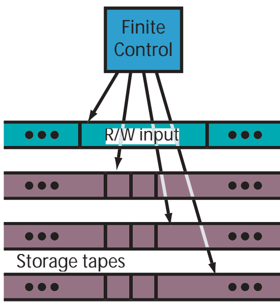
CS20a: NP problems

- The SAT problem
- Graph theory

Time-bounded TMs



- All tapes are 2-way infinite
- M is DTIME(T(n)) if
  - *M is deterministic*
  - *For any input of length n, M takes at most T(n) steps*
- M is NTIME(T(n))
  - *Nondeterministic case*

NP problems

- A problem is *in NP* if it can be decided yes/no by a nondeterministic TM in $O(n^c)$ steps
  - *n is the length of the input*
  - *c is a constant*
- How long does it take to decide a problem in NP?
  - *It takes $O(n^c)$ time to explore each possible nondeterministic choice*
  - *There are an exponential number of choices*
  - *So it takes $O(2^n)$ time worst case*
- How long does it take to *verify* an NP execution?
  - *There are $O(n^c)$ steps of size $O(n^c)$*
  - *So it takes $O(n^{2c})$ time*

CNF satisfiability

- A propositional formula is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctions of literals.
- A *literal* is a propositional letter, or the negation of a propositional letter.
- Every propositional formula can be represented in CNF.

$$
\begin{array}{ll}
 & (A \vee \neg B \vee C) \\
\wedge & (D \vee \neg B \vee \neg C) \\
\wedge & (\neg A \vee B \vee E \vee \neg F) \\
\vdots & \\
\wedge & (E \vee F \vee \neg D \vee A \vee \neg B)
\end{array}
$$

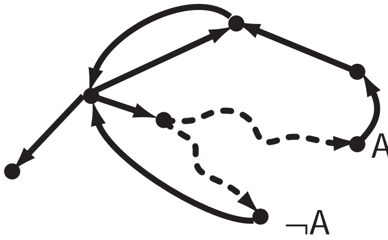
Satisfiability algorithm

- CNF should make the decision problem easier, since there is only conjunction, disjunction, negation
- An algorithm in NP
  - *Guess a truth assignment*
  - *Verify the assignment*
- An algorithm in P
  - *Must be deterministic (no "guessing")*
  - *Just figure out if the formula is true*

Building an algorithm for 2SAT

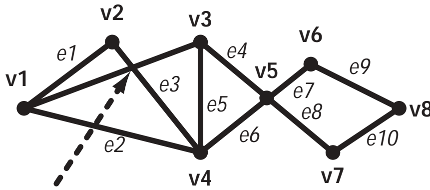
- 2SAT: each clause has at most two literals $A_1, \neg A_2, (A_3 \vee A_4), (A_5 \vee \neg A_6), \ldots$
- **Algorithm**
  - Consider a clause $(l \vee l')$
  - Rewrite the clauses as implications $(\neg l \Rightarrow l') \wedge (\neg l' \Rightarrow l)$
  - Draw a graph
  - The formula is unsatisfiable iff there are two paths
    - $l_1 \Rightarrow \cdots \Rightarrow A$
    - $l_1 \Rightarrow \cdots \Rightarrow \neg A$

2SAT graph



Graph theory

- A *graph* is a set of points *(vertices)* that are interconnected by a set of lines *(edges)*



Some common graphs

Formal definition of graphs

- A graph G is defined as a pair $G = (V, E)$ where
  - V is a set of vertices
  - E is a set of edges $(v_i, v_j), \ldots$
- $n = |V|$ is the size of the graph
- $|E|$ is the number of edges

Graph definitions

- For any edge $e = (v_i, v_j)$, where $(v_i, v_j) \in E$,
  - the edge e is *incident with* vertices v_i and v_j
  - the vertices v_i and v_j are *adjacent*.
  - the *degree* $d(v)$ of a vertex v is the number of edges that are incident to it.

A simple theorem

Theorem The number of vertices of odd-degree in a finite graph is even.

Proof Note that if we add up all the degrees, we get twice the number of edges.

$$\sum_i d(v_i) = 2 \cdot |E|$$

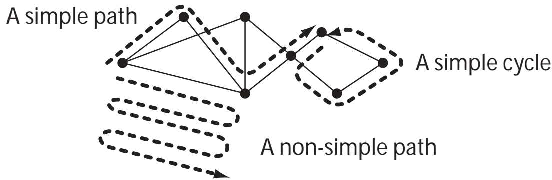
Since the rhs is even, so is the number of vertices with odd degree.

