TDDB56 – DALGOPT-D
Algorithms and optimization

Lecture 11

Graphs

Content:

- Basics, ADT Graph, Representations
- Graph Searching
  - Depth-First Search
  - Breadth-First Search
  - Example graph problems solved with search
- Directed Graphs

Basics

Graph $G = (V,E)$:
- $V$ a set of vertices;
- $E$ a set of pairs of vertices, called edges
  - Directed graph: the edges are ordered pairs
  - Undirected graph: the edges are unordered pairs
- Edges and vertices may be labelled by additional info

Example: A graph with
- vertices representing airports; info airport code
- edges representing flight routes; info mileage
  (see [G&T])

Terminology

- End vertices of an edge
- Edges incident on a vertex
- Adjacent vertices (neighbors)
- Degree of a vertex (undirected graph)
- Indegree/outdegree (directed graph)
- Path
- Simple path
- Cycle (loop)
Terminology cont…

- **G is connected**
  iff there is a path between any two vertices

- **G is a (free) tree**
  iff there is a unique simple path between any two vertices; notice: root is not distinguished

- **G is complete**
  iff any two vertices are connected by an edge

- **G’ = (V’, E’) is a subgraph of G = (V,E)**
  iff \( V’ \subseteq V \) and \( E’ \subseteq E \)

- A **connected component** of \( G \) is any maximal subgraph of \( G \) which is connected

ADT Graph

Defined differently by different authors.

[Goodrich & Tamassia]:

- endVertices(e): an array of the two endvertices of \( e \)
- opposite(v, e): the vertex opposite of \( v \) on \( e \)
- areAdjacent(v, w): true iff \( v \) and \( w \) are adjacent
- replace(v, x): replace element at vertex \( v \) with \( x \)
- replace(e, x): replace element at edge \( e \) with \( x \)
- insertVertex(o): insert a vertex storing element \( o \)
- insertEdge(v, w, o): insert an edge \((v, w)\) storing element \( o \)
- removeVertex(v): remove vertex \( v \) (and its incident edges)
- removeEdge(e): remove edge \( e \)
- incidentEdges(v): edges incident to \( v \)
- vertices(): all vertices in the graph; edges(): all edges in the graph

Representation of graphs (rough idea)

Graph \((V, E)\) with vertices \(\{v_1, \ldots, v_n\}\) represented as:

- **Adjacency matrix**
  \[ M[i,j] = 1 \text{ if } (v_i, v_j) \text{ in } E, \text{ and } 0 \text{ otherwise} \]

- **Adjacency list**
  for each \( v \) store a list of neighbors

The rough idea may need refinement

- Vertices and edges stored as cells: may contain additional information
- \( V \) and \( E \) are sets: use a set representation (list, table, dictionary?)
- Incident edges of a vertex should be accessible very efficiently.
- Neighbors of a vertex should be accessible very efficiently.

For refined variants of Adjacency list and Adjacency matrix see [G&T]
Adjacency List Structure [G&T]

Example graph

- List of vertices
- Vertex objects
- Lists of incident edges
- Edge objects linked to respective vertices
- List of edges

Adjacency Matrix Structure [G&T]

Example graph

- Vertex objects augmented with integer keys
- 2D-array adjacency array

Graph Searching

The problem: systematically visit the vertices of a graph reachable by edges from a given vertex.
Numerous applications:
- robotics: routing, motion planning
- solving optimization problems (see OPT part)

Graph Searching Techniques:
- Depth First Search (DFS)
- Breadth First Search (BFS)

DFS Algorithm

- Input: a graph G and a vertex s
- Visits all vertices connected with s in time $O(|V|+|E|)$

Procedure DepthFirstSearch(G=(V,E), s):
for each $v$ in V do explored(v) ← false;
RDFS(G,s)

Procedure RDFS(G,s):
explored(s) ← true;
previsit(s) {some operation on s before visiting its neighbors}
for each neighbor $t$ of $s$
if not explored(t) then RDFS(G,t)
postvisit(s) {some operation on s after visiting its neighbors}
Example (notation of [G&T])

- unexplored vertex
- visited vertex
- unexplored edge
- discovery edge
- back edge

Some applications of DFS

- Checking if G is connected
- Finding connected components of G
- Finding a path between vertices
- Checking acyclicity/finding a cycle
- Finding a spanning forest of G

Breadth First Search (BFS)

- Input: a graph G and a vertex s
- Visits all vertices connected with s in time $O(|V|+|E|)$ in order of increasing distance to s
- Apply Queue!

```plaintext
procedure BFS (G=(V,E), s):
for each v in V do explored(v) ← false;
S ← MakeEmptyQueue();
Enqueue(S,s); explored(s) ← true;
while not IsEmpty(S) do
    t ← Dequeue(S);
    visit(t);
    for each neighbor v of t do
        if not explored(v) then
            explored(v) ← true;
            Enqueue(S,v)
```

Graphs HT 2006 11.13

Graphs HT 2006 11.14

Graphs HT 2006 11.15

Graphs HT 2006 11.16
Example

- Unexplored vertex
- Visited vertex
- Unexplored edge
- Discovery edge
- Cross edge

Example (cont.)

Some applications of BFS

- Checking if $G$ is connected
- Finding connected components of $G$
- Finding a path of minimal length between vertices
- Checking acyclicity/finding a cycle
- Finding a spanning forest of $G$

Compare with DFS!
**Directed Graphs**
- **Digraphs**: edges are ordered pairs.
- Digraphs have many applications, like
  - Task scheduling
  - Route planning, ….
- Search algorithms apply, follow directions.
- DFS applications for digraphs:
  - Transitive closure but in $O(n(m+n))$
  - Checking strong connectivity
  - Topological sort of a directed acyclic graph DAG (scheduling)

**Transitive Closure**
- Given a digraph $G$, the transitive closure of $G$ is the digraph $G^*$ such that
  - $G^*$ has the same vertices as $G$
  - if $G$ has a directed path from $u$ to $v$ ($u \neq v$), $G^*$ has a directed edge from $u$ to $v$
- The transitive closure provides reachability information about a digraph

**Strong Connectivity Algorithm**
- Pick a vertex $v$ in $G$.
- Perform a DFS from $v$ in $G$.
  - If there's a $w$ not visited, print "no".
- Let $G'$ be $G$ with edges reversed.
- Perform a DFS from $v$ in $G'$.
  - If there's a $w$ not visited, print "no".
  - Else, print "yes".
- Running time: $O(n+m)$.

**DAGs and Topological Ordering**
- A directed acyclic graph (DAG) is a digraph that has no directed cycles.
- A topological ordering of a digraph is a numbering $v_1, \ldots, v_n$ of the vertices such that for every edge $(v_i, v_j)$, we have $i < j$.
- Example: in a task scheduling digraph, a topological ordering is a task sequence that satisfies the precedence constraints.
- Theorem
  - A digraph admits a topological ordering if and only if it is a DAG.
  - Theorem
Topological Sorting
Use modified DFS!

procedure TopologicalSort(G):
  nextnumber ← |G|
  for each vertex v in G do explored(v) ← false;
  for each vertex v in G do
    if not explored(v) then RDFS(G,v)

procedure RDFS(G,s):
  explored(s) ← true;
  for each neighbor t of s
    if not explored(t) then RDFS(G,t)
  number(s) ← nextnumber;
  nextnumber ← nextnumber - 1

Topological Sorting Example

© 2004 Goodrich, Tamassia