Edit distance and matrix chain multiplication

I. An algorithmic metric on strings

Version control

I. An algorithmic metric on strings

2 homework/homework-6.md

```
@@ -7,6 +7,6 @@
7      2. Design and analyze an algorithm[^2] to compute the even distance between every pair of vertices when the edge weights are real-valued, and there are no negative cycles.
8      2. Design and analyze an algorithm[^2] that, given a directed graph with real-valued edges, detects if there is a negative cycle.
9
10   - [^1]: We pushed back the deadline to Friday because the homework was also prepared late.
11     [^2]: As always, the faster the better.
12
```

```
@@ -7,6 +7,6 @@
7      2. Design and analyze an algorithm[^2] to compute the even distance between every pair of vertices when the edge weights are real-valued, and there are no negative cycles.
8      2. Design and analyze an algorithm[^2] that, given a directed graph with real-valued edges, detects if there is a negative cycle.
9
10   + [^1]: We pushed back the deadline to Friday because we were late in getting the homework up.
11     [^2]: As always, the faster the better.
12
```

Github highlights changes

I. An algorithmic metric on strings

2 homework/homework-6.md

@@ -7,6 +7,6 @@

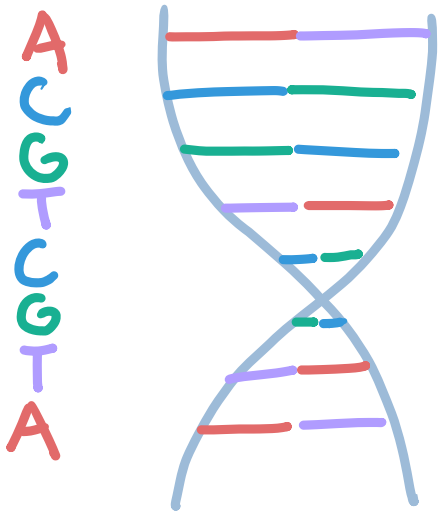
```
     2. Design and analyze an algorithm[^2] to compute the even distance between every pair of vertices when the edge weights are real-valued, and there are no negative cycles.
     2. Design and analyze an algorithm[^2] that, given a directed graph with real-valued edges, detects if there is a negative cycle.

-[^1]: We pushed back the deadline to Friday because the homework was also prepared late.
+[^1]: We pushed back the deadline to Friday because we were late in getting the homework up.
     [^2]: As always, the faster the better.
```

```
DIFF(1)                          User Commands                          DIFF(1)

NAME
       diff - compare files line by line

SYNOPSIS
       diff [OPTION]... FILES

DESCRIPTION
       Compare files line by line.

       -i  --ignore-case
              Ignore case differences in file contents.

       --ignore-file-name-case
              Ignore case when comparing file names.

       --no-ignore-file-name-case
              Consider case when comparing file names.

       -E  --ignore-tab-expansion
              Ignore changes due to tab expansion.

       -b  --ignore-space-change
              Ignore changes in the amount of white space.

       -w  --ignore-all-space
              Ignore all white space.

       -B  --ignore-blank-lines
              Ignore changes whose lines are all blank.
```

Genetics

I. An algorithmic metric on strings

A, C, G, T

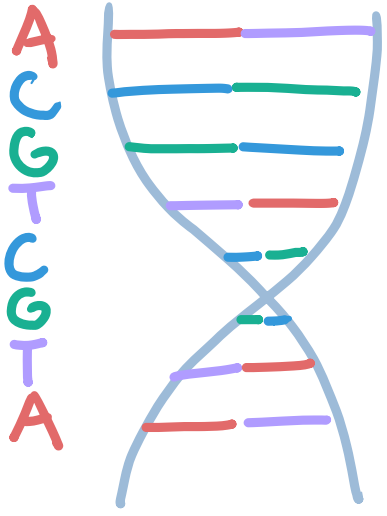


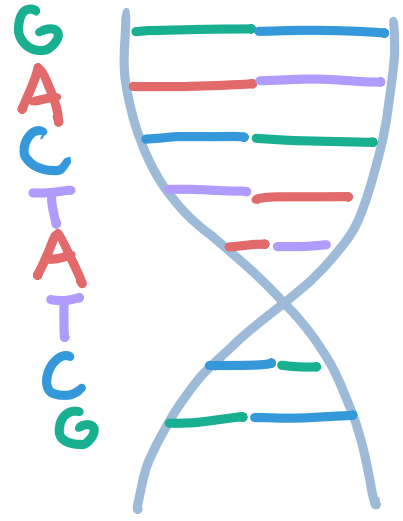
how to measure similarity?

important for:

(a) analyzing hereditary traits

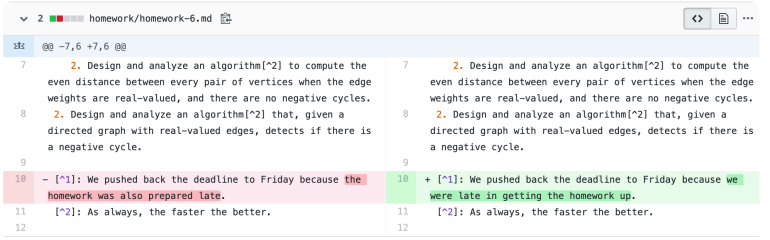
(b) distinguishing species

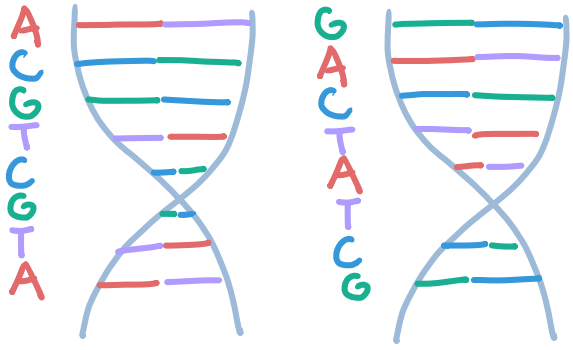




2 metrics for strings:

1. Hamming distance





for strings x,y w/ same length,

$$\text{Hamming}(x,y) = \left(\begin{array}{c}\#\text{ characters} \\ \text{differing in } x \text{ and } y\end{array}\right)$$

