NP-Complete

CSCI 570

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

DISCUSSION 12

HTTP://WWW-SCF.USC.EDU/~CSCI570

Outline

- Problems with Solutions

Problem 1

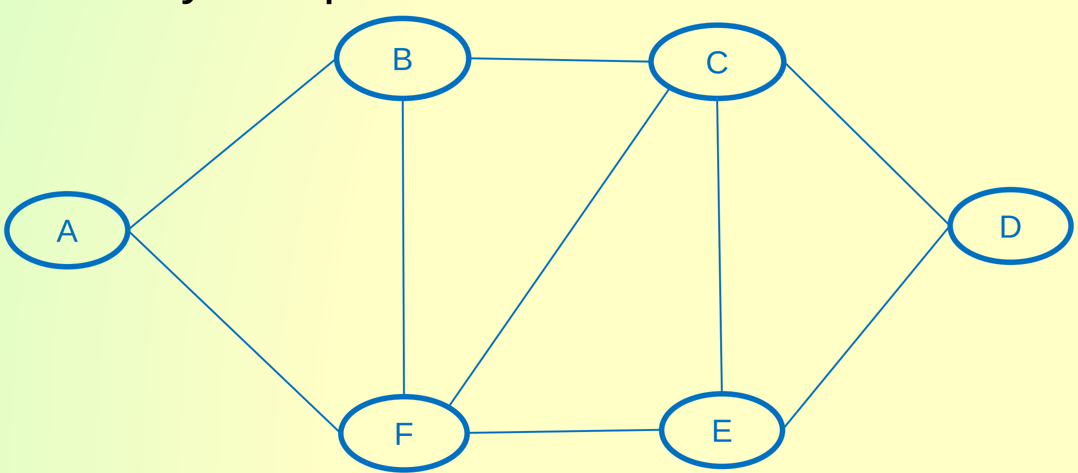
- The *Set Packing* problem is as follows. We are given m sets $S_1, S_2, \ldots S_m$ and an integer k. Our goal is to select k of the m sets such that none of the selected sets have any elements in common. Prove that this problem is **NP**-Complete.

Solution 1

- To prove a problem is NP complete, you must prove two things:
  - The problem is in NP – a certificate can be verified in polynomial time
  - The problem is NP-Hard with respect to NPC problems – given a solution to this problem, convert a known NPC problem to this problem in polynomial time
- To prove the Set Packing problem is in NP, we need to show we can verify a certificate in polynomial time.
- The certificate is a set C of sets
  - We can confirm in polynomial time that for each pair i,j in C, S_i and S_j have no common elements
  - Order the sets from 1 to k
  - Compare S_1 to $S_2..S_k$(k-1 set comparisons), then S_2 to $S_3..S_k$(k-2 set comparisons), then S_3 to $S_4..S_k$(k-3 set comparisons) all the way to S_{k-1} to S_k (1 set comparison)
  - This gives us (k(k+1) / 2) – k = $O(k^2)$ set comparisons

