Advanced Algorithms

Floriano Zini
Free University of Bozen-Bolzano
Faculty of Computer Science

Academic Year 2013-2014

Lecture 4 – Greedy algorithms
Chess vs Scrabble

- In chess the player must think ahead
- A winning strategy must consider the long term consequences of moves
- In scrabble, it is possible to do well by simply making the move that is best at the moment
- The player can adopt a greedy strategy

Greedy algorithms

- **Greedy** algorithms build up a solution step by step, always choosing the next step that offers the most obvious and immediate benefit
- This approach can be disastrous for some computational problems
- But there are many other for which it is optimal
- In this lecture we will see algorithms that adopt a greedy strategy:
  - Minimum spanning tree
  - Huffman coding
  - Set cover
Graphs

- A graph is an abstract representation of a set of objects where some pairs of the objects are connected by links.

- The interconnected objects are represented by mathematical abstractions called **nodes**.

- The links that connect some pairs of nodes are called **edges**.

**Graphs (cont.)**

- **Undirected graph**
  - An ordered pair $G = (V,E)$ where $V$ is the set of nodes and $E$ is the set edges (unordered pairs of nodes).

- **Directed graph or digraph**
  - An ordered pair $D = (V,A)$ where $V$ is the set of nodes and $A$ is the set of directed edges (ordered pairs of nodes).
Graphs (cont.)

- **Multigraph**
  - Multiple edges are allowed between the same pair of nodes

- **Weighted graph**
  - A weight is assigned to each edge
  - Weights represent, for example, costs, lengths, or capacities

Trees

- A tree is an undirected graph that is connected and acyclic

- **Properties**
  - A tree on n nodes has n-1 edges
  - Any connected, undirected graph G = (V, E) with |E| = |V| - 1 is a tree
  - An undirected graph is a tree if and only if there is a unique path between any pair of nodes
    - i.e., no cycles
Building a network

- Suppose you are asked to network a collection of computers and other hardware devices by linking selected pairs of them.
- This translates into a graph problem in which:
  - nodes are hardware devices
  - undirected edges are potential links, each with a maintenance cost or unit transfer cost
- The goal is to:
  - Pick enough edges so that the nodes are connected
  - Keep the total cost at minimum
- Finding the cheapest possible network is an example of minimum spanning tree search.

Minimum spanning tree examples

- Urban walks
- USA airport network
- Bacterial agent diffusion
Network building (cont.)

- The optimal set of edges cannot contain a cycle
  - removing an edge from this cycle would reduce the cost without compromising connectivity
- The solution is a connected acyclic graph, i.e., a tree
- The tree we want is the one with minimum total weight
  - The minimum spanning tree

Minimum spanning tree

- Definition
  - Input: An undirected graph $G = (V, E)$; edge weights $w_e$
  - Output: A tree $T = (V, E')$, with $E' \subseteq E$, that minimizes
    
    $$\text{weight}(T) = \sum_{e \in E'} w_e$$

- Example
  
  The cost is $1 + 4 + 2 + 4 + 5 = 16$
A greedy approach

- Kruskal’s minimum spanning tree algorithm on $G = (V, E)$
  - Start with the empty graph
  - Repeatedly add the next lightest edge from $E$ that does not produce a cycle

Example

Is the solution unique? Can you spot another MST? 

- The solution is not unique. Can you spot another MST?
Cut property

- The correctness of Kruskal’s method follows from the cut property
  - Suppose edges $X$ are part of a MST $G = (V, E)$
  - Pick any subset of nodes $S$ for which $X$ does not cross between $S$ and $V - S$
  - Let $e$ be the lightest edge across this partition
  - Then $X \cup \{e\}$ is part of some MST
  - A cut is any partition of the nodes $V$ into two groups, $S$ and $V - S$
  - The cut property says that it is always safe to add the lightest edge across any cut, provided $X$ has no edges across the cut

Proof of the cut property

- Edges $X$ are part of some MST $T$
- If the new edge $e$ also is part of $T$, then there is nothing to prove
- If $e$ is not in $T$, it is part of a different MST $T'$ built as follows
  - Add edge $e$ to $T$; since $T$ is connected adding $e$ creates a cycle
  - This cycle has another edge $e'$ across the cut $(S, V - S)$
  - $T' = T \cup \{e\} - \{e'\}$
Proof of the cut property (cont.)

