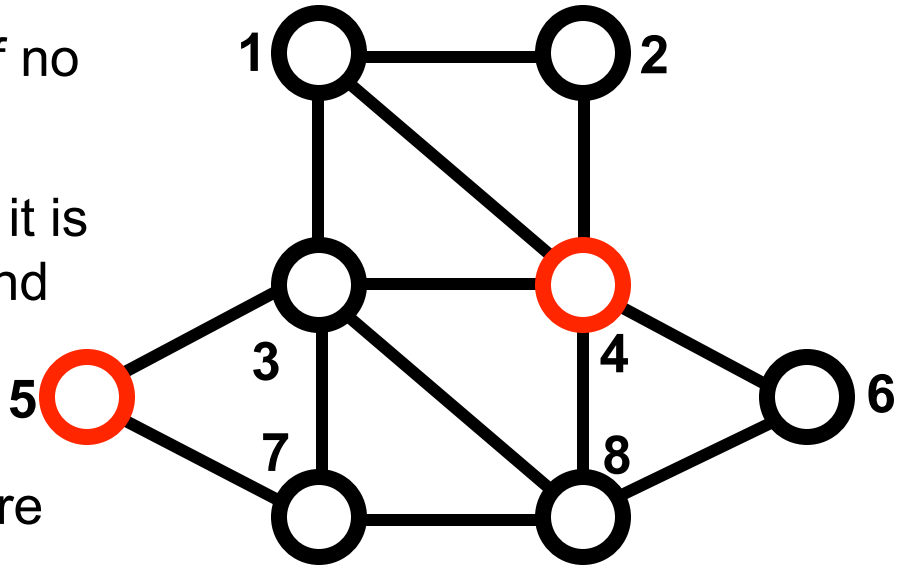
CS 140: Computation on Graphs – Maximal Independent Sets

A graph problem: Maximal Independent Set

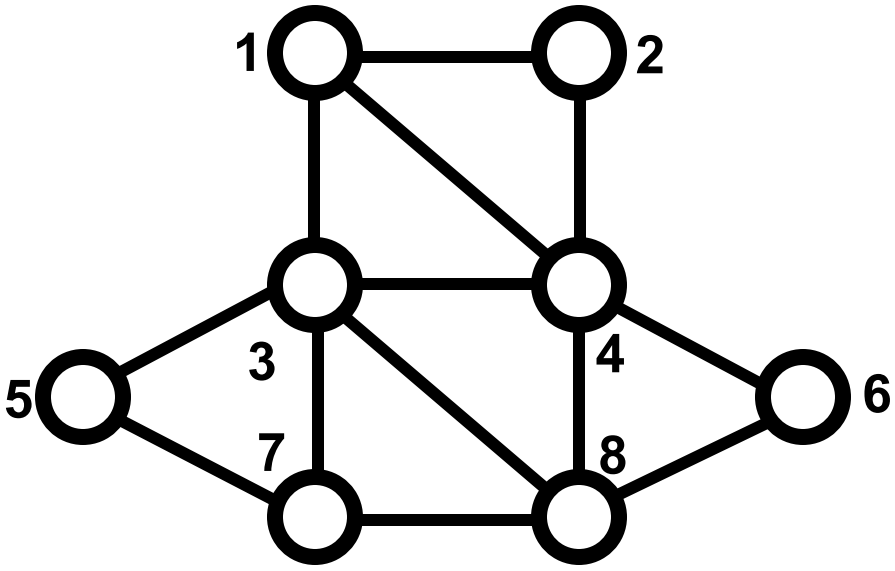
- Graph with vertices V = {1,2,…,n}
- A set S of vertices is independent if no two vertices in S are neighbors.
- An independent set S is maximal if it is impossible to add another vertex and stay independent
- An independent set S is maximum if no other independent set has more vertices
- Finding a *maximum* independent set is intractably difficult (NP-hard)
- Finding a *maximal* independent set is easy, at least on one processor.



