CS F364
Design & Analysis of Algorithms

23-02-2015

DYNAMIC PROGRAMMING

Graph Problems:

- All Pairs Shortest Paths

All Pairs Shortest Paths

- Problem:
  - Given a directed graph G = (V,E, W), find the distance between every pair of vertices (u, w) where u and w are in V.
- One solution:
  - For each u in V, run Dijkstra's Single Source Shortest Path algorithm with u as the source.
  - Cost: O(n*(m+n)*log(n))
    - Cost for a dense graph: $O(n^3 * \log(n))$

All Pairs Shortest Paths

- Assume the vertices in V are numbered (arbitrarily) as $(v_1, v_2, \ldots, v_n)$
- (Inductively) Define the cost function **D[k,i,j]**
  - as the distance from vertex i to vertex j using only intermediate vertices in $\{ v_1, v_2 \ldots, v_k \}$
- **Base case (k = 0):**
  - **D[0,i,j]**

    = **0** if i=j;

    = $\mathbf{w(v_i , v_j)}$ if there is an edge (v_i , v_j) in E;

    = **INFINITY** otherwise
- **Inductive step (Define D[k, _, _] in terms of D[k-1, _, _])**
  - Cost from i to j - with intermediate vertices $\{ v_1 , v_2 \ldots, v_k\}$:
    - If **k must be visited**, cost is **D[k-1, i, k] + D[k-1, k, j]**
    - If **k is not visited**, cost is **D[k-1, i, j]**

All Pairs Shortest Paths

- Recurrence for the cost function:
  - **D[k,i,j] = min(D[k-1,i,j],   D[k-1,i,k] + D[k-1,k,j])**

    for k>0
    - i.e. The cost function satisfies the optimal sub-structure property.

All Pairs Shortest Paths – DP Algorithm

- **Input:** Simple, weighted, directed graph G with no negative-weight cycles
- // D is a 3-D array of size n x n x n
- for (i=1; i<=n; i++) {
-   for (j=1; j<=n; j++) { // Initialize
-       if (i==j)   D[0,i,j] = 0;
-       else if ((v_i,v_j) in E)   D[0,i,j] = w((v_i,v_j));
-       else D[0,i,j] = MAXINT;
- }}
- ...
- return D; // Only D[n,i,j] is needed for all i,j

Induction Basis

ALL PAIRS SHORTEST PATHS – DP ALGORITHM

- **Input:** Simple, weighted, directed graph G with no negative-weight cycles

```
// D is a 3-D array of size n x n x n
for (i=1; i<=n; i++) {
  for (j=1; j<=n; j++) {  // Initialize
      if (i==j)   D[0,i,j] = 0;
      else if  ((v_i,v_j) in E)   D[0,i,j] = w((v_i,v_j));
      else  D[0,i,j] = MAXINT;
}}
for (k=1; k<=n; k++) {
  for (i=1; i<=n; i++) {
     for (j=1; j<=n; j++) {
          D[k,i,j] = min(D[k-1,i,j],   D[k-1,i,k] + D[k-1,k,j]);
 }}}
return D;  // Only D[n,i,j] is needed for all i,j
```

Induction Step

All Pairs Shortest Paths – DP Algorithm

- **Input:** Simple, weighted, directed graph G with no negative-weight cycles
- // D is a 3-D array of size n x n x n
- for (i=1; i<=n; i++) {
- for (j=1; j<=n; j++) { // Initialize
- if (i==j)   D[0,i,j] = 0;
- else if ((v_i,v_j) in E)   D[0,i,j] = w((v_i,v_j));
- else D[0,i,j] = MAXINT;
- }}
- for (k=1; k<=n; k++) {
- for (i=1; i<=n; i++) {
- for (j=1; j<=n; j++) {
- D[k,i,j] = min(D[k-1,i,j],   D[k-1,i,k] + D[k-1,k,j]);
- }}}
- return D; // Only D[n,i,j] is needed for all i,j

Time Complexity:
- $O(N^3)$

Space Complexity:
- $O(N^3)$
- Can this be reduced?

ALL PAIRS SHORTEST PATHS – DP ALGORITHM

- **Input:** Simple, weighted, directed graph G with no negative-weight cycles
- // D is a 3-D array of size n x n x n
- for (i=1; i<=n; i++) {
-   for (j=1; j<=n; j++) {
-     if (i==j)   D[0,i,j] = 0;
-     else if ((v_i,v_j) in E)   D[0,i,j] = w((v_i,v_j));
-     else D[0,i,j] = MAXINT;
- }}
- for (k=1; k<=n; k++) {
-   for (i=1; i<=n; i++) {
-     for (j=1; j<=n; j++) {
-       D[k%2,i,j] =
-         min(D[(k-1)%2,i,j],   D[(k-1)%2,i,k] + D[(k-1)%2,k,j]);
- }}}
- return D[n%2];

Time Complexity: $\Theta(N^3)$

Space Complexity: $2*N^2$

- Can you modify D in-place so that you need only N^2 space?

All Pairs Shortest Paths – DP Algorithm

- How do you recover the shortest paths, pairwise?
  - Construct a predecessor matrix, P[i,j] along with the distance matrix:
    - P[k][i,j] is the predecessor of j
      - in the shortest path from i to j
      - using intermediate vertices only from {1, 2, …, k}
  - Write a recurrence for P[k][i,j]