Department of Computer Science

CPS688 – Advanced Algorithms

Lab 3

General Instructions:

Due date: Week of March 13, two weeks from today's lab session.

Grading: Each student is required to demo the program during the lab session. Failure to show up will result in a grade of zero. No extensions.

Submission: Zip your code and submit it on D2L after your finish your demo.

Weight: 5% of your total grade.

Hint: Use code from previous labs whenever possible.

Lab Instructions:

Problem 1 – Rod Cutting Problem: Note: this problem was asked during a Microsoft interview

Given a rod of steel, you need to cut it into pieces in a way to optimize the total revenue. The decision is where to cut the rod given that different piece lengths have different revenue.

Formally,

Given a rod of length n inches and a table of prices p[i], i=1,2,…,n, find the maximum revenue r[n] obtainable by cutting up the rod and selling the pieces.

Rod lengths are integers

For i=1, 2,…,n we know the price(revenue) p[i] of a rod of length i inches.

Length i	1	2	3	4	5	6	7	8	9	10
Price p[i]	1	5	8	9	10	17	17	20	24	30

Cut possibilities for a rod of length 4:

1+1+1+1 for a total price of 1+1+1+1=4

2+1+1 for a total price of: 5+1+1+1=8

2+2 for a total price of 5+5=10

3+1 for a total price of 8+1=9

For a rod of length 4: 2+2 is optimal (p[2]+p[2]=10)

One approach to solve this problem is the dynamic programming bottom-up method. This approach typically depends on some natural notion of the "size" of a subproblem, such that solving any particular subproblem depends only on solving "smaller" subproblems. We sort the subproblems by size and solve them in size order, smallest first. When solving a particular subproblem, we have already solved all of the smaller subproblems its solution depends upon, and we have saved their solutions. We solve each subproblem only once, and when we first see it, we have already solved all of its prerequisite subproblems.

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

Here we proactively compute the solutions for smaller rods first, knowing that they will later be used to compute the solutions for larger rods. The answer will once again be stored in r[n].

Input

Your program will be tested against multiple test cases. Each test case begins with an integer n representing the length of the rod followed by n elements representing the prices.

Output

For each test case, determine the maximum value obtainable by cutting up the rod and selling the pieces.

Sample Input	Sample Output
8 1 5 8 9 10 17 17 20	22

Problem 2 – Strongly Connected Components

Given a directed graph, write a program that checks if the given graph is strongly connected or not. A directed graph is said to be strongly connected if every vertex is reachable from every other vertex.

Following is Kosaraju's DFS-based solution that does two DFS traversals of graph:

1. Initialize all vertices as not visited.
2. Do a DFS traversal of graph starting from any arbitrary vertex v. If DFS traversal doesn't visit all vertices, then return false.
3. Reverse all arcs (or find transpose or reverse of graph)
4. Mark all vertices as not-visited in reversed graph.
5. Do a DFS traversal of reversed graph starting from same vertex v (Same as step 2). If DFS traversal doesn't visit all vertices, then return false. Otherwise return true.

The idea is, if every node can be reached from a vertex v, and every node can reach v, then the graph is strongly connected. In step 2, we check if all vertices are reachable from v. In step 4, we check if all vertices can reach v (In reversed graph, if all vertices are reachable from v, then all vertices can reach v in original graph).

Input

Your program will be tested against multiple test cases. Each test case begins with two integers n and e, representing the number of antennas and cables. The next e lines represent the antennas that are connected by a cable.

Output

For each test case, print "yes" if all the antennas can communicate with each other; else print "no".

Sample Input	**Sample Output**
4 4 0 1 1 2 2 3 3 0	yes
6 7 0 1 0 2 2 4 3 1 3 5 4 5 5 0	no