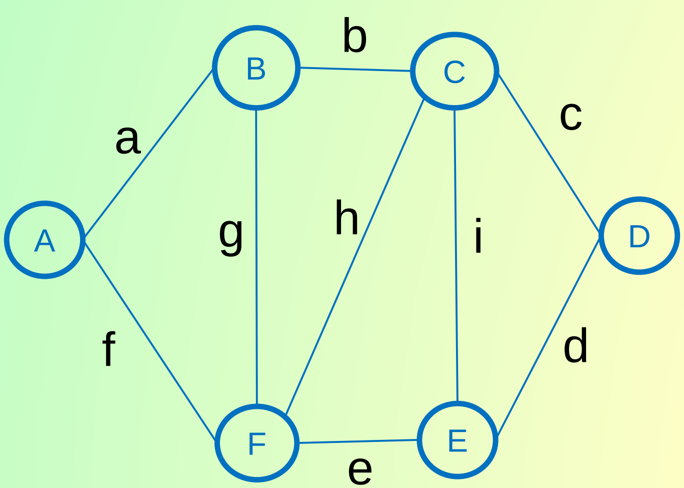
Solution 1

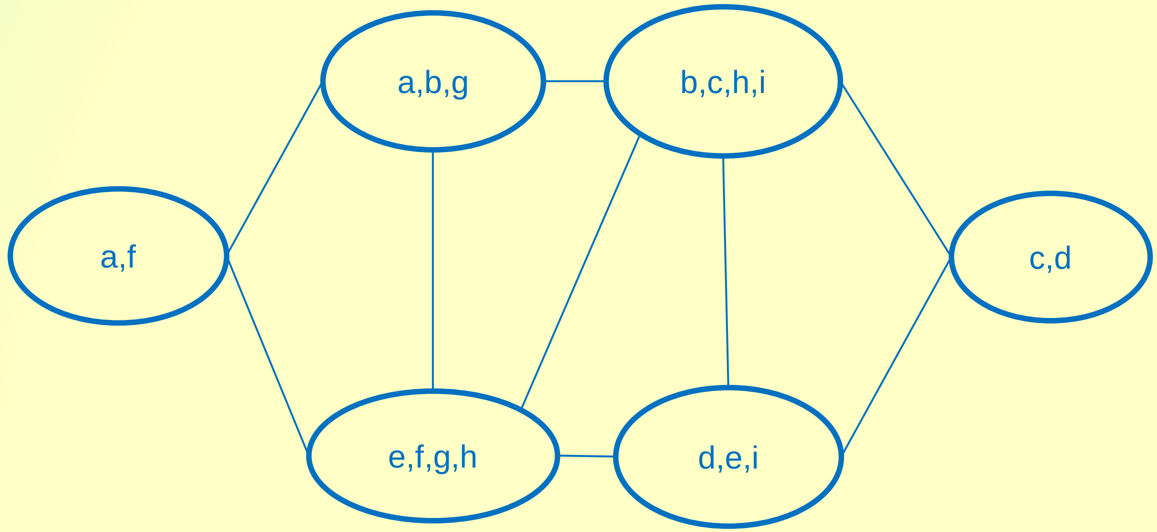
- To prove the Set Packing problem is in NPC, we need to show we can convert an existing NPC problem to the Set Packing problem in polynomial time
- Let's convert Independent Set to Set Packing
  - Independent Set provides a set of vertices in a graph in which no two vertices are adjacent
  - What independent sets exist in the following graph?
    - {A}, {B}, {C}, {D}, {E}, {F}
    - {A, C}, {A, D}, {A, E}, {B, E}, {B, D}, {D, F}
    - Are there any independent sets of size 3?



Solution 1

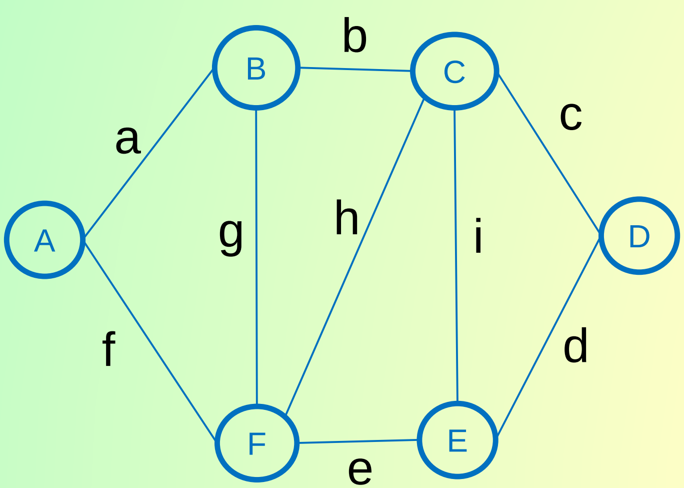
- To convert Independent Set to Set Packing, we assume we are given the Independent Set problem and have a solution to the Set Packing problem
  - We then need to convert the Independent Set problem to the Set Packing problem in polynomial time
- For each vertex in the independent set graph, create a set
  - Label each edge with a unique identifier
  - Make the elements in each set equal to the labels of the incident edges

