CS 218 Design and Analysis of Algorithms

Nutan Limaye

Indian Institute of Technology, Bombay
nutan@cse.iitb.ac.in

Module 1: Basics of algorithms

Bellman and Ford's algorithm

A dynamic programming approach to shortest path. (Very similar to DFS.)

Step 1 Figuring out the types of sub-problems.

Let $\mathbf{Opt}(v, t)$ be the minimum weight of v to t path.

We wish to compute $\mathbf{Opt}(s, t)$.

For any $v \in V$, if we are able to compute $\mathbf{Opt}(v, t)$, then we will be done.

Bellman and Ford's algorithm

A dynamic programming approach to shortest path.

Step 2 Designing the recursion.

For some $v \in V$, let P denote the optimal path from v to t corresponding to $\mathbf{Opt}(v, t)$.

The path P must use some vertex $u \in V$ among the neighbours of v

$\mathbf{Opt}(v, t) = \mathbf{Opt}(u, t) + w(v, u)$, where (v, u) is the first edge going out of v on P.

Bellman and Ford's algorithm

A dynamic programming approach to shortest path.

Step 2 Designing the recursion.

$$\mathbf{Opt}(v,t) = \min_{u \in V, (v,u) \in E} \{\mathbf{Opt}(u,t) + w(v,u)\}$$

Step 3 Deciding the memoization strategy.

Remember the values of $\mathbf{Opt}(u,t)$ for previously computed $u \in V$.

Step 4 Acyclic sub-problem dependencies.

Follows because the underlying graph is acyclic.

Step 5 Analysing the time complexity.

$\#$ sub-problems $\times$ time/sub-problem $=$
$\leq |V|(|V|+1) = O(|V|+|E|)$

What if the graph has cycles, but no negative weight cycles?

Bellman and Ford's algorithm

A dynamic programming approach to shortest path. (Very similar to Dijkstra's algorithm.)

Step 1 Figuring out the types of sub-problems.

Let $\mathbf{Opt}(v,i)$ be the minimum weight of v to t path that uses at most i-many edges.

We wish to compute $\mathbf{Opt}(s,n-1)$. This is because the graph is acyclic.

For any $v \in V$, and $i \in [n-1]$, if we are able to compute $\mathbf{Opt}(v,i)$, then we will be done.

$\boxed{i \text{ to } n}$ me number

v

$f(i) = \sum f(i-k)$ if $a(i \text{ to } k)$ is a valid encoding

Bellman and Ford's algorithm

A dynamic programming approach to shortest path.

Step 2 Designing the recursion.

For some $v \in V$ and $i \in [n-1]$, let P denote the optimal path from v to t corresponding to $\mathbf{Opt}(v, i)$.

If the path uses at most $i-1$ edges, then $\mathbf{Opt}(v, i) = \mathbf{Opt}(v, i-1)$.

If the path uses i edges, then $\mathbf{Opt}(v, i) = \mathbf{Opt}(u, i-1) + w(v, u)$, where (v, u) is the first edge going out of v on P.

Bellman and Ford's algorithm

A dynamic programming approach to shortest path.

Step 2 Designing the recursion.

For some $v \in V$ and $i \in [n-1]$, let P denote the optimal path from v to t corresponding to $\mathbf{Opt}(v, i)$.

If the path P uses at most $i-1$ edges, then $\mathbf{Opt}(v, i) = \mathbf{Opt}(v, i-1)$.

If the path P uses i edges, then $\mathbf{Opt}(v, i) = \mathbf{Opt}(u, i-1) + w(v, u)$, where (v, u) is the first edge going out of v on P.

Bellman and Ford's algorithm

A dynamic programming approach to shortest path.

Step 2 Designing the recursion.

This leads to the following recursion.

If $i > 0$,

$$\mathbf{Opt}(v, i) = \min\left\{\mathbf{Opt}(v, i{-}1), \min_{u \in V, (v,u) \in E}\{\mathbf{Opt}(u, i{-}1) + w(v, u)\}\right\} \quad ($$

Bellman and Ford's algorithm

Step 3 Decide on the memoization strategy.

We will create a table M with n rows and n columns.

The $[v, i]$th entry of M will contain the weight of the lowest weight path from v to t that uses at most i edges.

Bellman and Ford's algorithm

Step 4 Check that the sub-problem dependencies are acyclic.

From the recursion $(*)$, we know that the i sub-problem uses values of sub-problems $j < i$.

This immediately gives acyclicity of the underlying graph.

Bellman and Ford's algorithm

```
ShortestPath(G, s, t)
   M[t, 0] = 0
   for v ∈ V ∖ {t} and i ∈ {0, ..., n − 1} do
      M[v, i] = ∞
   end for
   for i = 1 to n − 1 do
      for v ∈ V (in any order) do
         Compute M[v, i] using the recurrence (*)
      end for
   end for
   Return M[s, n − 1]
```

Running time analysis of Bellman and Ford

- The algorithm fills the table M.
- The table has $O(n^2)$ entries.
- Each entry can be filled using $O(n)$ time.
- Hence the total time taken is $O(n^3)$.