CSE 549:
Computational Biology

Computer Science for ~~Biologists~~ Biology

Stony Brook University

What is Computer Science?
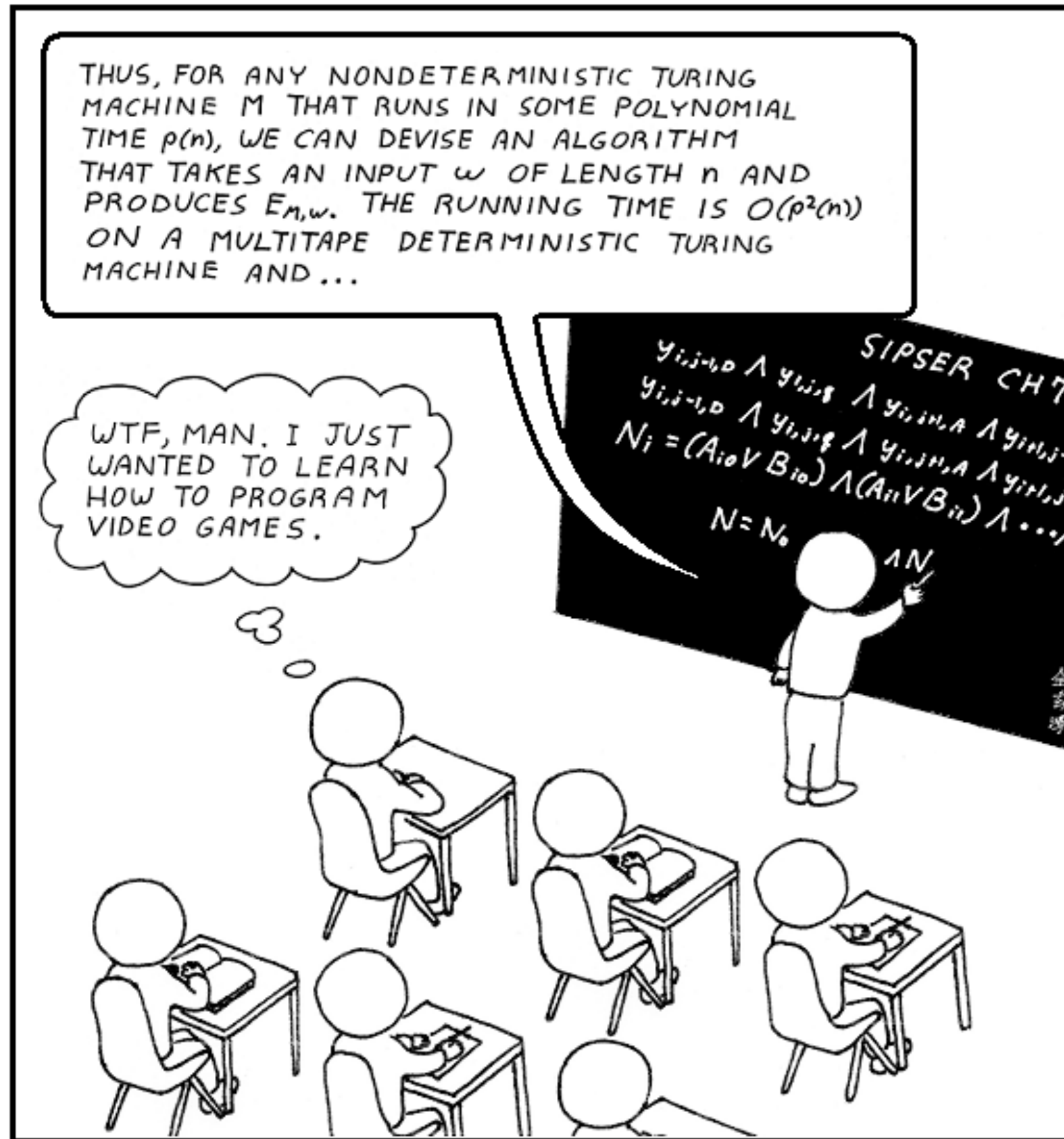
What is Computer Science?

Not actually simple to define constructively

Still debate whether certain areas constitute CS

Computer science is the scientific and practical approach to computation and its applications. It is the systematic study of the feasibility, structure, expression, and mechanization of the methodical procedures (or algorithms) that underlie the acquisition, representation, processing, storage, communication of, and access to information* …

What isn't Computer Science?

Don't install operating systems (may develop them)

**Don't** set up the office network (may study / design network protocols)

Not about Hacking together a program or learning a web-framework — programming ≠ CS (may study formal languages and develop new programming languages)

*http://www.cs.bu.edu/AboutCS/WhatIsCS.pdf

What is Computer Science?

Started as a branch of Mathematics — early computing machines

Charles Babbage (1791-1871)

Ada Lovelace (1791-1871)
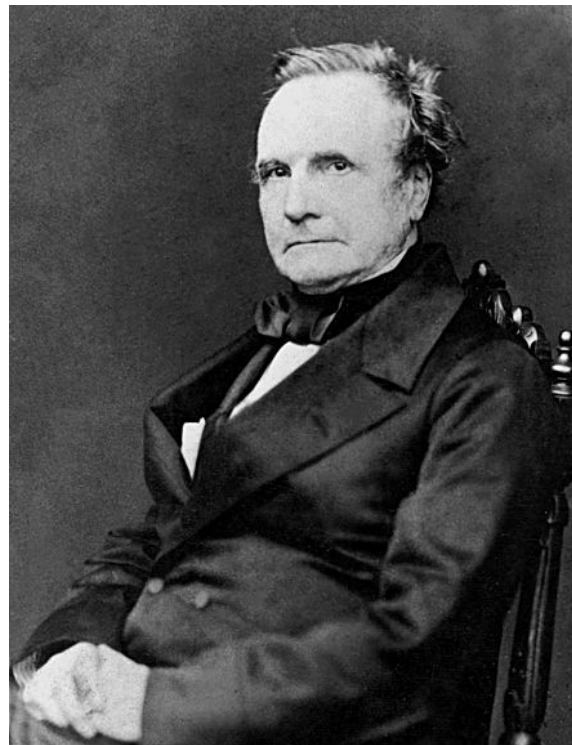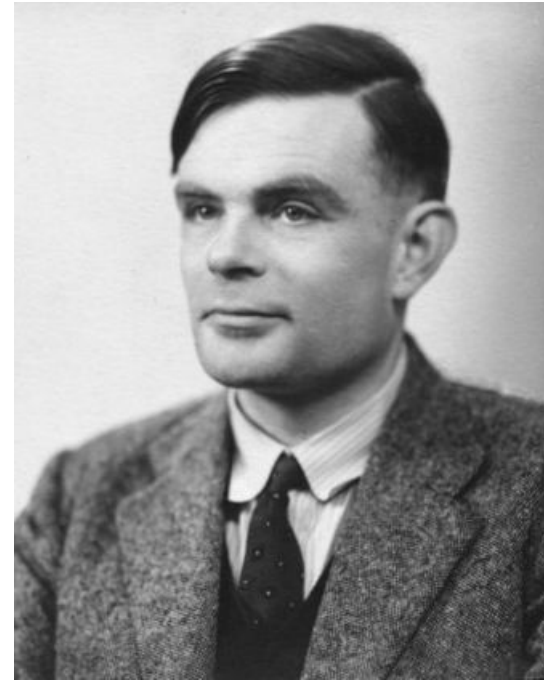