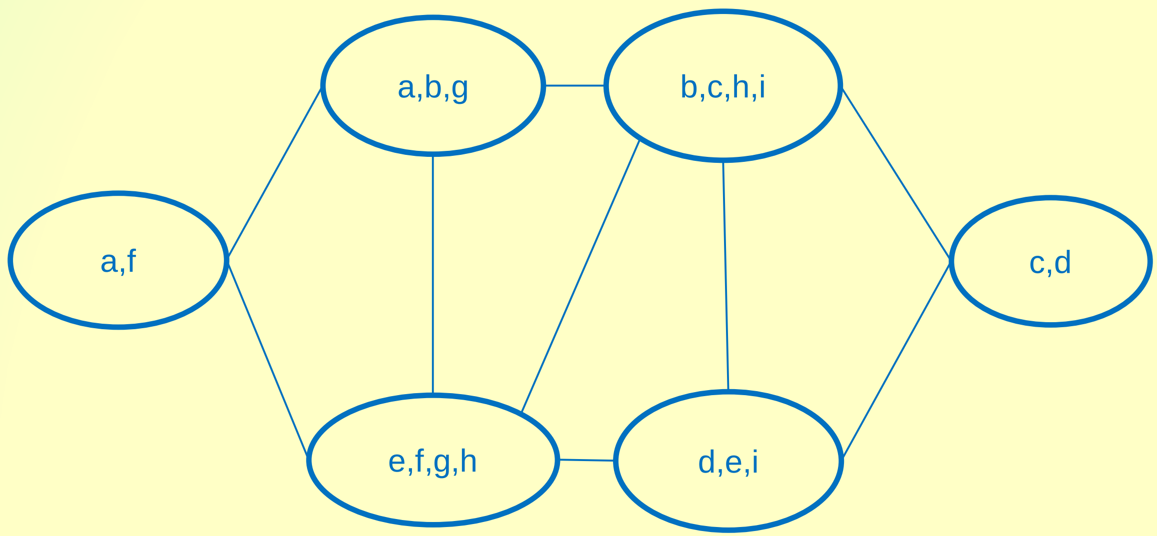




Solution 1

- A set packing of size k corresponds exactly to an independent set of size k
- Which sets can we choose that do not have any elements in common?
  - {A}={{a,f}}, {B}={a,b,g}}, {C}={{b,c,h,i}}, {D}={{c,d}}, {E}={{d,e,i}}, {F}={{e,f,g,h}}
  - {A,C}={{a,f},{b,c,h,i}, {A,D}={{a,f},{c,d}}, {A,E}={{a,f},{d,e,i}}, {B,E}={{a,b,g},{d,e,i}}, {B,D}={{a,b,g},{c,d}}, {D,F}={{c,d},{e,f,g,h}}
  - **Notice that there are no common edges in any of the above sets**





Problem 2

- Consider the partial 3-SAT problem, denoted as 3-SAT(α). We are given a collection of *k* clauses, each of which contains exactly three literals, and we are asked to determine whether there is an assignment of true/false values to the literals such that *at least* αk clauses will be true. Note that 3-SAT(1) is exactly the 3-SAT problem from lecture.
- Prove that 3-SAT(15/16) is **NP**-Complete.
- Consider the following as an example

  (a | b | !c) & (!a | !b | !d) & (!a | b | d) & (!b | !c | !d) &
  (a | c | !d) & (!a | b | d) & (b | c | !d) & (b | !c | d)

Solution 2

(a | b | !c) & (!a | !b | !d) & (!a | b | d) & (!b | !c | !d) & (a | c | !d) & (!a | b | d) & (b | c | !d) & (b | !c | d)

Clause 1

Clause 8

					Clause							
a	**b**	**c**	**d**		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	0	0		1	1	1	1	1	1	1	1
0	0	0	1		1	1	1	1	0	1	0	1
0	0	1	0		0	1	1	1	1	1	1	0
0	0	1	1		0	1	1	1	1	1	1	1
0	1	0	0		1	1	1	1	1	1	1	1
0	1	0	1		1	1	1	1	0	1	1	1
0	1	1	0		1	1	1	1	1	1	1	1
0	1	1	1		1	1	1	0	1	1	1	1
1	0	0	0		1	1	0	1	1	0	1	1
1	0	0	1		1	1	1	1	1	1	0	1
1	0	1	0		1	1	0	1	1	0	1	0
1	0	1	1		1	1	1	1	1	1	1	1
1	1	0	0		1	1	1	1	1	1	1	1
1	1	0	1		1	0	1	1	1	1	1	1
1	1	1	0		1	1	1	1	1	1	1	1
1	1	1	1		1	0	1	0	1	1	1	1

