CS630 Graduate Algorithms

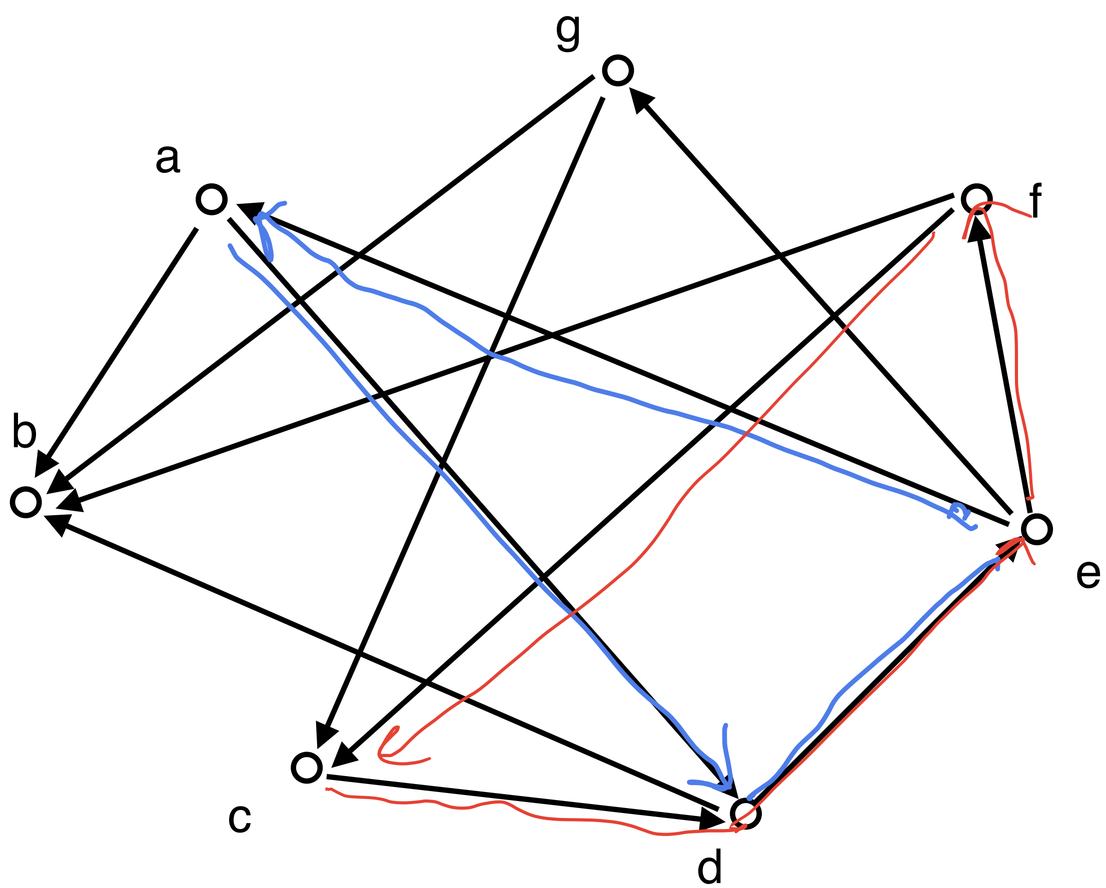
September 24, 2024

by Dora Erdos and Jeffrey Considine

- approximation algorithms
  - vertex cover 2-approximation
  - set cover
  - vertex cover ln n - approx
  - dominating set
  - independent set