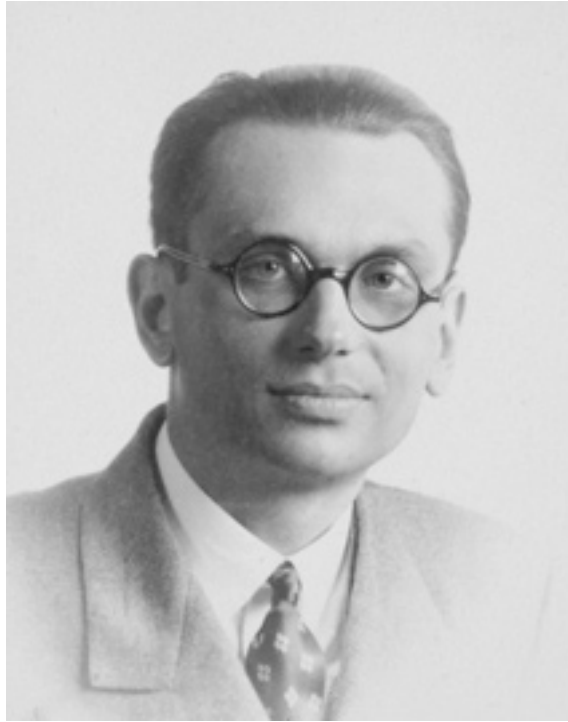Difference engine →
Analytical engine*

Commonly considered the first "programmer"; developed an algorithm for the analytical engine to compute the Bernoulli numbers

*Analytical engine (would have been the first Turing-complete, general purpose computer) was never completed

What is Computer Science?

What is "computable"? Early 20th century & birth of "modern" CS
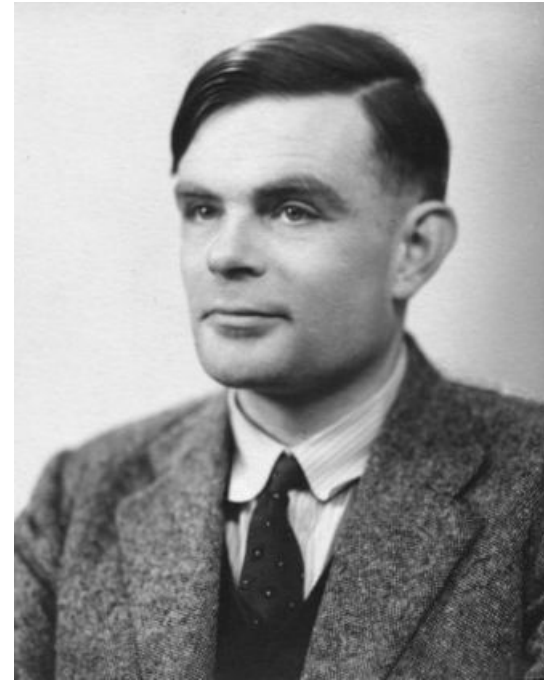
What is Computer Science?

What is "computable"? Early 20th century & birth of "modern" CS

Kurt
Gödel

What is Computer Science?
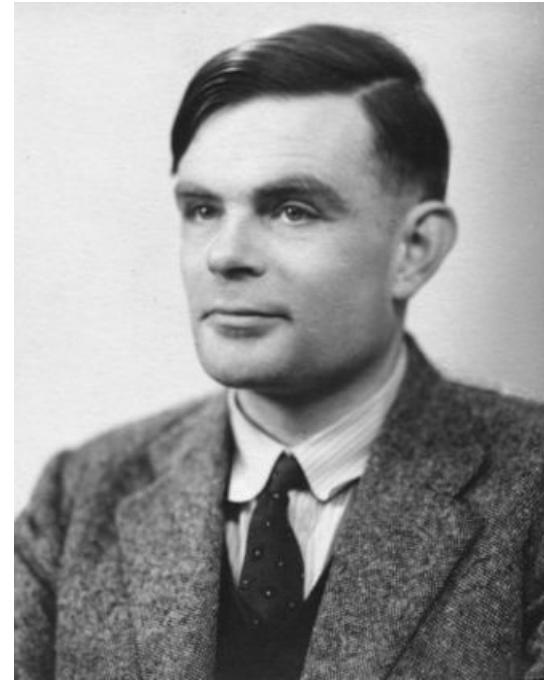
What is "computable"? Early 20th century & birth of "modern" CS

Kurt Gödel
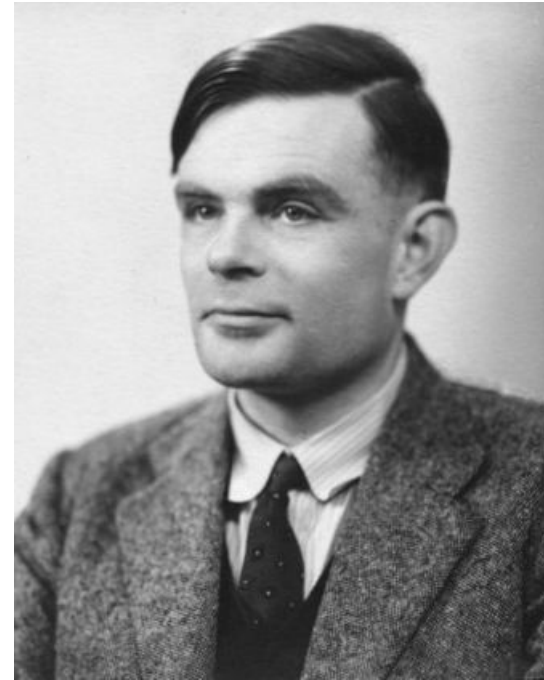
Alan Turing

What is Computer Science?

What is "computable"? Early 20th century & birth of "modern" CS

Kurt Gödel

Alan Turing

Alonzo Church

What is Computer Science?