The set of red vertices
S = {4, 5} is *independent*
and is *maximal*
but not *maximum*

Sequential Maximal Independent Set Algorithm

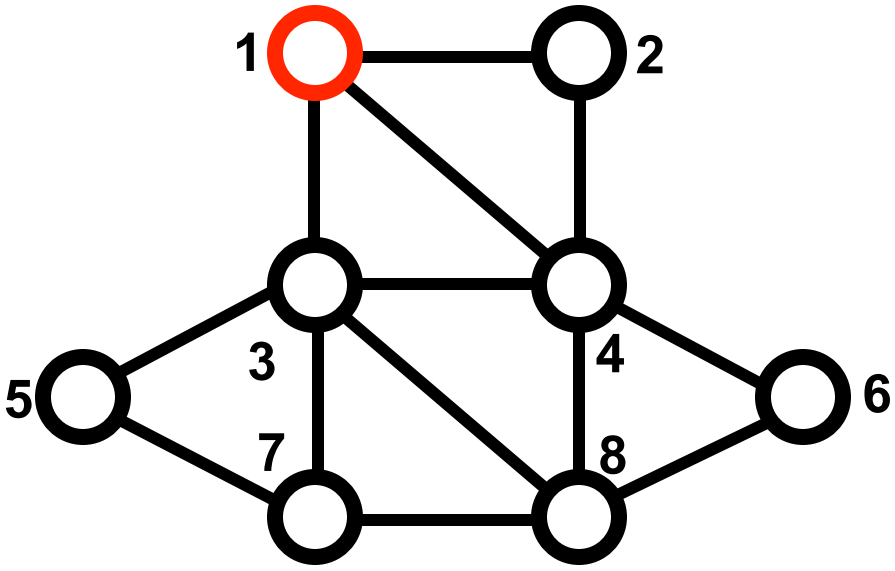
1. S = empty set;
2. for vertex v = 1 to n {
3. if (v has no neighbor in S) {
4. add v to S
5. }
6. }



S = { }

Sequential Maximal Independent Set Algorithm

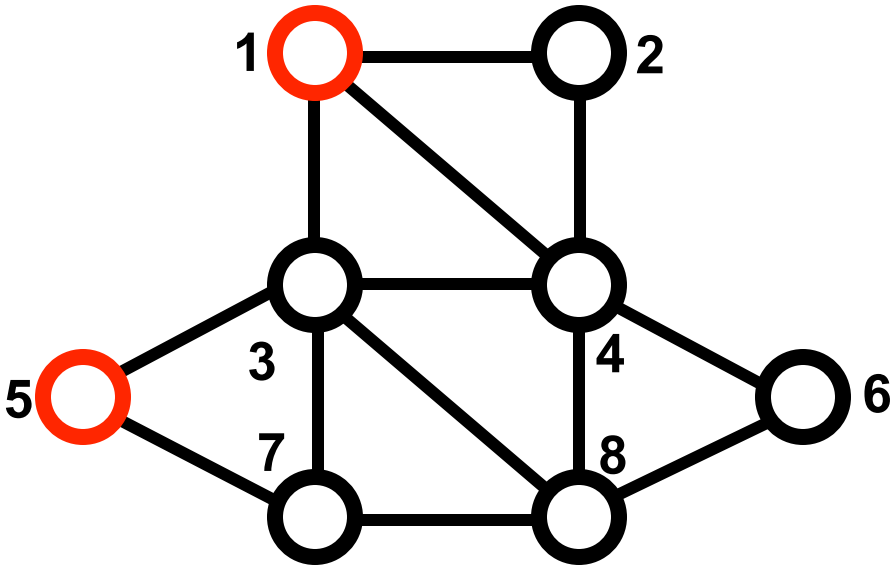
```
1.  S = empty set;
2.  for  vertex v = 1 to n {
3.      if (v has no neighbor in S) {
4.          add v to S
5.      }
6.  }
```



