Symmetric difference:
- A singleton set i:
- Checking if $i \in x$:
- The element in x with the smallest label:

More bit-twiddling hacks:
`https://graphics.stanford.edu/~seander/bithacks.html`

Review of Bitmask and Bit Operations

A bitmask represents a set: i-th bit is 1 if the i-th element is in set.

In C++ and Java, you have bitwise operations on integers:

`^`, `~`, `<<`, `>>`, `&`, `|`

What bitwise operations correspond to these set operations?

- Union of two sets x and y: `x | y`
- Intersection: `x & y`
- Symmetric difference: `x ^ y`
- A singleton set i:
- Checking if $i \in x$:
- The element in x with the smallest label:

More bit-twiddling hacks:
`https://graphics.stanford.edu/~seander/bithacks.html`

Review of Bitmask and Bit Operations

A bitmask represents a set: i-th bit is 1 if the i-th element is in set.

In C++ and Java, you have bitwise operations on integers:

`^`, `~`, `<<`, `>>`, `&`, `|`

What bitwise operations correspond to these set operations?

- Union of two sets x and y: `x | y`
- Intersection: `x & y`
- Symmetric difference: `x ^ y`
- A singleton set i: `(1 << i)`
- Checking if $i \in x$:
- The element in x with the smallest label:

More bit-twiddling hacks:
`https://graphics.stanford.edu/~seander/bithacks.html`

Review of Bitmask and Bit Operations

A bitmask represents a set: i-th bit is 1 if the i-th element is in set.

In C++ and Java, you have bitwise operations on integers:

`^`, `~`, `<<`, `>>`, `&`, `|`

What bitwise operations correspond to these set operations?

- Union of two sets x and y: `x | y`
- Intersection: `x & y`
- Symmetric difference: `x ^ y`
- A singleton set i: `(1 << i)`
- Checking if $i \in x$: `x & (1<<i)`
- The element in x with the smallest label:

More bit-twiddling hacks:
`https://graphics.stanford.edu/~seander/bithacks.html`

Review of Bitmask and Bit Operations

A bitmask represents a set: i-th bit is 1 if the i-th element is in set.

In C++ and Java, you have bitwise operations on integers:

`^`, `~`, `<<`, `>>`, `&`, `|`

What bitwise operations correspond to these set operations?

- Union of two sets x and y: `x | y`
- Intersection: `x & y`
- Symmetric difference: `x ^ y`
- A singleton set i: `(1 << i)`
- Checking if $i \in x$: `x & (1<<i)`
- The element in x with the smallest label: `x & (-x)`

More bit-twiddling hacks:
`https://graphics.stanford.edu/~seander/bithacks.html`

Problem 1 – Solution

DP State: $f(k, C) =$ number of ways to fill the first k columns such that column $k+1$ has configuration equal to bitmask C

Problem 1 – Solution

DP State: $f(k, C) =$ number of ways to fill the first k columns such that column $k+1$ has configuration equal to bitmask C

