CS173
More Dynamic Programming

Tandy Warnow

December 9, 2018

CS 173

More Dynamic Programming Algorithms

Tandy Warnow

Overview

Material to be covered:

- Review of Longest Increasing Substring
- How to get Longest Increasing Subsequence
- All pairs shortest path problem
- A dynamic programming algorithm to solve the all pairs shortest path problem

Longest Increasing Substring

Input: sequence (or array) $X = x_1 x_2 \ldots x_n$ of integers
Output: increasing substring of X that is as long as possible.

Subproblem formulation:

- Let $M[i]$ denote the length of the longest increasing substring that ends at x_i.

Computing $M[i]$

Let $M[i]$ denote the length of the longest increase substring that ends at x_i.

Then how we set $M[i]$ depends on the value of i:

1. If $i = 1$ then $M[i] := 1$
2. If $i \geq 2$ then:
    - $M[i] := 1$ if $x_{i-1} \geq x_i$
    - $M[i] := 1 + M[i-1]$ if $x_{i-1} < x_i$

Why is this correct?

DP Algorithm for Longest Increasing Substring

Given $X = x_1, x_2, \ldots, x_n$, to find the *length* of the longest increasing substring:

- $M[1] := 1$
- For $i = 2$ up to n do:

```
    If $x_{i-1} \geq x_i$ then $M[i] := 1$
    Else $M[i] := 1 + M[i-1]$
```

- Return $\max\{M[1], M[2], M[3], \ldots, M[n]\}$

Note: to find the actual longest increasing substring, you have to do backtracing.

DP Algorithm for Longest Increasing Subsequence

Input: $X = x_1, x_2, \ldots, x_n$

Suppose we want to find the length of the longest increasing subsequence (rather than substring).

Recall $M[i]$ was the length of the longest increasing substring that ends at x_i.

Let's let $Q[i]$ denote the length of the longest increasing subsequence that ends at x_i.

Recall the difference between substrings and subsequences!

DP Algorithm for the Longest Increasing Subsequence

Let $Q[i]$ denote the length of the longest increasing subsequence that ends at x_i.

Then $Q[1] = 1$

Let $Pred[i]$ denote the set of indices j where:

- $1 \leq j < i$
- $x_j < x_i$

Then

- $Q[i] = 1$ if $Pred[i] = \emptyset$ and
- $Q[i] = \max\{Q[j] + 1 | j \in Pred[i]\}$ else

For $X = 1, 3, 4, 1, 2, 8$:

- $Pred[1] = \emptyset$
- $Pred[2] = \{1\}$
- $Pred[3] = \{1, 2\}$

Class exercise: computing remaining entries of *Pred* array.

DP Algorithm for the Longest Increasing Subsequence

Input: array $X = [x_1, x_2, \dots, x_n]$

Output: length of longest increasing subsequence

```
▶ $Q[1] := 1$
▶ For $i = 2$ up to $n$:
    ▶ IF $Pred[i] = \emptyset$ THEN $Q[i] := 1$
    ▶ ELSE $Q[i] := \max\{Q[j] + 1 | j \in Pred[i]\}$
▶ Return $\max\{Q[1], Q[2], Q[3], \dots, Q[n]\}$
```

Note: to find the actual longest increasing substequence, you have to do backtracing.

Designing DP algorithm

Most important thing is to come up with the right subproblem formulation!

All Pairs Shortest Path

The input is an undirected graph $G = (V, E)$ with positive edge weights on the edges.

Given a path P between two vertices v_i and v_j, the **cost** (or **length**) of P is the sum of the weights on the edges in the path. We write this as $Cost(P)$.

A shortest path between two vertices is one with minimum cost.

We want to find the length of the shortest path between every pair of vertices, and store this in an $n \times n$ matrix (where $n = |V|$).

All Pairs Shortest Path

- Input: graph $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$, and edge weights given by $w : E \rightarrow R^+$. (Hence $w(e)$ is the weight of edge e.)
- Output: D, an $n \times n$ matrix, so that $D[i, j]$ is the length of the shortest path from v_i to v_j. Note we set $D[i, i] = 0$ for all $i = 1, 2, \ldots, n$.

Floyd-Warshall Algorithm

We present the Floyd-Warshall algorithm to solve All Pairs Shortest Path, beginning with the definition of its subproblems.

MAX(P): For $i \neq j$, and given a path P from v_i to v_j, we look at the *internal nodes* of the path (i.e., everything except the endpoints), and let $MAX(P)$ denote the maximum index of any internal node.

If the path is a single edge, then $MAX(P) = 0$.
Thus

- For $P = v_3, v_1, v_2, v_5, v_7$, $MAX(P) = 5$.
- For $P = v_5, v_1, v_2, v_7$, $MAX(P) = 2$.
- For $P = v_1, v_2$, $MAX(P) = 0$ (no internal nodes).

Floyd-Warshall Algorithm

The input to Floyd Warshall is a graph $G = (V, E)$ with non-negative weights on its edges, denoted by $w(v_i, v_j)$, where $(v_i, v_j) \in E$.
Floyd-Warshall computes subproblems **M[i,j,k]:**

- $M[i, j, k]$ is the length of the shortest path P from v_i to v_j such that $MAX(P) \leq k$.
- If $i = j$, we set $M[i, j, k] = 0$.
- If $i \neq j$ and there is no path between v_i and v_j satisfying $MAX(P) \leq k$, then we set $M[i, j, k] = \infty$.
- We let $k = 0, 1, 2, \ldots, n$, and $1 \leq i, j \leq n$.

Questions:

- What does $M[1, 2, 0]$ mean?
- What does $M[1, 2, 2]$ mean?
- What does $M[1, 2, 3]$ mean?
- What does $M[1, 2, 5]$ mean?