What is "computable"? Early 20th century & birth of "modern" CS

Kurt Gödel

Alan Turing

Alonzo Church

John von Neumann

What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

Assembly

Reads

Reference genome

+
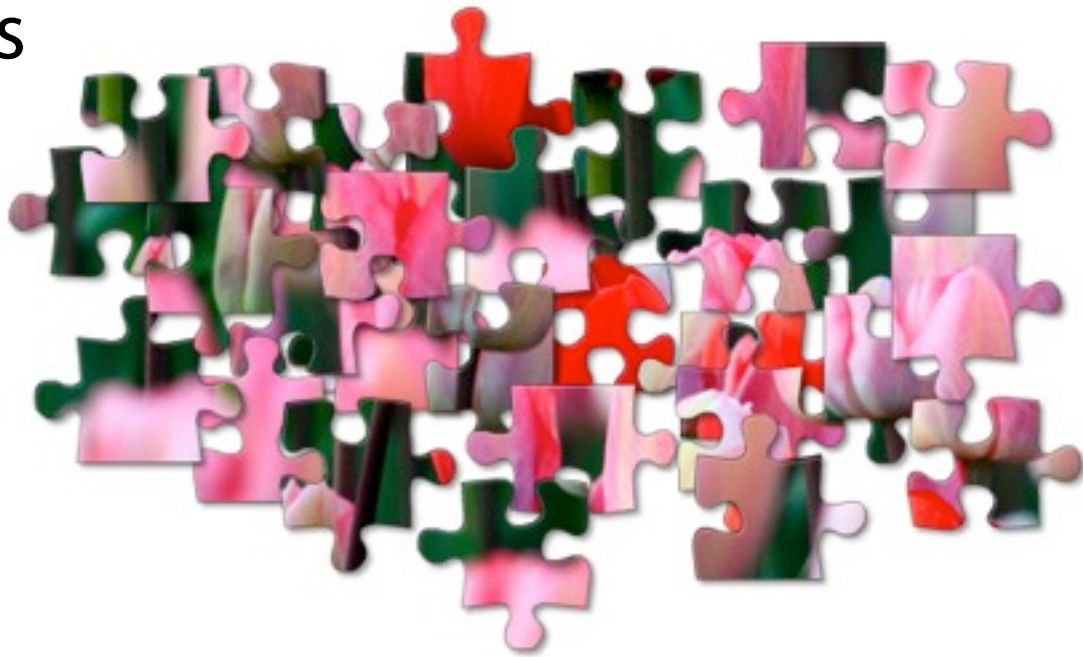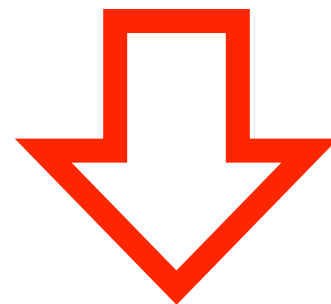
How to assemble puzzle without the benefit of knowing what the finished product looks like?

Input DNA

Assembly

Whole-genome "shotgun" sequencing starts by copying and fragmenting the DNA

("Shotgun" refers to the random fragmentation of the whole genome; like it was fired from a shotgun)

Input: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Copy: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
        GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
        GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
        GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Fragment: GGCGTCTA   TATCTCGG   CTCTAGGCCCTC   ATTTTTT
          GGC   GTCTATAT   CTCGGCTCTAGGCCCTCA   TTTTTT
          GGCGTC   TATATCT   CGGCTCTAGGCCCT   CATTTTTT
          GGCGTCTAT   ATCTCGGCTCTAG   GCCCTCA   TTTTTT

Assembly

Assume sequencing produces such a large # fragments that almost all genome positions are *covered* by many fragments...

Reconstruct this

From these

CTAGGCCCTCAATTTTT
CTCTAGGCCCTCAATTTTT
GGCTCTAGGCCCTCATTTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT

GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
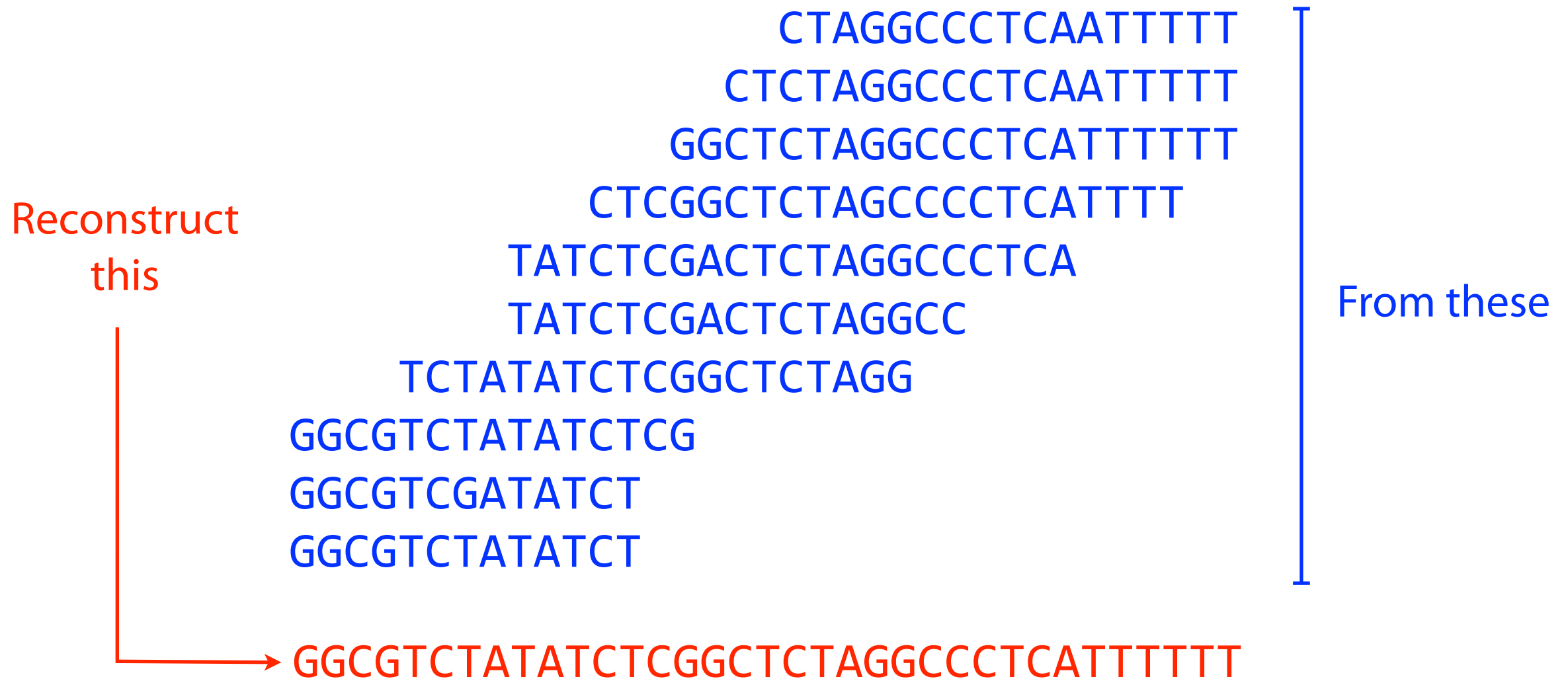
Assembly

