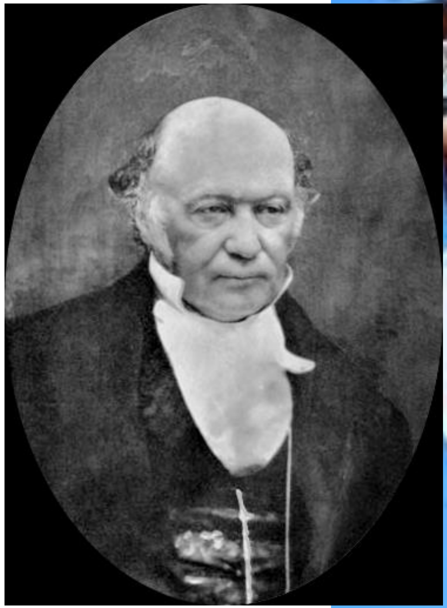
NP-Complete

Textbook Reading:

- Section 10.1 The Classes P and NP, pp. 422-426.
- Section 10.2 Reducibility, pp. 426-428
- Section 10.3 NP-Complete Problems: Cook's Theorem, pp. 429-430
- Subsections 10.4.1 3-CNF SAT and 10.4.2 Clique, pp. 431-433.

Hamiltonian Cycle

Sir William Rowen Hamilton's Icosian Game



Goal of Icosian Game

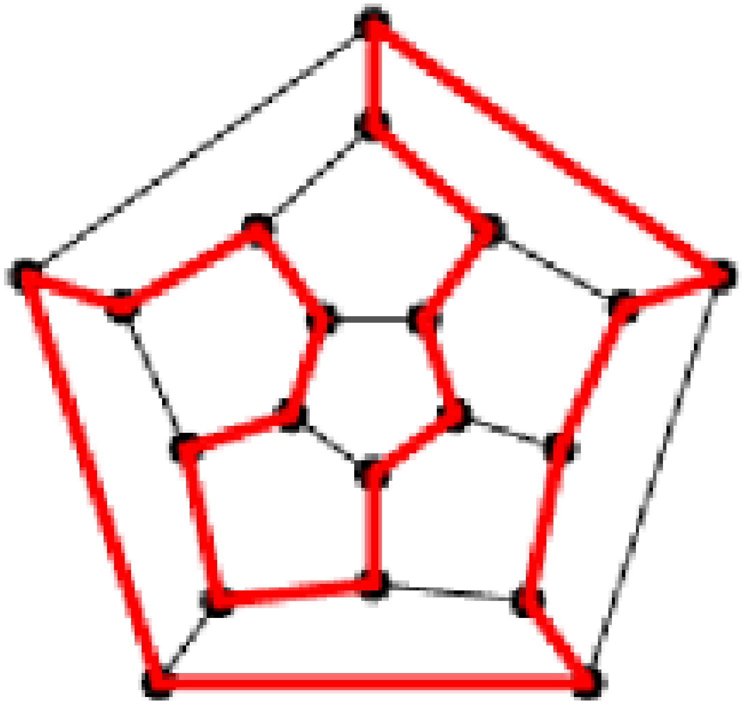
Vertices of the icosahedron represent cities. The goal is to perform a tour of the 20 cities and return to the starting vertex, following an edge of the icosahedron to move between two cities.

This is done by placing pegs on the board so that Peg i and Peg $i + 1$, $i = 1, 2, \ldots, 19$, and Peg 20 and Peg 1 are on adjacent positions, i.e., end vertices of an edge of the icosahedron.

Can you solve the problem?

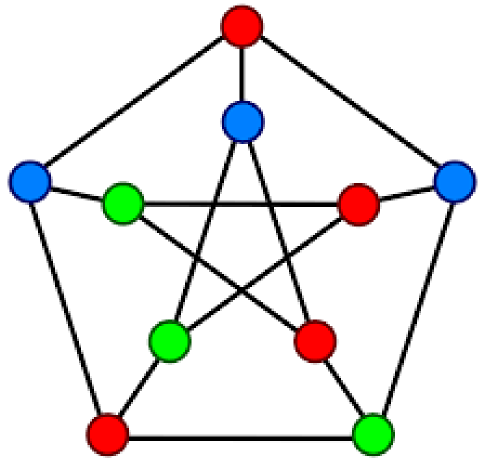
Solution to Icosian Game

Solution involves finding a **Hamiltonian cycle** in the icosahedron:



Graph Coloring Problem

Given *k* colors, color the vertices of a graph so that every edge is **properly** colored, i.e., both ends of the edge are colored differently. Below is a properly vertex 3-colored graph.