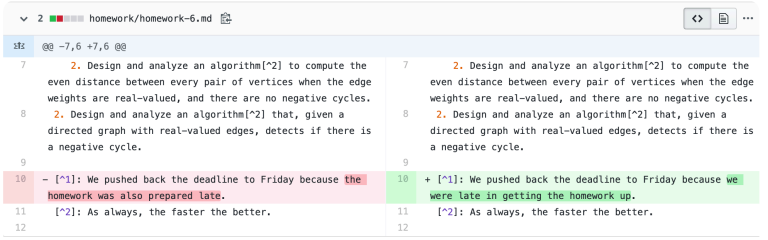
x= ABCDEFG

x x x x

y= ABCEFGD

=> Hamming dist. 4

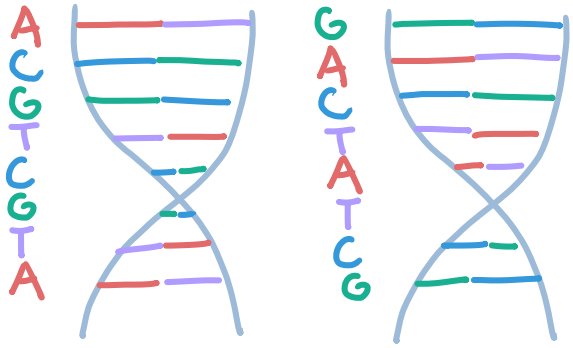
2 metrics for strings:



1. Hamming distance

$$\text{Hamming}(x,y) = \begin{pmatrix} \#\text{ characters} \\ \text{differing in } x \text{ and } y \end{pmatrix}$$

for strings x, y w/ same length



$x =$ ABCDEFG

$y =$ ABCEFGD (x x x x)

=> Hamming dist. 4

2. Edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

$x =$ ABC(del)DEFG

$y =$ ABCEFGD(ins)

=> edit distance 2

Hamming distance and edit distance are both metrics:

Hamming distance

characters differing in x and y

Edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

- $dist(x,y) \geq 0$
- $dist(x,y) = 0 \iff x = y$
- $dist(x,y) = dist(y,x)$
- $dist(x,y) \leq dist(x,z) + dist(z,y)$ "triangle inequality"

1. Hamming distance

characters differing in x and y

con: small misalignment ⇒ huge Hamming dist

pro: easy, fast to compute

2. Edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

pro: small misalignment ⇒ small edit distance
(better model for many applications)

cons: more complicated, how to compute?

Computing edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

Designing a recursive algorithm

1. Recursive specification
   - declares input/output of algo
   - "contract" algo must fulfill
   - induction hypothesis for correctness
2. Implement spec
3. Prove correctness
   argue algo satisfies recursive spec for all n by induction on n

1. Recursive specification

`edit`$(x[1..m], y[1..n])$: Given two strings x and y, returns the minimum number of edit operations needed to convert x into y.

Computing edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

Designing a recursive algorithm
1. Recursive specification
   - declares input/output of algo
   - "contract" algo must fulfill
   - induction hypothesis for correctness
2. Implement spec
3. Prove correctness
   argue algo satisfies recursive spec for all n by induction on n

① Recursive specification

$\texttt{edit}(x[1..m], y[1..n])$: Given two strings x and y, returns the minimum number of edit operations needed to convert x into y.

② Implement spec

$\texttt{edit}(x[1..k], y[1..\ell])$

1. If $k = 0$ or $\ell = 0$ then return $k + \ell$.
2. If $x[1] \neq y[1]$ then return the minimum of
   A. $1 + \texttt{edit}(x[2..k], y[1..\ell])$ // *delete* $x[1]$
   B. $1 + \texttt{edit}(x[1..k], y[2..\ell])$ // *insert* $y[1]$
   C. $1 + \texttt{edit}(x[2..k], y[2..\ell])$ // *replace* $x[1]$ *with* $y[1]$
3. If $x[1] = y[1]$ then return the minimum of
   A. $\texttt{edit}(x[2..k], y[2..\ell])$ // *do nothing*
   B. $1 + \texttt{edit}(x[1..k], y[2..\ell])$ // *insert* $y[1]$
   C. $1 + \texttt{edit}(x[2..k], y[1..\ell])$ // *delete* $x[1]$

Computing edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

Designing a recursive algorithm
1. Recursive specification
   - declares input/output of algo
   - "contract" algo must fulfill
   - induction hypothesis for correctness
2. Implement spec
3. Prove correctness
   argue algo satisfies recursive spec for all n by induction on n

1. Recursive specification

$\texttt{edit}(x[1..m], y[1..n])$: Given two strings x and y, returns the minimum number of edit operations needed to convert x into y.

2. Implement spec

$\texttt{edit}(x[1..k], y[1..\ell])$

1. If $k = 0$ or $\ell = 0$ then return $k + \ell$.
2. If $x[1] \neq y[1]$ then return the minimum of
   A. $1 + \texttt{edit}(x[2..k], y[1..\ell])$ // *delete* $x[1]$
   B. $1 + \texttt{edit}(x[1..k], y[2..\ell])$ // *insert* $y[1]$
   C. $1 + \texttt{edit}(x[2..k], y[2..\ell])$ // *replace* $x[1]$ *with* $y[1]$
3. If $x[1] = y[1]$ then return the minimum of
   A. $\texttt{edit}(x[2..k], y[2..\ell])$ // *do nothing*
   B. $1 + \texttt{edit}(x[1..k], y[2..\ell])$ // *insert* $y[1]$
   C. $1 + \texttt{edit}(x[2..k], y[1..\ell])$ // *delete* $x[1]$

Running time: $T(n) = 3T(n-1) + 1 = O(3^n)$ 

Computing edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

Designing a recursive algorithm
1. Recursive specification
   - declares input/output of algo
   - "contract" algo must fulfill
   - induction hypothesis for correctness
2. Implement spec
3. Prove correctness
   argue algo satisfies recursive spec for all n by induction on n

① Recursive specification

$\texttt{edit}(x[1..m], y[1..n])$: Given two strings x and y, returns the minimum number of edit operations needed to convert x into y.

