DESIGN AND ANALYSIS OF ALGORITHMS
LAB 09: Dynamic Programming (cont.)

I. Dynamic Programming Idea

1. "Programming" relates to planning/use of tables, rather than computer programming.
2. Solve smaller problems first, record solutions in a table; use solutions of smaller problems to get solutions for bigger problems.
3. Differs from Divide and Conquer in that it stores the solutions, and subproblems are "overlapping"

II. Programming Exercises

A. Requirements for all problems:

1. Implement a Dynamic Programming algorithm to solve problem in Python programming language
2. Implement a Divide-and-Conquer algorithm to solve problem in Python programming language
3. Generate different inputs of different size
4. Draw the running time of programs as a function of input size

B. Problems

1. Determine if there is a non-trivial directed path from node i to node j

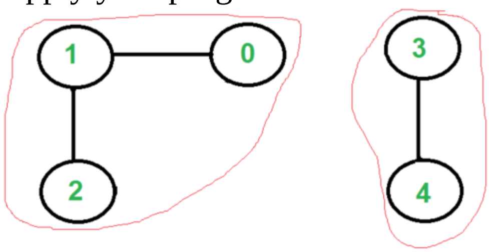
Intuition

- Given adjacency matrix A
- $R^k(i,j)$ denotes if there is a path from i to j which has intermediate vertices only among $\{1,2,\ldots,k\}$.
- Thus, $R^0(i,j) = A(i,j)$.
- $R^{k+1}(i,j)$ from $R^k(i,j)$?
- If $R^k(i,j) = 1$, then $R^{k+1}(i,j) = 1$
- If $R^k(i,k+1) = 1$ and $R^k(k+1,j) = 1$, then $R^{k+1}(i,j) = 1$
- Thus, we compute $R^0, R^1, \ldots$ one by one.
- Can consider this as 3D matrix (involving the parameters i,j,k)

Warshall's Algorithm is described as follows

Warshall's Algorithm
Input $A[1..n, 1..n]$.
$R[0, i, j] = A[i, j]$.
For $k = 1$ to n do
For $i = 1$ to n do
For $j = 1$ to n do
If $R[k-1, i, j] = 1$ or $(R[k-1, i, k] = 1$ and $R[k-1, k, j] = 1)$,
then $R[k, i, j] \leftarrow 1$ Else $R[k, i, j] \leftarrow 0$ Endif
Endfor
Endfor
Endfor

Apply your programs for the following graph



The adjacency matrix of the graph is

	0	1	2	3	4
0	0	1	0	0	0
1	1	0	1	0	0
2	0	1	0	0	0
3	0	0	0	0	1
4	0	0	0	1	0

Hint for Python code:

```
from collections import defaultdict
A = defaultdict(int)
#input
A[(1,2)] = 1
A[(2,1)] = 1
A[(2,3)] = 1
A[(3,2)] = 1
A[(4,5)] = 1
A[(5,4)] = 1

def warshall(A, n, u, v):
    #function to check if vertices u, v are connected
    R = defaultdict(int)

    for key in A.keys():
        R[(0,) + key] = A[key]
```

```
    for k in range(1, n+1):
        for i in range(1,n+1):
            for j in range(1,n+1):
                if (R[(k-1,i,j)]==1) or (R[(k-1,i,k)]==1 and R[(k-1,k,j)]==1):
                    R[(k,i,j)] = 1
    return R[(n, u+1, v + 1)]

print(warshall(A,5,0,2))
print(warshall(A,5,0,4))
```

Use three-dimentional array instead of defaultdict to store temporary results, reimplement the algorithm

Hint:

-how to create three-dimentional array with size (m, n, k) in Python

C =[[[0 for _ in range(k)] for _ in range(n)] for _ in range(m)]

-how to access an element in C

C[k][i][j]

2. Find Single Source Shortest Paths in an undirected weighted graph.

Introduction:

- Suppose A is the adjacency matrix of a weighted graph, that is, A[i, j] gives the weight of the edge (i, j).
- Here the vertices are numbered 1 to n.
- Here we assume that A[i, j] is non-negative. One can handle negative weights also as long as there is no negative circuits (because going through negative circuits repeatedly can reduce the weight of the path by arbitrary amount).
- If edge does not exist then the weight is taken to be ∞.
- We want to find shortest path between all pairs of vertices.

For the Single Source Shortest Paths Problem we can apply a Dynamic Programming algorithm, which is called Floyd's Algorithm. The idea of the algorithm is as follows

- Idea similar to Warshall's algorithm.
- $D_k[i,j]$ denote the length of the shortest path from i to j where the intermediate vertices (except for the end vertices i and j) are all $\le k$.
- $D_0[i,j] = A[i,j]$.
- Here we assume $A[i,i] = 0$ for all i. If initially not so, then update $A[i,i]$ to be so.
- To find, $D_k[i,j]$, for $1 \le k \le n$:
- $D_k[i,j] = min(D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j])$.
- In the algorithm, $next[i,j]$ denotes the vertex which appears just after i in the shortest (known) path from i to j.
- Note: $D_k[i,k] = D_{k-1}[i,k]$ and $D_k[k,j] = D_{k-1}[k,j]$. So, we can do the computation in place! (using D itself to compute $D_0, D_1, D_2, \ldots$).

Pseudocode of Floyd's Algorithm

```
Input A[1..n, 1..n]
 For i = 1 to n do, For j = 1 to n do
      next[i,j] = j; D[i,j] = A[i,j]
 EndFor EndFor
 For i = 1 to n do D[i,i] = 0 EndFor
 For k = 1 to n do
      For i = 1 to n
           For j = 1 to n
                If D[i,j] > D[i,k] + D[k,j], then
                     D[i,j] = D[i,k] + D[k,j]
                     next[i,j] = next[i,k]
                     Endif
           EndFor
      EndFor
 EndFor
```

Apply your programs for the following graph, considering vertex 0 as the source vertex.