**Euler Circuits**

Can you traverse all edges exactly once, starting and finishing at the same vertex?

Possible if and only if:
1. Graph is connected
2. Each vertex has even degree

**Finding Euler Circuits: DFS and then Splice**

Given a graph $G = (V,E)$, find an Euler circuit in $G$.

How?

- Basic Euler Circuit Algorithm:
  1. Do a depth-first search (DFS) from a vertex until you are back at this vertex
  2. Pick a vertex on this path with an unused edge and repeat 1.
  3. Splice all these paths into an Euler circuit

Running time =

**Euler Circuit Example**

**Euler with a Twist: Hamiltonian Circuits**

An Euler circuit: A cycle that goes through each edge exactly once.

A Hamiltonian circuit: A cycle that goes through each vertex exactly once.

Does graph I have:
1. An Euler circuit?
2. A Hamiltonian circuit?

Does graph II have:
1. An Euler circuit?
2. A Hamiltonian circuit?

**Finding Hamiltonian Circuits in Graphs**

Problem: Find a Hamiltonian circuit in a graph $G = (V,E)$.

Is there an easy (linear time) algorithm for checking this?

Runtime?
Polynomial versus Exponential Time

- Most of our algorithms so far have been $O(\log N)$, $O(N)$, $O(N \log N)$ or $O(N^2)$ running time for inputs of size $N$
- These are all polynomial time algorithms
- Their running time is $O(N^k)$ for some $k > 0$
- Exponential time $B^k$ is asymptotically worse than any polynomial function $N^k$ for any $k$
  - For any $k$, $N^k$ is $o(B^k)$ for any constant $B > 1$
- Polynomial time algorithms are generally regarded as “fast” algorithms – these are the kind we want!
- Exponential time algorithms are generally inefficient – avoid these!

The “complexity” class P

- The set $P$ is defined as the set of all problems that can be solved in polynomial worst case time
- Also known as the polynomial time complexity class – contains problems whose time complexity is $O(N^k)$ for some $k$
- Examples of problems in $P$: searching, sorting, topological sort, single-source shortest path, Euler circuit, etc.

The “complexity” class NP

- Definition: NP is the set of all problems for which a given candidate solution can be checked in polynomial time
- Example of a problem in NP:
  - Our new friend, the Hamiltonian circuit problem: Why is it in NP?
- NP = “Non-Deterministic Polynomial Time”

Other problems in NP

- Sorting: Can test in linear time if a candidate ordering is sorted
- But sorting is also in $P$.
  - Are any other problems in $P$ also in NP?

The Intimate Relationship between P and NP

- Sorting is in $P$. Are any other problems in $P$ also in NP?
  - YES!
- All problems in $P$ are also in NP i.e. $P \subseteq NP$
- If you can solve a problem in polynomial time, can definitely verify a solution in polynomial time
- So, some problems in NP like searching, sorting, etc. are also in $P$.
- Question: Are all problems in NP also in $P$?
  - Is $NP \subseteq P$?

Your chance to win a Turing award: $P = NP$?

- Nobody knows whether $NP \subseteq P$
  - Proving or disproving this will bring you instant fame!
- It is generally believed that $P \neq NP$ i.e. there are problems in NP that are not in $P$
  - But no one has been able to show even one such problem
- A very large number of problems are in NP (such as the Hamiltonian circuit problem) but not known to be in $P$
  - No one has found fast (polynomial time) algorithms for these problems
  - No one has been able to prove such algorithms don’t exist (i.e. that these problems are not in $P$)!
P, NP, and Exponential Time Problems

- All algorithms for NP-complete problems so far have tended to run in nearly exponential worst case time
- But this doesn’t mean fast sub-exponential time algorithms don’t exist! Not proven yet...
- Diagram depicts relationship between P, NP, and EXPTIME (class of problems that can be solved within exponential time)

NP-complete problems

- The “hardest” problems in NP are called NP-complete (NPC) problems
- Why “hardest”? A problem X is NP-complete if:
  1. X is in NP and
  2. any problem Y in NP can be converted to X in polynomial time such that solving X also provides a solution for Y
     (If only 2 holds, X is said to be NP-hard)
- Input to Y: “Converter” Algorithm: Input to X (runs in poly time)
- We say that problem Y can be reduced to X
- Note: X is NP-hard if all problems in NP can be reduced to X

The Power of NP-completeness

- Thus, if you find a poly time algorithm for just one NPC problem X, all problems in NP can be solved in poly time
- Example: The Hamiltonian circuit problem can be shown to be NP-complete (not so easy to prove from scratch!)

The “graph” of NP-completeness

- Cook first showed (in 1971) that satisfiability of Boolean formulas (SAT) is NP-complete
- Hundreds of other problems (from scheduling and databases to optimization theory) have since been shown to be NPC
- How? By giving an algorithm for converting a known NPC problem to your pet problem in poly time. Then, your problem is also NPC!

Showing NP-completeness: An Example

- Consider the Traveling Salesperson (TSP) Problem: Given a fully connected, weighted graph G = (V,E), is there a cycle that visits all vertices exactly once and has total cost ≤ K?
- TSP is in NP (why?)
- Can we show TSP is NP-complete? How?
**TSP is NP-complete!**

We can show TSP is also NPC if we can convert any input for HC to an input for TSP in poly time. Here’s one way:

Just assign weight of 1 for all existing edges and 2 to new edges.

Can prove: This graph has a Hamiltonian circuit iff this fully connected graph has a TSP cycle of total cost $K = |V|$ (here, $K = 5$).

---

**Coping with NP-completeness**

Given that it is difficult to find fast algorithms for NPC problems, what do we do?

Alternatives:

1. **Dynamic programming**: Avoid repeatedly solving the same subproblem – use table to store results (see Chap. 10).
2. **Settle for algorithms that are fast on average**: Worst case still takes exponential time, but doesn’t occur very often.
3. **Settle for fast algorithms that give near-optimal solutions**: In TSP, may not give the cheapest tour, but maybe good enough.
4. **Try to get a “wimpy exponential” time algorithm**: It’s okay if running time is $O(1.00001^N)$ – bad only for $N > 1,000,000$. 