② Implement spec

$\texttt{edit}(x[1..k], y[1..\ell])$

1. If $k = 0$ or $\ell = 0$ then return $k + \ell$.
2. If $x[1] \neq y[1]$ then return the minimum of
   A. $1 + \texttt{edit}(x[2..k], y[1..\ell])$
   B. $1 + \texttt{edit}(x[1..k], y[2..\ell])$
   C. $1 + \texttt{edit}(x[2..k], y[2..\ell])$
3. If $x[1] = y[1]$ then return the minimum of
   A. $\texttt{edit}(x[2..k], y[2..\ell])$
   B. $1 + \texttt{edit}(x[1..k], y[2..\ell])$
   C. $1 + \texttt{edit}(x[2..k], y[1..\ell])$

$T(n) = 3T(n-1) + 1 = O(3^n)$

Summary

edit distance is good model

can be computed recursively

recursive algo is exponential time

Q: How to make recursive algo fast?

caching!

Caching edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

Designing a recursive algorithm
1. Recursive specification
   - declares input/output of algo
   - "contract" algo must fulfill
   - induction hypothesis for correctness
2. Implement spec
3. Prove correctness
   argue algo satisfies recursive spec for all n by induction on n

Idea:

- save answers to each subproblem
- never solve same subproblem twice.

1. index parameters to be more cache-friendly

① Recursive specification let x[1..m], y[1..n] be fixed.

$\texttt{edit}(i, j)$ = the minimum number of edit operations to convert $x[i..m]$ into $y[j..n]$.

② Implement spec

$\texttt{edit}(i, j)$

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $x[1] \neq y[1]$ then return $\min\{1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1), 1 + \texttt{edit}(i+1, j+1)\}$.
5. Otherwise return $\min\{\texttt{edit}(i+1, j+1), 1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1)\}$.

Caching edit distance

min # single character insertions, deletions, and substitutions to convert x into y.

Idea:

- save answers to each subproblem
- never solve same subproblem twice.

Designing a recursive algorithm
1. Recursive specification
   - declares input/output of algo
   - "contract" algo must fulfill
   - induction hypothesis for correctness
2. Implement spec
3. Prove correctness
   argue algo satisfies recursive spec for all n by induction on n

2. augment recursive code w/ caching

1. index parameters to be more cache-friendly

1. Recursive specification let $x[1..m]$, $y[1..n]$ be fixed.
   $\texttt{edit}(i,j)$ = the minimum number of edit operations to convert $x[i..m]$ into $y[j..n]$.
2. Implement spec

$\underline{\texttt{cached-edit}(i,j)}$

/ Implements the same recursive spec as* $\texttt{edit}(i,j)$*, but assumes there is an* $m \times n$ *array* $A[1..m, 1..n]$ *("A" for "Answer") allocated in global memory, with all entries initially set to* `null`*.* */

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $A[i,j]$ is `null`:
   A. If $x[1] \neq y[1]$ then set
   $$A[i,j] = \min\{1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1), 1 + \texttt{edit}(i+1, j+1)\}.$$
   B. Otherwise set
   $$A[i,j] = \min\{\texttt{edit}(i+1, j+1), 1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1)\}.$$
5. Return $A[i,j]$.

$\underline{\texttt{edit}(i,j)}$

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $x[1] \neq y[1]$ then return $\min\{1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1), 1 + \texttt{edit}(i+1, j+1)\}$.
5. Otherwise return $\min\{\texttt{edit}(i+1, j+1), 1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1)\}$.

A[1..m,1..n] is a 2-dim array in global memory

Caching edit distance

- save answers to each subproblem
- never solve same subproblem twice.

min # single character insertions, deletions, and substitutions to convert x into y.

Designing a recursive algorithm

① Recursive specification
- declares input/output of algo
- "contract" algo must fulfill
- induction hypothesis for correctness

② Implement spec

③ Prove correctness
argue algo satisfies recursive spec for all n by induction on n

2. augment recursive code w/ caching

cached-edit(i,j)

/* *Implements the same recursive spec as* edit(i,j)*, but assumes there is an* $m \times n$ *array* $A[1..m, 1..n]$ *("A" for "Answer") allocated in global memory, with all entries initially set to* null. */

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $A[i,j]$ is null:
   A. If $x[1] \neq y[1]$ then set
   $$A[i,j] = \min\{1 + \texttt{edit}(i+1,j), 1 + \texttt{edit}(i,j+1), 1 + \texttt{edit}(i+1,j+1)\}.$$
   B. Otherwise set
   $$A[i,j] = \min\{\texttt{edit}(i+1,j+1), 1 + \texttt{edit}(i+1,j), 1 + \texttt{edit}(i,j+1)\}.$$
5. Return $A[i,j]$.

1. index parameters to be more cache-friendly

① Recursive specification let x[1..m], y[1..n] be fixed.
edit(i,j) = the minimum number of edit operations to convert $x[i..m]$ into $y[j..n]$.

② Implement spec

edit(i,j)
1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $x[1] \neq y[1]$ then return $\min\{1 + \texttt{edit}(i+1,j), 1 + \texttt{edit}(i,j+1), 1 + \texttt{edit}(i+1,j+1)\}$.
5. Otherwise return $\min\{\texttt{edit}(i+1,j+1), 1 + \texttt{edit}(i+1,j), 1 + \texttt{edit}(i,j+1)\}$.

A[1..m,1..n] is a 2-dim array in global memory

cached-edit(i,j)

/* *Implements the same recursive spec as* edit(i,j)*, but assumes there is an* $m \times n$ *array* $A[1..m, 1..n]$ *("A" for "Answer") allocated in global memory, with all entries initially set to* null. */

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $A[i,j]$ is null:
   A. If $x[1] \neq y[1]$ then set

$$A[i,j] = \min\{1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1), 1 + \texttt{edit}(i+1, j+1)\}.$$

   B. Otherwise set

$$A[i,j] = \min\{\texttt{edit}(i+1, j+1), 1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1)\}.$$

5. Return $A[i,j]$.

Running time

mn subproblems

$O(1)$ time per subproblem
(excl. recursive calls)

$\times$

$O(mn)$ time total

Caching reduced running time exponentially!

