Recitation 12

Dynamic Programming

12.1 Announcements

- *DPLab* has been released and is due *Dec 1*.
- Happy Thanksgiving

12.2 Matrix Chain Product

Definition 12.1. *In the* matrix chain product problem*, we are attempting to find the cheapest way to multiply a chain of n matrices. I.e., determine a parenthesization of the expression*

$$A_1 \times A_2 \times \ldots \times A_n$$

such that cost of evaluating the expression is minimized.

Task 12.2. *Write a top-down solution to the matrix chain product problem. Specifically, assuming you have a cost function*

val $\mathtt{cost}$: $\mathtt{int} \star \mathtt{int} \star \mathtt{int} \rightarrow \mathtt{real}$

where $\mathtt{cost}(x, y, z)$ is the cost of multiplying two matrices with dimension (x, y) and (y, z), write a function

val $\mathtt{MCP}$: $(\mathtt{int} \star \mathtt{int})\ \mathtt{Seq.t} \rightarrow \mathtt{real}$

which takes a sequence of pairs (h_i, w_i) (the dimensions of the i^{th} matrix) and returns the cheapest cost of multiplying those matrices.

Be sure to clearly specify your subproblems. Determine the work of your algorithm by assuming that each distinct subproblem is computed only once. Determine the span of your algorithm by describing the DAG of subproblem dependencies and identifying a bottom-up ordering of the subproblems.

Algorithm 12.3. *Top-down matrix chain product.*

```
fun MCP(D) =
   let
      % useful helpers for the $i^{th}$ height and width
      fun $h(i)$ = let val $(x, \_)$ = $D[i]$ in $x$ end
      fun $w(i)$ = let val $(\_, x)$ = $D[i]$ in $x$ end

      % Subproblem: assuming $i < j$, the cheapest cost of multiplying
      % matrices $i$ through $j$ (exclusive at $j$)
      fun MCP'$(i, j)$ =
         if $j - i \leq 1$ then 0
         else $\min_{i<k<j} (\mathtt{MCP'}(i, k) + \mathtt{cost}(h(i), h(k), w(j-1)) + \mathtt{MCP'}(k, j))$
   in
      MCP'$(0, |D|)$
   end
```

Work: in the algorithm above, there are $O(|D|^2)$ distinct subproblems. At each subproblem (i, j), we require $O(j - i) \subseteq O(|D|)$ work, and therefore the total work required is $O(|D|^3)$.

Span: let the "length" of a subproblem `MCP'` (i, j) be $j - i$. In the DAG, we notice that

1. subproblems of length 1 have no dependencies,
2. subproblems of length 2 depend upon subproblems of length 1,
3. subproblems of length 3 depend upon subproblems of length 1 and 2,
4. etc.

In general, subproblems of length k depend upon other subproblems which have lengths strictly less than k. So, our bottom-up ordering should be to compute subproblems in increasing order of their length.

We can implement line 11 of the given algorithm using `(reduce min` ∞`)` after a `tabulate`. For a subproblem of length k, this requires $O(\log k) \subseteq O(\log |D|)$ span. The longest chain of dependencies in the DAG is length $|D|$, and therefore the total span of the bottom-up approach is $O(|D| \log |D|)$.

The *Bellman-Ford* algorithm, which we covered in the shortest path section, is another example of dynamic programming.

Task 12.4. *The code for Bellman-Ford given in the textbook is written in a bottom-up fashion. Rewrite this code in a top-down style. Once again, be sure to identify the subproblems and the DAG of dependencies. Use these observations to re-derive the work and span of Bellman-Ford.*

Algorithm 12.5. *Top-down Bellman-Ford*

```
 1  fun BF $(G = (V, E), s)$ =
 2    let
 3      % Subproblem: the shortest distance from $s$ to $v$ using $k$ or fewer hops
 4      fun $\delta(v, k)$ =
 5        if $(v = s \wedge k = 0)$ then $0$
 6        else if $k = 0$ then $\infty$
 7        else $\min\left(\delta(v, k-1), \min_{(u,v)\in E}(\delta(u, k-1) + w(u, v))\right)$
 8    in
 9      $\{v \mapsto \delta(v, |V| - 1) : v \in V\}$
10    end
```

Work: there are $|V|^2$ distinct subproblems, since for each $v \in V$, there are $|V|$ possible values for the argument "k". On line 6, we require linear work in the number of in-neighbors of the vertex v. This observation lets us naively upper bound the work by $O(|V|^2|E|)$, since each vertex has at most $|E|$ in-neighbors:

$$\begin{aligned} &\sum_v \sum_k O(|N_G^-(v)|) \\ \le &\sum_v \sum_k O(|E|) \\ = &\,O(|V|^2|E|). \end{aligned}$$

However, this is a crude bound. If we instead just flip around the summation, then we notice a familiar equation: the sum of in-degrees of all vertices in a graph is equal to the number of edges.