Path definitions

- A *subgraph* of G is a graph obtained by removing some edges and vertices from G.
- A *path* from v_1 to v_n is a sequence $P = v_1, e_1, \ldots, e_{n-1}, v_n$ such that $e_i = (v_i, v_{i+1})$
- If $v_n = v_1$, we say the path is a *cycle*, or *circuit*
- If each vertex appears only once, the path is *simple*.

If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

Path example



If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

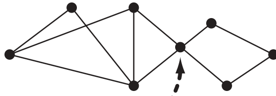
Connected components

- Two vertices v_i, v_j are *connected* if there is a path from v_i to v_j
- The *connected components* of a graph are a partition $V_1, V_2, \ldots, V_k \subseteq V$ in which all vertices are connected.
- A *connected* graph has one component, otherwise it is *disconnected*
- An *articulation point* is a vertex v whose removal disconnects the graph.

If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

Graph components

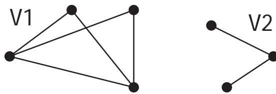
A connected graph



Articulation point

A disconnected graph

V1

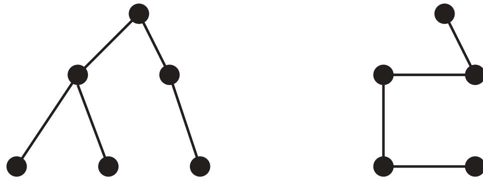


V2

If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

Trees

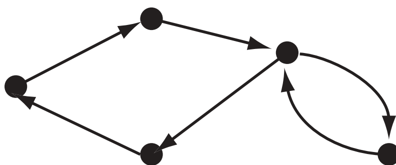
- A tree is a connected graph with no cycles
- A forest is a set of trees



If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

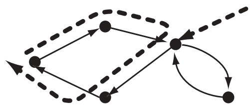
Directed graphs

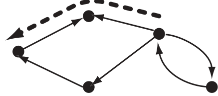
- A *directed graph* assigns a direction to the edges

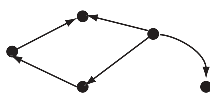


If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

Paths, cycles







Connected components

- A *directed* graph G is defined as a pair $G = (V, E)$ where
  - V is a set of vertices
  - E is a set of edges $(v_i \to v_j), \ldots$
- Two vertices v_i, v_j are *strongly connected* iff there is a directed path from v_i to v_j, **and** a directed path from v_j to v_i.
- Two vertices v_i, v_j are *weakly connected* iff there is an undirected path from v_i to v_j
- Use these to define *strongly-connected* and *weakly-connected* components.

Graph algorithms: Depth-First-Search (DFS)

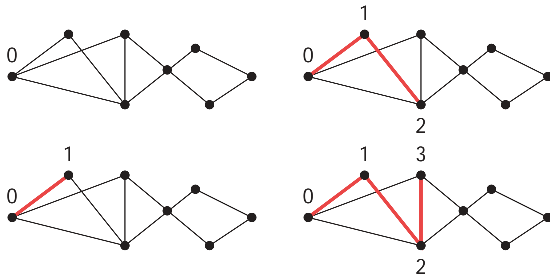
- DFS does the following:
  - *Assigns a unique number to each vertex*
  - *Forms a* spanning tree *(called the DFS spanning tree)*
- Basic algorithm
  - *We need a stack of edges (initially empty)*
  - *A counter c (initially 1)*

DFS algorithm

- Choose an arbitrary vertex v, assign it the number 0, and push all edges incident with v
- **While** the stack is not empty
  - Pop an edge (v_i, v_j) from the stack
  - The edge v_i already has a number
  - **If** v_j is not assigned a number
    - ∗ Assign v_j the number c
    - ∗ Increment c
    - ∗ Push all edges incident with v_j

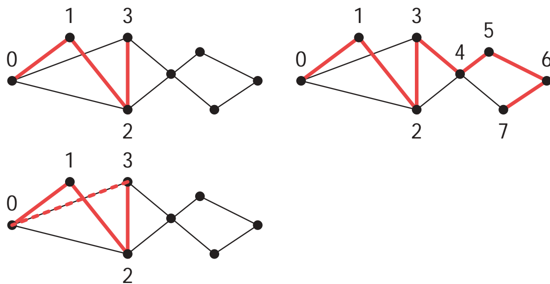
If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

DFS: Initial search



If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

DFS: Backtracking



If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

DFS: Finalize

Biconnected components

Definition For $e, e' \in E$, define $e \equiv e'$ if e and e' lie on a simple cycle.
Theorem The relation $\equiv$ is an equivalence relation.

