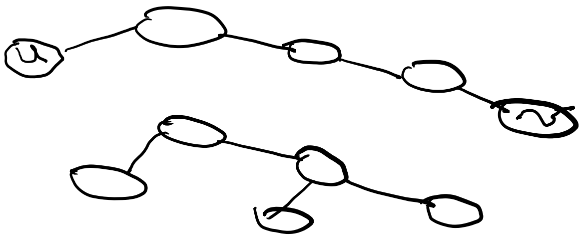


Professor Ruby's solutions

Prove that Hamiltonian Path is a member of NP:

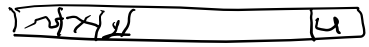
Need to analyze the verify algorithms to show that they run in polynomial time!

Version 1 using the BFS Template

1. Randomly select n-1 the edges and put into Sequence T

```
Algorithm verifyHP(G, u, v, T)
        for each e in G.edges() do
           setInT(e, NO)
        for each e in T.elements() do  // label the edges in T
           setInT(e, YES)
        (isConnected, maxDeg) <- BFS(G)
        if isConnected /\ getDegree(u) = 1 /\ getDegree(v) = 1 /\ maxDeg = 2 then
           return yes
        else return NOT_A_Solution

Algorithm initResult(G)
        maxDeg <- 0
        components <- 0
Algorithm result(G)
        return (components=1, maxDeg)
Algorithm postInitEdge(e)
        if getInT(e) = NO then
                setLabel(e, SKIP)
Algorithm preComponentVisit(G, v)
        components <- components + 1
Algorithm preVertexVisit(G, v)
        degree <- 0
Algorithm unexploredEdgeVisit(G, v, e, w)
        degree <- degree + 1
Algorithm postVertexVisit(G, v)
        setDegree(v, degree)
        maxDeg <- max(maxDeg, degree)
```



Version 2 without using the Template:

1. Randomly select all vertices in G once and put into Sequence T (i.e., a permutation of vertices in G that will be the possible Hamiltonian Path that we will check)

```
Algorithm verifyHP2(G, u, v, T)
    if T.first().element() != u ∨ T.last().element() != v then
        return NOT_A_Solution
    p <- T.first()
    while p != T.last() do
        x <- p.element()
        p <- T.after(p)
        y <- p.element()
        if !G.areAdjacent(x, y) then
            return NOT_A_Solution
    return yes
```

Version 3 using the BFS Template (Inspired by Abdalgalil Amin Abdalgalil Mustafa)

1. Randomly select n-1 the edges and put into Sequence T

```
Algorithm verifyHP(G, u, v, T)
    start = u  // start is a subclass field
    for each e in G.edges() do
        setInT(e, NO)
    for each e in T.elements() do  // label the edges in T
        setInT(e, YES)
    BFS(G)
    if getDepth(v) = G.numVertices()-1 then
        return yes
    else return NOT_A_Solution

Algorithm isNextComponent(G, v)
    return v = start  // start is a subclass field = u
Algorithm postInitVertex(v)
    setDepth(v, 0)
Algorithm postInitEdge(e)
    if getInT(e) = NO then
        setLabel(e, SKIP)
Algorithm preDiscEdgeVisit(G, v, e, w)
    setDepth(w, getDepth(v) + 1)
```

Prove that Longest Path is a member of NP:

1. Select all vertices of G once and put into Sequence T (subset of vertices in G, all unique, no vertex more than once)

```
Algorithm verifyLP(G, u, v, min, T)
1       if T.first().element() != u ∨ T.last().element() != v then
1               return NOT_A_Solution
1       p <- T.first()
1       sum <- 0
n       while p != T.last() do
n               x <- p.element()
n               p <- T.after(p)
n               y <- p.element()
n               areAdjacent <- false
m               for each e in G.incidentEdges(x) do  // find the edge connecting x and y if it exists
m                       if G.opposite(x, e) = y then
n                               sum <- sum + weight(e)
n                               areAdjacent <- true
1               if !areAdjacent then
1                       return NOT_A_Solution
1       if sum ≥ min then
1               return yes
1       else return NOT_A_Solution
```

Therefore, LP is a member of NP.

Reduce HP to LP

(G,u,v) -> (G,u,v,MIN)

```
Algorithm reduceHP2LP(G,u,v)
        for each e in G.edges() do
                setWeight(e, 1)
        return (G,u,v,G.numVertices()-1)
```

Any longest path of length n-1 must be a Hamiltonian Path (since edge weights are 1).

Therefore, LP is NPH (because of the reduction of HP to LP) and LP is NP (because of the polynomial time verifier), so LP is a member of NPC.

Homework 16: Show that SAT is a member of NPC.
Exp is my boolean formula.

1. Randomly assign true/false to each variable in Exp and
   put in a Dictionary D that maps (var->{true, false})

Algorithm verifySAT(Exp, D)
    Evaluate Exp using the mapping D for the variables in Exp.
    If evaluates to true, then return yes else return NOT_A_Solution.
    Runs in polynomial time.
Therefore, SAT is in NP.
Algorithm reduce Circuit-SAT2SAT(circuit)
The and-gates become and /\-operators, the or-gates become \/-operators, not-gates become !-operator. The inputs are assigned a variable name. Return the logical expression.

Therefore, SAT is NPH and since SAT is NP, therefore SAT is NPC!!!

A -> B
(A, B) ➔ (easy, easy), (easy, hard), (hard, hard), not possible for (hard, easy)
B is at least as hard as A. Why?

Solution to Homework problem C13-2 of Assignment 15.

Accepter for M={5}
Algorithm acceptM(x)
    if x = 5 then return yes
    else return no

L member of P implies there exists an accepter for L that runs in $O(n^k)$, so let's call it verifyL (members of P do not need a guess in the interface since they can ignore it and answer yes or no deterministically).
Algorithm reduceL2M(x) // let x be an arbitrary instance of L
    if verifyL(y) = yes then
        return 5
    return 10