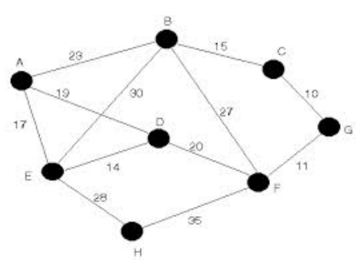
Graph Algorithms

This is not a pipe.

Graph Algorithms

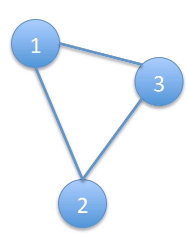


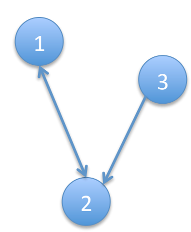
This is not a graph

Graph Algorithms

- A graph is an ordered set (V,E), where V is a set of vertices and E is a set of 2 element subsets of V.
- For example V = {1,2,3}, E = {{1,2},{3,2},{1,3}}
- A directed graph is an ordered set (V,A), where V is a set of vertices and A is an ordered pair of vertices.
- For example V= {1,2,3}, E= {(1,2), (2,1), (3,2)}

Graph Algorithms





V = {1,2,3}, E = {{1,2},{3,2},{1,3}}

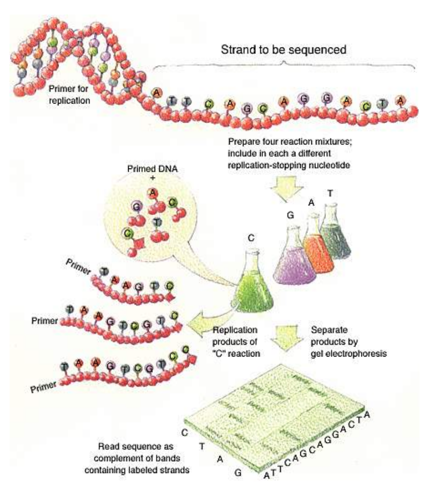
V= {1,2,3}, E= {(1,2), (2,1), (3,2)}

Graph Algorithms

- We will see how some graph algorithms play an important role in genome sequencing.

DNA Sequencing

- Shear DNA into millions of small fragments
- Read 500 – 700 nucleotides at a time from the small fragments (Sanger method)



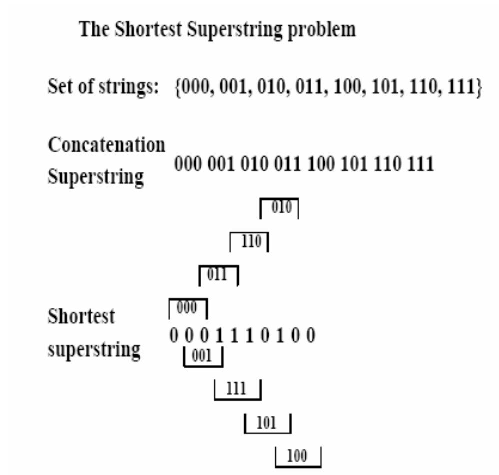
Fragment Assembly

- **<u>Computational Challenge</u>:** assemble individual short fragments (reads) into a single genomic sequence ("superstring")
- Until late 1990s the shotgun fragment assembly of human genome was viewed as an intractable problem

Shortest Superstring Problem

- <u>Problem:</u> Given a set of strings, find a shortest string that contains all of them
- <u>Input</u>: Strings $s_1, s_2, \ldots, s_n$
- <u>Output</u>: A string s that contains all strings $s_1, s_2, \ldots, s_n$ as substrings, such that the length of s is minimized
- **Complexity:** NP – complete
- **Note:** this formulation does not take into account sequencing errors

Shortest Superstring Problem: Example



Reducing SSP to TSP

- Define *overlap (s_i, s_j)* as the length of the longest prefix of s_j that matches a suffix of s_i.

aaaggcatcaaatctaaaggcatcaaa

aaaggcatcaaatctaaaggcatcaaa

What is overlap (s_i, s_j) for these strings?

