IE170: Algorithms in Systems Engineering: Lecture 26

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

April 2, 2007

Taking Stock

Last Time

- Flows

This Time

- Review!

Stuff We Learned

- Dynamic Programming (15.[1,3])
- Greedy Algorithms (16.[1,2])
- Graphs and Search (22.*)
- Spanning Trees (23.*)
- (Single Source) Shortest Paths (24.[1,2,3])
- (All Pairs) Shortest Paths (25.[1,2])
- Max Flow (26.[1,2,3])

Dynamic Programming

Dynamic Programming in a Nutshell

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution "from the bottum up"
4. Construct optimal solution (if required)

Examples

- Assembly Line Balancing
- Lot Sizing

Assembly Line Balancing

- Let $f_i(j)$ be the fastest time to get through S_{ij} $\forall i = 1,2$ $\forall j = 1,2,\ldots n$

$$\begin{aligned} f^* &= \min(f_1(n) + x_1, f_2(n) + x_2) \\ f_1(1) &= e_1 + a_{11} \\ f_2(1) &= e_2 + a_{21} \\ f_1(j) &= \min(f_1(j-1) + a_{1j}, f_2(j-1) + t_{2,j-1} + a_{1j}) \\ f_2(j) &= \min(f_2(j-1) + a_{2j}, f_1(j-1) + t_{1,j-1} + a_{2j}) \end{aligned}$$

Lot Sizing

- Let $f_t(s)$: be the minimum cost of meeting demands from $t, t+1, \ldots T$ (t until the end) if s units are in inventory at the beginning of period t

$$f_t(s) = \min_{x \in 0,1,2,\ldots} \{c_t(x) + h_t(s + x - d_t) + f_{t+1}(s + x - d_t)\}.$$

Greedy

- Greedy is not always optimal!
- But it sometimes works:

Activity Selection

- Let $S_{ij} \subseteq \mathcal{A}$ be the set of activities that start after activity i needs to finish and before activity j needs to start:

$$S_{ij} \stackrel{\text{def}}{=} \{k \in S \mid f_i \le s_k, f_k \le s_j\}$$

- Let's assume that we have sorted the activities such that

$$f_1 \le f_2 \le \cdots \le f_n$$

- Schedule jobs in $S_{0,n+1}$

- c_{ij} be the size of a maximum-sized subset of mutually compatible jobs in S_{ij}.
- If $S_{ij} = \emptyset$, then $c_{ij} = 0$
- If $S_{ij} \neq \emptyset$, then $c_{ij} = c_{ik} + 1 + c_{kj}$ for some $k \in S_{ij}$. We pick the $k \in S_{ij}$ that maximizes the number of jobs:

$$c_{ij} = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max_{k \in S_{ij}} c_{ik} + c_{kj} + 1 & \text{if } S_{ij} \neq \emptyset \end{cases}$$

- Note we need only check $i < k < j$

To Solve S_{ij}

1. Choose $m \in S_{ij}$ with the earliest finish time. The Greedy Choice
2. Then solve problem on jobs S_{mj}

Graphs!

- Adjacency List, Adjacency Matrix
- Breadth First Search
- Depth First Search

BFS

- Input: Graph $G = (V, E)$, source node $s \in V$
- Output: $d(v)$, distance (smallest # of edges) from s to v $\forall v \in V$
- Output: $\pi(v)$, predecessor of v on the shortest path from s to v

BFS

BFS(V, E, s)
```
for each u in V \ {s}
do d(u) ← ∞
    π(u) ← NIL
d[s] ← 0
Q ← ∅
ADD(Q, s)
while Q ≠ ∅
do u ← POLL(Q)
    for each v in Adj[u]
    do if d[v] = ∞
          then d[v] ← d[u] + 1
               π[v] = u
               ADD(Q, v)
```


DFS

DFS

- Input: Graph $G = (V, E)$
- Output: Two timestamps for each node $d(v)$, $f(v)$,
- Output: $\pi(v)$, predecessor of v
  - not on shortest path necessarily

DFS(V, E)
```
for each u in V
do color(u) ← GREEN
    π(u) ← NIL
time ← 0
for each u in V
do if color[u] = GREEN
      then DFS-VISIT(u)
```


DFS (Visit Node—Recursive)

DFS-VISIT(u)
```
color(u) ← YELLOW
d[u] ← time++
for each v in Adj[u]
do if color[v] = GREEN
      then π[v] ← u
           DFS-VISIT(v)

color(u) ← RED
f[u] = time++
```


Classifying Edges in the DFS Tree

Given a DFS Tree G_π, there are four type of edges (u, v)

1. **Tree Edges**: Edges in E_π. These are found by exploring (u, v) in the DFS procedure
2. **Back Edges**: Connect u to an ancestor v in a DFS tree
3. **Forward Edges**: Connect u to a descendent v in a DFS tree
4. **Cross Edges**: All other edges. They *can* be edges in the same DFS tree, or can cross trees in the DFS forest G_π

Modifying DFS to Classify Edges

- DFS can be modified to classify edges as it encounters them...
- Classify $e = (u, v)$ based on the color of v when e is first explored...
- GREEN: Indicates Tree Edge
- YELLOW: Indicates Back Edge
- RED: Indicates Forward or Cross Edge

