

PRACTICE PROBLEMS

Graph isomorphism is an example of a problem which is in \mathcal{NP} , but is not known to be \mathcal{NP} -complete, nor is it known to be in $\text{co-}\mathcal{NP}$.

INPUT: Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

QUESTION: Can the vertices of G_1 be renamed so that G_1 becomes G_2 ? (Is there a one-to-one onto function $f : V_1 \rightarrow V_2$ so that $\forall x, y \quad (x, y) \in E_1$ **iff** $(f(x), f(y)) \in E_2$?)

EXERCISE: Show that GRAPH ISOMORPHISM is in \mathcal{NP} .

A graph with n vertices can be represented as an $n \times n$ binary matrix which has a 1 in position (i, j) if and only if there is an edge (v_i, v_j) . If you “unroll” this matrix (say by rows), you will have a vector of n^2 bits and you can consider this to be a number in standard binary notation. So, there is a correspondence between n vertex graphs and n^2 bit numbers. If we re-label the vertices of the graph, we don’t change the graph properties. Different re-labelings of the graph will (usually) give different numbers. Clearly among all re-labelings of the graph, there is some re-labeling which gives the smallest value for this binary number. We would like to represent a graph by the minimum number we can get by re-labeling. We’ll call this minimal number the *canonical number* of the graph. It’s easy to see that two graphs are isomorphic **iff** they have the same canonical number.

EXERCISE: Show that if finding the *canonical number* of a graph is easy, then GRAPH ISOMORPHISM is easy.

However, *canonical number* may be harder than GRAPH ISOMORPHISM. If I can tell that two graphs are NOT isomorphic, I know that their canonical numbers are different, but I don’t know what their canonical numbers are. Further, if I know that two graphs are isomorphic, I know that their canonical numbers are identical, but again I don’t know what these canonical numbers are.

EXERCISE: IS-CANONICAL

INPUT: A graph G and an integer I .

QUESTION: Is $I \geq$ the canonical number of G ?

Show that IS-CANONICAL is in $\text{co-}\mathcal{NP}$.

EXERCISE: ORDER-CANONICAL

INPUT: A graph G and an integer I .

QUESTION: Is the canonical number of G less than I ?

Show that ORDER-CANONICAL is in \mathcal{NP} .

EXERCISE: The graph $v_1 - - - v_2 - - - v_3$ is isomorphic to $v_1 - - - v_3 - - - v_2$ and is also isomorphic to $v_2 - - - v_1 - - - v_3$.

Find the canonical number of $v_1 - - - v_2 - - - v_3$.

Assume that you have an algorithm $\mathbf{YS}(\)$ so that when you input a Boolean expression \mathbf{E} ,
 $\mathbf{YS}(\mathbf{E})$ outputs *YES* if \mathbf{E} is satisfiable, and
 $\mathbf{YS}(\mathbf{E})$ outputs *NO* if \mathbf{E} is not satisfiable.

EXERCISE: Show how to use $\mathbf{YS}(\)$ to construct an algorithm $\mathbf{FIND}(D(x_1, \dots, x_n))$ which when given a satisfiable Boolean expression $D(x_1, \dots, x_n)$, returns an assignment $x_1 = a_1, x_2 = a_2, \dots, x_n = a_n$, so that $D(a_1, \dots, a_n)$ is *TRUE*.

EXERCISE: Assume that $\mathbf{YS}(D(x_1, \dots, x_n))$ has run time $\mathcal{O}(n^k)$ and find the run time of $\mathbf{FIND}(D(x_1, \dots, x_n))$.

EXERCISE: Use what we've learned about nonnegative difference equations to show that:

$$\sum_{i=0}^n i^K = \Theta(n^{K+1}).$$

EXERCISE: To multiply polynomials faster, Prof. Nutt suggested breaking the polynomials into 4 quarters instead of into 2 halves. To beat the three half-size multiplication algorithm, how many quarter size multiplications does Prof. Nutt have to have in his algorithm.

EXERCISE: Charlie Tuna found a theorem in a book that said:

For the difference equation:

$$x_n = x_{n-1} + x_{n-2},$$

every solution can be written in the form

$$x_n = \alpha F_n + \beta F_{n-1},$$

where α and β are constants and F_n and F_{n-1} are respectively the n^{th} and $(n-1)^{\text{st}}$ Fibonacci numbers.

Charlie concluded that if x_n is a solution to $x_n = x_{n-1} + x_{n-2}$, then $x_n = \Theta(\lambda_0^n)$ where $\lambda_0 = (1 + \sqrt{5})/2$. Freddy Flounder retorted that Charlie was wrong, but couldn't explain why. Please help out Charlie and Freddy by finding an example x_n which satisfies the difference equation but does **NOT** have $x_n = \Theta(\lambda_0^n)$. (Harry Halibut said $x_n = 0$ is a solution with $\alpha = 0$ and $\beta = 0$, but Charlie said that Harry's example didn't count.)

Also, give an *EXTRA* hypothesis, so that Charlie's claim would be true.

s-t Hamiltonian Path:

INPUT: A graph G and two specified vertices s and t .

QUESTION: Does G have a Hamiltonian Path which starts at s and ends at t ?

EXERCISE: Assume that you know that Hamiltonian Circuit is \mathcal{NP} -Complete, show that s-t Hamiltonian Path is \mathcal{NP} -Complete.

EXERCISE: Assume that you know that s-t Hamiltonian Path is \mathcal{NP} -Complete, show that Hamiltonian Circuit is \mathcal{NP} -Complete.

EXERCISE: Show that TSP (**YES/NO** version) with all edge weights in $\{1, 2\}$ is \mathcal{NP} -Complete. (You should assume that Hamiltonian Circuit is \mathcal{NP} -Complete.)

INPUT: A disk array $D[1], D[2], \dots, D[n]$ with each $D[I]$ in $\{A, B, C\}$.

QUESTION: Does the configuration defined by the array $D[1..n]$ ever occur in the minimal move solution to the Towers of Hanoi problem of moving n disks from tower A to tower C ?

EXERCISE: Give a *subtract-and-conquer* algorithm for this decision version of Towers of Hanoi.

EXERCISE: Calculate the run time for your algorithm for this decision version of Towers of Hanoi.

EXERCISE:

```
PROCEDURE TWO(A, n, BIG, SEC )
  IF n = 2
    THEN IF A[1] ≥ A[2]
           THEN BIG := A[1], SEC := A[2]
           ELSE BIG := A[2], SEC := A[1]
  IF n > 2
    THEN split A into two half-size arrays A1 and A2
           TWO(A1, n/2, B1, S1 )
           TWO(A2, n/2, B2, S2 )
           IF B1 ≥ B2
             THEN IF B2 ≥ S1
                    THEN BIG := B1, SEC := B2
                    ELSE BIG := B1, SEC := S1
             ELSE IF B1 ≥ S2
                    THEN BIG := B2, SEC := B1
                    ELSE BIG := B2, SEC := S2
```

Assume that \mathbf{A} is an \mathbf{n} element array. Assume that \mathbf{n} is a power of 2. Write down a *Difference Equation* for the number of array element comparisons used by this algorithm. (Do **NOT** count comparisons involving \mathbf{n} .) Write down appropriate *Initial Conditions*.

Solve your difference equation to find the number of array element comparisons as a function of \mathbf{n} .

EXERCISE: Give an inductive proof that this algorithm, **TWO**, correctly finds the two largest elements in the array \mathbf{A} . (When \mathbf{n} is a power of 2.)