Lecture 26:
- Intro to Dynamic Programming
- Floyd-Warshall

Announcements
- PS-6 due
- PS-7 out
- HW-7 due Fri
- Quiz 6 Thurs

Dynamic Programming Intro

Basic Idea of Dynamic (aka “Bottom Up”) Programming:
- can help make tricky and expensive recursive algorithms more efficient
- leverages “memoization” – i.e., remembering results of computations
- we’ll look at some examples then introduce general strategy

Fibonacci Numbers: Often used as an example for learning recursion ...
- Recall: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-2} + F_{n-1}$ ... for $n \geq 0$

Algorithm: Naive Fib(n)

```
begin
  if $n = 0$ then return 0
  else if $n = 1$ then return 1
  else return Fib($n - 2$) + Fib($n - 1$)
end
```

The problem: too much repeated computation
- note: not all recursive solutions have this issue, e.g., mergesort
Dynamic Programming Intro

Examples:

• \( F_2 = F_0 + F_1 = 1 \)

• \( F_3 = F_1 + F_2 = F_1 + F_0 + F_1 = 1 + 0 + 1 = 2 \)

• \( F_4 = F_2 + F_3 = F_0 + F_1 + F_1 + F_0 + F_1 = 3 \)

• \( F_5 = F_3 + F_4 = F_1 + F_0 + F_1 + F_0 + F_1 + F_1 + F_0 + F_1 = 5 \)

Check In: Can you spot the problem?

Check In: What is the cost?

• in a full binary tree there are \( 2^{n-1} \) nodes

• above tree is full at height \( n/2 \) ... since we subtract 2 on left-most path

• thus, \( O(2^n) \) recursive calls ... thus exponential!
Dynamic Programming Intro

Check In: Can we do better? How?
- instead of calculating same values over and over, only calculate them once
- for example, maintain an array of Fib values and lookup before calculating

```
Algorithm: Dynamic Programming Fib(n)
begin
  A ← an n + 1 element array initialized to 0's
  A[1] ← 1
  for i = 2 to n do
  return A[n]
end
```

Check In: What is the cost?
- Only $O(n)$ ... $O(n)$ to initialize array, $O(n)$ iterations

All-Pairs Shortest Paths: Floyd Warshall

Similar to single-source shortest paths, but for all “sources”
- Given a directed, weighted graph $G = (V, E)$
- Either find the shortest weighted paths $(u, v)$ for each $u, v \in V$
- Or report a negative cycle (if one exists)

A simple (naive) approach:
- for each $u \in V$, use Bellman-Ford to compute shortest paths from $u$
- cost: Bellman-Ford is $O(VE)$, so $O(V^2E)$
- however, grows with $E$ up to “quartic” if dense ... i.e., $O(V^4)$

We can do better using a dynamic programming solution ... Floyd-Warshall
- Not hard to see there would be a lot of recomputation in “naive” approach
- Idea: structure problem as subproblems we only have to compute once
Floyd Warshall

Subproblems:
- assume vertices labeled 1, 2, ..., n (we already do, but from 0, ..., n − 1)
- a subproblem restricts graph “size” based on an index \( k \)
- the subgraphs are “indexed by \( k \)” (size is based on \( k \))

Given a source vertex \( u \) and destination vertex \( v \)
- an internal path vertex is a node other than \( u \) and \( v \) (on the path)

\( W_{k,u,v} \) is the minimum path weight from \( u \) to \( v \) such that:
- the path only contains internal vertices labeled 1, ..., \( k \) (not exceeding \( k \))
- the path does not contain a directed cycle
- if there isn’t a path from \( u \) to \( v \), then \( W_{k,u,v} = +\infty \)

Check In: What are the following for \( W_{k,u,v} \)?
- \( W_{0,1,5}, W_{1,1,5}, W_{2,1,5} \) ... +\( \infty \) (no such 1 \( \rightarrow \) 5 paths for \( k \))
- \( W_{3,1,5} \) ... 3 (the path 1 \( \rightarrow \) 2 \( \rightarrow \) 3 \( \rightarrow \) 5)
- \( W_{4,1,5} \) ... −20 (the path 1 \( \rightarrow \) 4 \( \rightarrow \) 5)
- \( W_{5,1,5} \) ... −20 (same as \( W_{4,1,5} \))
- \( W_{5,2,4} \) ... +\( \infty \) (no path from 2 to 4)
Floyd Warshall

Properties of the $W_{k,u,v}$ subproblems ...