slides by Dora Erdos

Acyclic Subgraph

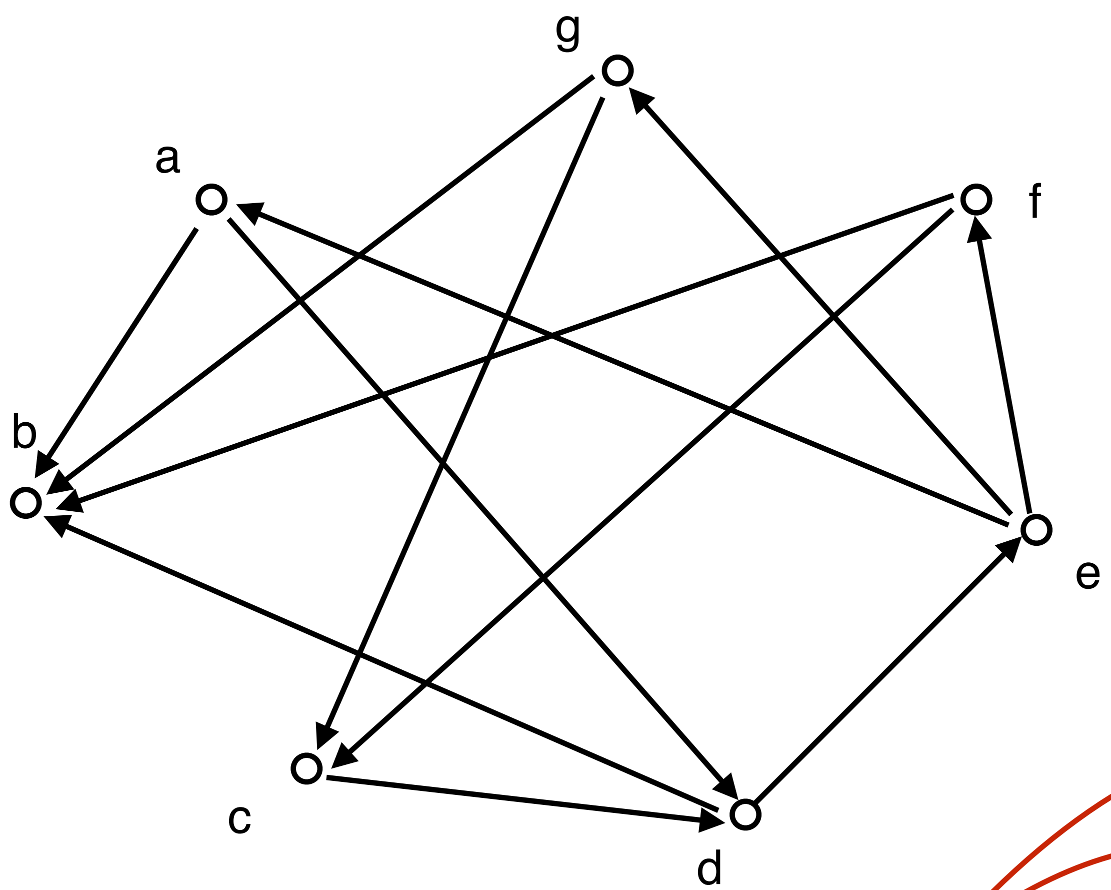


directed cycle:
a sequence of directed edges that start and finish in the same vertex

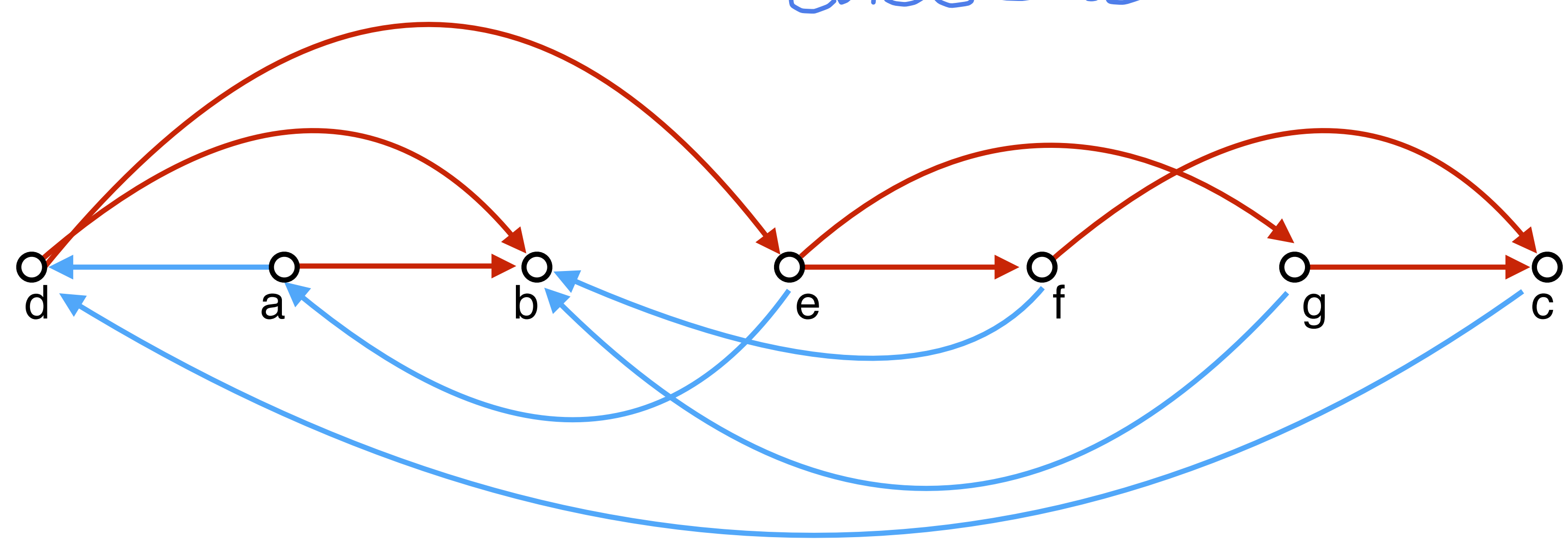
Given graph G, find it's largest acyclic subgraph:

