Algorithm

```
VertexCover(graph G, int k)
     if(G has no edges) //we've covered them all!
          return true
     if(k < 0)
          return false
     H1 = copy of G
     H2 = copy of G
     pick any edge (u,v)
     H1 = H1.remove(u) //removes u and all edges with u as
an endpoint
     H2 = H2.remove(v) //removes u and all edges with u as
an endpoint
     return VertexCover(H1,k-1) || VertexCover(H2, k-1)
```

Running Time

Recurrence: $T(k) = \begin{cases} 2T(k-1) + O(n+m) & \text{if } k \geq 1 \\ O(1) & \text{if } k < 0 \end{cases}$

Running time? Unroll or use recursion tree

Approximation Ratio

For a minimization problem (find the shortest/smallest/least/etc.)

If $OPT(\mathrm{G})$ is the value of the best solution for G, and $ALG(G)$ is the value that your algorithm finds, then ALG is an α approximation algorithm if for every G,

$$\alpha \cdot OPT(G) \geq ALG(G)$$

i.e. you're within an α factor of the real best.

Vertex Cover LP

Minimize $\sum w(u) \cdot x_u$

Subject to:

$x_u + x_v \geq 1$ for all $(u, v) \in E$

$0 \leq x_u \leq 1$ for all u.

Don't worry about the weights for today.

We got an exact solution for bipartite graphs….