...but we don't know what came from where
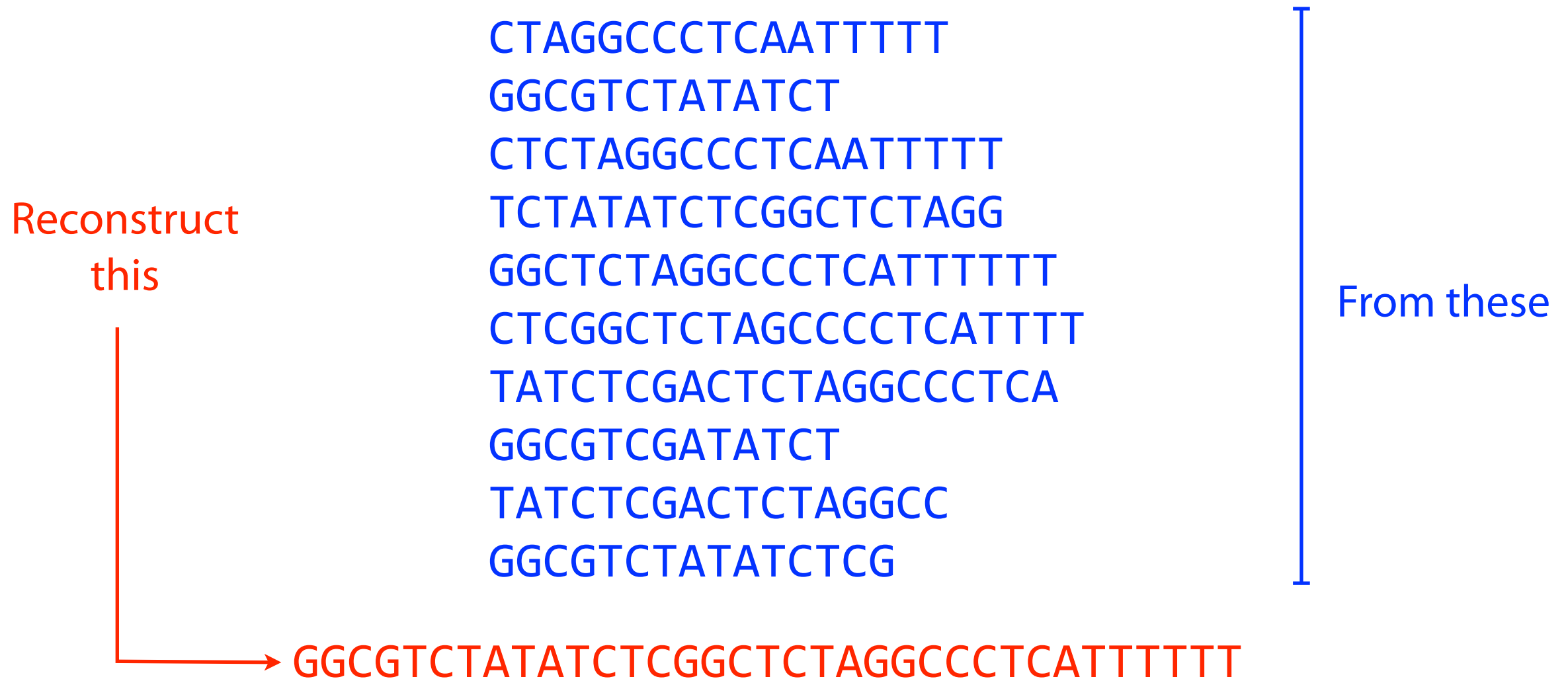
CTAGGCCCTCAATTTTT
GGCGTCTATATCT
CTCTAGGCCCTCAATTTTT
TCTATATCTCGGCTCTAGG
GGCTCTAGGCCCTCATTTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
GGCGTCGATATCT
TATCTCGACTCTAGGCC
GGCGTCTATATCTCG

From these

Reconstruct this

GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

Given: a collection, R, of sequencing reads (strings)

Find: The genome (string), G, that generated them

What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

Given: a collection, R, of sequencing reads (strings)

Find: The genome (string), G, that generated them

Not well-specified.

What makes one genome more likely than another?
What constraints do we place on the space of solutions?

What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

Given: a collection, R, of sequencing reads (strings)

✔

Find: The shortest genome (string), G, that contains all of them

Shortest Common Superstring

Given: a collection, $S = \{s_1, s_2, \ldots, s_k\}$, of sequencing reads (strings)

Find*: The shortest possible genome (string), G, such that $s_1, s_2, \ldots, s_k$ are all substrings of G

How, might we go about solving this problem?

*for reasons we'll explore later, this isn't actually
a great formulation for genome assembly.

Shortest common superstring

Given a collection of strings S, find $SCS(S)$: the shortest string that contains all strings in S as substrings

Without requirement of "shortest," it's easy: just concatenate them

Example: S: BAA AAB BBA ABA ABB BBB AAA BAB

Concatenation: BAAAABBBAABAABBBBAAABAB

├────────────── 24 ──────────────┤

$SCS(S)$: AAABBBABAA

├──── 10 ────┤

AAA
 AAB
  ABB
   BBB
    BBA
     BAB
      ABA
       BAA

Shortest common superstring

Can we solve it?

Imagine a modified overlap graph where each edge has cost = - (length of overlap)

SCS corresponds to a path that visits every node once, minimizing total cost along path

That's the *Traveling Salesman Problem* (*TSP*), which is NP-hard!

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBA
AAA
 AAB
  ABB
   BBB
    BBA
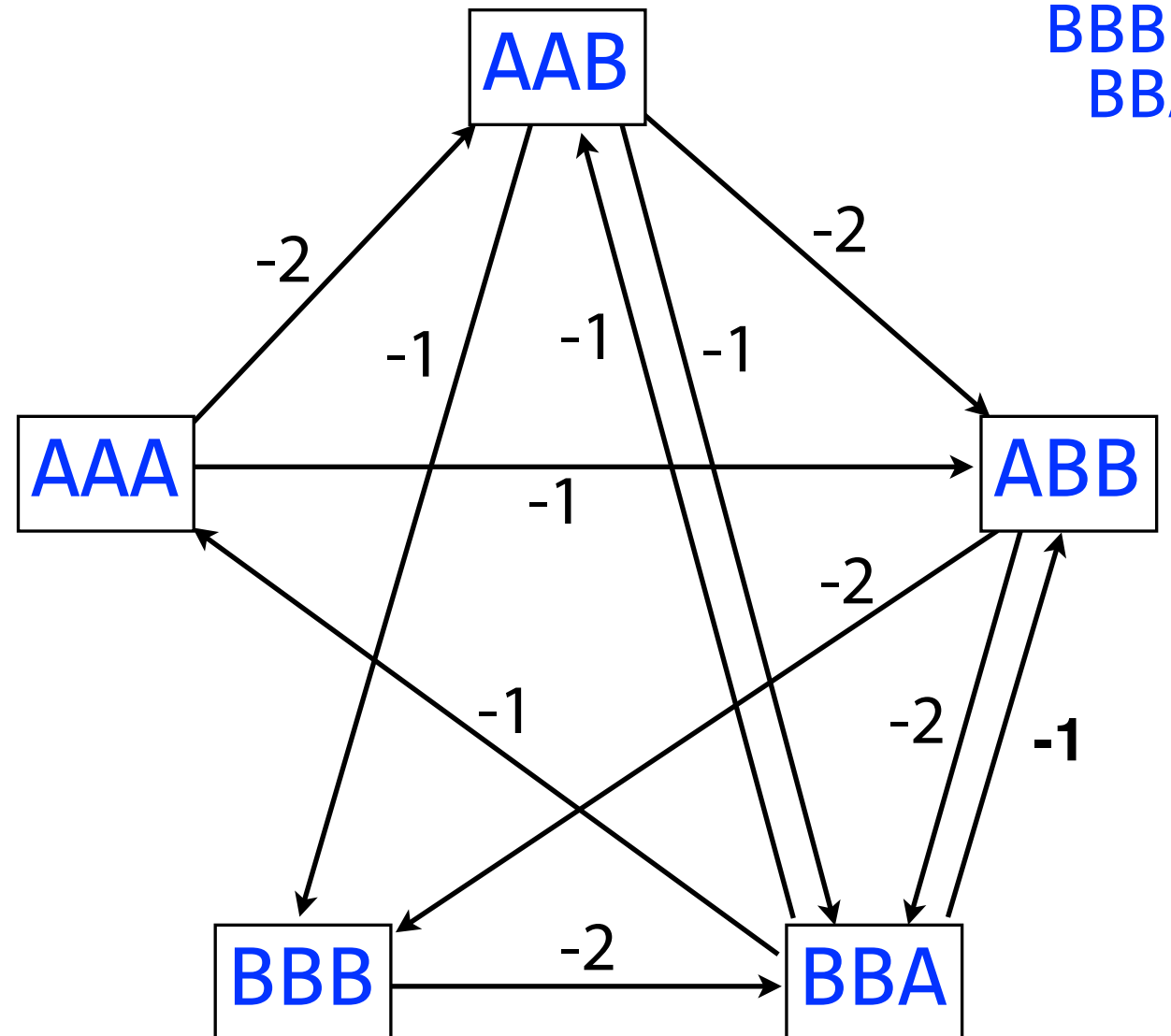
Shortest common superstring