delete some of its edges, such that the remaining graph doesn't contain any directed cycles and has the maximum number of edges.

Acyclic Subgraph



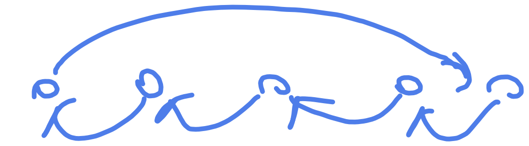
Why draw nodes in this particular order?
– no reason!
chose at random

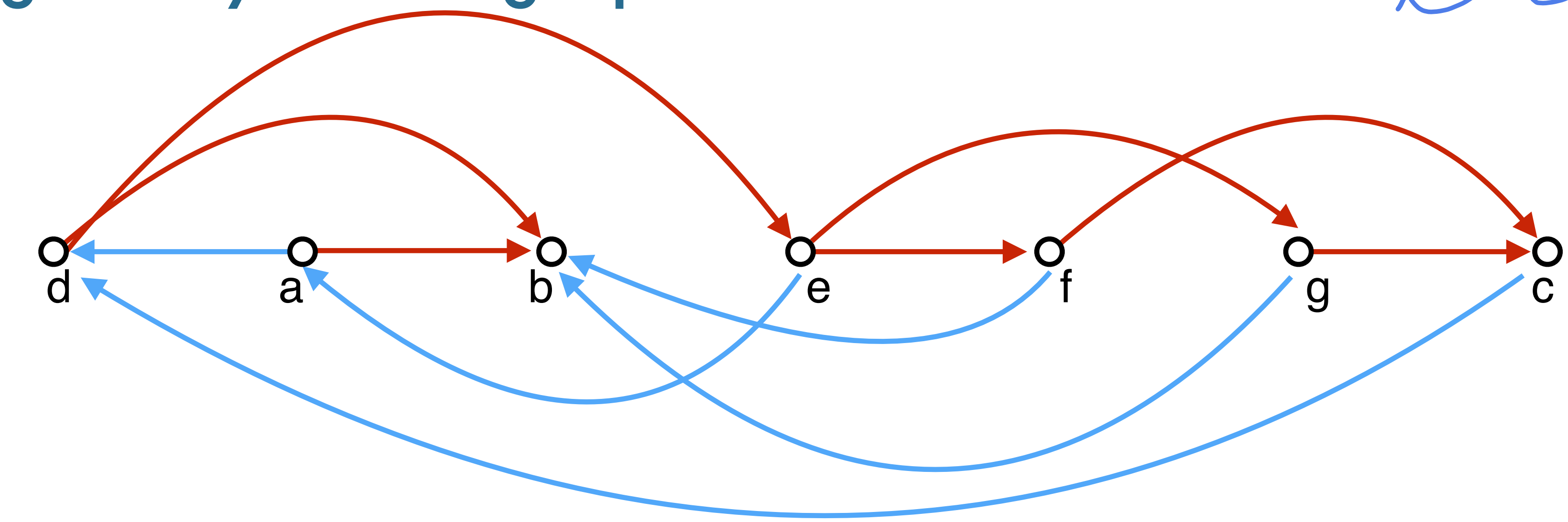


Given graph G, find it's largest acyclic subgraph:

delete some of its edges, such that the remaining graph doesn't contain any directed cycles and has the maximum number of edges.

TopHat - largest acyclic subgraph





Select all true statements for a graph created by the described process.