Caching, a.k.a. dynamic programming:

If recursive algo has polynomial # of subproblems, then

caching => polynomial time algo

cached-edit(i, j)

/* *Implements the same recursive spec as* edit(i, j)*, but assumes there is an* $m \times n$ *array* $A[1..m, 1..n]$ *("A" for "Answer") allocated in global memory, with all entries initially set to* null. */

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $A[i, j]$ is null:
   A. If $x[1] \neq y[1]$ then set

$$A[i, j] = \min\{1 + \texttt{edit}(i + 1, j), 1 + \texttt{edit}(i, j + 1), 1 + \texttt{edit}(i + 1, j + 1)\}.$$

   B. Otherwise set

$$A[i, j] = \min\{\texttt{edit}(i + 1, j + 1), 1 + \texttt{edit}(i + 1, j), 1 + \texttt{edit}(i, j + 1)\}.$$

5. Return $A[i, j]$.

Running time

mn subproblems

$O(1)$ time per subproblem (excl. recursive calls)

$\times$

$O(mn)$ time total

template for this class:

1. recursive spec
2. recursive implementation
3. mention "caching" or "dynamic programming"
4. analyze run time
5. how to use recursive algo to solve problem
6. proof (by induction)

See notes for full explanation & sample solutions

template for this class:

1. recursive spec
2. recursive implementation
3. mention "caching" or "dynamic programming"
4. analyze run time
5. how to use recursive algo to solve problem
6. proof (by induction)

`cached-edit(i, j)`

/* *Implements the same recursive spec as* `edit(i, j)`*, but assumes there is an* $m \times n$ *array* $A[1..m, 1..n]$ *("A" for "Answer") allocated in global memory, with all entries initially set to* `null`. */

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $A[i, j]$ is `null`:
   A. If $x[1] \neq y[1]$ then set

$$A[i, j] = \min\{1 + \texttt{edit}(i + 1, j), 1 + \texttt{edit}(i, j + 1), 1 + \texttt{edit}(i + 1, j + 1)\}.$$

   B. Otherwise set

$$A[i, j] = \min\{\texttt{edit}(i + 1, j + 1), 1 + \texttt{edit}(i + 1, j), 1 + \texttt{edit}(i, j + 1)\}.$$

5. Return $A[i, j]$.

Running time

mn subproblems

$O(1)$ time per subproblem (excl. recursive calls)

$\times$

$O(mn)$ time total

We don't want:

- code implementing caching (which is mechanical)
- iterative code (w/ loops)
- space usage

Sample solution.

Problem. Design and analyze an algorithm that, given two strings $x[1..m]$ and $y[1..n]$, computes the edit distance from x to y.

Solution.

1. Recursive spec / induction hypothesis. Let $x[1..m]$ and $y[1..n]$ be two strings. We define

$$\texttt{edit}(i,j) = \text{the minimum number of edit operations to convert } x[i..m] \text{ into } y[j..n].$$

2. Recursive implementation.

`edit(`i`,`j`)`

1. If $i > m$ and $j > n$ then return 0.
2. If $i > m$ then return $n - j$.
3. If $j > n$ then return $m - i$.
4. If $x[1] \neq y[1]$ then return $\min\{1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1), 1 + \texttt{edit}(i+1, j+1)\}$.
5. Otherwise return $\min\{\texttt{edit}(i+1, j+1), 1 + \texttt{edit}(i+1, j), 1 + \texttt{edit}(i, j+1)\}$.

3. Solving the original problem. $\texttt{edit}(1,1)$ gives the edit distance between $x[1..m]$ and $y[1..n]$.

4. Running time with caching / dynamic programming. We apply dynamic programming to the recursive algorithm above. We have $O(mn)$ subproblems, each of which take $O(1)$ time excluding recursive subcalls. Thus the total running time is bounded by

$$(\#\text{ subproblems}) \begin{pmatrix} \text{time per} \\ \text{subproblem} \\ \text{excl. recursive calls} \end{pmatrix} \leq O(mn).$$

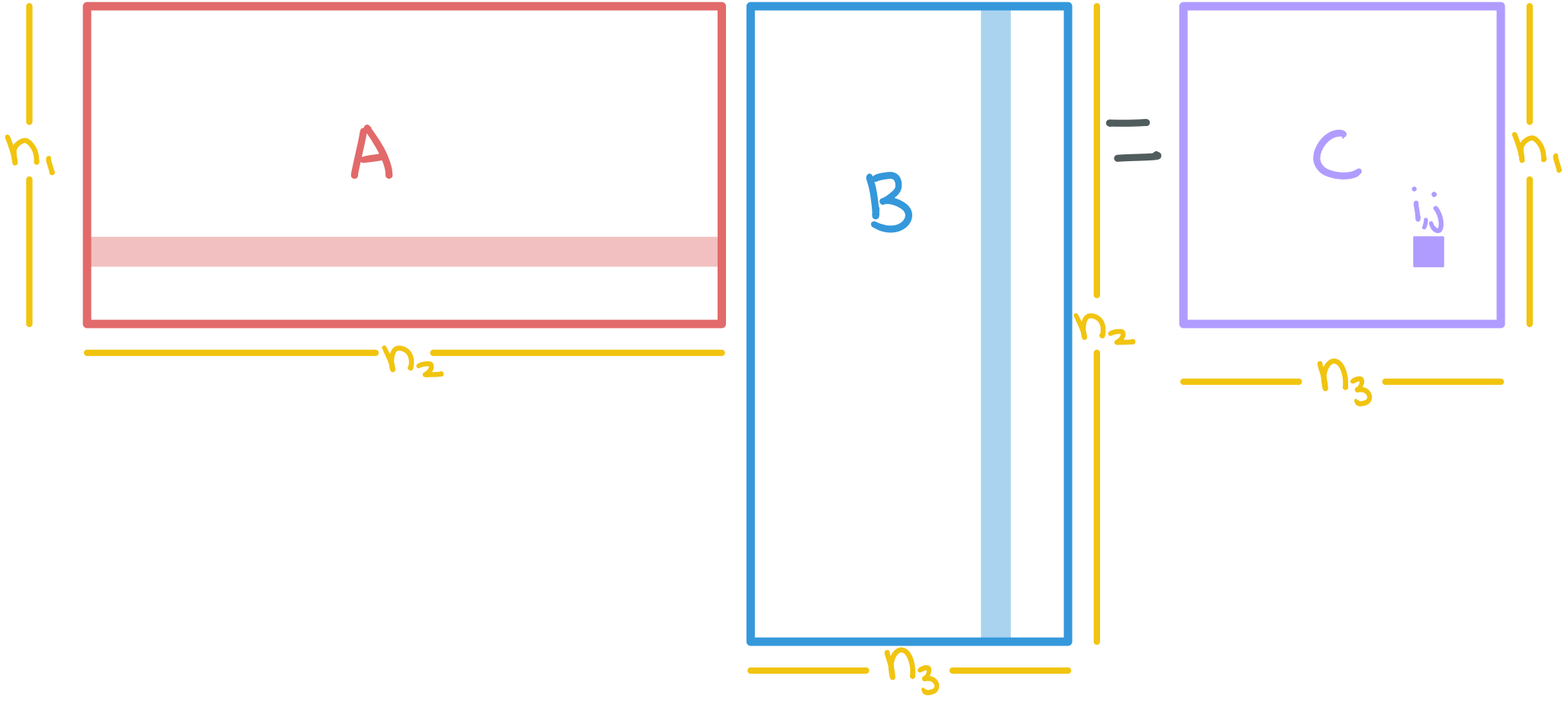
5. Proof. We prove that `edit(`i`,`j`)` satisfies the recursive spec for all $i \in \{1, \ldots, m+1\}$ and $j \in \{1, \ldots, n+1\}$, by induction on $\min\{m - i, n - j\}$.