Say we disregard edge weights and just look for a path that visits all the nodes exactly once

That's the *Hamiltonian Path* problem: NP-complete

Indeed, it's well established that SCS is NP-hard

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBA
AAA
AAB
ABB
BBB
BBA
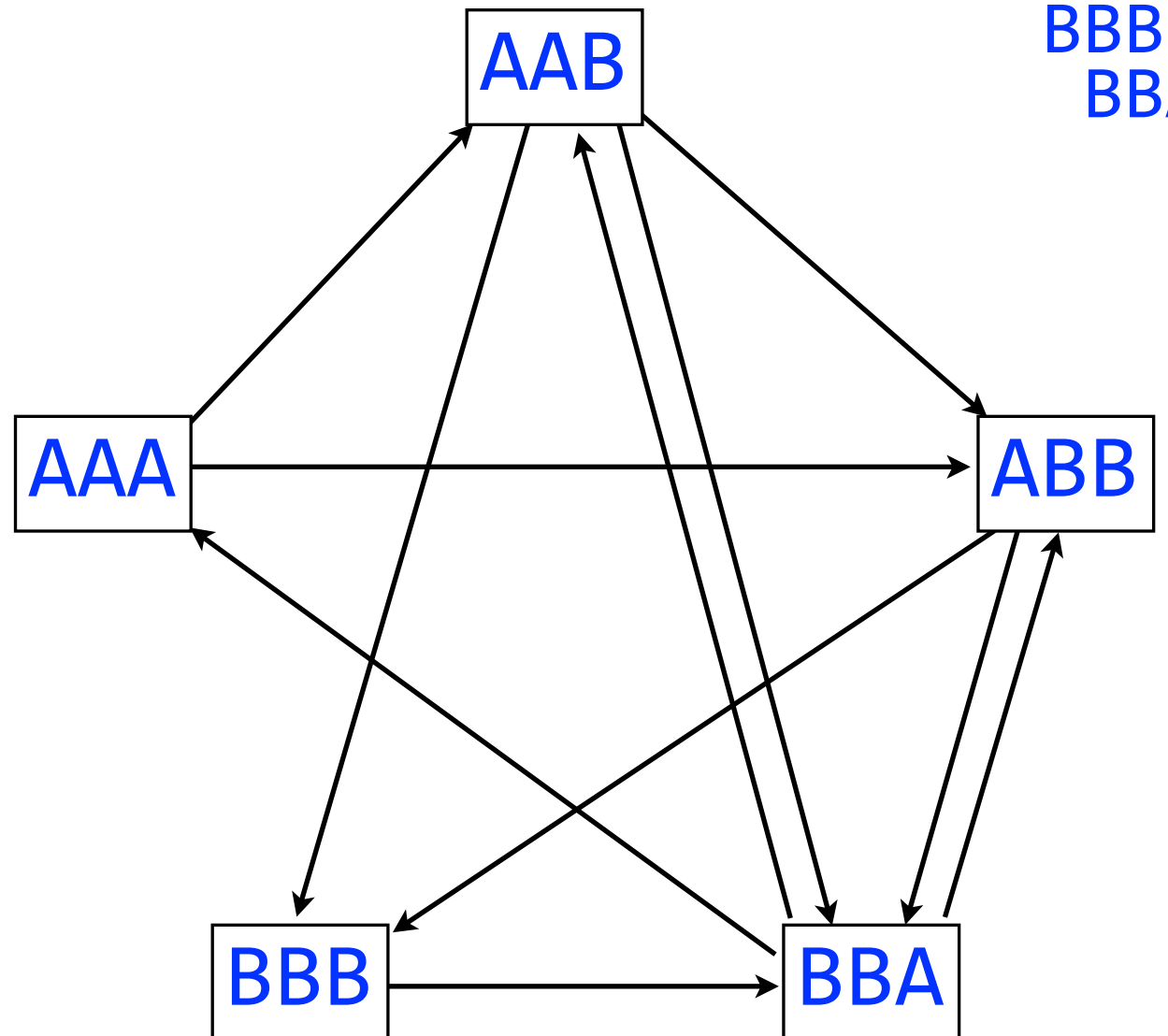
Shortest common superstring & friends

Traveling Salesman, Hamiltonian Path, and Shortest Common Superstring are all NP-hard

For refreshers on Traveling Salesman, Hamiltonian Path, NP-hardness and NP-completeness, see Chapters 34 and 35 of "Introduction to Algorithms" by Cormen, Leiserson, Rivest and Stein, or Chapters 8 and 9 of "Algorithms" by Dasgupta, Papadimitriou and Vazirani (free online: http://www.cs.berkeley.edu/~vazirani/algorithms)

Important note: The fact that we modeled SCS as NP-hard problems (TSP and HP) **does not** prove that (the decision version of) SCS is NP-complete. To do that, we must **reduce** a known NP-complete problem to **SCS**.

Given an instance I of a known hard problem, generate an instance I' of SCS such that if we can solve I' in polynomial time, then we can solve I in polynomial time. This *implies* that SCS is *at least* as hard as the hard problem.

This can be done e.g. with HAMILTONIAN PATH

transformation
(computable in poly time)

Arbitrary instance of HP

Constructed instance of SCS

solve SCS instance

HP known to be NP-complete

reverse transformation
(computable in poly time)
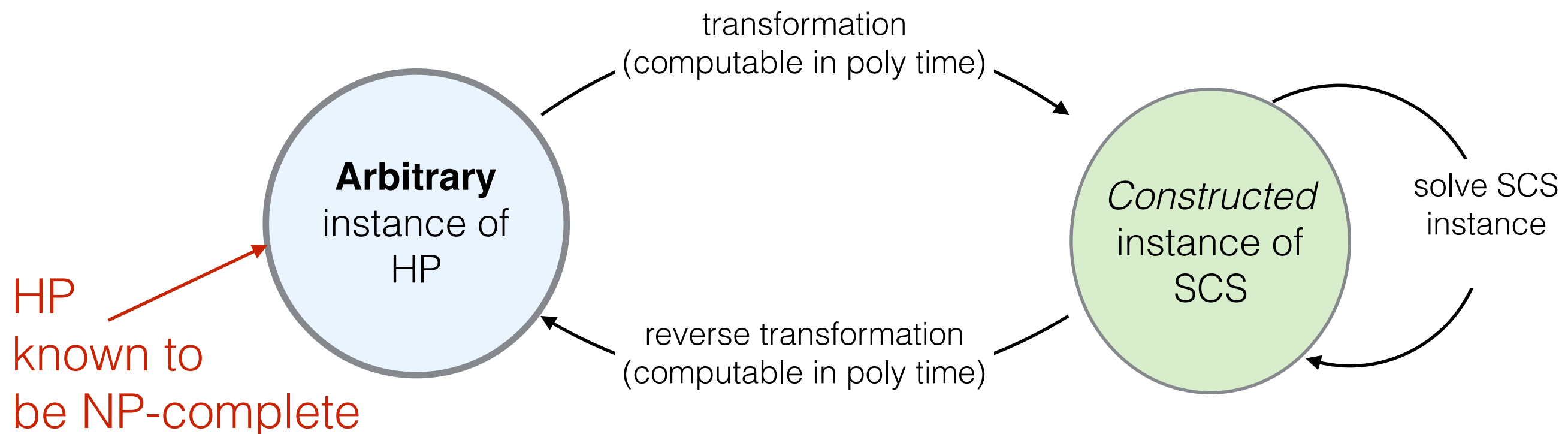
Shortest Common Superstring

The fact that SCS is **NP-complete** means that it is unlikely that there exists *any* algorithm that can solve a general instance of this problem in time polynomial in n — the number of strings.