~~A.~~ it is possible to have a cycle with only red edges

red edges point in the same direction → no way of going back to the first node

✓ B. any cycle in G has to have both red and blue edges

~~C.~~ at least half of the edges in G is red

✓ D. at least half of the edges in G are the same color

if both of them are less than half, then their union wouldn't make up the whole.

✓ E. the largest acyclic subgraph in G has at least $\frac{|E|}{2}$ edges

subgraph of same color is acyclic.

Vertex Cover 2x-optimal greedy algorithm

Vertex Cover: Given a graph G(V,E) find the smallest subset of vertices S, such that it forms a vertex cover. That is, every edge (u,v) has at least one of its nodes in S.

Algorithm 1: GreedyVC($G(V, E)$)

$S \leftarrow$ empty set of vertices;
for (u, v) *is an edge* **do**
  if $u \notin S$ *AND* $v \notin S$ **then**
    $S \leftarrow S \cup \{u, v\}$;
return S;

order of iteration doesn't matter!

Vertex Cover 2x-optimal greedy algorithm

Claim: The GreedyVC( ) algorithm returns a set S that is *at most twice* as large as the smallest vertex cover. → we use at most twice as many vertices to cover every edge.

proof:

- meaning of VC: every edge has at least one endpoint in S.
- the edges selected for the output (their nodes are selected) ⟹ don't have any nodes in common.

set A of edges

⟹ the optimal VC has to contain at least |A| vertices

- our algo returns 2|A| vertices

Vertex Cover 2x-optimal greedy algorithm

Claim: The GreedyVC( ) algorithm returns a set S that is *at most twice* as large as the smallest vertex cover.

proof:

- Consider the set A of edges that this algorithm chooses.
- None of these edges share a vertex, hence any vertex cover must include *at least* |A| vertices
- Set S contains $2 \cdot |A|$ vertices

Approximation algorithms

Suppose that the optimal solution to an optimization problem P has value m*, and algorithm A returns a solution with value m. We say that A is an approximation algorithm with approximation factor c (also called approx. ratio) if on *any* input

if P is a *minimization* problem then $m^* \leq m \leq c \cdot m^*$

↳ in GreedyVC $c=2$

- sometimes we use the notation $\frac{1}{c} \cdot m \leq m^*$

or if P is a *maximization* problem then $m \leq m^* \leq c \cdot m$

$m \leq \frac{m^*}{c}$

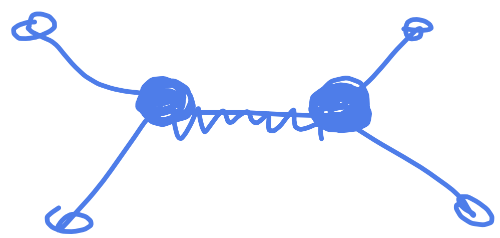
We say that A is a c-approximation algorithm

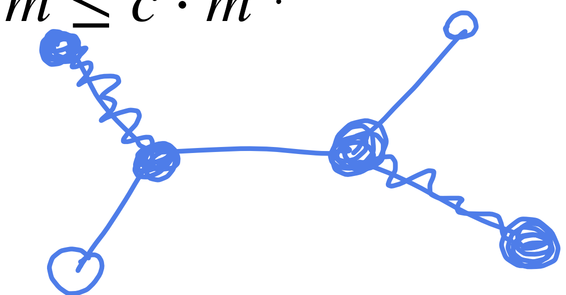
teeny tiny number

for best approx algorithms out there $c = (1+\varepsilon)$

Approximation algorithms

A is a c-approximation algorithm if it returns the value m and we have $m \leq c \cdot m^*$ or $m^* \leq c \cdot m$ for the min/max problem.





Select all True statements when A is a c - approximation algorithm for a *minimization* problem. Let m* be the optimal solution, m is the output of A.

A. the size (value) of m is always $c \cdot m^*$

B. sometimes $m = c \cdot m^*$, sometimes $m < c \cdot m^*$

C. for some inputs A may return the optimal solution, that is $m = m^*$

D. if c = 1, then A always yields the optimal solution

Approximation algorithms

A is a c-approximation algorithm if it returns the value m and we have $m \leq c \cdot m*$ or $m* \leq c \cdot m$ for the min/max problem.

