Attendance

Are you here on time? (As noted on syllabus, timely attendance counts positively towards your participation grade.)

A) No
B) Maybe
C) Yes

COSC 290 Discrete Structures

Lecture 19: Graph search

Prof. Michael Hay
Wednesday, Mar. 21, 2018
Colgate University

Plan for today

1. Review: converting to CNF
2. Graphs and graph algorithms
3. Data representations for graphs
4. Algorithms for topological sort

Review: converting to CNF

Proving every φ can be expressed in CNF

Claim: For any φ that is in NNF, there exists a proposition φ' that is in CNF and is logically equivalent to φ.

Proof by induction (excerpt): A key step in proof is handling case 2, when $\varphi := \alpha \vee \beta$.

Inductive hypothesis tells us that *hasCNF*(α) and *hasCNF*(β). Let α' be such that $\alpha \equiv \alpha'$ and *isCNF*(α'); similarly for β'.

Let $\alpha' := c_1 \wedge c_2 \wedge \cdots \wedge c_m$ where each c_i is a clause (disjunction of one or more literals).

Let $\beta' := d_1 \wedge d_2 \wedge \cdots \wedge d_n$ where each d_j is a clause.

Last time, math of the proof. Today, a small example illustrating the math. (This example is *not* part of the proof!)

Example: $\alpha' := (\neg p \vee q) \wedge (\neg r)$ and $\beta' := (s \vee \neg t) \wedge (\neg u \vee v) \wedge (w)$

Inductive case 2 continued...

$$
\begin{aligned}
\varphi &\equiv \alpha' \vee \beta' && \text{inductive hypothesis} \\
&\equiv \left(\bigwedge_{i=1}^{m} c_i\right) \vee \beta' && \text{definition of } \alpha' \\
&\equiv \bigwedge_{i=1}^{m} (c_i \vee \beta') && \text{distribute OR over ANDs of } \alpha' \\
&\equiv \bigwedge_{i=1}^{m} \left(c_i \vee \left(\bigwedge_{j=1}^{n} d_j\right)\right) && \text{definition of } \beta' \\
&\equiv \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} (c_i \vee d_j) && \text{distribute OR over ANDs of } \beta'
\end{aligned}
$$

Quick aside about schedule

We skipped over Ch. 6, but information from Ch. 6 does creep in a little bit to some of the later chapters. For now, here is all you need to know.

Let $f(n)$ be runtime of an algorithm on inputs of size n.

$f(n) = O(n^2)$ means that function f grows no faster than n^2.

$f(n) = \Omega(n)$ means that function f grows no slower than n.

$f(n) = \Theta(n \log n)$ means that function f grows at the same rate as $n \log n$.

Graphs and graph algorithms

Directed Graphs

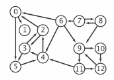

Figure 1: Directed graph $G = (V, E)$ where $V = \{0, 1, \ldots, 12\}$ and E can be determined from figure.

Figure taken from Sedgewick and Wayne, Algorithms, 4th edition

Poll: deciphering notation

Consider this expression,

$$|\{ y \in V : (y, x) \in E \}|$$

What does this represent?

A) The neighbors of x
B) The neighbors of y
C) The number of incoming edges to x
D) The number of outgoing edges to y
E) None of the above

Directed Acyclic Graphs

Definition (Directed Acyclic Graph)

A directed acyclic graph (DAG) is a directed graph that contains no directed cycles.

Example

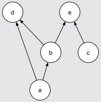

Figure 2: Example DAG

Example application of DAGs

A to do list,

$$[\mathit{attendClass}, \mathit{sleep}, \mathit{borrowBook}, \mathit{eat}, \mathit{brushTeeth}, \mathit{study}]$$

with precedence constraints:

- $\mathit{borrowBook} \to \mathit{study}$ (book borrowing must *precede* studying)
- $\mathit{study} \to \mathit{attendClass}$
- $\mathit{sleep} \to \mathit{attendClass}$
- $\mathit{eat} \to \mathit{brushTeeth}$
- $\mathit{brushTeeth} \to \mathit{sleep}$

(DAG shown on board.)

Topological ordering

Definition

Given a DAG, a topological ordering is an ordering of the vertices (a sequence) such that for every directed edge $(u, v) \in E$, vertex u comes before v in the ordering.

Example

Continuing the todo list example, a topological ordering is valid way to order the tasks such that all precedence constraints are obeyed.

- borrowBook, study, eat, brushTeeth, sleep, attendClass
- eat, brushTeeth, sleep, borrowBook, study, attendClass
- eat, borrowBook, brushTeeth, study, sleep, attendClass
- ~~eat, brushTeeth, sleep, attendClass, borrowBook, study~~

Algorithms for topological ordering

Today, we will look at three algorithms:

- version 1: repeatedly find a “minimal” vertex
- version 2: same idea, more efficient than v1
- version 3: based on depth-first search (as efficient as v2)

Data representations for graphs

Adjacency matrix

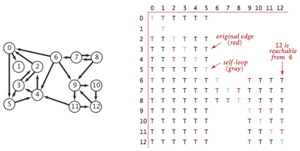

Figure 3: Directed graph G (on left) and adjacency matrix for the *transitive closure* of G (on right). The adjacency matrix of G would be *only* the red **T**s.

In Java: `boolean[][] adjMatrix`

Figure taken from Sedgewick and Wayne, Algorithms, 4th edition

Adjacency list

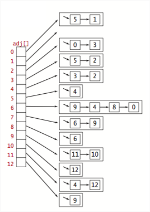

Figure 4: Adjacency list for G.

Figure taken from Sedgewick and Wayne, Algorithms, 4th edition

In Java:

Using a Map:
`Map<Integer, List<Integer>> adjList`

or, using a *jagged array*:
`int[][] adjList`

- `adjList[i].length` equals number of neighbors of i
- `adjList[i][k]` gives the k^{th} neighbor of i

Poll: comparison between data representations

Compare the adjacency matrix with the adjacency list representation. Which operations would be faster with the adjacency matrix representation?

A) checking whether $(x, y) \in E$
B) computing the degree (number of neighbors) of x
C) printint out the neighbors of x
D) counting the number of edges in the graph.

Algorithms for topological sort

Version 1: repeatedly find minimal

Input: Directed graph $G = (V, E)$.
Output: list of vertices, in some topological order

```
Initialize order to empty list
S := V
while S is not empty do
    X := findMinimal(S)
    choose any x from X
    S := S − {x}
    append x to order                    ▷ put x at end of order
return order
```

Where *findMinimal*(S) returns a subset of $X \subseteq S$ of vertices that are minimal with respect to S.

x is minimal with respect to S if $x \in S$ and $\forall y \in S - \{x\} : (y, x) \notin E$.

(Apply to example on board.)