Reflexive $e \equiv e$ since the edge and its two endpoints define a simple cycle.

Symmetric If $e_1 \equiv e_2$, then $e_2 \equiv e_1$, since e_1 and e_2 can be intechanged in the definition.

Transitive Suppose $e_1 \equiv e_2$ and $e_2 \equiv e_3$.

- Let c, c' be the two simple cycles.
- Those two cycles can be combined.

Biconnected components

Transitive case (part 1)

- Let $(u_1, v_1) \equiv (u_2, v_2)$ and $(u_2, v_2) \equiv (u_3, v_3)$
- Let c_1, c_2 be the two cycles
- Assume u_1, u_2, v_2, u_1 occur in that order around c
- Let x be the first vertex on the segment of c from u_1 to u_2 that also lies on c_2
- Let y be the first vertex on the segment of c from v_1 to v_2 that also lies on c_2

Transitive case (part 2)

- $x \neq y$ since c is simple
- Let p be the path fro x to y containing (u_1, v_1)
- Let p' be the path from x to y containing (u_3, v_3)
- Then p and p' intersect only in x, y and form a simple cycle and form a simple cycle containing (u_1, v_1) and (u_3, v_3)

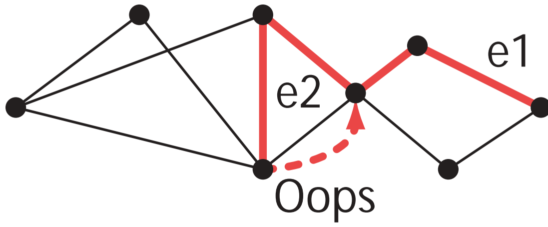
Articulation points

The equivalence classes of $\equiv$ are called the *biconnected components.*
Theorem The vertex a is an articulation point iff a is contained in at least two biconnected components.

Proof

- If a disconnects the graph, then there are u, v such that every path from u to v goes through a. Then (u, a) and (a, v) can't lie on a simple cycle.
- Suppose $(u, a) \not\equiv (a, v)$. Then all paths from u to v must go through a.

Articulation points



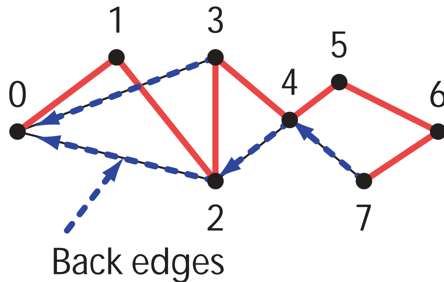
Using DFS to get biconnected components

Theorem Let (u, v) and (v, w) be two adjacent edges in a DFS tree of G. Then $(u, v) \equiv (v, w)$ iff there is a back-edge from some descendent of w to some ancestor of u.

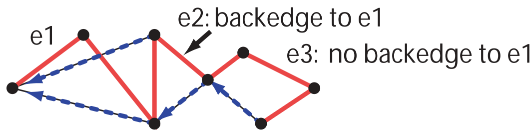
Proof

- If there is a back-edge from some descendent of w to some ancestor of u, then there is a simple cycle.
- Suppose $(u, v) \equiv (v, w)$. Then there is a simple cycle containing them. The DFS tree rooted at w must contain a back-edge because it hat to get to u.

DFS backedges



DFS biconnected components



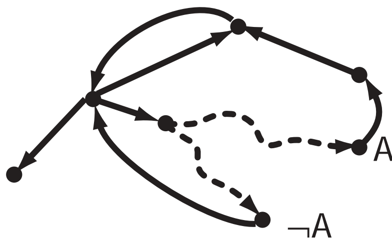
If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

Solving 2SAT

- We can define DFS for *directed graphs*
  - *Follow edges only in the forward direction*
- Use DFS to determine strongly-connected components
  - *Strongly-connected component*
    - *For each u, v, there is a path from u to v, and from v to u*

If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

2SAT graph



If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences

2SAT solution

- The formula is satisfiable iff there is a strongly-connected component that contains A and not A for some letter A
  - *If it does, then A implies not A and not A implies A (contradiction)*
  - *If not, we have to define a satisfying assignment (next time)*

If this page displays slowly, try turning off the "smooth line art" option in Acrobat, under Edit->Preferences