PSN. Give an linear time algorithm for solving the problem $k = 2$?

Clique Problem

- Consider the Friendship Network on Facebook, i.e., the vertex set *V* are users of Facebook and the edges set *E* consists of all pairs of users that are friends.
- A **clique** is a group of users, such that every pair are friends, i.e., form a clique. In graph theory, a ***k*-clique** is a subset *K* of *k* vertices that that every pair of vertices in *K* is joined with an edge.
- **Clique Problem.** Given a positive integer *k*, does there exists a *k*-clique, i.e., *k* users, such that every pair are friends.

PSN. Show that the *k*-clique can be solved in polynomial time for *k* constant.

a) The 3-clique problem can be solve in time $O(n^3)$

b) For *k* constant, the *k*-clique problem can be solved in time $O(n^k)$

Sum of Subsets

Given the input integers $A = \{a_0, a_1, \dots, a_{n-1}\}$, and the target sum c, is there are subset of the integers whose sum equals c?

No Worst-Case Polynomial-Time Algorithms Known

Mathematicians worked for years on trying to obtain efficient algorithms for solving these problems in general and many other natural problems, such as the 0/1 Knapsack Problem we discussed earlier in this course, without success. In fact, to this day, no known worst-case polynomial algorithms are known for solving any of them. It turns out that they are all **NP-hard**.

Decision version of the problem

The decision version of a problem asks whether a solution to the problem exists, “yes” or “true” if its exists, “no” or “false”, otherwise. For example,

- Does graph *G* contain a Hamiltonian cycle?
- Can graph *G* be properly vertex *k*-colored?
- Does graph *G* contain a *k*-clique?
- Does there exists a subset of a set *A* whose elements sum to *c*?

Class P

A decision problem is in P if it can be solved with a polynomial-time algorithm, i.e., an algorithm with worst case complexity $O(n^k)$ for some positive integer k.

These problems are sometimes called ***tractable***. The class P is the set of all such problems.

Class NP

NP stands for **nondeterministic polynomial.** It applies to decision problems.

Given a decision problem, we associate a **certificate** with the problem. Examples:

Hamiltonian Cycle Problem. Certificate is sequence of *n* distinct vertices: "yes-certificate" if it is corresponds to a Hamiltonian cycle; otherwise "no-certificate".

Coloring Problem. Certificate is *k*-coloring of vertices: "yes-certificate" if it is a proper coloring; otherwise, it is a "no-certificate".

Clique Problem. Certificate is subset of k vertices: "yes-certificate" if it is a k-clique; otherwise it is a "no-certificate".

Sum of Subsets Problem. Certificate is a subset S of A: "yes-certificate" if the elements of A sum to *c*; otherwise it is a "no-certificate".

High-Level Pseudocode for NP

```
function NPAlgorithm(A,I)
Input:  A (a decision problem), I (an instance of problem A)
Output:  “yes” or “don’t know”
   1.  In polynomial time, guess a candidate certificate C for the
                                                          problem A
   2.   In polynomial time, use C to deterministically verify that I is
                                                          a yes instance.
     if a yes instance is verified in step 2 then
          return (“yes”)
     else
          return(“don’t know”)
     endif
end NPAlgorithm
```

Verifying problems are in NP

Hamiltonian Cycle Problem. Certificate is sequence of *n* distinct vertices $u_1, u_2, \ldots, u_n$ in the graph *G*. It can be verified in time *O*(*n*) whether this certificate is a "yes-certificate", i.e., is a Hamiltonian cycle by simply checking that $\{u_i, u_{i+1}\}$, $i = 1, 2, \ldots, n - 1$, and $\{u_n, u_1\}$, are all edges of *G.*

Coloring Problem. It can be verified whether a *k*-coloring is proper in time *m*, where *m* is the number of edges of *G*, by scanning all the edges to see whether both end vertices of the edge are colored the same.

Clique Problem. It can be verified whether a subset of *k* vertices forms a *k*-clique in time $C(k,2) = k(k - 1)/2$ by checking whether every pair of vertices of the subset is adjacent in *G***.**

Sum of Subsets Problem. It can be verified whether the elements of a subset *S* of a set *A* of size *n* sum to *c* by performing $|S| - 1 < n$ additions.

For each of these problems a candidate certificate that is guessed can be verified to be a yes-certificate or no-certificate in polynomial time, so **they are all in NP**.