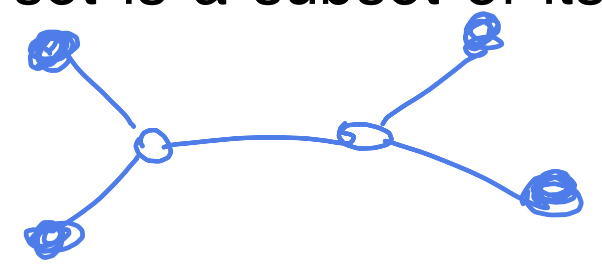
- c is always $c \geq 1$
- if c = 1, then A always yields the optimal solution

Goal:

- find A for which we can prove that it is a c-approximation for all inputs
- the smaller the c the better
- sometimes for efficiency we may use an approximation algorithm even if there exist a (slow) polynomial optimal algorithm

Greedy approximation algorithm for Independent Set

Independent Set: Given a graph G(V,E), an independent set is a subset of its vertices S*, such that for each edge (u,v) at most one of u or v is in S*



Greedy algorithm to find the max independent set:

Iterate over vertices in any order

→ if the vertex has no edges in common with the current S

→ add to S

instead of any order:

iterate lowest degree first

GreedyIS is a $(D+1)$-approximation

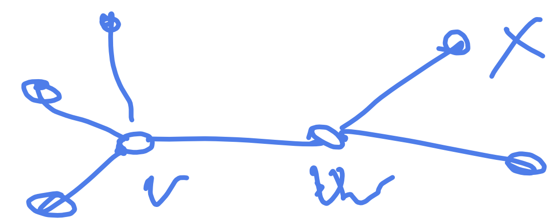
$\deg(v) = 4$

$\deg(w) = 3$

$\deg(x) = 1$

$D = 4$

Let D be the maximum degree in G and let S be the set returned by GreedyIS



proof:

goal: greedy algo returns at most $D|S|$ less vertices as the max

- v is a node in $V - S$ (means: v is not in S)
- v is not in S, b/c at least one of its neighbors is in S.
- D is the max degree $\Rightarrow$ v can have $\leq D$ neighbors in S.

$\Rightarrow |V - S| \leq D|S|$

$\Rightarrow |V| = |V - S| + |S| \leq D|S| + |S| = (D+1)|S|$

$\Rightarrow |S| \leq OPT \leq |V| \leq (D+1)|S|$

GreedyIS is a $(D+1)$-approximation

Let D be the maximum degree in G and let S be the set returned by GreedyIS

Goal: find a lower bound on |S|

- a node u is in V—S (thus not in S) because it has a neighbor v in s
- Each v in S has at most D neighbors
- we get $|V-S| \leq D \cdot |S|$
- Adding up the two we get $|V| = |V-S| + |S| \leq D \cdot |S| + |S| = (D+1)|S|$
- in conclusion $|OPT| \leq |V| \leq (D+1)|S|$

Set Cover greedy algorithm

Set Cover: Given a universe U of items $i_1, i_2, \ldots, i_n$ and subsets of items $S_1, S_2, \ldots, S_m$, select a minimum number of the subsets so that their union contains every item in U.

Greedy algorithm:

In each iteration select the set that contains the largest number of so far uncovered items.

Set Cover greedy algorithm

Set Cover: Given a universe U of items $i_1, i_2, \ldots, i_n$ and subsets of items $S_1, S_2, \ldots, S_m$, select a minimum number of the subsets so that their union contains every item in U.

Greedy algorithm:

In each iteration select the set that covers the most additional items.

Algorithm 1: GreedySC($U, S_1, \ldots S_m$)

$X \leftarrow U$ /* uncovered elements in U */
$C \leftarrow$ empty set of subsets;
while X *is not empty* **do**
    Select S_i that covers the most items in X;
    $C \leftarrow C \cup S_i$;
    $X \leftarrow X \setminus S_i$;
return C;

TopHat - Set Cover

number of sets needed is k

Let k be the size of the optimal solution to Set Cover and |U| = n

Select all True statements!

✓ A. the largest set in the minimum Set Cover contains at least $\frac{n}{k}$ items

✗ B. every set in the minimum Set Cover contains at least $\frac{n}{k}$ items

✗ C. the *second* set chosen by GreedySC contains at least $\frac{1}{k}$ fraction of the items

✓ D. the *second* set chosen by GreedySC contains at least $\frac{1}{k}$ fraction of the *so far uncovered* items

SC greedy approximation

ln n = natural log = $\log_e n$

Theorem: if the optimal solution to SC uses k sets, than the greedy solution uses at most $k \ln n$ sets

reminder from calculus: for any t > 0 we have $\left(1 - \frac{1}{t}\right)^t < \frac{1}{e}$

key ideas in proof:

- The largest set in the optimal set cover must cover at least $\frac{n}{k}$ items.
- In each iteration, the next set chosen by GreedySC always covers at least $\frac{1}{k}$ fraction of the so far uncovered items.