S = { 1 }

Sequential Maximal Independent Set Algorithm

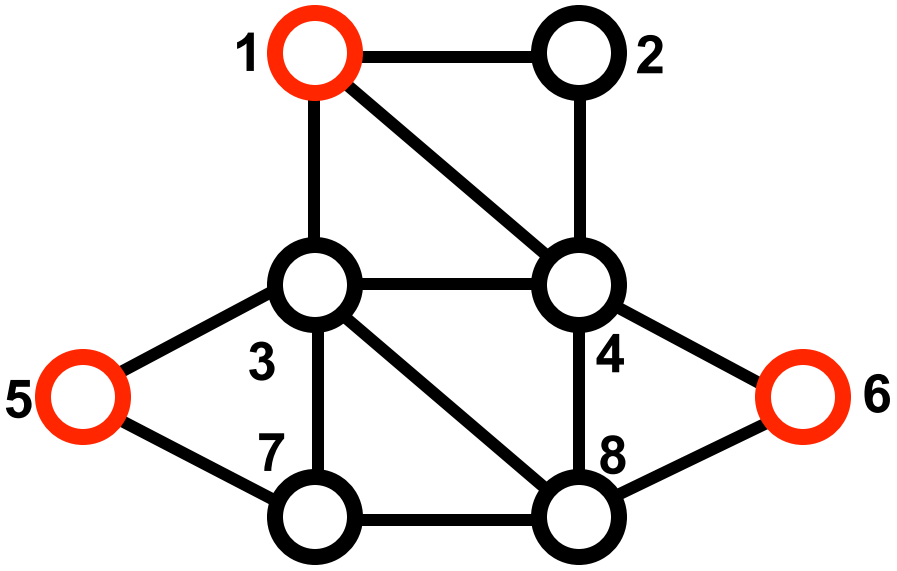
```
1.  S = empty set;
2.  for  vertex v = 1 to n {
3.      if (v has no neighbor in S) {
4.          add v to S
5.      }
6.  }
```



S = { 1, 5 }

Sequential Maximal Independent Set Algorithm

```
1.  S = empty set;
2.  for  vertex v = 1 to n {
3.      if (v has no neighbor in S) {
4.          add v to S
5.      }
6.  }
```