If we give up on finding the *shortest* possible superstring G, how does the situation change?

Shortest Common Superstring

There's a "greedy" *heuristic* that turns out to be an *approximation algorithm* (provides a solution within a constant factor of the the optimum)

At each step, chose the pair of strings with the maximum overlap, merge them, and return the merged string to the collection.

Greedy conjecture factor of 2-OPT *is* the worst case

Different approx. (**not all greedy**)

ratio	authors	year
	approximating SCS	
3	Blum, Jiang, Li, Tromp and Yannakakis [4]	1991
$2\frac{8}{9}$	Teng, Yao [23]	1993
$2\frac{5}{6}$	Czumaj, Gasieniec, Piotrow, Rytter [8]	1994
$2\frac{50}{63}$	Kosaraju, Park, Stein [15]	1994
$2\frac{3}{4}$	Armen, Stein [1]	1994
$2\frac{50}{69}$	Armen, Stein [2]	1995
$2\frac{2}{3}$	Armen, Stein [3]	1996
$2\frac{25}{42}$	Breslauer, Jiang, Jiang [5]	1997
$2\frac{1}{2}$	Sweedyk [21]	1999
$2\frac{1}{2}$	Kaplan, Lewenstein, Shafrir, Sviridenko [12]	2005
$2\frac{1}{2}$	Paluch, Elbassioni, van Zuylen [18]	2012
$2\frac{11}{23}$	Mucha [16]	2013
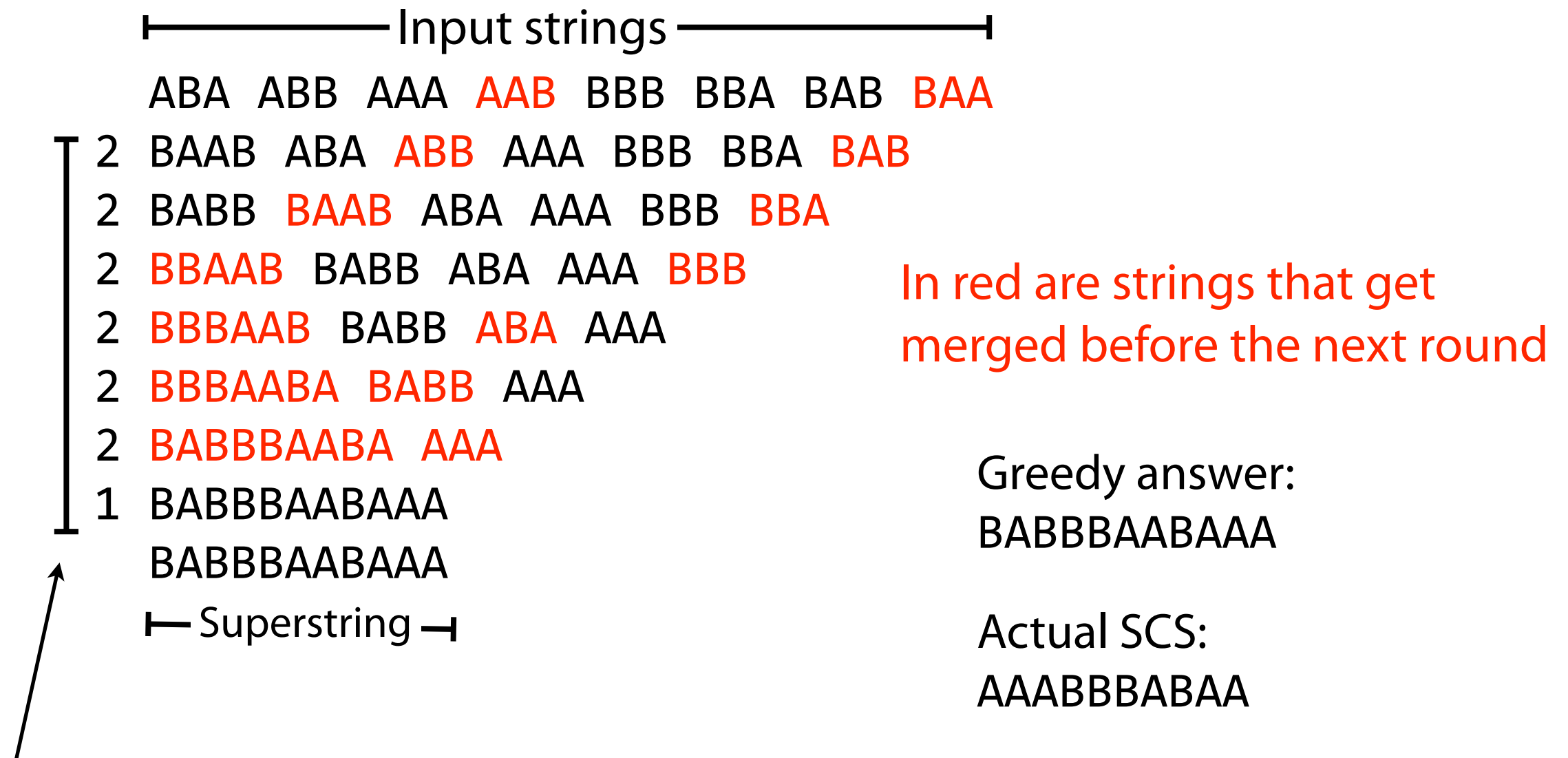
Golovnev, Kulikov, & Mihajlin. "Approximating Shortest Superstring Problem Using de Bruijn Graphs." Combinatorial Pattern Matching. Springer Berlin Heidelberg, 2013.

Shortest common superstring: greedy

Greedy-SCS algorithm in action ($l = 1$):

|————————— Input strings —————————|

ABA ABB AAA AAB BBB BBA BAB BAA

2 BAAB ABA ABB AAA BBB BBA BAB

2 BABB BAAB ABA AAA BBB BBA

2 BBAAB BABB ABA AAA BBB

2 BBBAAB BABB ABA AAA

2 BBBAABA BABB AAA

2 BABBBAABA AAA

1 BABBBAABAAA

   BABBBAABAAA

|— Superstring —|

In red are strings that get merged before the next round

Greedy answer:
BABBBAABAAA

Actual SCS:
AAABBBABAA

Rounds of merging, one merge per line.
Number in first column = length of overlap merged before that round.