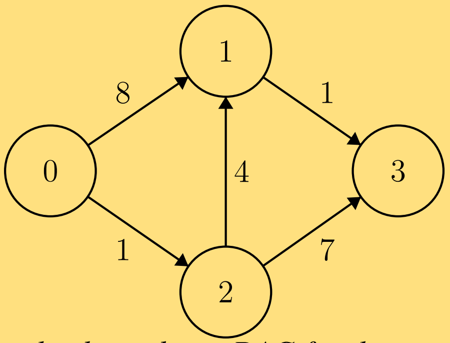
$$\begin{aligned} &\sum_v \sum_k O(|N_G^-(v)|) \\ = &\sum_k \boxed{\sum_v O(|N_G^-(v)|)} \\ = &\sum_k O(|E|) \\ = &\,O(|V||E|). \end{aligned}$$

Span: in the DAG, we notice that each subproblem $\delta(v, k)$ could be dependent upon $\delta(u, k-1)$ for every $u \in V$ (if, for example, every other vertex is an in-neighbor of v), but no others. Therefore we can use a bottom-up ordering which calculates subproblems in increasing order of k. We require $O(\log |V|)$ span at line 6, and the longest chain of dependencies is $O(|V|)$. Therefore the span of Bellman-Ford is $O(|V| \log |V|)$.

Remark 12.6. *We're assuming constant-time lookup for the results of subproblems. In Bellman-Ford, if we assume vertices are integer labeled, then this assumption is perfectly acceptable. If the vertices were not integer labeled, however, then these lookups require logarithmic work. This contributes a multiplicative factor of* $\log |V|$ *in the work of Bellman-Ford.*

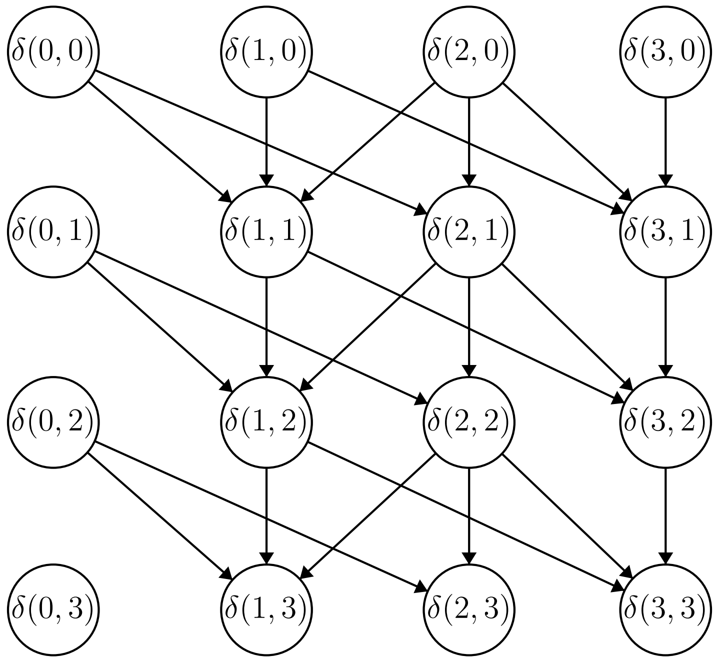
12.3 Additional Exercises

Exercise 12.7. *Describe the DAG of dependencies between subproblems $\delta(v, k)$ in the Bellman-Ford algorithm; i.e., each vertex is a pair (v, k), and there is an arc from (v', k') to (v, k) if we need to know the value $\delta(v', k')$ in order to calculate $\delta(v, k)$.*



Draw the dependency DAG for the example graph given above.

Except for $v = s$, each $\delta(v, k)$ is dependent upon $\delta(v, k-1)$ as well as $\delta(u, k-1)$ for every u which is an in-neighbor of v.


$\delta(0,0)$
$\delta(1,0)$
$\delta(2,0)$
$\delta(3,0)$
$\delta(0,1)$
$\delta(1,1)$
$\delta(2,1)$
$\delta(3,1)$
$\delta(0,2)$
$\delta(1,2)$
$\delta(2,2)$
$\delta(3,2)$
$\delta(0,3)$
$\delta(1,3)$
$\delta(2,3)$
$\delta(3,3)$