It is easy to see that `edit(`i`,`j`)` returns the length of *some* edit sequence from $x[1..m]$ to

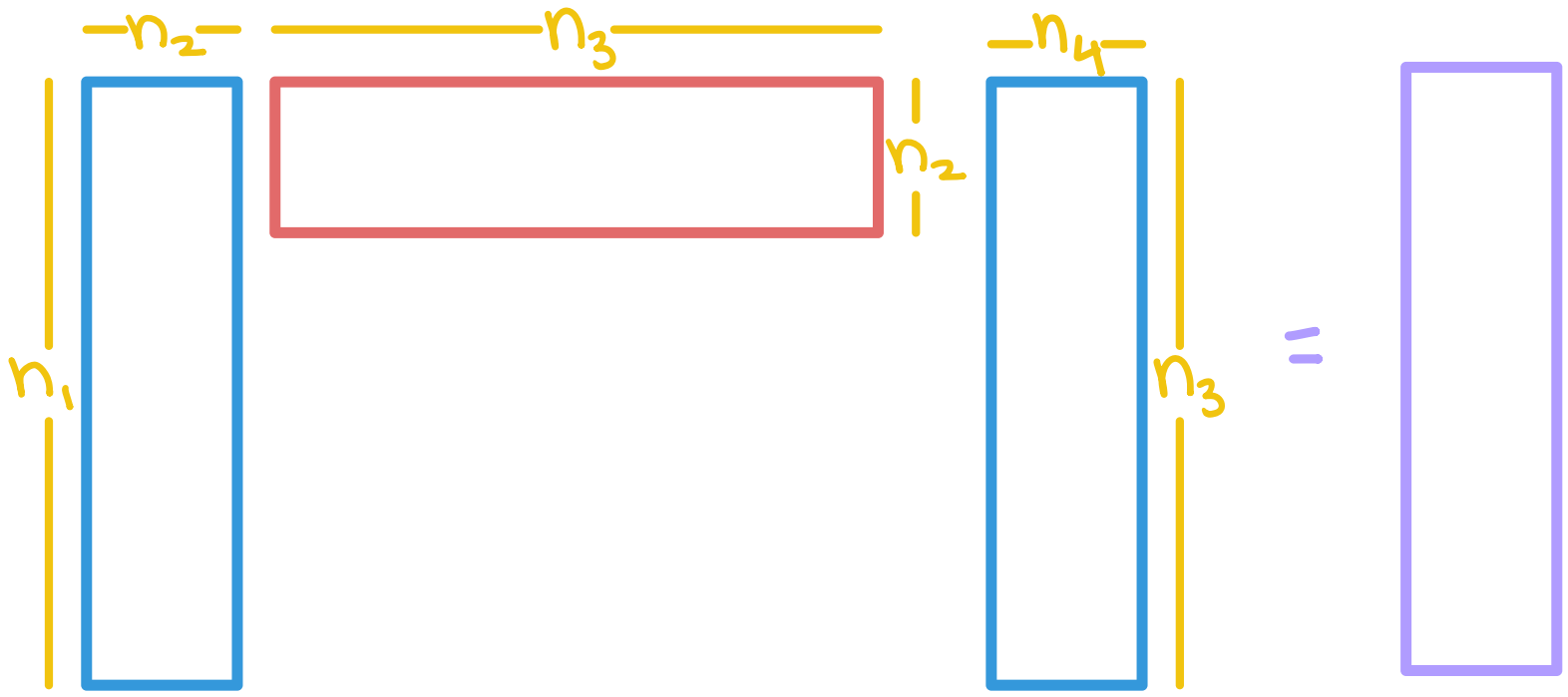
template for this class:
1. recursive spec
2. recursive implementation
3. mention "caching" or "dynamic programming"
4. analyze run time
5. how to use recursive algo to solve problem
6. proof (by induction)