Is $T'$ a minimum spanning tree?
- $T'$ is a tree (see properties of trees)
- $T'$ is a minimum spanning tree
  - $weight(T') = weight(T) + w(e) - w(e')$
  - Both $e$ and $e'$ cross between $S$ and $V - S$, and $e$ is the lightest edge of this type
  - Therefore $w(e) \leq w(e')$, and $weight(T') \leq weight(T)$
- Since $T$ is a MST, it must be the case that $weight(T') = weight(T)$ and $T'$ is also a MST

Cut property example

Undirected graph

$X$ has 3 edges and is part of MST $T$

The cut: $\delta = \{A,B,C,D\} \rightarrow$ one of the minimum-weight edges across the cut $(S, V - S)$ is $e = \{D,E\}$

$X \cup \{e\}$ is part of MST $T'$

MST $T$: 

MST $T'$:
Kruskal’s algorithm

- At any moment, the edges already chosen form a partial solution:
  - a collection of connected components each of which has a tree structure
- The next edge $e$ to be added connects two components $T_1$ and $T_2$:
  - $e$ is the lightest edge that doesn’t produce a cycle
  - $e$ is certain to be the lightest edge between $T_1$ and $V - T_1$
  - Therefore $e$ is part of a MST, as it satisfies the cut property

---

Kruskal’s algorithm (cont.)

```plaintext
procedure kruskal(G, w)
Input: A connected undirected graph $G = (V,E)$ with edge weights $w$
Output: A minimum spanning tree defined by the edges $X$

for all $u \in V$:
  makeset($u$)

$X = {}$

Sort the edges $E$ by weight
for all edges $\{u,v\} \in E$, in increasing order of weight:
  if $\text{find}(u) \neq \text{find}(v)$:
    add edge $\{u,v\}$ to $X$
  union($u,v$)
```

At each stage the algorithm chooses an edge to add to the partial solution:
- test each candidate edge $\{u,v\}$ to see whether the endpoints $u$ and $v$ lie in different components
- Once an edge is chosen, the two components are merged
Kruskal’s algorithm (cont.)

- The state is a collection of disjoint sets, containing the nodes of a particular component.
- Initially each node is in a component by itself:
  - makeset(x): create a singleton set containing just x.
- Pairs of nodes are tested to see if they are in the same set.
  - find(x): to which set does x belong?
- Adding an edge means merging two components.
  - union(x, y): merge the sets containing x and y.
- The algorithm uses \(|V|\) makeset, \(2*|E|\) find, and \(|V|-1\) union.

A data structure for disjoint sets

We store a set by a directed tree.

- Nodes of the tree are elements of the set, arranged in no particular order.
- Each node has parent pointers that eventually lead up to the root of the tree.
- The root element is a convenient representative, or name, for the set.
- Each node has a rank, the height of the subtree rooted in that node.

A directed-tree representation of two sets \(\{B, E\}\) and \(\{A, C, D, F, G, H\}\)

\[
\text{rank}(E) = 1, \quad \text{rank}(H) = 2, \quad \text{rank}(D) = 1, \quad \text{rank}(G) = 0
\]
Makeset and find

- \textit{makeset} is a \textbf{constant-time operation}

\begin{verbatim}
procedure makeset(x)
  π(x) = x
  rank(x) = 0

function find(x)
  while x ≠ π(x): x = π(x)
  return x
\end{verbatim}

- \textit{find} follows pointers to the root and \textbf{takes time proportional to the height of the tree}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{quiz}
\caption{Quiz}
\end{figure}

The tree is built by \textit{union}, which merges two sets of nodes.
Any idea about how union can be implemented?

Merging two sets is make the root of one point to the root of the other

\begin{itemize}
  \item Option 1
  \item Option 2
\end{itemize}

\textbf{Which is the best option?}

Option 2 is better than option 1, as it makes the three shallower
Union by rank

The tree is built by union
- **Merging** two sets is **make the root of one point to the root of the other**
- Since tree height is the main impediment to computational efficiency, a good strategy is to **make the root of the shorter tree point to the root of the taller tree**
- The overall **height increases only if the two trees being merged are equally tall**
- Instead of explicitly computing heights of trees, we use the **rank** numbers of their root nodes—which is why this scheme is called **union by rank**

---

Union by rank (procedure)

```plaintext
procedure union(x, y)
    r_x = find(x)
    r_y = find(y)
    if r_x = r_y: return
    if rank(r_x) > rank(r_y):
        \( \pi(r_y) = r_x \)
    else:
        \( \pi(r_x) = r_y \)
        if rank(r_x) = rank(r_y): rank(r_y) = rank(r_y) + 1
```

---
**Total time for Kruskal’s algorithm**

- \( O(|V|) \) for creating the initial trees
- \( O(|E| \cdot \log |E|) = O(|E| \cdot \log |V|) \) for sorting the edges
- \( O(|E| \cdot \log |V|) \) for the union and find operations that dominate the rest of the algorithm

A single union or find operation is \( O(\log |V|) \)