Polynomial-Time Reducibility

Given two decision problems *A* and *B*, we say that *A* is (polynomially) *reducible* to *B*, denoted $A \propto B$, if there is a mapping *f* from the inputs to problem *A* to the inputs to problem *B*, such that

1. The function *f* can be computed in polynomial time (that is, there is a polynomial algorithm that for input I to problem *A* outputs the input *f*(*I*) to problem *B*), and

2. the answer to a given input I to problem *A* is yes if, and only if, the answer to the input *f*(*I*) to problem B is yes.

Examples of Reductions

We will illustrate the concept of reduction with two examples.

An ***independent set of size k*** is a set *N* of *k* vertices, such that no two vertices in *N* are adjacent.

A ***vertex cover of size k*** is a subset *C* of *k* vertices such that every edge of *G* is incident to a vertex in *C*.

Clique $\propto$ Independent Vertex Set

Clique $\propto$ Vertex Cover

Clique $\propto$ Independent Vertex Set

Consider and instance of the clique problem (*G*,*k*), i.e., a graph *G* = (*V*,*E*) and a positive integer *k*.

The **complement graph** $\boldsymbol{\bar{G} = (V, \bar{E})}$ is the graph having the same vertex set as *G*, but its edge set $\bar{E}$ consists of all unordered pairs of vertices from *V* that are **not** adjacent in *G*.

Observe that a subset *K* of *V* is a *k*-clique in *G* if and only if it is a set of *k* independent vertices in $\bar{G}$.

Define the reduction function *f* to map instance (*G*,*k*) of the clique problem to instance ($\bar{G}$,*k*) of the independent vertex set problem.

Clique $\propto$ Vertex Cover

Consider and instance of the clique problem (G,k), i.e., a graph $G = (V,E)$ having n vertices and a positive integer k.

Observe that a subset K of V is a k-clique in G if and only if it is a set of $V - K$ *is a vertex cover* in $\bar{G}$.

Define the reduction function f to map instance (G,k) of the clique problem to instance ($\bar{G}$,$n - k$) of the vertex cover problem.

Properties of Reduction Relation $\propto$

- The relation $\propto$ is transitive; that is, if $A \propto B$ and $B \propto C$ then $A \propto C$.

- If $A \propto B$ and B has polynomial complexity, then so does A.

Crazy Definition?

A problem *B* is **NP-complete**, if it is in NP and **any** problem *A* in NP is reducible to *B*, i.e., $A \propto B$.

At first, this definition may seem crazy, because it suggests that you can use *B* to solve every other problem in NP? It would mean that if *B* is in P, every NP problem is in P, i.e., P = NP.

NP-Complete Problems Exist!

Surprising it was shown by Stephen Cook and independently by Leonid Levin, that NP-complete problems exist!

Cook's Famous Theorem

Cook's Theorem. CNF SAT is NP-complete.

Independently, at about the same time, Levin showed that graph tiling is NP-complete.

The Reduction CNF SAT $\propto$ CNF 3-SAT.

Consider any instance of CNF SAT

$$I = C_1 \wedge C_2 \wedge . . . \wedge C_m.$$

C_i is of the form $y_1 \vee \cdots \vee y_j$ where

$$y_1, \ldots, y_j \in \{ x_1, \ldots, x_n\} \cup \{\bar{x}_1, \ldots, \bar{x}_n\} \text{ and } \bar{x}_i = \neg x_i$$

We need to construction a reduction *f* mapping an input *I* of CNF SAT to an input *I’* of CNF 3-SAT, so that *I* is satisfiable iff *I’* is satisfiable.

Replacing clauses of size 1 and 2

a) Replacing clause of size 1 with logically equivalent clause of size 3

$x \lor ((y \land \neg y) \lor (z \land \neg z)) \Leftrightarrow x \lor (\mathrm{F} \lor \mathrm{F}) \Leftrightarrow x$

Using the distributive law of $\lor$ over $\land$ we have

$x \Leftrightarrow x \lor ((y \land \neg y) \land (z \land \neg z))$

$\Leftrightarrow (x \lor y \lor z) \land (x \lor \neg y \lor z) \land (x \lor y \lor \neg z) \land (x \lor \neg y \lor \neg z)$

b) Replacing clause of size 2 with logically equivalent clause of size 3

$(x \lor y) \lor (z \land \neg z) \Leftrightarrow (x \lor y) \lor \mathrm{F} \Leftrightarrow x \lor y$

Using the distributive law of $\lor$ over $\land$ we have

$x \lor y \Leftrightarrow (x \lor y) \lor (z \land \neg z) \Leftrightarrow (x \lor y \lor z) \land (x \lor y \lor \neg z)$