Slide courtesy of Ben Langmead

Shortest common superstring: greedy

Greedy-SCS algorithm in action ($l = 1$):

Input strings

ABA ABB AAA AAB BBB BBA BAB BAA
2 BAAB ABA ABB AAA BBB BBA BAB
2 BABB BAAB ABA AAA BBB BBA
2 BBAAB BABB ABA AAA BBB
2 BBBAAB BABB ABA AAA
2 BBBAABA BABB AAA
2 BABBBAABA AAA
1 BABBBAABAAA

BABBBAABAAA

├─ Superstring ─┤

In red are strings that get
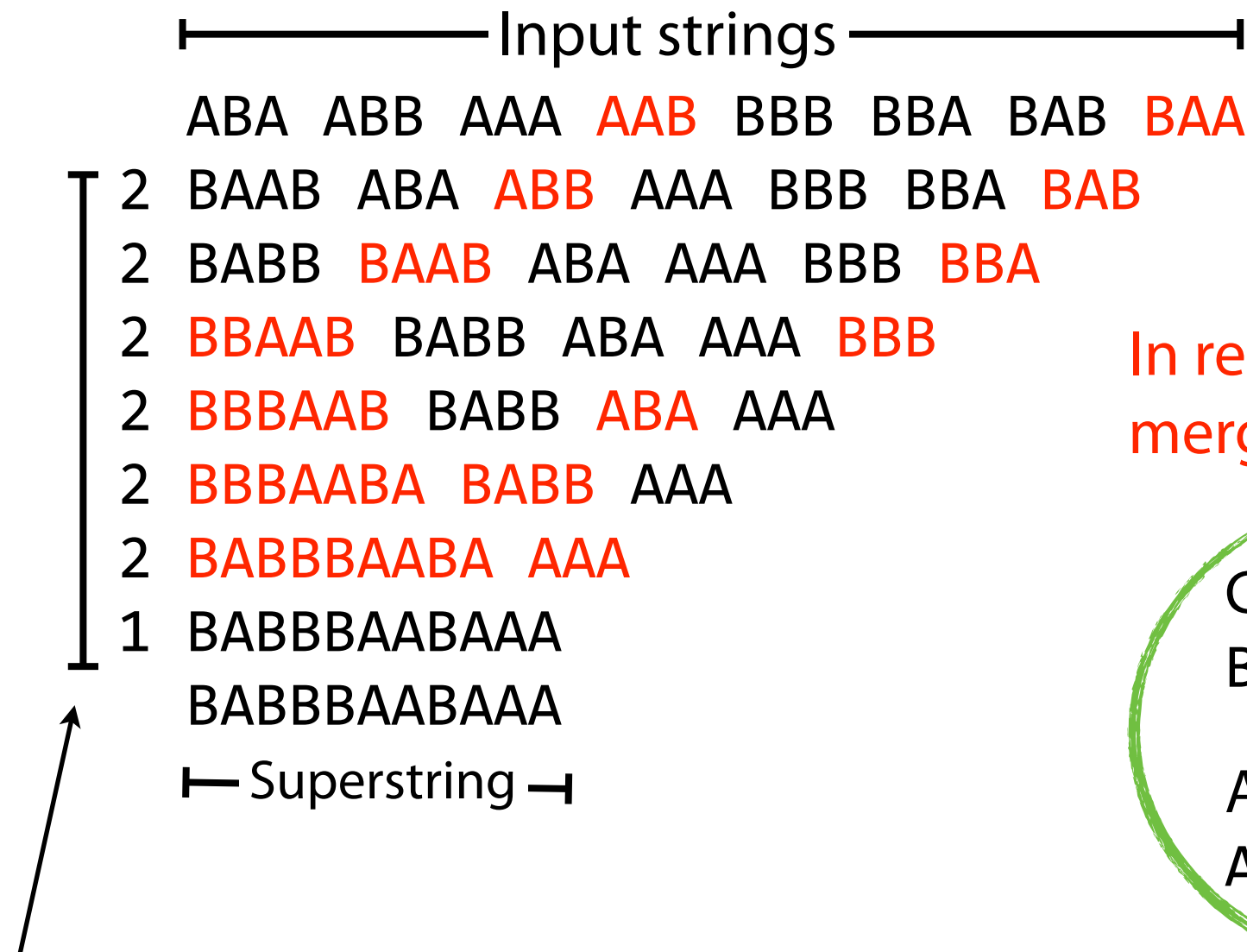merged before the next round

Greedy answer:
BABBBAABAAA
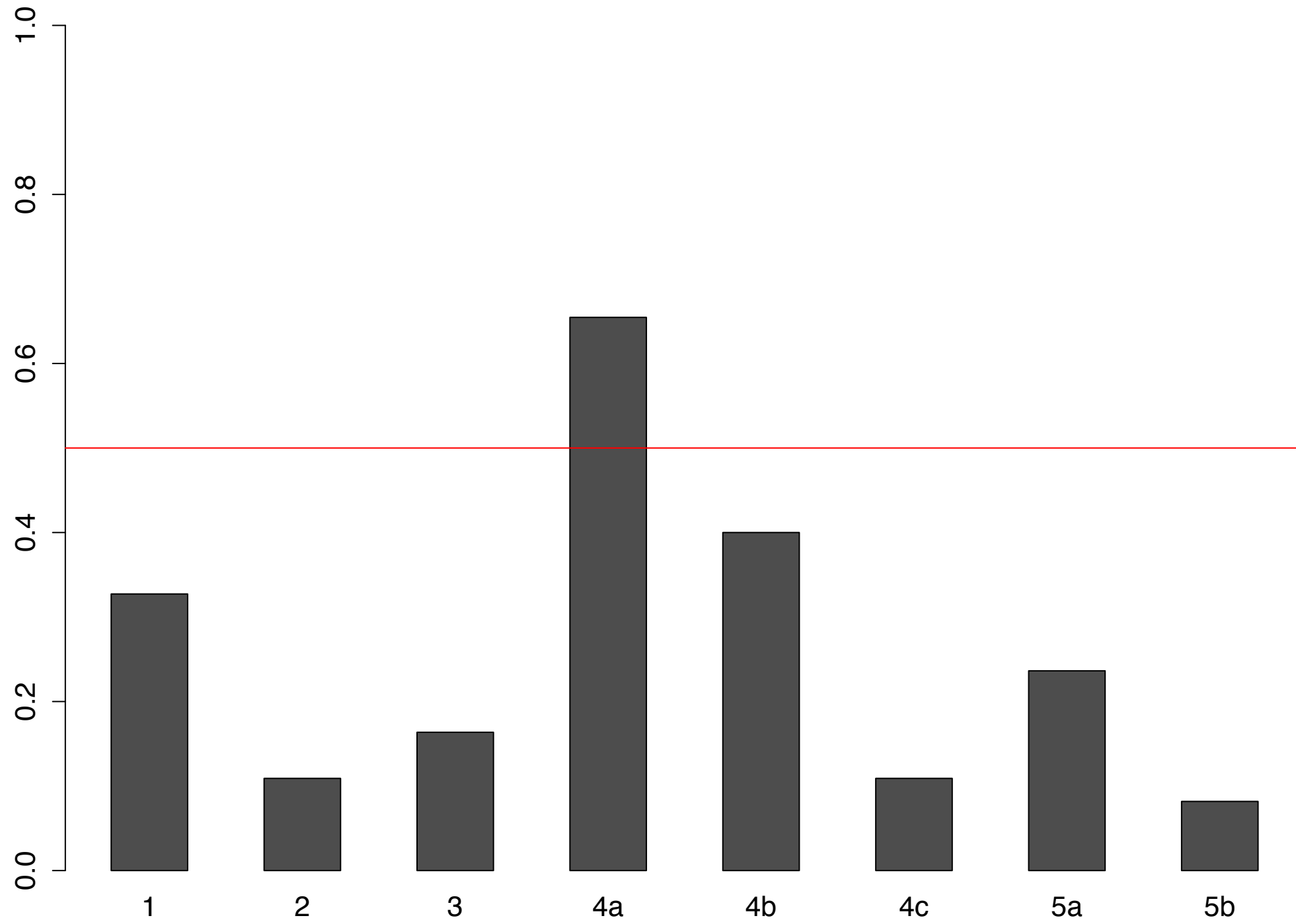
Actual SCS:
AAABBBABAA

Rounds of merging, one merge per line.
Number in first column = length of overlap merged before that round.

