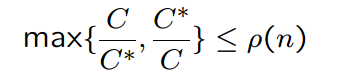
C为approxiamation algorithm的解

C\*为最优解

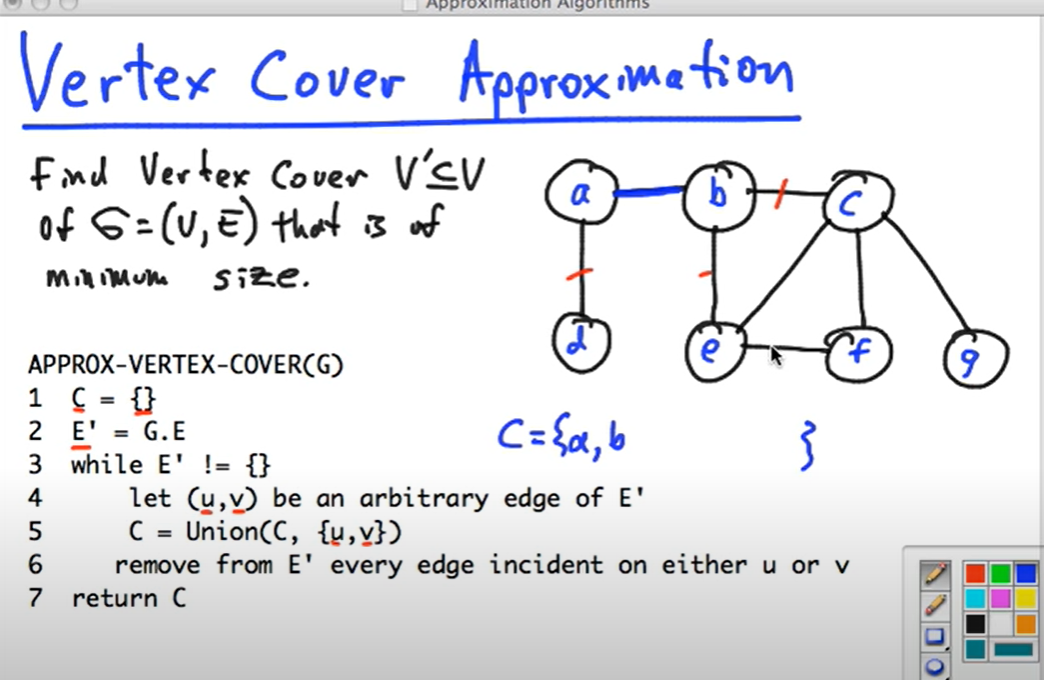


如果在一个比值下，叫做

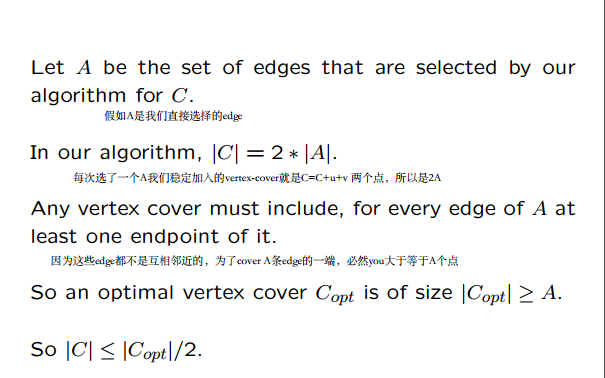


ρn有时候并不是一个固定常数，也有可能跟着n变化

Vertex Cover



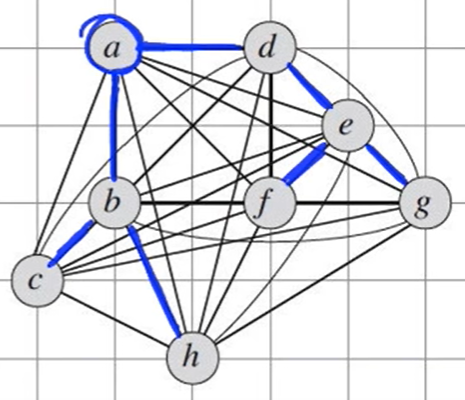
任选一条边，C加上uv两个点， 移去uv接触到的任意一条edge,例如ab,ad,be,bc