Solution 2

- To prove the 3-SAT(α) problem is in NP, we need to show we can verify a certificate in polynomial time.
- The certificate is a given a truth value assignment
- We can count how many clauses are satisfied and compare it to 15k / 16 in polynomial time

Solution 2

- To prove the 3-SAT(α) problem is in NPC, we need to show we can convert an existing NPC problem to the 3-SAT(α) problem in polynomial time.
- Let's convert 3-SAT to 3-SAT(α)
  - Assume we have a solution to 3-SAT(α) and a 3-SAT problem
- If I add 3 new variables, I can add 8 clauses so that only 7 of those clauses can be satisfied (all 8 distinct possible clauses)

  (x | y | z), (!x | y | z), (x | !y | z), (x | y | !z),
  (!x | !y | z), (!x | y | !z), (x | !y | !z), (!x | !y | !z)
- If I add all 8 of these clauses k/8 times (where k is the number of original clauses in 3-SAT), I will have 2k clauses, of which 15k/16 can be satisfied *if and only if* 100% of the original *k* could be satisfied.

Solution 2

(a | b | !c) & (!a | !b | !d) & (!a | b | d) & (!b | !c | !d) &
(a | c | !d) & (!a | b | d) & (b | c | !d) & (b | !c | d)

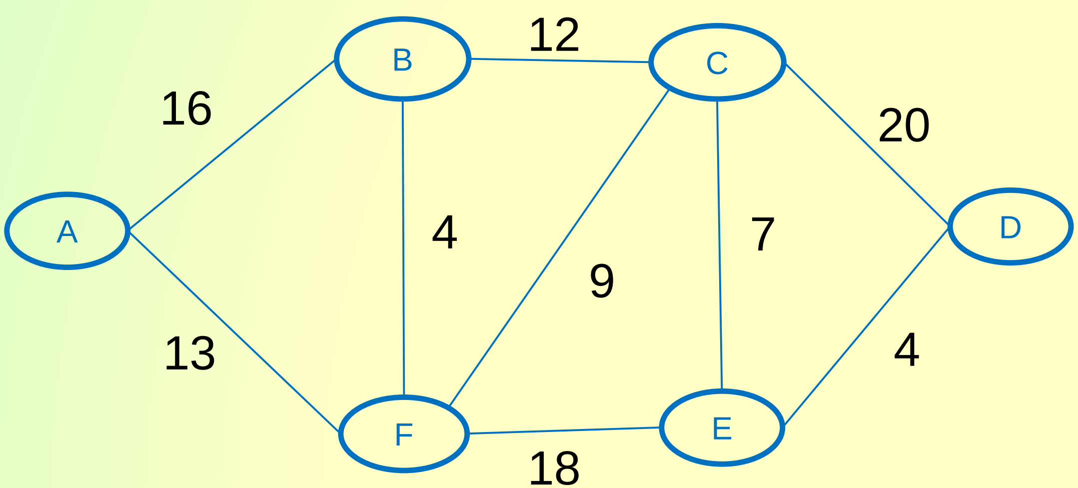
- If I add all 8 of the following clauses to the above equation k/8 times (where k=8 in this case), one of the 8 just added will always be false

  (a | b | c) & (!a | b | c) & (a | !b | c) & (a | b | !c) &
  (!a | !b | c) & (!a | b | !c) & (a | !b | !c) & (!a | !b | !c)

- Then only if the original formula was satisfiable will I ever end up with 15/16 of the clauses being satisfiable for a given arrangement of values