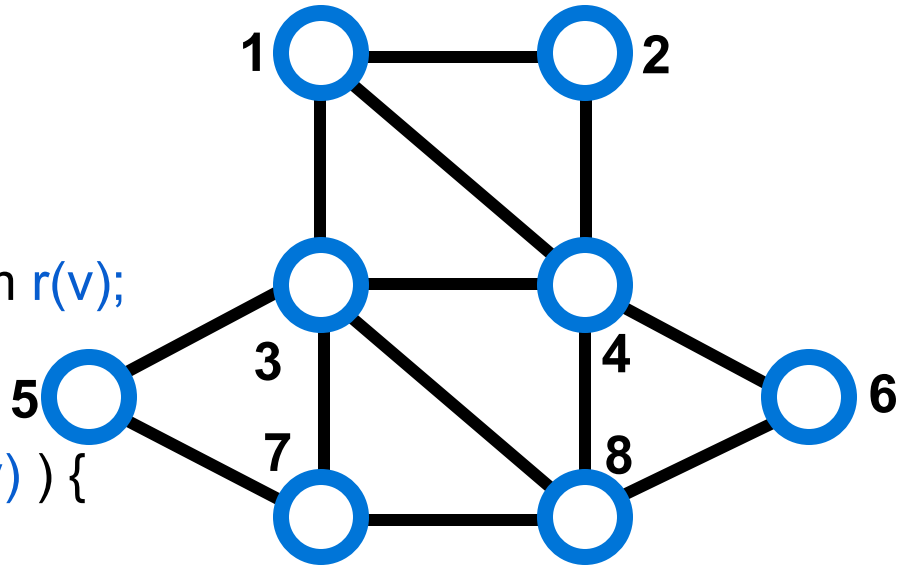


S = { 1, 5, 6 }

work **~ O(n), but** ***span*** **~O(n)**

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

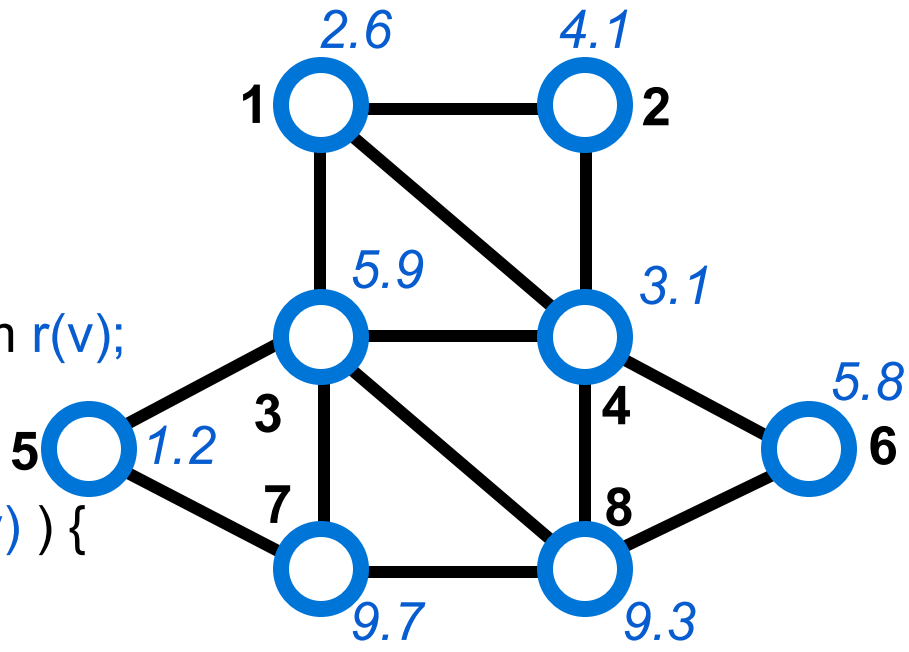


S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

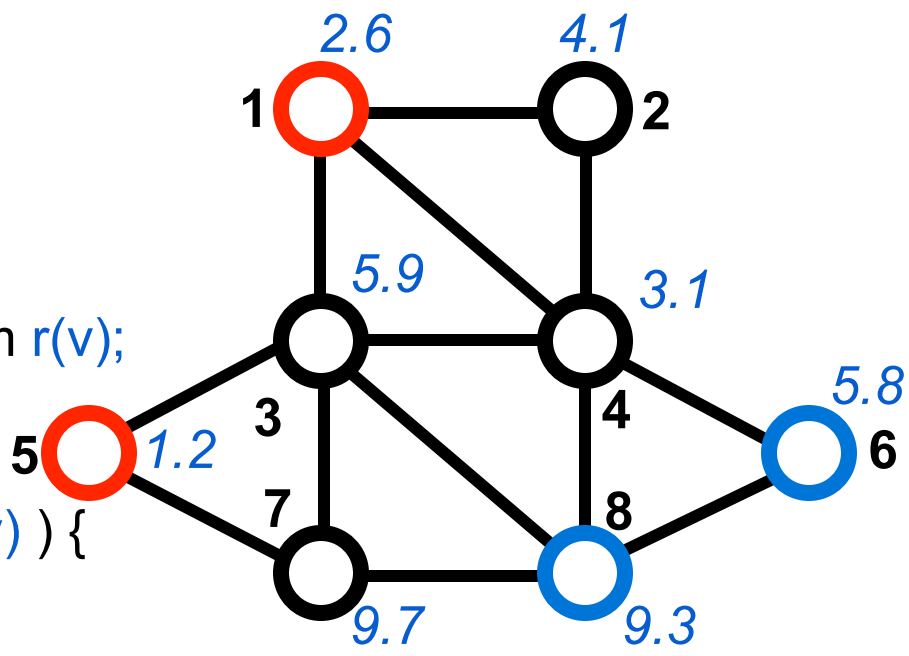


S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

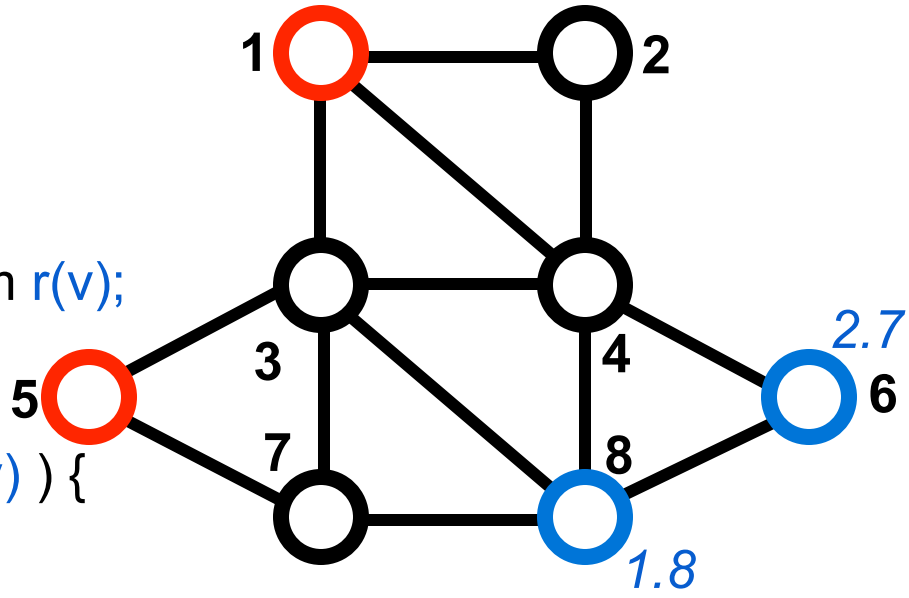


S = { 1, 5 }

C = { 6, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

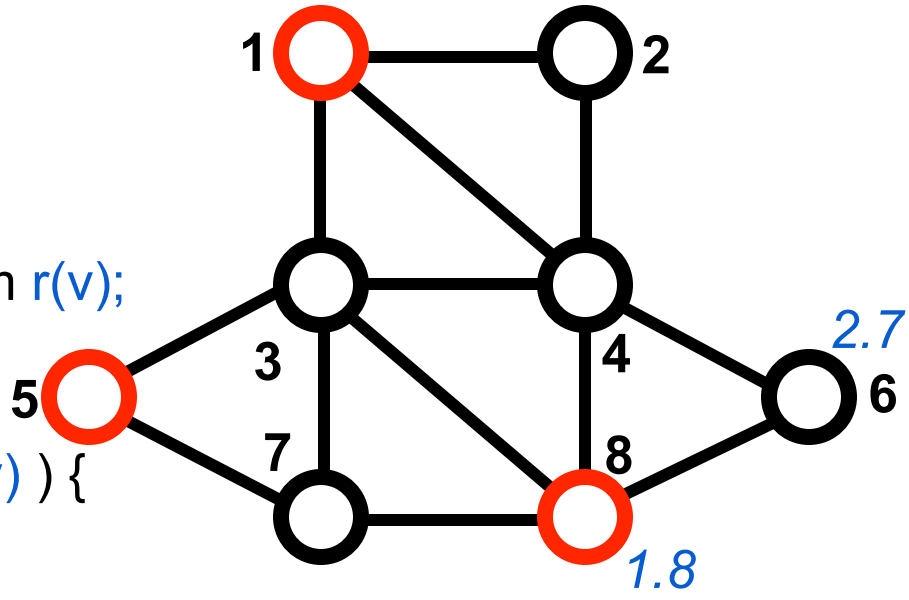


S = { 1, 5 }

C = { 6, 8 }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```