- $(n + 1) \cdot n \cdot n$ total subproblems (for $n = |V|$) ... thus $O(n^3)$
- when $k = n$, we’ve computed all of the shortest paths (for all pairs)
- for a $u, v$, possible paths increase and $W_{k,u,v}$ can only decrease as $k$ grows

Optimal Substructure

Assume $P$ is an optimal (shortest) $u$-$v$ path for a specific value of $k$

- then either $k$ is an internal node in $P$ or it isn’t

(1) If vertex $k$ is not an internal node of $P$ ...

- then $P$ is also an optimal (shortest) path for $k - 1$
- i.e., $W_{k-1,u,v} = W_{k,u,v}$ is the optimal (shortest) path weight

(2) If vertex $k$ is an internal node of $P$ ...

- let $P_1$ be the part of $P$ from $u$ to $k$ ... “prefix”
- let $P_2$ be the part of $P$ from $k$ to $v$ ... “suffix”

and note that:

- $P_1$ is a shortest $u$-$k$ path weighing $W_{k-1,u,k}$ ... $k - 1$ since $k$ ends $P_1$
- $P_2$ is a shortest $k$-$v$ path weighing $W_{k-1,k,v}$ ... $k - 1$ since $k$ starts $P_1$

... proof via contradiction (e.g., if $P_1$ isn’t shortest, then $P$ can be shorter)

(1) and (2) mean that: ... optimal substructure!

$$W_{k,u,v} = \min\{W_{k-1,u,v}, W_{k-1,u,k} + W_{k-1,k,v}\}$$

which suggests a dynamic programming algorithm for all-pairs shortest paths!
Floyd Warshall

**Base Cases:** Recall we assume vertices labeled from $1 \ldots |V|$

**Check In:** For $k=0$ what is $W_{0,u,v}^k$

- when $u=v$? ... $0$
- when $u \neq v$ but $(u,v) \in E$? ... the edge weight $w(u,v)$
- when $u \neq v$ and $(u,v) \not\in E$? ... $+\infty$

**Check In:** How do we know if there is a negative cycle?

- Note that a path from $u-u$ has a weight of 0 (base case)
- Thus, a $u-u$ path weight less than 0 implies a negative cycle!
- We can check this by checking if $W_{|V|,u,u} < 0$ for each vertex $u \in V$

---

**Algorithm:** Floyd Warshall

**Input:** Directed weighted graph $G=(V,E)$ with $n=|V|$

**Result:** Minimum path distances $dist(u,v)$ for each $u,v \in V$

1. begin
2. let $A$ be an $(n+1) \times n \times n$ 3-d array
3. for $u=1$ to $n$ do // base cases
4.  for $v=1$ to $n$ do
5.   if $u=v$ then $A[0][u][v] \leftarrow 0$
6.   else if $(u,v) \in E$ then $A[0][u][v] \leftarrow w(u,v)$
7.   else $A[0][u][v] \leftarrow +\infty$
8. for $k=1$ to $n$ do // iteratively solve subproblems
9.  for $u=1$ to $n$ do
10.   for $v=1$ to $n$ do
11.     $A[k][u][v] \leftarrow \min\{A[k-1][u][v], A[k-1][u][k]+A[k-1][k][v]\}$
12. for $u=1$ to $n$ do // check for negative cycles
13.  if $A[n][u][u] < 0$ then return false
14. for $u=1$ to $n$ do // set the $dist$ values
15.  for $v=1$ to $n$ do
16.     $dist(u,v) \leftarrow A[n][u][v]$
17. return true