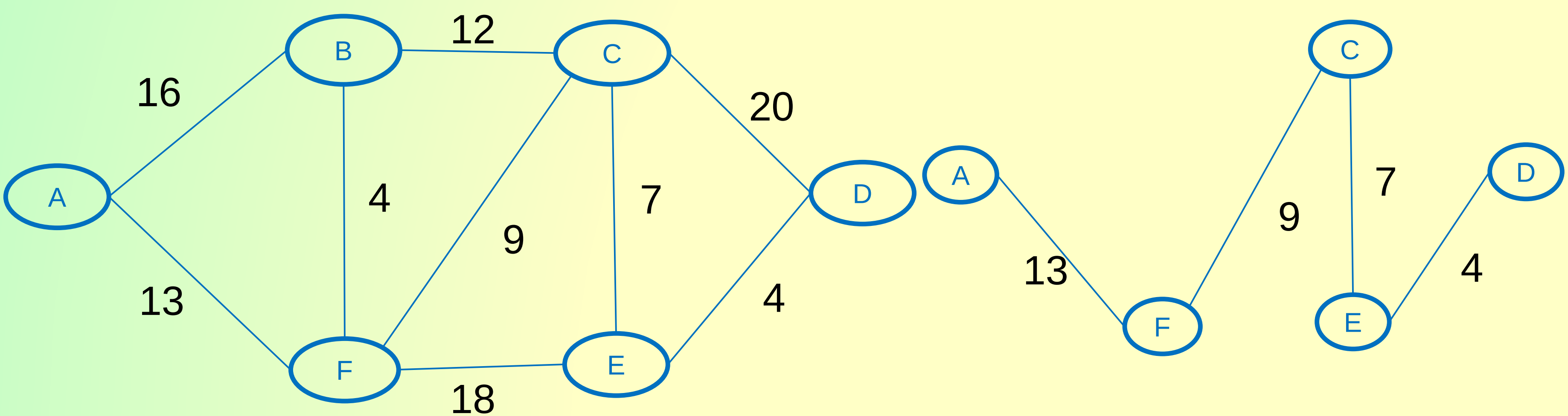
Problem 3

- The *Steiner Tree* problem is as follows. Given an undirected graph *G=(V,E)* with nonnegative edge costs and whose vertices are partitioned into two sets, *R* and *S*, find a tree $T \subseteq G$ with total cost at most C such that for every *v* in *R*, *v* is in *T*. That is, the tree T contains every vertex in *R* (and possibly some in *S*) with a total edge cost of at most *C*.
- Prove that this problem is **NP**-Complete.
- Let's first understand Steiner Trees a little better and see if we can find one in the following graph with R={A,D,F} and C=34.



