Dynamic Programming

EECS 477
Lecture 16, 11/7/2002

Floyd’s algorithm

- Shortest paths in a directed graphs between all the pairs of vertices
  - Dijkstra does paths from one seed vertex
- Graph $G=\{N=\{1,..,N\},A\}$
  - Arrows $A$ – stored in the edge length matrix $L[i,j] = \text{distance from } i \text{ to } j$, infinity if no edge
  - If $k$ is on the shortest path from $i$ to $j$, then $(i \text{ to } k)$, and $(k \text{ to } j)$ is optimal too

Floyd’s algorithm

- Constructing matrix $D$ of shortest path distances
  - $D_k$ is the matrix of shortest paths using only vertices $1..k$ as intermediate
  - $D_k[i,j] = \min \{ D_{k-1}[i,j], D_{k-1}[i,k]+D_{k-1}[k,j] \}$
  - Start with $D_0 = L$
  - $N \times N$ matrix $N$ times
    - Runtime $\Theta(N^3)$ (Dijkstra $N\Theta((A+N) \log N)$)

Dynamic Programming

- Solution splits into parts
- If a solution is optimal then its parts have to be optimal too

$$d(A,B) = \min_k \{d(A,M_k) + d(M_k,B)\}$$
**Floyd’s algorithm: an example**

- **Graph**

```
   B  | 7  |  C
 ---+---+---
   A  | 6  |  3  |  D
```

- **Matrix**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>inf</td>
</tr>
<tr>
<td>C</td>
<td>inf</td>
<td>0</td>
<td>5</td>
<td>inf</td>
</tr>
<tr>
<td>D</td>
<td>inf</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**TSP**

- **Traveling Salesman Problem**
  - Cities 1..n
  - Starts in city 1
  - Flies through all the remaining cities
  - Returns to city 1
  - Cost from city x to city y is $d(x,y)$
  - Need to minimize the total cost along the path

**TSP: algorithms**

- **Trivial algorithm**
  - Each solution is a permutation
    - Check all $O(n!)$ permutations
  - Dynamic programming
    - Held&Karp 1962

- **S** is subset of $\{2..n\}$, $x \in S$
- **Opt[S; x]** = length of the cheapest path starting in city 1 visiting all the cities in $S\{x\}$ and stopping in city x

**TSP: dynamic programming**

- **Optimality requires**
  - $\text{Opt}[\{x\}; x] = d(1,x)$
  - $\text{Opt}[S; x] = \min_{y \in S\{x\}} \{\text{Opt}[S\{x\}; y] + d(y,x)\}$

- **Optimal travel cost can be obtained as**
  - The minimum value of
    - $\text{Opt}[\{2,3,\ldots,n\}; y] + d(y,1)$
  - for all $y$. 
**TSP: DP algorithm**

- Run on sets of increasing cardinality
- \{1, 2, 3, 4\}, n=4
  - \text{Opt}\{2\}, \text{Opt}\{3\}, \text{Opt}\{4\}
  - \text{Opt}\{2, 3\}, \text{Opt}\{2, 3\}, \text{Opt}\{2, 4\}, \ldots
  - \text{Opt}\{2, 3, 4\}, \text{Opt}\{2, 3, 4\}, \text{Opt}\{2, 3, 4\}, \text{Opt}\{2, 3, 4\}, \text{Opt}\{2, 3, 4\}
- Memory required
  - \(\Theta(n \times 2^n)\)
  - Why?

**TSP: DP algorithm**

- Runtime
- There are \(C(n, k)\) subsets of size \(k\)
- There are \(k \times C(n, k)\) \text{Opt} values for each \(k\)
- To compute \text{Opt} value for \(k\)-subset requires \(\Theta(k)\) operations
- \(T(n) = \sum k \times C(n, k) = \Theta(n^2 \times 2^n)\)
- Later we'll see improved algorithm for a restricted Euclidean version of TSP

**Chained matrix multiplication**

- \(c_{ij} = \sum_k a_{ik} b_{kj}\)
- \(A\) – \(p\) by \(q\) matrix
- \(B\) – \(q\) by \(r\) matrix
- \(AB\) takes \(pqr\) scalar multiplication
- Example ABCD: what is the best order?
  - 2x3, 3x5, 5x2, 2x7
- Greedy algorithm does not work

**Chained matrix multiplication**

- \(D[0], D[1], \ldots, D[N]\) dimensions
- Matrix \(M_i\) has dimensions \(D[i-1] \times D[i]\)
- Optimality
  - \(P(i,i+s) = \min_{k,i<i+s} \{P(i,k) + P(k+1,i+s) + \sum D[i-1]D[k]D[i+s]\}\)
- Start with \(P(i,i+1) = D[i-1]D[i]D[i+1]\)
- Go to higher \(s\)
Chained multiplication

- Trivial algorithm enumerates all possibilities
  \[ T(n) = \sum_{k=1}^{n-1} T(k) T(n-k) \]
- \( T(n) \) are Catalan numbers
  - Number of binary trees
  - Grows like \( \Omega(4^n/n^2) \)
  - Each check takes \( \Omega(n) \) operations
  - Trivial runtime is in \( \Omega(4^n/n) \)

CMM: dynamic programming

- Fill the table
  \[ \text{Best}[x, x+s] \]
- Start at \( s=0 \) – diagonal
- Proceed for \( s=1..n \)
- Runtime \( \sum_{s=1}^{n-1} (n-s)s = \Theta(n^3) \)
  - For level \( s \) have \( n-s \) elements each has \( s \) choices to split