Stuff You Can Do with DFS

Topological Sort: The Whole Algorithm

1. DFS search the graph
2. List vertices in order of decreasing finishing time

Strongly Connected Components

1. Call $\mathrm{DFS}(G)$ to topologically sort G
2. Compute G^T
3. Call $\mathrm{DFS}(G^T)$ but consider vertices in topologically sorteded order (from G)
4. Vertices in each tree of depth-first forest for SCC

Spanning Tree

Kruskal's Algorithm

1. Start with each vertex being its own component
2. Merge two components into one by choosing the light edge that connects them
3. Scans the set of edges in increasing order of weight

Prim's Algorithm

- Builds one tree, so A is always a tree
- Let V_A be the set of vertices on which A is incident
- Start from an arbitrary root r
- At each step find a light edge crossing the cut $(V_A, V \setminus V_A)$

Kruskal's Algorithm

```
KRUSKAL(V, E, w)
  A ← ∅
  for each v in V
  do MAKE-SET(v)
  SORT(E, w)
  for each (u, v) in (sorted) E
  do if FIND-SET(u) ≠ FIND-SET(v)
        then A ← A ∪ {(u, v)}
              UNION(u, v)return A
```

Pseudocode for Prim

PRIM(V, E, w, r)

$Q \leftarrow \emptyset$
for each $u \in V$
do $key[u] \leftarrow \infty$
  $\pi[u] \leftarrow$ NILINSERT(Q, u)
  $key[r] = 0$
while $Q \neq \emptyset$
do $u \leftarrow$ EXTRACT-MIN(Q)
  for each $v \in Adj[u]$
  do if $v \in Q$ and $w_{uv} < key[v]$
    then $\pi[v] \leftarrow u$
      $key[v] = w_{uv}$

Shortest Paths

- (Single Source) shortest-path algorithms produce a label: $d[v] = \delta(s, v)$.
- Initially $d[v] = \infty$, reduces as the algorithm goes, so always $d[v] \geq \delta(s, v)$
- Also produce labels $\pi[v]$, predecessor of v on a shortest path from s.

Relax!

- The algorithms work by improving (lowering) the shortest path estimate $d[v]$
- This operation is called relaxing an edge (u, v)
- Can we improve the shortest-path estimate for v by going through u and taking (u, v)?

RELAX(u, v, w)

if $d[v] > d[u] + w_{uv}$
  then $d[v] \leftarrow d[u] + w_{uv}$
    $\pi[v] \leftarrow u$

Lemma, Lemma, Lemma

Path Relaxation Property

Let $P = \{v_0, v_1, \ldots v_k\}$ be a shortest path from $s = v_0$ to v_k. If the edges (v_0, v_1), (v_1, v_2), (v_{k-1}, v_k) are relaxed in that order, (there can be other relaxations in-between), then $d[v_k] = \delta(s, v_k)$

Bellman-Ford Algorithm

- Works with Negative-Weight Edges
- Returns true is there are no negative-weight cycles reachable from s, false otherwise

$\textsc{Bellman-Ford}(V, E, w, s)$
$\quad\textsc{Init-Single-Source}(V, s)$
$\quad$**for** $i \leftarrow 1$ to $|V| - 1$
$\quad$**do for each** (u, v) **in** E
$\qquad$**do** $\textsc{Relax}(u, v, w)$
$\quad$**for each** (u, v) **in** E
$\quad$**do if** $d[v] > d[u] + w_{uv}$
$\qquad$**then return** $False$

$\quad$**return** $True$

SSSP Dag

$\text{DAG-}\textsc{Shortest-Paths}(V, E, s, w)$
$\quad\textsc{Init-Single-Source}(V, s)$
$\quad$topologically sort the vertices
$\quad$**for each** u **in** topologically sorted$V$
$\quad$**do for each** $v \in Adj[u]$
$\qquad$**do** $\text{RELAX}(u, v, w)$

Dijkstra

$\textsc{Dijkstra}(V, E, w, s)$
$\quad\textsc{Init-Single-Source}(V, s)$
$\quad S \leftarrow \emptyset$
$\quad Q \leftarrow V$
$\quad$**while** $Q \neq \emptyset$
$\quad$**do** $u \leftarrow \textsc{Extract-Min}(Q)$
$\qquad S \leftarrow S \cup \{u\}$
$\qquad$**for each** $v \in Adj[u]$
$\qquad$**do** $\textsc{Relax}(u, v, w)$

- Dijkstra's Algorithm Runs in $O(E \lg V)$, with a binary heap implementation.

