14.1 Maximal Independent Set

Given a graph \( G = (V, E) \), an independent set is a set of vertices \( S \subseteq V \) such that if \( u, v \in S \), then \( (u, v) \notin E \). A maximal independent set is an independent set to which no more vertices can be added without violating the independence property. Let \( d(v) \) denote the degree of a vertex \( v \).

**Algorithm 1 MaxIndSet(V, E)**

1: MIS ← ∅
2: repeat
3: Select \( v \in V \) with probability \( 1/2d(v) \) in to set \( S \)
4: for \((u, v) \in E\) do
5: if \( u, v \) are in \( S \) then
6: if \( d(u) > d(v) \) then
7: remove \( v \)
8: end if
9: if \( d(u) = d(v) \) then
10: remove one of \( u, v \) from \( S \)
11: end if
12: end if
13: end for
14: Call this reduced set \( I \)
15: Add \( I \) to MIS
16: Remove \( I \) and all it’s neighboring vertices and all incident edges from \( G \)
17: \( S \leftarrow \emptyset \)
18: until \( E = \emptyset \)

You can find more details for the proof in the link on the course webpage.

We define good vertices and edges as follows:

- \( D(v) = \{ u : (u, v) \in E | d(u) \leq d(v) \} \)
- Good Vertices : \( V_G = \{ v \in V | |D(v)| > d(v)/3 \} \)
- Good Edges : \( E_G = \{ (u, v) \in E | u \in V_G \text{ OR } v \in V_G \} \)

Proof outline: 1/2 the edges are good for good vertices a constant probability that will be deleted therefore a constant probability that an edge will be deleted.
Lemma 14.1.1 \(|E_G| \geq |E|/2\).

**Proof:** edges from smaller to larger. For bad nodes count two edges out for everyone in. By simple counting at most 1/2 the vertices can be bad.

Lemma 14.1.2 \(\forall v \in V_G, \sum_{(u,v) \in E} d(u)/2 > 1/6\).

**Proof:** Just consider the neighbors \(u\) with \(d(u) \geq d(v)\). \((d(u)/3)/2d(v) > (d(u)/3)/2d(u) > 1/6\).

Lemma 14.1.3 \(\Pr[v \in S \cap v \notin I] \leq 1/2\)

Lemma 14.1.4 \(p(v \in I) \geq 1/4d(v)\).

Lemma 14.1.5 \(\forall v \in V_G, \Pr(v \in N(I)) \geq 1/36\).

**Proof:** if \(d < 3\) then simple otherwise...

Since half the edges are good, the probability that an edge is removed is at least 1/72.

### 14.2 Biconnected components

A biconnected component of an undirected graph \(G\) is a maximal set of edges such that any two edges in the set lie on a common simple cycle. A *bridge* is an edge that does not belong to any cycles.

Given a graph \((V, E)\), the following procedure creates a graph \(G'(V', E')\) on the edges \(E\) of \(G\) such that any pair of vertices \(u_e, u_{e'} \in V'\) that correspond to edges \(u, u'\) that are in a cycle in \(G\) will be in the same connected component in \(G'\).

![Figure 14.2.1: An example graph G and it’s associated G’](image)

Algorithm Outline:
• Generate spanning tree \( T \) of \( G \) and root it. Denote the parent of vertex \( v \) in this tree by \( p(v) \)

• Calculate preorder number, \( pre(v) \), and size, \( size(v) \), for every vertex

• For each vertex \( v \), calculate the lowest neighbor, \( low(v) \), of any vertex in its subtree.

• Similarly for highest, \( high(v) \).

• Add edges to \( G' \) as follows:
  
  - **R1**: add \((v, p(v)), (p(v), p(p(v)))\) to \( G' \) if \((low(v) < pre(p(v))) \) OR \((high(v) > pre(p(v)) + size(p(v)))\)
  
  - **R2**: add \((v, p(v)), (v, u))\) to \( G' \) if \((pre(u) < pre(v)) \) OR \((pre(u) > pre(v) + size(v)))\)

• Find connected components of \( G' \)

Consider an arbitrary rooted spanning tree \( T \) on \( G \) Note that any edge in \( E - T \) cannot be a bridge. Generating a prefix label \( pre(.) \) for every vertex can be done with tree contraction. Computing the size of each subtree \( size(.) \) can also be done with tree contraction (leaffix).

Compute for every vertex \( u \) the minimum and maximum labels \( min(v), max(v) \) over all neighbors \( v \) of \( u \). Using this computation, we can calculate the \( low(.) \), \( high(.) \) for all vertices as follows:

• Calculate leaffix min on the minimum to get \( low(v) \)

• Calculate leaffix max on the maximums to get \( high(v) \)

Consider the predicate \( c(v) = low(v) < pre(v) \) OR \( high(v) > pre(v) + size(v) \)

**Lemma 14.2.1** A tree edge \((v, p(v))\) is a bridge iff \( c(v) \)
Proof: If there is such a condition then there is an edge out of the tree rooted at \( v \) and we must have a cycle involving \((v, p(v))\). If there is not such a condition then there is no edge out of the tree and removing \((v, p(v))\) would disconnect the graph.

Now let's consider connecting the cycles. Recall:

- **R1**: add \((v, p(v)), (p(v), p(p(v)))\) to \( G' \) if \((\text{low}(v) < \text{pre}(p(v))) \text{ OR } (\text{high}(v) > \text{pre}(p(v)) + \text{size}(p(v)))\)

- **R2**: add \((v, p(v)), (v, u)\) to \( G' \) if \((\text{pre}(u) < \text{pre}(v)) \text{ OR } (\text{pre}(u) > \text{pre}(v) + \text{size}(v))\)

Claim: this only connects pairs of edges that are in a cycle

**Theorem 14.2.2** The connected components in \( G' \) are biconnected components in \( G \)

Proof: By claim, this only connects edges in cycle. We will now argue that vertices of \( G' \) that correspond to edges in a cycle in graph \( G \) are connected in \( G' \). Note that the pair of edges at the least common ancestor of the cycle will not be connected.

Look at cases

- cycle edges go up tree ... connected by R1
- cycle loops from a vertex to an ancestor ... connected by R2 and possibly R1
- Cycle crosses between two branches ... connected by R1 and R2