Conclusion: GreedySC is an $\ln n$ -approximation

SC greedy approximation

Theorem: if the optimal solution to SC uses k sets, than the greedy solution uses at most $k \ln n$ sets

reminder from calculus: for any t > 0 we have $\left(1 - \frac{1}{t}\right)^t < \frac{1}{e}$

- opt covers $\geq \frac{1}{k}$ fraction of n items

  $\Rightarrow$ largest set in S also covers at least that

  $\Rightarrow$ remaining uncovered $\leq n - \frac{n}{k} = n\left(1 - \frac{1}{k}\right)$

- set selected second covers $\frac{1}{k}$ of remaining

  $\Rightarrow$ remaining $\left(n\left(1-\frac{1}{k}\right)\right) \cdot \left(1-\frac{1}{k}\right) = n\left(1-\frac{1}{k}\right)^2$

  ⋮

  after r iters: remaining $n\left(1-\frac{1}{k}\right)^r$

- continue until no items remain

why chose $k \ln n$? → see next slide.

$\Rightarrow$ we know for sure that this happens after at most $(k \ln n)$ iterations

remaining $= n\left(1-\frac{1}{k}\right)^{k \ln n} = n\left(\left(1-\frac{1}{k}\right)^k\right)^{\ln n} < n\left(\frac{1}{e}\right)^{\ln n} = n\frac{1}{e^{\ln n}} = n\frac{1}{n} = 1$

Conclusion: GreedySC is an $\ln n$ -approximation

$\Rightarrow$ remaining < 1

SC greedy approximation

Theorem: if the optimal solution to SC uses k sets, than the greedy solution uses at most $k \ln n$ sets .

reminder from calculus: for any t > 0 we have $\left(1 - \frac{1}{t}\right)^t < \frac{1}{e}$

proof:

- since the optimal solution uses k sets, there is at least one set in the opt that covers 1/k fraction of all items
- since GreedySC selects the largest set, it also covers at least $\frac{n}{k}$ items
- after the first iteration at most $n\left(1 - \frac{1}{k}\right)$ remain uncovered
- again, there must be a set in the cover that contains at least 1/k of the remaining
- thus after two iterations $n\left(1 - \frac{1}{k}\right)^2$ are uncovered
- after $k \ln n$ rounds there are at most $n\left(1 - \frac{1}{k}\right)^{k \ln n}$ uncovered items left
- $n\left(1 - \frac{1}{k}\right)^{k \ln n} < \left(\frac{1}{e}\right)^{\ln n} = 1$
- there are at most $k \ln n$ sets returned by the greedy algorithm

SC greedy approximation — how to get k ln n

Reminder from calculus for any t > 0

$$\left(1-\frac{1}{t}\right)^{t} < \frac{1}{e}$$

After r iterations the number of uncovered elements is

$$n\left(1-\frac{1}{k}\right)^{r}$$

what value should r be? This is how we can figure out:

Use trick to get $1 \leq n\left(\left(1-\frac{1}{k}\right)^{k}\right)^{\frac{r}{k}} < n\left(\frac{1}{e}\right)^{\frac{r}{k}}$

Some manipulations:

$$e^{\frac{r}{k}} < n \Rightarrow \frac{r}{k} < \ln n \Rightarrow r < k \ln n$$

After r iterations there are no uncovered vertices left.

GreedySC for Vertex Cover

Can we use the approximate solution for Set Cover to solve Vertex Cover?

Dominating Set

Dominating Set: Given a graph G(V,E) a dominating set is a subset of its vertices S, such that for each node v either v is in S or it has a neighbor in S.

DS problem: Given G, find a minimum size dominating set.

Dominating Set

Dominating Set: Given a graph G(V,E) a dominating set is a subset of its vertices S, such that for each node v either v is in S or it has a neighbor in S.

DS problem: Given G, find a *minimum* size dominating set.

Independent Set: Given a graph G(V,E), an independent set is a subset of its vertices S, such that for each edge (u,v) at most one of u or v is in S

claim: The *maximum* independent set is also a dominating set.

What is the relationship between DS and IS?

Dominating Set and Set Cover

Design an $\ln n$ -approximation algorithm for Dominating Set.