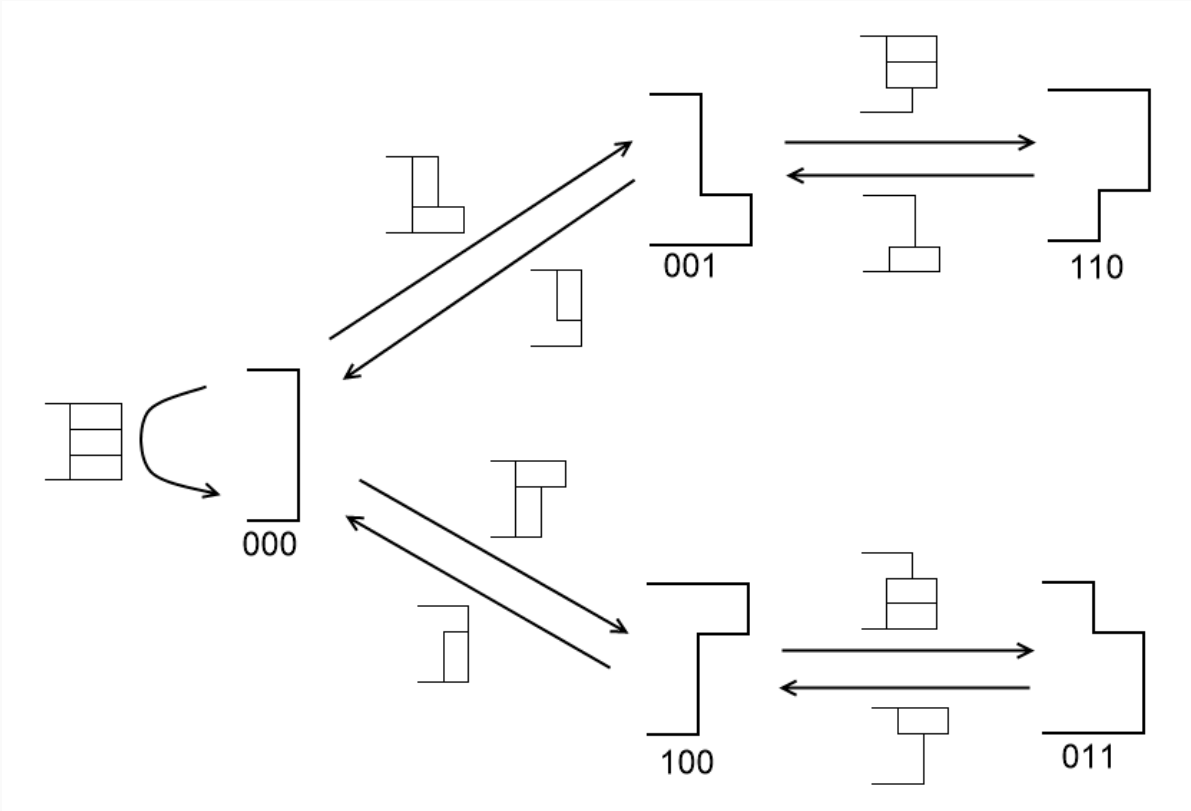
Base case: $f(0, 000) = 1$, $f(0, z) = 0$ for $z \neq 0$

Problem 1 – Solution

DP State: $f(k, C) =$ number of ways to fill the first k columns such that column $k+1$ has configuration equal to bitmask C

Base case: $f(0, 000) = 1$, $f(0, z) = 0$ for $z \neq 0$

Recurrence: try to fill last unfilled column – don't double count!



Problem 1 – Solution

DP State: $f(k, C) =$ number of ways to fill the first k columns such that column $k+1$ has configuration equal to bitmask C

Base case: $f(0, 000) = 1$, $f(0, z) = 0$ for $z \neq 0$

Recurrence: try to fill last unfilled column – don't double count!

$$
\begin{aligned}
f(k, 000) &= f(k-2, 000) + f(k-1, 001) + f(k-1, 100) \\
f(k, 001) &= f(k-1, 000) + f(k-1, 110) \\
f(k, 100) &= f(k-1, 000) + f(k-1, 011) \\
f(k, 110) &= f(k-1, 001) \\
f(k, 011) &= f(k-1, 100)
\end{aligned}
$$

Problem 1 – Solution

DP State: $f(k, C) =$ number of ways to fill the first k columns such that column $k+1$ has configuration equal to bitmask C

Base case: $f(0, 000) = 1$, $f(0, z) = 0$ for $z \neq 0$

Recurrence: try to fill last unfilled column – don't double count!

$$
\begin{aligned}
f(k, 000) &= f(k-2, 000) + f(k-1, 001) + f(k-1, 100) \\
f(k, 001) &= f(k-1, 000) + f(k-1, 110) \\
f(k, 100) &= f(k-1, 000) + f(k-1, 011) \\
f(k, 110) &= f(k-1, 001) \\
f(k, 011) &= f(k-1, 100)
\end{aligned}
$$

Answer: $f(n, 000)$

Problem 1 – Solution

DP State: $f(k, C) =$ number of ways to fill the first k columns such that column $k+1$ has configuration equal to bitmask C