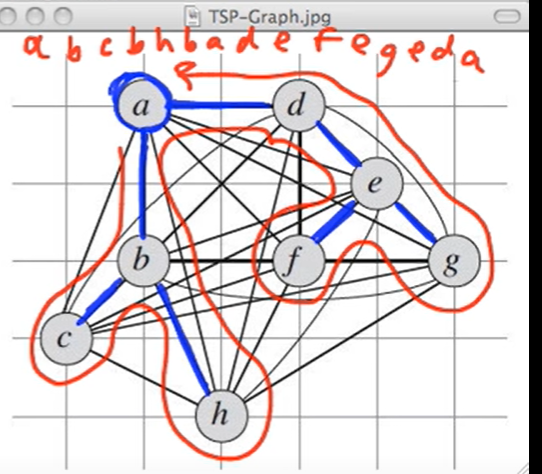
Traveling sales man,

给你一个complete graph,每个cost c，找到最小值，

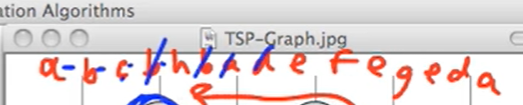
第一步，找到minimum spanning tree //KRUSKAL PRIM-JARNIK都行

 //因为他已经是minimum spanning tree了，他必然是lower bound,这是经过所有点edge最短的方法，

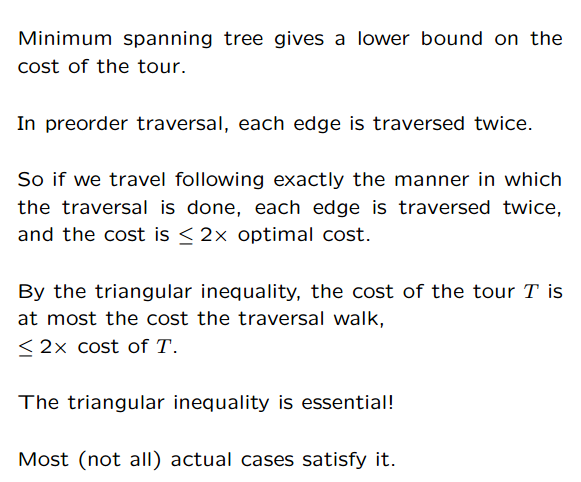
第二步，对spanning tree进行preorder traversal

每一条edge最多被浏览两次，最少一次

第三步，利用三角定理对重复的点修饰



重复的直接划，cb+bh>=ch



Set Covering Problem

Subset sum

An [approximate](https://en.wikipedia.org/wiki/Approximation_algorithm) version of the subset sum would be: given a set of {\displaystyle N} numbers {\displaystyle x\_{i},\ldots ,x\_{N}} and a number {\displaystyle s}, output:

* Yes, if there is a subset that sums up to {\displaystyle s}.
* No, if there is no subset summing up to a number between {\displaystyle (1-c)s} and {\displaystyle s} for some small {\displaystyle c>0}.
* Any answer, if there is a subset summing up to a number between {\displaystyle (1-c)s} and {\displaystyle s} but no subset summing up to {\displaystyle s}.

If all numbers are non-negative, the approximate subset sum is solvable in time polynomial in {\displaystyle N} and {\displaystyle 1/c}.

The solution for subset sum also provides the solution for the original subset sum problem in the case where the numbers are small (again, for non-negative numbers). If any sum of the numbers can be specified with at most {\displaystyle P} bits, then solving the problem approximately with {\displaystyle c=2^{-P}} is equivalent to solving it exactly. Then, the polynomial time algorithm for approximate subset sum becomes an exact algorithm with running time polynomial in {\displaystyle N} and {\displaystyle 2^{P}} (i.e., exponential in {\displaystyle P}).

The algorithm for the approximate subset sum problem is as follows:

initialize a list *S* to contain one element 0.

**for each** *i* from 1 to *N* **do**

let *T* be a list consisting of *xi* + *y*, for all *y* in *S*

let *U* be the union of *T* and *S*

sort *U*

make *S* empty

let *y* be the smallest element of *U*

add *y* to *S*

**for each** element *z* of *U* in increasing order **do**

// Trim the list by eliminating numbers close to one another

// and throw out elements greater than *s*.

**if** *y* + *cs*/*N* < *z* ≤ *s* **then**

*y* = *z*

add *z* to *S*

**if** *S* contains a number between (1 − *c*)*s* **and** *s* **then**

return *yes*

**else**

return *no*

The algorithm is polynomial time because the lists {\displaystyle S}, {\displaystyle T} and {\displaystyle U} always remain of size polynomial in {\displaystyle N} and {\displaystyle 1/c} and, as long as they are of polynomial size, all operations on them can be done in polynomial time. The size of lists is kept polynomial by the trimming step, in which we only include a number {\displaystyle z} into {\displaystyle S} if it is greater than the previous one by {\displaystyle cs/N} and not greater than {\displaystyle s}.

This step ensures that each element in {\displaystyle S} is smaller than the next one by at least {\displaystyle cs/N} and do not contain elements greater than {\displaystyle s}. Any list with that property consists of no more than {\displaystyle N/c} elements.

The algorithm is correct because each step introduces an additive error of at most {\displaystyle cs/N} and {\displaystyle N} steps together introduce the error of at most {\displaystyle cs}.