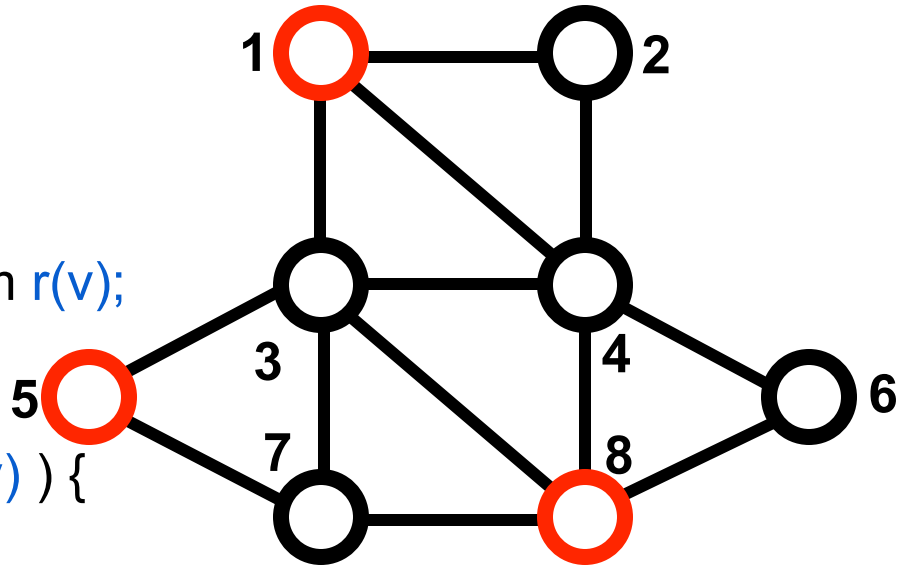


S = { 1, 5, 8 }

C = { }

Parallel, Randomized MIS Algorithm *[Luby]*

```
1.  S = empty set;  C = V;
2.  while  C  is not empty {
3.      label each v in C with a random r(v);
4.      for all v in C in parallel {
5.          if r(v) < min( r(neighbors of v) ) {
6.              move v from C to S;
7.              remove neighbors of v from C;
8.          }
9.      }
10. }
```



Theorem: This algorithm "very probably" finishes within O(log n) rounds.

work **~ O(n log n), but** ***span*** **~O(log n)**