Base case: $f(0, 000) = 1$, $f(0, z) = 0$ for $z \neq 0$

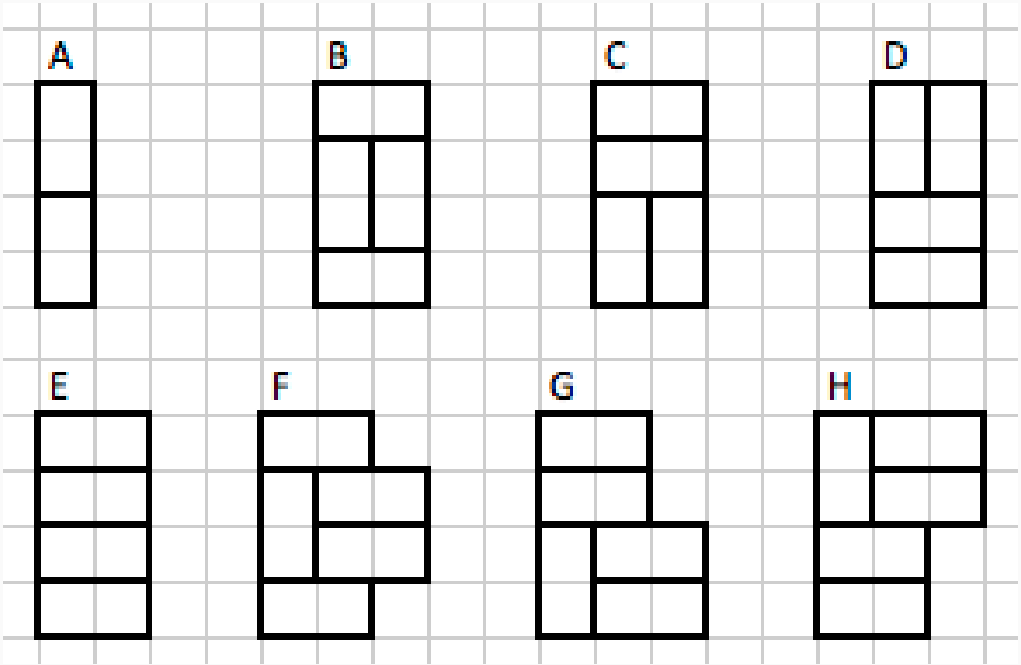
Recurrence: try to fill last unfilled column – don't double count!

$$
\begin{aligned}
f(k, 000) &= f(k-2, 000) + f(k-1, 001) + f(k-1, 100) \\
f(k, 001) &= f(k-1, 000) + f(k-1, 110) \\
f(k, 100) &= f(k-1, 000) + f(k-1, 011) \\
f(k, 110) &= f(k-1, 001) \\
f(k, 011) &= f(k-1, 100)
\end{aligned}
$$

Answer: $f(n, 000)$
Time Complexity: $O(n)$

Problem 2 – An Even Harder Problem

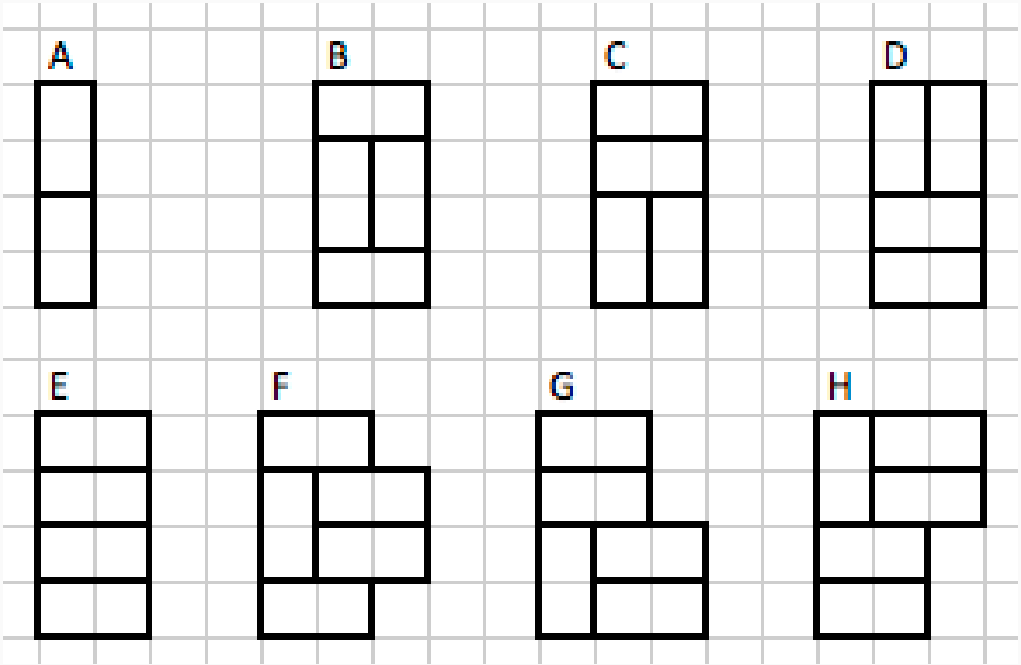
How many ways are there to tile a $4 \times n$ grid with 2×1 tiles?
Constraint: $n \leq 1,000,000$



What about $5 \times n$, $6 \times n$, ..., or in general $k \times n$?