All Pairs Shortest Paths

- The output of an all pairs shortest path algorithm is a matrix $D = (d)_{ij}$, where $d_{ij} = \delta(i, j)$
- DP: $\ell_{ij}^{(m)}$ be the shortest path from $i \in V$ to $j \in V$ that uses $\leq m$ edges

$$\ell_{ij}^{(m)} = \min_{1 \leq k \leq n} (\ell_{ik}^{(m-1)} + w_{kj})$$

$\textsc{Extend}(L, W)$
$\quad$create $(n \times n)$ matrix L'
$\quad$**for** $i \leftarrow 1$ **to** n
$\quad$**do for** $j \leftarrow 1$ **to** n
$\qquad$**do** $\ell'_{ij} \leftarrow \infty$
$\qquad\quad$**for** $k \leftarrow 1$ **to** n
$\qquad\quad$**do** $\ell'_{ij} \leftarrow \min(\ell'_{ij}, \ell_{ik} + w_{kj})$

- This is just like matrix multiplication.
- We can speed this up.

Faster All-Pairs-Shortest-Paths

APSP2(W)
$L^{(1)} = W$
$m \leftarrow 1$
while $m \leq n - 1$
do $L^{(2m)} = \text{EXTEND}(L^m, L^m)$
    $m \leftarrow 2m$
    return $L^{(m)}$

- OK to "overshoot" $n - 1$, since shortest path labels don't change after $m = n - 1$ (since no negative cycles)
- "Repeated squaring" is a technique used to improve the efficiency of lots of other algorithms
- Analysis:

Floyd Warshall

- Floyd-Warshall Labels: Let $d_{ij}^{(k)}$ be the shortest path from i to j such that all intermediate vertices are in the set $\{1, 2, \ldots, k\}$.
- This simple observation, immediately suggests a DP recursion

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k = 0 \\ \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) & k \geq 1 \end{cases}$$

- We look for $D^{(n)} = (d)_{ij}^{(n)}$

Floyd-Warshall

FLOYD-WARSHALL(W)
$D^{(0)} = W$
for $k \leftarrow 1$ **to** n
do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
        do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{k-1}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
            return $D^{(n)}$

Flows

- A net flow is a function $f : V \times V \rightarrow \mathbb{R}^{|V| \times |V|}$ that satisfies three conditions:

1. Capacity Constraints:

$$0 \leq f(u, v) \leq c(u, v)$$

2. Skew Symmetry:

$$f(u, v) = -f(v, u) \ \forall u \in V, v \in V$$

3. Flow Conservation:

$$\sum_{v \in V} f(u, v) = 0 \ \forall u \in V \setminus \{s, t\}$$

The Maximum Flow Problem

Given $G = (V, E)$. source node $s \in V$, sink node $t \in V$, edge capacities c. Find a flow whose value is maximum.

Phlow Phacts

- For any cut (S,T), $f(S,T) = |f|$
- Resdiual capacity of arcs given flow:

$$c_f(u,v) \stackrel{\text{def}}{=} c(u,v) - f(u,v) \geq 0.$$

- Give flow f, we can create a residual network from the flow. $G_f = (V, E_f)$, with

$$E_f \stackrel{\text{def}}{=} \{(u,v) \in V \times V \mid c_f(u,v) > 0\},$$

so that each edge in the residual network can admit a positive flow.

Max-Flow Min-Cut Theorem

The following statements are equivalent

1. f is a maximum flow
2. f admits no augmenting path. (No (s,t) path in residual network)
3. $|f| = c(S,T)$ for some cut (S,T)

Ford-Fulkerson(V, E, c, s, t)

for $i \leftarrow 1$ **to** n
do $f[u,v] \leftarrow f[v,u] \leftarrow 0$
while $\exists$ augmenting path P in G_f
do augment f by $c_f(P)$

Analysis of this? Do better algorithms exist?

Maximum Bipartite Matching

- A graph $G = (V, E)$ is bipartite if we can partition the vertices into $V = L \cup R$ such that all edges in E go between L and R
- A matching is a subset of edges $M \subseteq E$ such that for all $v \in V$, ≤ 1 edge of M is incident upon it.

Maximum Bipartite Matching

Given (undirected) bipartite graph $G = (L \cup R, E)$, find a matching M of G that contains the most edges

Stuff To Know: EVERYTHING!

DP and Greedy

- Develop (and potentially solve small) problems via DP
- Activity Selection (or related problems): Greedy Works

Graphs

- BFS, DFS, and Analysis.
- Classifying edges in directed and undirected graphs
- Topological Sorting
- Finding Strongly Connected Components

Spanning Trees

- Kruskal's Algorithm (and analysis)
- Prim's Algorithm (and analysis)

More Stuff To Know...

Single Source Shortest Paths

- Distance Labels and RELAX
- Path Relaxation Property
- Bellman-Ford Algorithm
  - How to do it
  - When (Why?) it works
  - Analysis
- SSSP Dag
  - How to do it
  - When (Why?) it works
  - Analysis
- Dijkstra's Algorithm
  - How to do it
  - When (Why?) it works
  - Analysis

Even More Stuff To Know...

All Pairs Shortest Paths

- Analogue to Matrix Multiplication
- Floyd-Warshall
  - How to do it?
  - When (Why?) it works?
  - Analysis

Flows

- What is a flow?
- What is a cut?
- What is MFMC Theorem?
- How to create residual graph G_f?
- How to do Augmenting Paths algorithm (Ford Fulkerson/Edmonds Karp)
- Analysis

Next Time

- Quiz! April 4
- No Class: Friday April 6. Have a nice holiday! We start numerical methods on Monday