Reduction from CNF SAT to CNF 3-SAT

- We construct instance *I′* of CNF 3-SAT from instance *I* of SAT by replacing clauses of size 1 and 2 using the logically equivalent conjunction of clauses from previous slide. Otherwise, we replace $C_i = y_1 \vee \cdots \vee y_j$ with

$$f = (y_1 \vee y_2 \vee z_1) \wedge (\bar{z}_1 \vee y_3 \vee z_2) \wedge (\bar{z}_2 \vee y_4 \vee z_3) \wedge (\bar{z}_3 \vee y_5 \vee z_4) \wedge \cdots \wedge (\bar{z}_{j-2} \vee y_{j-1} \vee y_j)$$

- If *I* is satisfiable then at least one of $y_1, \ldots, y_j$ are true, and the z_i's can be chosen so the formula *f* is true.
- Conversely, if *I* is not satisfiable, then for any truth assignment some clause C_i is false, i.e., the literals $y_1, \ldots, y_j$ are all false. In which case no matter what values are chosen for the variables z_i, *f* has the value false, so that *I′* is not satisfiable (convince yourself of this).
- We have shown that instance *I* of CNF SAT is satisfiable iff instance *I′* of CNF 3-SAT is satisfiable.

CNF-SAT $\propto$ Clique

To show that Clique is NP-complete it is sufficient to reduce CNF-SAT to Clique (not the other way round, which is trivial since CNF-SAT is NP-complete and Clique is in NP).   .

The Reduction CNF SAT ∝ Clique.

- Consider any Conjunctive Normal Form (CNF) formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

where

$$C_i \;=\; y_1 \vee \ldots \vee y_j \vee \bar{y}_{j+1} \vee \ldots \vee \bar{y}_k,$$

$$y_1, \ldots, y_j \in \{\, x_1, \ldots, x_n\},\;\; \bar{y}_{j+1}, \ldots, \bar{y}_k \in \{\bar{x}_1, \ldots, \bar{x}_n\}.$$

- Construct graph *G* = (*V*,*E*) with vertex set $V = \bigcup_{i=1}^{m} V_i$ where

$$V_i = \{(y_1, i), \ldots, (y_j, i), (\bar{y}_{j+1}, i), \ldots, (\bar{y}_k, i)$$

- Two vertices (a, i) and (b, j) are joined with an edge iff $i \neq j$ and the literal a is not the negation (not) of the literal b.
- Reduction is from $I = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ to $I' = (G, m)$

Proof of Correctness of Reduction

Suppose that *G* has a clique *K* of size *m*.

- Then $K = \{(a_i, i) : i \in \{1, \ldots, m\}\}$
- Assign a_i the value *T* (if $a_i = x_j$ assign variable x_j the value *T*. If $a_i = \bar{x}_j$ assign x_j the value *F*).
- This assignment is well defined, since by definition of adjacency in *G*, none of the literals (in the first component) of a vertex is the negation of another.
- Moreover, this assignment makes each of the clauses C_i evaluate to *T*, so that *I* evaluates as true, i.e., *I* is satisfiable.

Conversely, suppose that *I* is satisfiable.

- Then each clause C_i must contain at least one literal a_i that is true.
- The set of vertices $\{(a_i,i) : i \in \{1, . . . , m\}\}$ forms a clique of size *m*.

NP-complete

Hundreds of natural and important problems have been shown to be NP-complete, including Vertex Coloring and Sum of Subsets. In fact, the coloring problem is NP-complete even for 3 colors.

Vertex Cover and Independent Set

Since we gave a

reduction from Clique to the Vertex Cover and from Clique Independent Vertex Set

and have just shown that

Clique is NP-complete

it follows that

Vertex Cover and Independent Vertex Set are both NP-complete.

NP-Hard.

A problem, not necessary a decision problem, is ***NP-hard***, if it could be applied to solve an NP-complete problem in polynomial time, i.e., it is at least as difficult as an NP-complete problem.

There are thousands of important and practical problems that are known to be NP-hard. In fact, most algorithmic problems that are encountered in practice in science and engineering are NP-hard.

P = NP?

This is one of the most celebrated and important problems in all of computer science and mathematics.

To show P = NP, all you would have to do is design a polynomial-time algorithm for **one** NP-complete problem.

Some of the greatest minds in mathematics and computer science have tried to crack this problem for about 50 years now, without success.

Bagel Challenge

If you solve the famous P = NP? problem, I will buy you a baker's dozen bagels and write you are check for \$($e^{i\pi}$ + 1) .