Slide courtesy of Ben Langmead

Results from survey & review

1. Given a string S of length $|S|$, how many (non-empty) substrings does S contain?

1. Given a string S of length $|S|$, how many (non-empty) substrings does S contain?

   $\binom{|S|}{2}$

1. Given a string S of length $|S|$, how many (non-empty) substrings does S contain?

$\binom{|S|}{2}$

2. Given a directed acyclic graph (DAG) $G = (V, E)$, give a (tight) asymptotic upper bound for the complexity of finding the shortest path between two vertices $s, t \in V$?

1. Given a string S of length $|S|$, how many (non-empty) substrings does S contain? $\binom{|S|}{2}$

2. Given a directed acyclic graph (DAG) $G = (V, E)$, give a (tight) asymptotic upper bound for the complexity of finding the shortest path between two vertices $s, t \in V$? $O(V + E)$

1. Given a string S of length $|S|$, how many (non-empty) substrings does S contain? $\binom{|S|}{2}$

2. Given a directed acyclic graph (DAG) $G = (V, E)$, give a (tight) asymptotic upper bound for the complexity of finding the shortest path between two vertices $s, t \in V$? $O(V + E)$

3. *Sketch* (in English or pseudocode) an algorithm that returns the *length* of the shortest path between s, t in the DAG G.

1. Given a string S of length $|S|$, how many (non-empty) substrings does S contain? $\binom{|S|}{2}$

2. Given a directed acyclic graph (DAG) $G = (V, E)$, give a (tight) asymptotic upper bound for the complexity of finding the shortest path between two vertices $s, t \in V$? $O(V + E)$

3. *Sketch* (in English or pseudocode) an algorithm that returns the *length* of the shortest path between s, t in the DAG G.

```
for each vertex v in V:
    sp[v] = infinity
sp[s] = 0
for each vertex v in topo(V):
    for each w in predecessors(v):
        sp[v] = min(sp[w] + 1, sp[v])
return sp[t]
```

4. I am given two coins, which I will flip simultaneously. Let $X = H$ be the event that the first coin comes up heads and $X = T$ be the event that the first coin comes up tails. Let $Y = H$ and $Y = T$ be defined similarly for the second coin. If you know that $\Pr(X = H, Y = H) = 0.25$ and $\Pr(X = T, Y = H) = 0.5$ and $\Pr(X = H, Y = T) = 0.05$.

  (a) What is $\Pr(X = T, Y = T)$?

  (b) What is $\Pr(X = T)$?

	Y=H	Y=T
X=H	0.25	0.5
X=T	0.05	?

4. I am given two coins, which I will flip simultaneously. Let $X = H$ be the event that the first coin comes up heads and $X = T$ be the event that the first coin comes up tails. Let $Y = H$ and $Y = T$ be defined similarly for the second coin. If you know that $\Pr(X = H, Y = H) = 0.25$ and $\Pr(X = T, Y = H) = 0.5$ and $\Pr(X = H, Y = T) = 0.05$.

   (a) What is $\Pr(X = T, Y = T)$?
   (b) What is $\Pr(X = T)$?

	Y=H	Y=T
X=H	0.25	0.5
X=T	0.05	(1-0.8) = 0.2

4. I am given two coins, which I will flip simultaneously. Let $X = H$ be the event that the first coin comes up heads and $X = T$ be the event that the first coin comes up tails. Let $Y = H$ and $Y = T$ be defined similarly for the second coin. If you know that $\Pr(X = H, Y = H) = 0.25$ and $\Pr(X = T, Y = H) = 0.5$ and $\Pr(X = H, Y = T) = 0.05$.

   (a) What is $\Pr(X = T, Y = T)$? 0.2

   (b) What is $\Pr(X = T)$?

4. I am given two coins, which I will flip simultaneously. Let $X = H$ be the event that the first coin comes up heads and $X = T$ be the event that the first coin comes up tails. Let $Y = H$ and $Y = T$ be defined similarly for the second coin. If you know that $\Pr(X = H, Y = H) = 0.25$ and $\Pr(X = T, Y = H) = 0.5$ and $\Pr(X = H, Y = T) = 0.05$.

   (a) What is $\Pr(X = T, Y = T)$? 0.2

   (b) What is $\Pr(X = T)$? $\Pr(X = T, Y = H) + \Pr(X = T, Y = T) = 0.5 + 0.2 = 0.7$

   (c) Are the two coin flips statistically independent (say why or why not)?

4. I am given two coins, which I will flip simultaneously. Let $X = H$ be the event that the first coin comes up heads and $X = T$ be the event that the first coin comes up tails. Let $Y = H$ and $Y = T$ be defined similarly for the second coin. If you know that $\Pr(X = H, Y = H) = 0.25$ and $\Pr(X = T, Y = H) = 0.5$ and $\Pr(X = H, Y = T) = 0.05$.

   (a) What is $\Pr(X = T, Y = T)$? 0.2

   (b) What is $\Pr(X = T)$? $\Pr(X = T, Y = H) + \Pr(X = T, Y = T) = 0.5 + 0.2 = 0.7$

   (c) Are the two coin flips statistically independent (say why or why not)? No, since e.g. $\Pr(X = T, Y = T) \neq \Pr(X = T) \cdot \Pr(Y = T)$.

5. The program `grep` allows one to quickly find a target pattern in a file. In practice `grep` works quickly by using a combination of a well-engineered implementation, and an efficient *algorithm* for pattern search. Assume that you are given a large string T representing the file and a string P representing the pattern the user wishes to find.

   (a) What *algorithm* might you use to search for the pattern P in T?
   (b) What is the running time of your algorithm in terms of $|T|$ and $|P|$?

5. The program `grep` allows one to quickly find a target pattern in a file. In practice `grep` works quickly by using a combination of a well-engineered implementation, and an efficient *algorithm* for pattern search. Assume that you are given a large string T representing the file and a string P representing the pattern the user wishes to find.

   (a) What *algorithm* might you use to search for the pattern P in T? Knuth-Morris-Pratt

   (b) What is the running time of your algorithm in terms of $|T|$ and $|P|$?

5. The program `grep` allows one to quickly find a target pattern in a file. In practice `grep` works quickly by using a combination of a well-engineered implementation, and an efficient *algorithm* for pattern search. Assume that you are given a large string T representing the file and a string P representing the pattern the user wishes to find.

   (a) What *algorithm* might you use to search for the pattern P in T? Knuth-Morris-Pratt

   (b) What is the running time of your algorithm in terms of $|T|$ and $|P|$? $O(|T| + |P|)$