Problem 2 – An Even Harder Problem

How many ways are there to tile a $4 \times n$ grid with 2×1 tiles?
Constraint: $n \leq 1,000,000$



What about $5 \times n$, $6 \times n$, ..., or in general $k \times n$?

Solution: find the recurrence with $O(2^{2k})$ BFS!

Problem 3 – Even Harder?!

How many ways are there to tile a $k \times n$ grid with 2×1 tiles?
Constraints: $2 \leq k \leq 10, 0 \leq n \leq 10^{18}$

Skipping Recurrence Steps Quickly

Consider the case of $k = 2$, i.e. $f_n = f_{n-1} + f_{n-2}$

Skipping Recurrence Steps Quickly

Consider the case of $k = 2$, i.e. $f_n = f_{n-1} + f_{n-2}$

Expand the term f_{n-1} we get "2-step" recurrence

$$\begin{aligned} f_n &= f_{n-2} + f_{n-3} + f_{n-2} \\ &= 2f_{n-2} + f_{n-3} \end{aligned}$$

Skipping Recurrence Steps Quickly

Consider the case of $k = 2$, i.e. $f_n = f_{n-1} + f_{n-2}$

Expand the term f_{n-1} we get "2-step" recurrence

$$\begin{aligned} f_n &= f_{n-2} + f_{n-3} + f_{n-2} \\ &= 2f_{n-2} + f_{n-3} \end{aligned}$$

Now use the above two formulas, we get "4-step" recurrence

$$\begin{aligned} f_n &= 4f_{n-4} + 2f_{n-5} + f_{n-4} + f_{n-5} \\ &= 5f_{n-4} + 3f_{n-5} \end{aligned}$$

Skipping Recurrence Steps Quickly

Consider the case of $k = 2$, i.e. $f_n = f_{n-1} + f_{n-2}$

Expand the term f_{n-1} we get "2-step" recurrence

$$\begin{aligned} f_n &= f_{n-2} + f_{n-3} + f_{n-2} \\ &= 2f_{n-2} + f_{n-3} \end{aligned}$$

Now use the above two formulas, we get "4-step" recurrence

$$\begin{aligned} f_n &= 4f_{n-4} + 2f_{n-5} + f_{n-4} + f_{n-5} \\ &= 5f_{n-4} + 3f_{n-5} \end{aligned}$$

Do this again, we get "8-step" recurrence

$$f_n = 34f_{n-8} + 21f_{n-9}$$

Skipping Recurrence Steps Quickly

Consider the case of $k = 2$, i.e. $f_n = f_{n-1} + f_{n-2}$

Expand the term f_{n-1} we get "2-step" recurrence

$$\begin{aligned} f_n &= f_{n-2} + f_{n-3} + f_{n-2} \\ &= 2f_{n-2} + f_{n-3} \end{aligned}$$

Now use the above two formulas, we get "4-step" recurrence

$$\begin{aligned} f_n &= 4f_{n-4} + 2f_{n-5} + f_{n-4} + f_{n-5} \\ &= 5f_{n-4} + 3f_{n-5} \end{aligned}$$

Do this again, we get "8-step" recurrence

$$f_n = 34f_{n-8} + 21f_{n-9}$$

This seems tedious – can we automate this procedure efficiently?

Matrix Multiplication

Why did it work? The recurrence was linear!

For $k = 2$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_{n-2} \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} f_{n-1} \\ f_n \end{bmatrix}$$

In other words,

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} = \begin{bmatrix} f_{n-1} \\ f_n \end{bmatrix}$$

Matrix Multiplication

$k = 3$ is a bit more complicated...

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_{n-2,0} \\ f_{n-1,0} \\ f_{n-1,1} \\ f_{n-1,4} \\ f_{n-1,6} \\ f_{n-1,5} \end{bmatrix} = \begin{bmatrix} f_{n-1,0} \\ f_{n,0} \\ f_{n,1} \\ f_{n,4} \\ f_{n,6} \\ f_{n,5} \end{bmatrix}$$

Fast Matrix Exponentiation

All we need now is a way to get M^n quickly – in $O(\log n)$ time!

Fast Matrix Exponentiation

All we need now is a way to get M^n quickly – in $O(\log n)$ time!