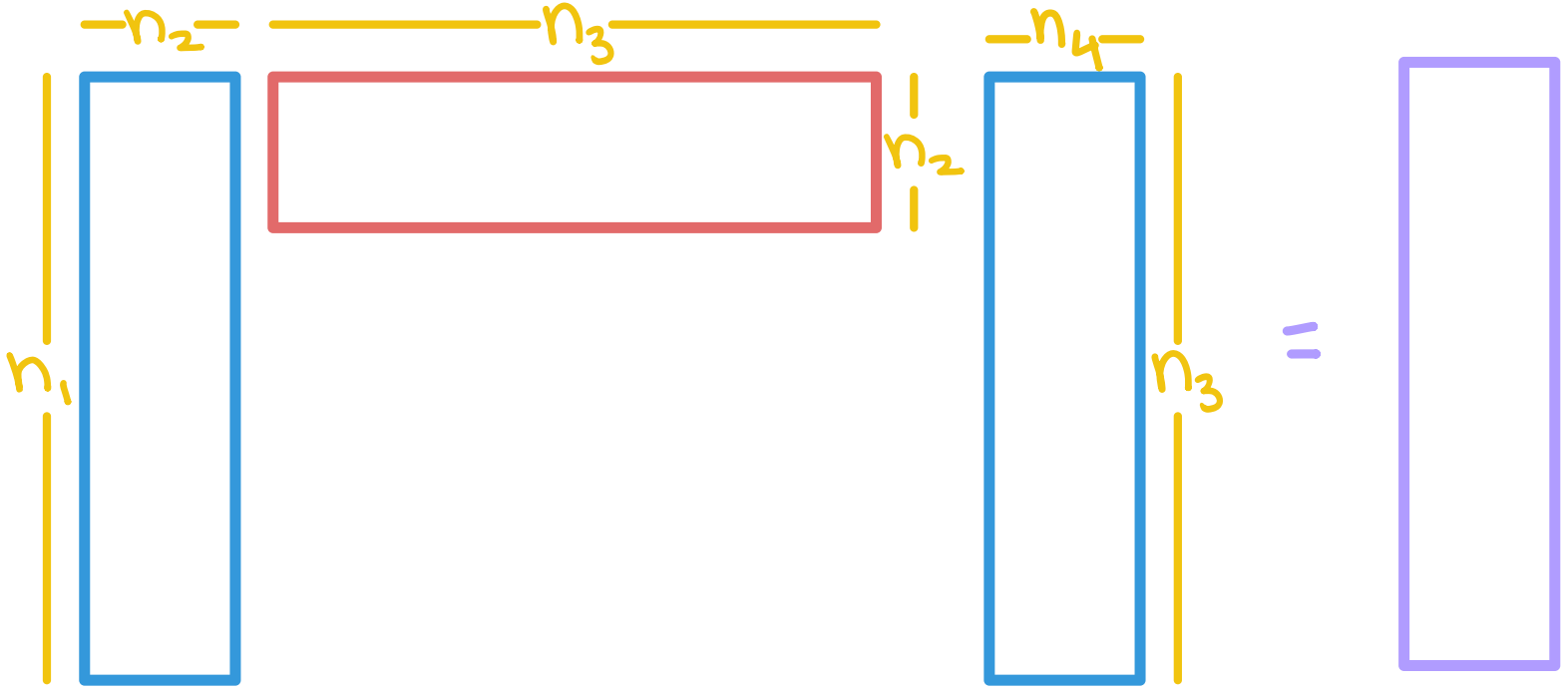
II Matrix chain multiplication

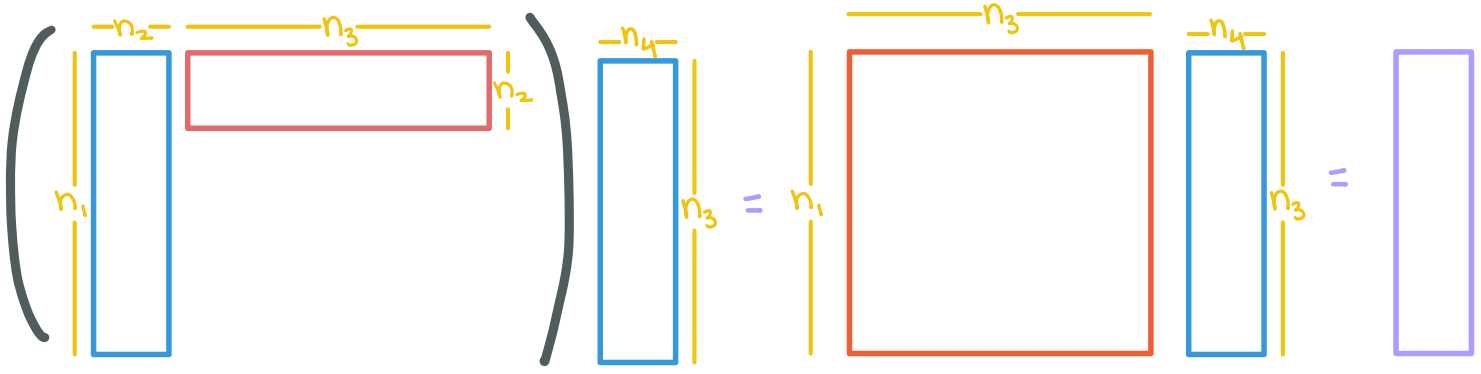
$$C_{ij} = \sum_{k=1}^{n_2} A_{ik} B_{kj}$$



