All-Pairs Shortest Paths (no negative cycles)

Sergei Uversky
2/5/2015

1 Shortest paths

The shortest path problem is as it sounds - given a graph, we want to find the shortest path between two vertices. (We are dealing with weighted graphs, as this problem is solved rather trivially for unweighted graphs with breadth-first search).

In this lecture, we dealt with a variant called all-pairs shortest path, which wants to find the shortest path between all pairs of vertices in the graph. In particular, we are dealing with directed graphs with no negative cycles (i.e. some weights can be negative, but there cannot be a cycle whose net weight is negative).

2 All-pairs shortest paths on directed graphs

Assume our graph has indexed vertices \( V = v_1, v_2, ..., v_n \). Let \( l(v_i, v_j) \) be the length of an edge from \( v_i \) to \( v_j \). We want to determine the cost of a shortest path from \( v_i \) to \( v_j \) \( \forall i, j \in \{1, 2, ..., n\} \).

Let \( C_{ij}^k \) be the length of a shortest path from \( v_i \) to \( v_j \) such that the path has no intermediate vertex of index \( > k \). We can set up the dynamic programming equations for this problem as follows:

\[
C_{ij}^k = \begin{cases} 
0 & \text{if } i = j \\
l(v_i, v_j) & \text{if } i \neq j, k = 0 \\
\min\{C_{ij}^{k-1}, C_{ik}^{k-1} + C_{kj}^{k-1}\} & \text{if } k > 0 
\end{cases}
\]

These equations readily lend themselves to translation into a dynamic programming-based algorithm, which is jointly credited to Floyd and Warshall.
2.1 Floyd-Warshall’s Algorithm

ShortestPathCosts(V, E):
1: \( n \leftarrow V.length \)
2: Let \( b \) be a new two-dimensional array of size \( n \times n \).
3: Let \( C \) be a new three-dimensional array of size \( n \times n \times n \).
4: for \( i \leftarrow 1 \) to \( n \) do
5: \hspace{1em} for \( j \leftarrow 1 \) to \( n \) do
6: \hspace{2em} \( b_{ij} \leftarrow 0 \)
7: \hspace{2em} if \( i = j \) then
8: \hspace{3em} \( C_{ij}^0 \leftarrow 0 \)
9: \hspace{2em} else
10: \hspace{3em} \( C_{ij}^0 \leftarrow l(v_i, v_j) \)
11: \hspace{2em} end if
12: \hspace{1em} end for
13: \hspace{1em} end for
14: for \( k \leftarrow 1 \) to \( n \) do
15: \hspace{1em} for \( i \leftarrow 1 \) to \( n \) do
16: \hspace{2em} for \( j \leftarrow 1 \) to \( n \) do
17: \hspace{3em} \( C_{ij}^k \leftarrow C_{ij}^{k-1} \)
18: \hspace{3em} value \leftarrow C_{ik}^{k-1} + C_{kj}^{k-1} \)
19: \hspace{3em} if \( value < C_{ij}^k \) then
20: \hspace{4em} \( C_{ij}^k \leftarrow value \)
21: \hspace{4em} \( b_{ij}^k \leftarrow k \)
22: \hspace{3em} end if
23: \hspace{2em} end for
24: \hspace{1em} end for
25: \hspace{1em} end for
26: return \( C, b \)

Unwind(b, i, j):
1: if \( b_{ij} \neq 0 \) then
2: \hspace{1em} \( k \leftarrow b_{ij} \)
3: \hspace{1em} Unwind(i, k)
4: \hspace{1em} print \( k \)
5: \hspace{1em} Unwind(k, j)
6: end if

At the end of the call to ShortestPathCosts, we get \( C \), which contains the costs of the shortest paths between all pairs of vertices in the graph, and \( b \), which contains back-pointers used to track the shortest paths that involve passing through the \( k \)th vertex. Unwind is used to print out the actual path for a pair of vertices \((v_i, v_j)\).
2.2 Analysis

The structure of this algorithm implies clearly that its runtime is $O(n^3)$. It would appear that the algorithm requires $O(n^3)$ space as well; however, it is possible to rewrite it to not use $k$ directly, by removing line 17 from \texttt{SHORTESTPATHCOSTS} and removing references to $k$. This might make the algorithm converge to an answer more slowly, but reduces space usage to $O(n^2)$. (This is mentioned in exercise 25.2-4 in Cormen 2E).

3 Related reading

\texttt{Wikipedia} as always, has a related article. The textbook also has a section on Floyd-Warshall (§25.2 in Cormen 2E).