Department of Electrical and Computer Engineering

Dynamic Programming

ECE 1762 Algorithms and Data Structures
Spring Semester, 2004 — U Toronto

The main difficulty when dealing with dynamic programming problems is finding the appropriate (recursive) formulas for the quantities needed to be computed. What follows, including the problems discussed in class, is a terse, but representative, list of dynamic programming problems and a short representation of their solutions:

• **Binomial Coefficients.** We would like to efficiently compute $\begin{pmatrix} n \\ k \end{pmatrix}$. Let $C(n,k) = \begin{pmatrix} n \\ k \end{pmatrix}$, then we could have:

$C(n,0) = 1$,

$C(n,n) = 1$, and

$C(n,k) = C(n-1,k-1) + C(n-1,k)$

• **Context Free Language Membership (The Cocke-Younger-Kasami or CYK Algorithm).** Suppose we have a grammar G in Chomsky Normal Form[1], and a string x, $\mid x \mid = n$. Determine whether x is in the language generated by the grammar G. Let use x_{ij} for the substring consisting of characters i through j (inclusive) whenever $i \leq j$. Let $u[i,j]$ be the set of all nonterminals symbols generating x_{ij}. It is easy to see that $u[i,i]$ can be generated directly from all productions of the form $A \rightarrow x_{ii}$, while for $i < j$ we can have:

$$u[i,j] = \cup_{k=i+1}^{j-1} A \mid A \rightarrow BC \ for\ some\ B \in u[i,k]\ and\ C \in u[k+1,j].$$

u has size $O(n^2)$ and it is not that difficult to show that the entire algorithm can run in time $O(n^3)$ given an appropriate representation for grammar G.

• **Transitive Closure.** Let R be a *binary relation* over an ordered set S, $\mid S \mid = n$. We define the *transitive closure* of R (denoted R^+) as follows:

(a) For $a, b \in S$, if $(a,b) \in R$ then $(a,b) \in R^+$.

(b) For $a, b, c \in S$, if $(a,b) \in R$ and $(b,c) \in R$ then $(a,c) \in R^+$.

(c) Nothing else is in R^+ unless it follows from (a) or (b).

[1] Every production is of the form $A \rightarrow BC$ or $A \rightarrow \alpha$, for nonterminals A, B, C and terminal symbols α.

Let the elements of the set S be ordered $1, 2, \ldots, n$. Let $m_k[i,j]$ be true exactly when (i,j) belongs to the transitive closure of R if and only if it uses elements of the subset of S, $S' = S - n, n-1, \ldots, k+1$. Then

$m_0[i,j]$ = TRUE if and only if $(i,j) \in R$

and for $k > 0$ we have that:

$m_k[i,j]$ = TRUE if and only if $m_{k-1}[i,j] \vee (m_{k-1}[i,k] \wedge m_{k-1}[k,j])$ is TRUE.

In written words, $(i,j) \in R^+$ by using pairs of elements of R from the set $1, \ldots, k$ if and only if $(i,j) \in R^+$ already, or $(i,k) \in R^+$ and $(k,j) \in R^+$. The desired matrix is m_n, the above construction can be easily done in $O(n^3)$ time using no more than $O(n^2)$ space.

• **All–Pairs Shortest Path, non-negative weights (Floyd's Algorithm).** Given a graph $G = (V, E)$, and a *non-negative* cost $cost[i,j]$ for each edge $(i,j) \in E$, compute the length of the shortest path from i to j, for all possible pairs of vertices $i, j \in E$ (for all graph problems, assume that vertices are numbered as $1, \ldots, n$).

Let $m_k[i,j]$ be the length of the shortest path from i to j that uses *only* intermediate vertices from the set $1, \ldots, k$. Then for any k we have that

$m_k[i,i] = 0$

and for $i \neq j$,

$m_0[i,j] = cost[i,j]$.

Finally, for $k > 0$ we can write:

$m_k[i,j] = min(m_{k-1}[i,j], (m_{k-1}[i,k] + m_{k-1}[k,j]))$.

The latter formula means that the shortest path from i to j is the shorter of the following: the path from i to j which passes through vertices $1, \ldots, k-1$ only; and the path we get if we link the shortest past from i to k through vertices $1, \ldots, k-1$ and the shortest path from k to j if we use vertices from the set $1, \ldots, k-1$. The matrix m_n contains the cost of the shortest path for each pair of vertices. If we compute matrices m_i in order $(m_0, m_1, \ldots, m_n)$ then we clearly need $O(n^2)$ space, and the whole algorithm can run in $O(n^3)$ time.

• **All–Pairs Shortest Paths with arbitrary weights.** The problem is formulated as the one above with the exception that we impose no restriction on the *cost* function assigned to each edge of the graph. Similarly,

$m_k[i,i] = 0, for\ all\ k$, and $m_0[i,j] = cost[i,j]$,

while for $k > 0$ we can recursively obtain our values if we define $m_k[i,j]$ as follows:

$m_k[i,j] = min m_{k-1}[i,j], (m_{k-1} + \sum_{l=0}^{\infty} m_{k-1}[k,k] + m_{k-1}[k,j])$.

The above formula reminds the one of the previous problem with the addition of the case of a *infinite cycles:* if there's a cycle with an overall negative sum of weights from vertex k back to itself, we can always decrease the length of a shortest path going through k by taking an extra (negative) cycle, hence making the whole path arbitrarily small. In other words, if this is the case, *no* shortest path exists (since the addition of extra negative cycles makes the value smaller and

smaller), and we define the cost of this path to be $-\infty$. Similarly, space and time requirements for the above algorithm are $O(n^2)$ and $O(n^3)$ respectively.

• **Regular Expressions from Automata (Kleene's Algorithm).** Given a graph $G = (V, E)$ that represents a deterministic finite automaton, with start state s and (unique) final state f. Let Σ be a finite alphabet, and for every edge (u, v), let $sym[u, v]$ be the character which labels that edge. Constrcut a regular expression which describes the set of all paths from s to f.

Let $r_k[i, j]$ be the regular expression which describes the set of all paths from i to j, using only intermediate vertices (states) in the set $1, \ldots, k$. Then

$$r_o[i, j] = sym[i, j]$$

and when $k > 0$

$$r_k[i, j] = r_{k-1}[i, j] + (r_{k-1}pi, k]r^\star_{k-1}[k, k]r_{k-1}[k, j]).$$

Once again, the desired expression is $r_n[s, f]$ and with proper representation for regular expressions (e.g. trees), the algorithm runs in $O(n^3)$.

Note: The last four problems are all instances of a much more general theory of path algebras (see textbook), where a graph is labeled with elements of a closed semi-ring, and there are operations $\oplus$ and $\otimes$, for which $\oplus_{i=0}^{\infty} a^i$ is defined. The general form of the solution is

$$r_o[i, j] = label[i, j]$$

and when $k > 0$

$$r_k[i, j] = r_{k-1}[i, j] \oplus (r_{k-1}[i, k] \otimes (\oplus_{i=0}^{\infty} r_{k-1}[k, k])^i r_{k-1}[k, j]).$$