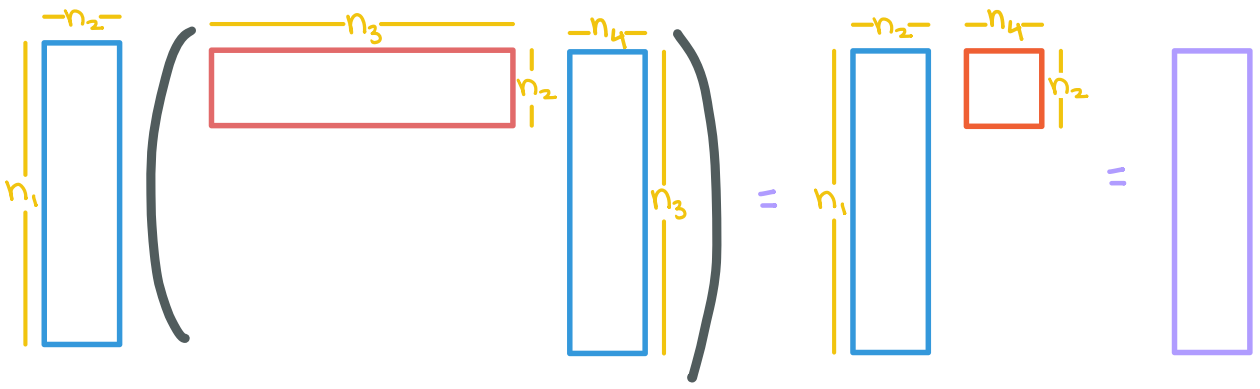
$O(n_1 n_2 n_3)$ time to compute C


n_2
n_3
n_4
n_2
n_1
n_3
=



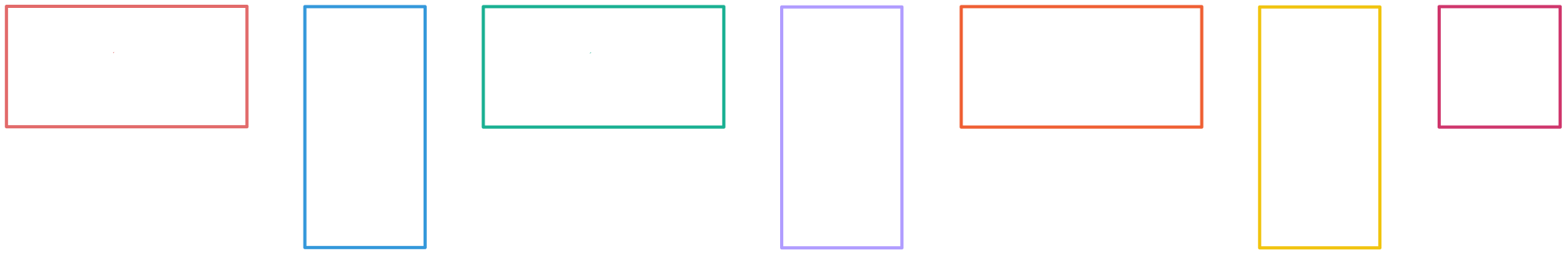


$$O(n_1n_2n_3 + n_1n_3n_4)$$
$$= O(n_1n_3(n_2+n_4))$$



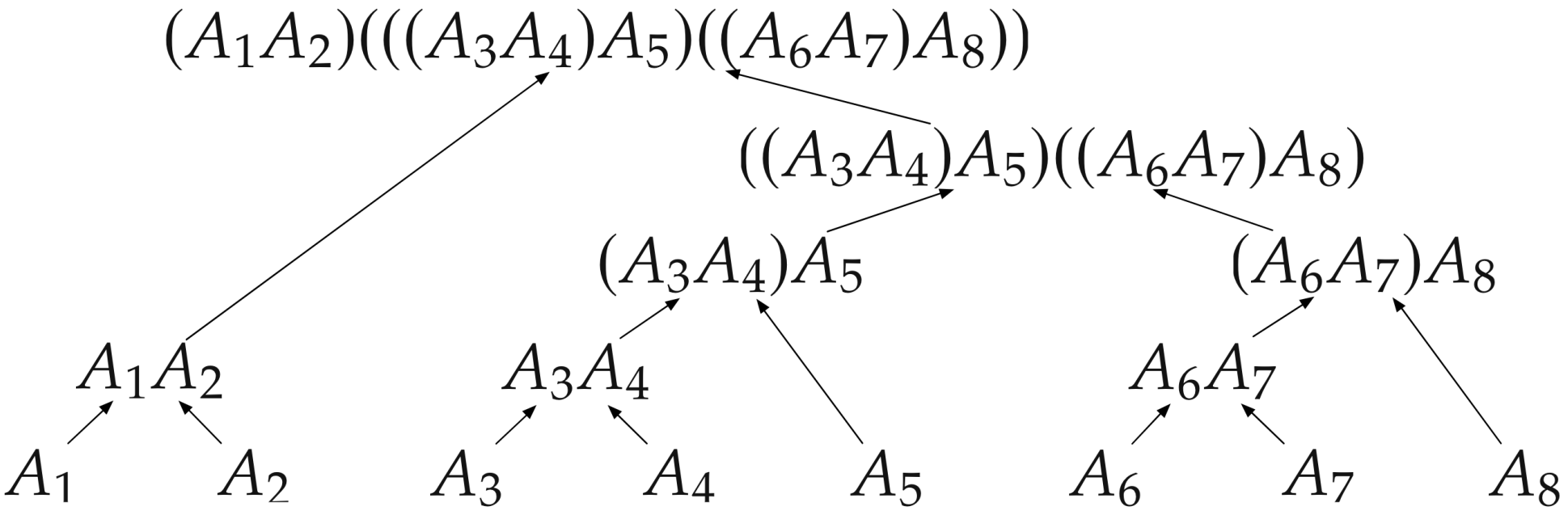
$$O(n_2n_3n_4 + n_1n_2n_4)$$
$$= O(n_2n_4(n_1+n_3)$$

Input: dimensions $n_1, n_2, \sim, n_k, n_{k+1} \in \mathbb{N}$



Output: min # (ordinary) multiplications needed to compute $A_1 A_2 A_3 \sim A_k$, where $A_i \in \mathbb{R}^{n_i \times n_{i+1}}$

Input: dimensions $n_1, n_2, \ldots, n_k, n_{k+1} \in \mathbb{N}$

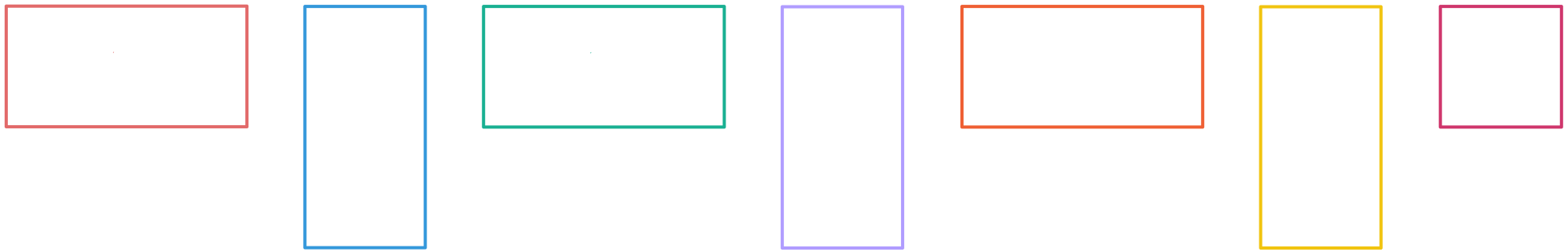


Output: min # (ordinary) multiplications needed to compute $A_1 A_2 A_3 \cdots A_k$, where $A_i \in \mathbb{R}^{n_i \times n_{i+1}}$

Input: dimensions $n_1, n_2, \sim, n_k, n_{k+1} \in \mathbb{N}$

Output: min # (ordinary) multiplications needed to compute

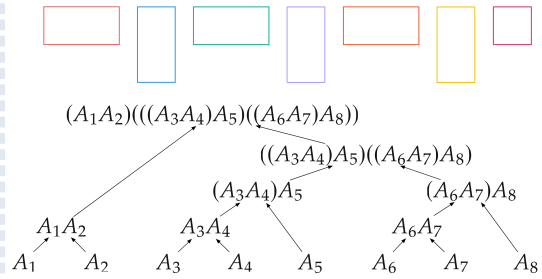
$A_1 A_2 A_3 \sim A_k$, where $A_i \in \mathbb{R}^{n_i \times n_{i+1}}$



Ideas?