Hmm... if we know M, then we can square it to get M^2, but squaring that again gives M^4, and then M^8, and so on...

Fast Matrix Exponentiation

All we need now is a way to get M^n quickly – in $O(\log n)$ time!

Hmm... if we know M, then we can square it to get M^2, but squaring that again gives M^4, and then M^8, and so on...

$\rightarrow$ We can compute M^{2^k} with $O(k)$ matrix multiplications

Fast Matrix Exponentiation

All we need now is a way to get M^n quickly – in $O(\log n)$ time!

Hmm... if we know M, then we can square it to get M^2, but squaring that again gives M^4, and then M^8, and so on...

$\rightarrow$ We can compute M^{2^k} with $O(k)$ matrix multiplications

Matrix multiplication is associative, so we can break up the exponent into sum of powers of two by using its base-2 representation:

$$M^{30} = M^{11110_2} = M^{16} M^8 M^4 M^2$$

Fast Matrix Exponentiation

All we need now is a way to get M^n quickly – in $O(\log n)$ time!

Hmm... if we know M, then we can square it to get M^2, but squaring that again gives M^4, and then M^8, and so on...

$\rightarrow$ We can compute M^{2^k} with $O(k)$ matrix multiplications

Matrix multiplication is associative, so we can break up the exponent into sum of powers of two by using its base-2 representation:

$$M^{30} = M^{11110_2} = M^{16}M^8M^4M^2$$

$\rightarrow$ compute M^{2^k} for $0 \leq k \leq \log n$, and then multiply the appropriate ones together in a total of $O(\log n)$ multiplications!

Fast Matrix Exponentiation

```
1 FAST_POW(M, n):
2   initialize A = Identity matrix
3   for k from 0 to log_2(n):
4     if n AND 2^k != 0:
5       A = A * M
6     M = M * M
7   return A
```

Traveling Salesman

Problem 4 – Traveling Salesman

Given undirected weighted graph of $n \leq 16$ vertices, find a minimum cost path starting at vertex 0 that goes though all vertices!

Problem 4 – Traveling Salesman Solution

Intuition: to figure out where to go next, we need to know where we went, and where we are.

Problem 4 – Traveling Salesman Solution

Intuition: to figure out where to go next, we need to know where we went, and where we are.

DP State: $f(S, k) =$ minimum length path that goes through all vertices in S once and ends at vertex k

Problem 4 – Traveling Salesman Solution

Intuition: to figure out where to go next, we need to know where we went, and where we are.

DP State: $f(S, k) =$ minimum length path that goes through all vertices in S once and ends at vertex k

Base case: $f(S, k) = 0$ if S has exactly one vertex, k, otherwise ∞

Problem 4 – Traveling Salesman Solution

Intuition: to figure out where to go next, we need to know where we went, and where we are.

DP State: $f(S, k) =$ minimum length path that goes through all vertices in S once and ends at vertex k

Base case: $f(S, k) = 0$ if S has exactly one vertex, k, otherwise ∞

Recurrence: $f(S, k) = \min_{i \in S, i \neq k} f(S - k, i) + d(i, k)$

Problem 4 – Traveling Salesman Solution

Intuition: to figure out where to go next, we need to know where we went, and where we are.

DP State: $f(S, k) =$ minimum length path that goes through all vertices in S once and ends at vertex k

Base case: $f(S, k) = 0$ if S has exactly one vertex, k, otherwise ∞

Recurrence: $f(S, k) = \min_{i \in S, i \neq k} f(S - k, i) + d(i, k)$

Answer: $f(\{0, \ldots, n-1\}, 0)$

Problem 4 – Traveling Salesman Solution

Intuition: to figure out where to go next, we need to know where we went, and where we are.

DP State: $f(S, k) =$ minimum length path that goes through all vertices in S once and ends at vertex k

Base case: $f(S, k) = 0$ if S has exactly one vertex, k, otherwise ∞

Recurrence: $f(S, k) = \min_{i \in S, i \neq k} f(S - k, i) + d(i, k)$

Answer: $f(\{0, \ldots, n-1\}, 0)$

Time complexity: $O(n^2 2^n)$

Advanced DP Optimization

Other DP Optimization Methods

There are various other ways to optimize DP when it has specific properties, the most useful ones being:

- Convex hull optimization
- Divide and conquer optimization
- Knuth optimization

For details see: `http://codeforces.com/blog/entry/8219`

End of material for Assignment 2

Next Class

DP on Trees and Graphs