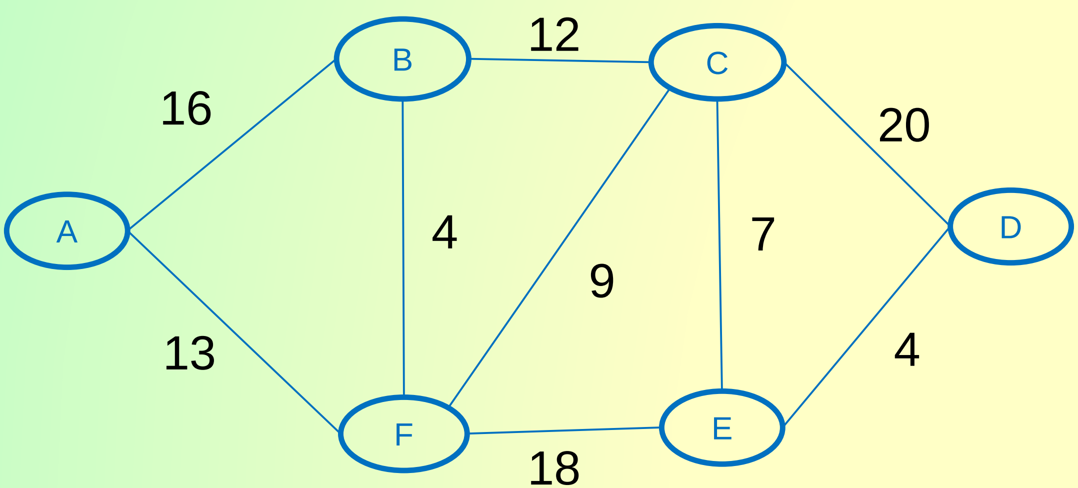
Solution 3

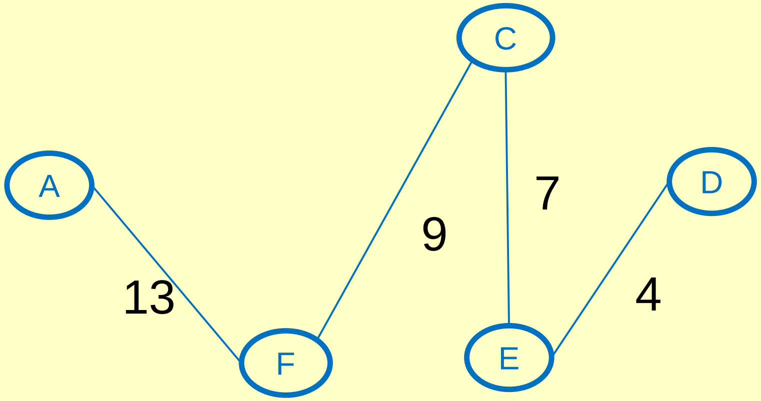
- Let’s first understand Steiner Trees a little better and see if we can find one in the following graph with R={A,D,F} and C=34.
- We have included A, D, and F in the tree, but we also needed to include C and E.
- The total cost is 33, which is less than 34, so we have found a Steiner tree with the given constraints.



Solution 3

- To prove the Steiner Tree problem is in NP, the certificate will be the tree itself. Confirm every vertex in the tree is in R and that the total edge weight is at most C. This can be done in O(R)

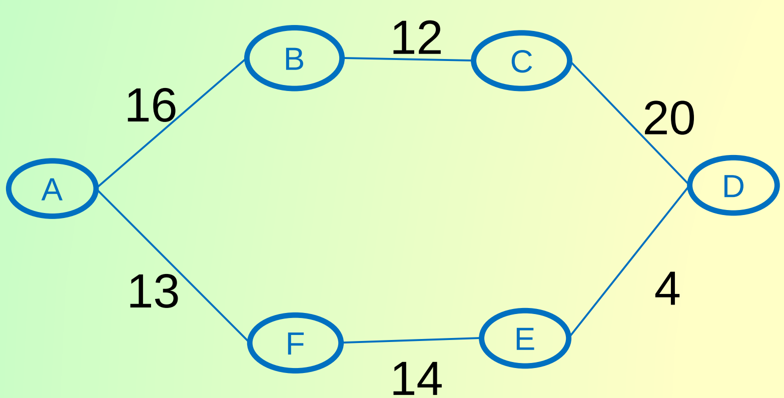


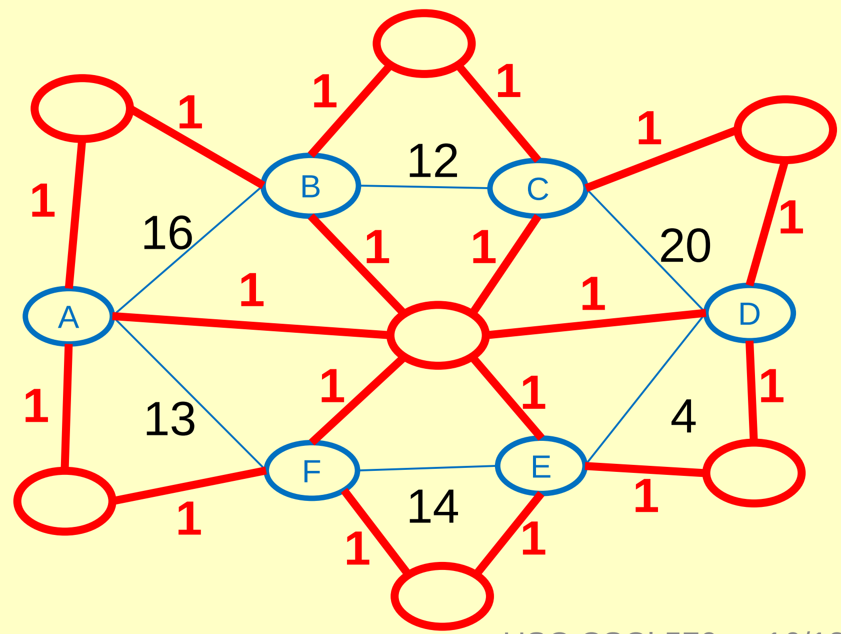


Solution 3

- To prove the Steiner Tree problem is in NPC, assume we have a solution to the Steiner Tree problem. Let’s convert the Vertex Cover problem to a Steiner Tree problem.
- We can create a new graph G’ that starts with a copy of G and adds:
  - for each edge, a new vertex connected to the two endpoints (which will be m new vertices)
  - a new vertex connected to all of the original vertices
  - Give every new edge a cost of one
  - Set the required vertex set R to be only the new vertices
  - This new graph has a Steiner tree of cost C = m+k if and only if the original graph had a vertex cover of size k