Input: dimensions $n_1, n_2, \ldots, n_k, n_{k+1} \in \mathbb{N}$

Output: min # (ordinary) multiplications needed to compute $A_1 A_2 A_3 \cdots A_k$, where $A_i \in \mathbb{R}^{n_i \times n_{i+1}}$



1. recursive spec

$\texttt{Fastest}(i, j)$ = the minimum number of operations to compute the matrix product $A_i \ldots A_j$.

template for this class:

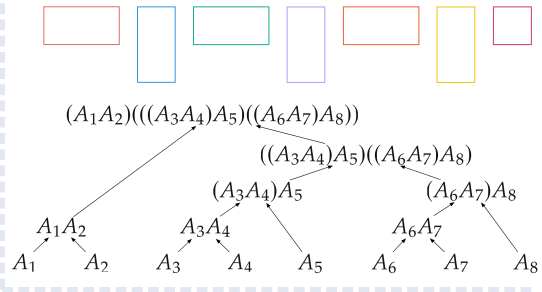
1. recursive spec
2. recursive implementation
3. mention "caching" or "dynamic programming"
4. analyze run time
5. how to use recursive algo to solve problem
6. proof (by induction)

Input: dimensions $n_1, n_2, \ldots, n_k, n_{k+1} \in \mathbb{N}$

Output: min # (ordinary) multiplications needed to compute $A_1 A_2 A_3 \cdots A_k$, where $A_i \in \mathbb{R}^{n_i \times n_{i+1}}$

1. recursive spec

$\mathsf{Fastest}(i, j)$ = the minimum number of operations to compute the matrix product $A_i \ldots A_j$.



2. recursive implementation

$\mathsf{Fastest}(i, j)$

1. If $i \geq j$ then return 0.
2. Return the minimum, over all $k = i, \ldots, j-1$, of

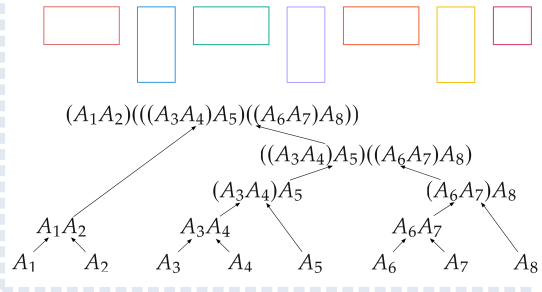
$$\mathsf{Fastest}(i, k) + \mathsf{Fastest}(k+1, j) + n_i n_{k+1} n_j.$$

template for this class:

1. recursive spec
2. recursive implementation
3. mention "caching" or "dynamic programming"
4. analyze run time
5. how to use recursive algo to solve problem
6. proof (by induction)

Input: dimensions $n_1, n_2, \ldots, n_k, n_{k+1} \in \mathbb{N}$

Output: min # (ordinary) multiplications needed to compute $A_1 A_2 A_3 \cdots A_k$, where $A_i \in \mathbb{R}^{n_i \times n_{i+1}}$



1. recursive spec

$\texttt{Fastest}(i, j)$ = the minimum number of operations to compute the matrix product $A_i \ldots A_j$.

2. recursive implementation

$\texttt{Fastest}(i, j)$

1. If $i \geq j$ then return 0.
2. Return the minimum, over all $k = i, \ldots, j-1$, of

$$\texttt{Fastest}(i,k) + \texttt{Fastest}(k+1,j) + n_i n_{k+1} n_j.$$

3. how to use recursive algo to solve problem

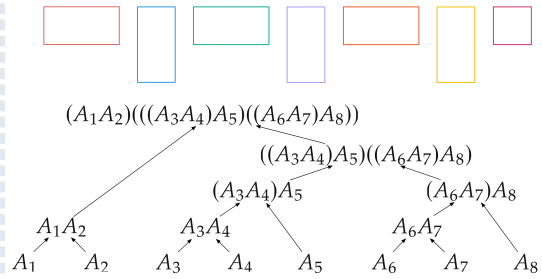
Fastest (1,n)

template for this class:

1. recursive spec
2. recursive implementation
3. how to use recursive algo to solve problem
4. mention "caching" or "dynamic programming"
5. analyze run time
6. proof (by induction)

Input: dimensions $n_1, n_2, \ldots, n_k, n_{k+1} \in \mathbb{N}$

Output: min # (ordinary) multiplications needed to compute $A_1 A_2 A_3 \cdots A_k$, where $A_i \in \mathbb{R}^{n_i \times n_{i+1}}$



1. recursive spec

`Fastest`(i, j) = the minimum number of operations to compute the matrix product $A_i \ldots A_j$.

2. recursive implementation

`Fastest`(i, j)

1. If $i \geq j$ then return 0.
2. Return the minimum, over all $k = i, \ldots, j-1$, of

$$\texttt{Fastest}(i,k) + \texttt{Fastest}(k+1,j) + n_i n_{k+1} n_j.$$

3. how to use recursive algo to solve problem

Fastest (1,n)

4/5: Running time w/ dyn prog.

k^2 subproblems

$O(k)$ time per subproblem

$O(k^3)$ time total

template for this class:

1. recursive spec
2. recursive implementation
3. how to use recursive algo to solve problem
4. mention "caching" or "dynamic programming"
5. analyze run time
6. proof (by induction)