Floyd-Warshall Algorithm

Remember $M[i,j,k]$ is the length of the shortest path P from v_i to v_j such that $MAX(P) \leq k$.
Suppose we were able to compute (correctly) all $M[i,j,k]$, for $1 \leq i,j \leq n$ and $0 \leq k \leq n$.

- Question: How could we compute the length of the shortest path from v_i to v_j?

Floyd-Warshall Algorithm

Remember $M[i,j,k]$ is the length of the shortest path P from v_i to v_j such that $MAX(P) \leq k$.
Suppose we were able to compute (correctly) all $M[i,j,k]$, for $1 \leq i,j \leq n$ and $0 \leq k \leq n$.

- Question: How could we compute the length of the shortest path from v_i to v_j?
- Answer: it is the same as $M[i,j,n]$.

So once we have computed all entries in the 3-dimensional matrix M, we return the 2-dimensional matrix obtained by setting $k = n$.

Floyd-Warshall Algorithm, k=0

When $k = 0$ we are asking about the lengths of paths that have no internal nodes.

Case: $i = j$**:** Set $M[i, i, 0] = 0$

Case: $i \neq j$**:** $M[i, j, 0]$ is the length of the shortest path P from v_i to v_j with $MAX(P) = 0$, or ∞ if no such path exists.

If the path P exists, it is a single edge e, and its weight is $w(e)$.

Hence, $M[i, j, 0] = w(v_i, v_j)$ if $(v_i, v_j) \in E$, and otherwise $M[i, j, 0] = \infty$.

Floyd-Warshall Algorithm

After we compute all entries of the matrix M with $k = 0$, can we compute the entries with $k = 1$?

If $i = j$, we set $M[i, j, 1] = 0$.

For $i \neq j$, consider a shortest path P from v_i to v_j with $MAX(P) \leq 1$. What does it look like?

Floyd-Warshall Algorithm

Consider a shortest path P from v_i to v_j with $MAX(P) \leq 1$.
Cases:

- P is a single edge e, and so $Cost(P) = w(e) = M[i,j,0]$.
- P has an internal vertex, which must be v_1.
  Hence P has two edges, (v_i, v_1) and (v_1, v_j).
  Then $Cost(P) = w(v_i, v_1) + w(v_1, v_j)$.
  Note that $w(v_i, v_1) = M[i,1,0]$ and $w(v_1, v_j) = M[1,j,0]$.

Hence $M[i,j,1] = \min\{M[i,j,0], M[i,1,0] + M[1,j,0]\}$.

Floyd-Warshall Algorithm

After we compute all entries of the matrix M with $k = 0, 1, \ldots, K-1$, can we compute the entries with $k = K$?

Consider a shortest path P from v_i to v_j with $MAX(P) \leq K$. Cases:

- P satisfies $MAX(P) \leq K-1$. Then $Cost(P) = M[i, j, K-1]$.
- P satisfies $MAX(P) = K$.
  Hence $v_K \in P$. Analyzing this is a bit more complicated, but we will show the path P satisfies
  $Cost(P) = M[i, K, K-1] + M[K, j, K-1]$.

Floyd-Warshall Algorithm

Consider a shortest path P from v_i to v_j with $MAX(P) = K$.
Hence $v_K \in P$.
Write P as the concatenation of two paths, P_1 from v_i to v_K and P_2 from v_K to v_j.
Do you agree that $Cost(P) = Cost(P_1) + Cost(P_2)$?
Questions:

- What is $MAX(P_1)$?
- What is $MAX(P_2)$?

Floyd-Warshall Algorithm

P is a shortest path P from v_i to v_j with $MAX(P) = K$.
We write P as the concatenation of P_1 from v_i to v_K and P_2 from v_K to v_j.
Note that $MAX(P_1) \leq K - 1$ and $MAX(P_2) \leq K - 1$.
Hence,

- $Cost(P_1) = M[i, K, K - 1]$
- $Cost(P_2) = M[K, j, K - 1]$
- $Cost(P) = M[i, K, K - 1] + M[K, j, K - 1]$.

Floyd-Warshall Algorithm

$$M[i,j,K] = \min\{M[i,j,K-1], M[i,K,K-1] + M[K,j,K-1]\}$$

Floyd-Warshall Algorithm

Algorithm:

- %Set base cases:
- $M[i, i, 0] := 0$
- For $i \neq j$, if $(v_i, v_j) \in E$ then $M[i, j, 0] := w(v_i, v_j)$, else $M[i, j, 0] := \infty$
- For $k = 1$ up to n DO:
  - For all $i = 1, 2, \ldots, n$, $M[i, i, k] := 0$
  - For all pairs i, j with $i \neq j$, $M[i, j, k] := \min\{M[i, j, K-1], M[i, K, K-1] + M[K, j, K-1]\}$

Easy to see this is $O(n^3)$ (in fact, $\Theta(n^3)$).

Floyd-Warshall Algorithm

- As presented, this algorithm only returns the length (i.e., cost) of the shortest path between every pair of nodes. It does not return the path itself, or directly help you find the path. The assigned reading shows you how to can store information to help you reconstruct the shortest path.
- This algorithm, as described, was only for undirected graph with positive edge weights. It's a trivial modification to deal with directed graphs or with weights that can be negative. (It just gets strange to think about shortest paths when there are negative cycles...)

Dynamic Programming Algorithms

When you are asked to create a DP algorithm, remember to:

- Define what your variables mean
- Show the base cases and how you'll compute them
- Show how you'll compute the other variables, using previously computed variables (this is the "recursive definition")
- State the order in which you compute the variables (remember - bottom-up)
- Show where your "answer" will be (or how you'll compute your answer from your variables)
- Explain why your algorithm is correct
- Analyze the running time