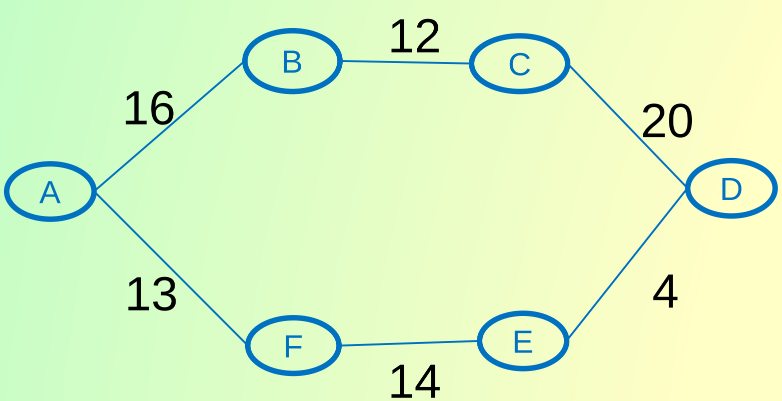
To simplify, consider the following graph instead of the one on the previous slide

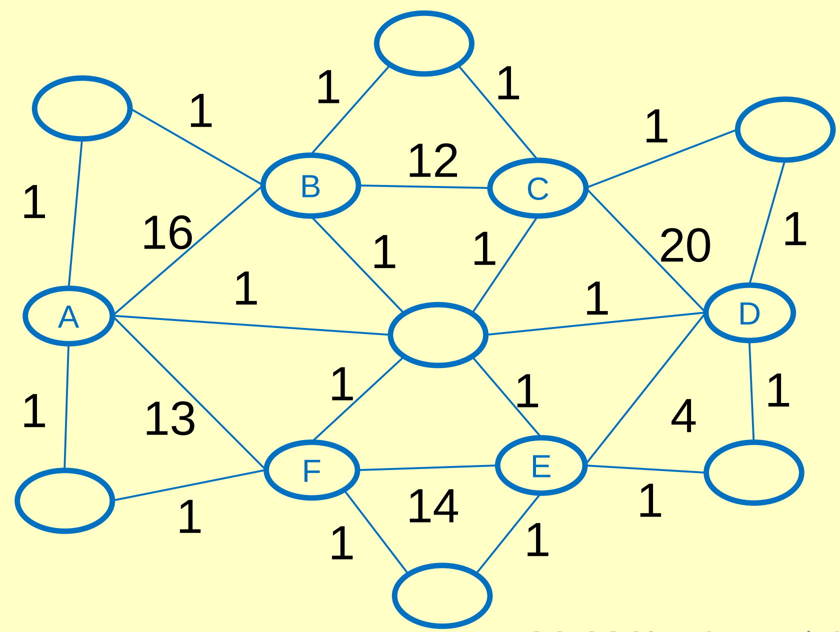




Solution 3

- Set the required vertex set R to be only the new vertices. This new graph has a Steiner tree of cost C = m+k if and only if the original graph had a vertex cover of size k.
  - m edges will be necessary to connect each new one on the “between” edges to existing vertices, and can connect to k original vertices
  - those k original vertices then each connect to the last added new vertex with one edge each to form a tree connecting all new vertices
  - those k incident original vertices constitute the vertex cover in the original graph.





Solution 3

- Suppose we wanted to find a vertex cover of size k=3 in the original graph on the left. One vertex cover set is {A, C, E}.
- For the Steiner Tree, look at the graph on the right.
  - Add the vertex in the middle. We will need k edges to connect it to the vertex cover of size k, should one exist.
  - We can then connect each of the *m* new vertices on the outside to the relevant vertex cover, thus making all outer nodes connected.
  - If all of added nodes make a connected tree (which has a cost of m+k), then there was a vertex cover of size k in the original graph.
  - R={new nodes that were added}