Reducing SSP to TSP

- Define *overlap* (s_i, s_j) as the length of the longest prefix of s_j that matches a suffix of s_i.

aaaggcatcaaatctaaaggcatcaaa

aaaggcatcaaatctaaaggcatcaaa

shortest superstring: aaaggcatcaaatctaaaggcatcaaa

Reducing SSP to TSP

- Construct a graph with n vertices representing the n strings $s_1, s_2, \ldots, s_n$.
- Insert edges of length *overlap* (s_i, s_j) between vertices s_i and s_j.
- Find the shortest path which visits every vertex exactly once. This is the **Traveling Salesman Problem** (TSP).

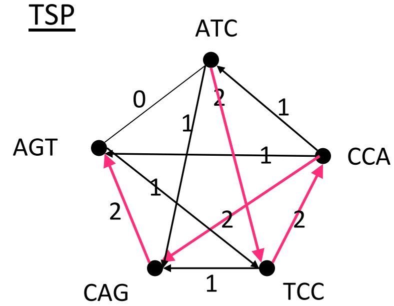
SSP to TSP: An Example

S = { ATC, CCA, CAG, TCC, AGT }

SSP

AGT
CCA
ATC
ATCCAGT
TCC
CAG

TSP



ATCCAGT

What if a problem has:

An exponential upper bound

A polynomial lower bound

The only algorithms that are known to solve the TSP have worst case exponential time complexity. Thus they have an exponential upper bound.

A lower bound for the complexity of a problem is mathematical proof that shows that a certain amount of time is required to solve a problem. The only known lower bounds for the TSP are polynomial.

We have only found exponential algorithms, so it appears that the problem is "intractable".

But... we can't prove that an exponential solution is needed, we can't prove that a polynomial algorithm cannot be developed, so we can't say the problem is intractable...

Cook, Stephen (1971). "The complexity of theorem proving procedures". Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158.

- This paper introduces a concept that handles the conundrum of the mis-match between upper and lower bounds of problems that we strongly believe are hard.

“I can’t find an efficient algorithm. I guess I’m just too dumb”

"I can't find an efficient algorithm, because no such algorithm is possible!"

"I can't find an efficient algorithm, but neither can all these famous people."

- **What is NP?**
- NP is the set of all decision problems (question with yes-or-no answer) for which the 'yes'-answers can be **verified** in polynomial time ($O(n^k)$ where n is the problem size, and k is a constant)

Polynomial time is sometimes used as the definition of *fast* or *quickly*.

TSP (Travelling Salesman Problem)

Input: Weighted (directed) graph G = (V,E).
Output: A least cost tour that visits very vertex in V exactly once.

Cannot easily verify that a tour is of least cost. So TSP in not known to be in NP.

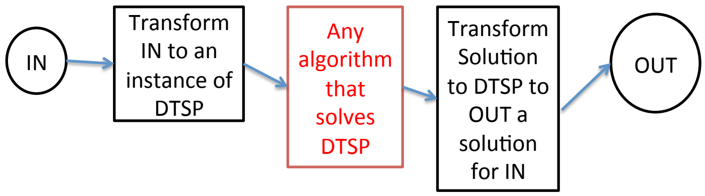
DTSP (Decision version of TSP)
Input: Weighted (directed) graph G = (V,E), and a number K.
Output: Is there a tour that visits every vertex in V exactly once of cost K or less?

Given a tour it's easy to verify whether the cost is less than or equal to K. So we just showed that DTSP is in NP.

What is NP-Complete?

A problem x that is in NP is also in NP-Complete if and only if every other problem in NP can be quickly (ie. in polynomial time) transformed into x. In other words:
x is in NP, and every problem in NP is reducible to x
if any one of the NP-Complete problems was to be solved quickly then all NP problems can be solved quickly.

To prove that a problem is NP-complete one needs to provide polynomial transformation algorithms as shown in the black boxes below. IN is an instance of a problem that is know to be NP-complete and OUT the correct YES/NO answer to IN.



Cook showed the first known NP-complete problem (It is known as SAT). Currently there must be thousands of problems that have been shown to be NP-complete.