CS2020: Data Structures and Algorithms (Accelerated)

Problems 19–21

Released: Tuesday, March 29th 2011, Due: Wednesday, April 6th 2011, 13:59

Overview. You have three short programming tasks this week. The first two tasks are related to DAGs. The third one is related to the APSP problem. All three have Dynamic Programming (DP) flavor and can be solved with the knowledge from Lecture18-19.

Collaboration Policy. As always, you are encouraged to work with other students on solving these problems. However, you **must** write up your solution **by yourself**. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Problem 19. (Building A Road)

Remember your MST-related task in PS7? Now, your company win the tender to build one long road to connect two of those remote villages. Your company asked you, their best programmer, to estimate the shortest time to complete the whole project.

Note that to build a road, you need to[1] first clear the land for the road → create even surfaces → lay foundation → pour (several?) layers of asphalt → put markings and lightings, etc...

There are some tasks that you have to complete first before moving on to the next task, i.e. we must lay foundation first before pouring asphalt. But there are some tasks that you can do in parallel to save time, i.e. we can put markings and lightings at the same time.

Now, given those tasks, their dependencies, and the time required to complete each task, what is the earliest completion time to build this single road segment?

You are given a file `road.java` that contains a pre-written main method. The main method will read in a dependency graph from input and store it in `Vector < Vector < ii > > AdjList`. There are V vertices, up to 40 vertices. Each vertex represents one task. If there is a directed edge (u, v) between two vertices u and v, it means that task u must be completed first before task v. You can assume that this dependency graph will always be a DAG. The main method will also read in a sequence of V integers that will be stored in `Vector < Integer > T`. The positive value `T[i]` (or `T.get(i)` in Java) represents the amount of hours to complete task i. These values are ≤ 1000 (i.e. less than 42 days for a task).

Your task is to implement the function `public static int shortestTime()` that can access the two data structure above and reports the shortest time (in hours) to build a road. For this task, your code must implement some form of topological sort (refer to `GraphDemo.java` for an example). Submit the updated `road.java` via our game system.

Example 1

```
Suppose there are three tasks/vertices, labeled as 0, 1, 2.
Let T[0] = 10 hours, T[1] = 10 hours, T[2] = 20 hours.
If the dependency graph is like this (0->1, 0->2).
0---->1
 \
  --->2
Then the shortest time to build the road is 30 hours (if your workers don't sleep).
```

Example 2

```
Suppose there are four tasks/vertices, labeled as 0, 1, 2, 3.
Let T[0] = 10 hours, T[1] = 10 hours, T[2] = 20 hours, T[3] = 10 hours.
If the dependency graph is like this (0->1, 0->2, 1->3, 2->3).
0---->1---->3
 \        /
  --->2-->
Then the shortest time to build the road is 40 hours (if your workers don't sleep).
```

[1] If you are curious, you can read how construction workers build roads:
http://www.wakeweekly.com/archives/2004/Aug5-5.html

Problem 20. (Live Your Life to the Fullest)

Every living human being must have gone through one important life event: his/her own birth. Let's give an index of this event as index 0 (the source of it all). Then, this human being will be able to go through V life-events. Some of these life-events have more than one outgoing edges that represent important decision points/choices throughout his/her life, like whether he/she wants to get married or not; and if he/she wants to get married, how many children that he/she wants to have (0, 1, 2, or many); etc. Then, eventually, whatever choices that he/she made during his/her lifetime, every living human being must end up with the last event of his/her life where he/she has no more choice: the death. Let's give an index of this event as index $V - 1$ (the end of it all).

You are given a file `life.java` that contains a pre-written main method. The main method will read in a life-events graph from input and store it in `Vector < Vector < ii > > AdjList`. You can assume that this life-events graph will always be a DAG. You can assume that this life-events graph model your own life. There are at most $V = 30$ vertices in this graph.

Your task is to implement the function `public static int numWays(int u)` that returns the number of different ways to live a particular life if you are currently at event/vertex u. The main method will call `numWays(0)`, i.e. number of different ways to live a particular life since birth (vertex 0). If you need to pre-process something before calling `numWays(0)`, implement the function `public static void init(int V)`. Submit the updated `life.java` via our game system.

Note that this time, your implementation *cannot* use any form of topological sort. That is, you have to use Dynamic Programming (DP) technique shown in class (refer to `LISDPDemo.java` for an example). Your DG leader will check this.

Example 1

```
Suppose there are only five events in a particular simple life:
0->birth, 1->married, 2->single, 3->have two kids, 4->death; with this DAG model:
0---->1---->3---->4
 \              /
  --->2-------->
Then there are only two different ways to live this life: 0->1->3->4 or 0->2->4
```

Example 2

```
Suppose there are 10 events in another life model:
          ---------------->
         /                  \
        /    ---->4---->     \
       /    /           \     \
0---->1---->3---->5---->7---->8---->9
 \        /  \                    / ^
  --->2-->    >-->6-------------->  |
       \                          /
        -------------------------->
Verify that there are 8 different ways to live this life.
```

Problem 21. (Small World Network)

With the rise of technology, our world is getting *smaller*. There is even an idea that any two random people on earth are only six steps away from each other via some chain of social connection[2].

You are given a file `smallworld.java` that contains a pre-written main method. The main method will read in a direct friendship graph between V persons from input and store it in `public static int[][] AdjMatrix`. You can assume that there are only up to $V = 20$ distinct persons in this friendship graph. If there is a direct friendship between person u and v, then `AdjMatrix[u][v] = 1`, otherwise `AdjMatrix[u][v] = 0`. You can assume that the friendship is mutual, i.e. if `AdjMatrix[u][v] = 1` then `AdjMatrix[v][u] = 1` too. You can assume that the friendship graph is connected and everyone is a direct friend of him/herself.

Your task is to implement the function `public static int diameter()` that returns the diameter of the given direct friendship graph. The definition of 'diameter of graph' is given in Lecture19. Submit the updated `smallworld.java` via our game system.

Note that this time, your implementation *cannot* use any form of `queue` data structure. That is, no standard queue and no priority queue. Moreover, your solution must run in $O(V^3)$ even for a complete (dense) graph.

Example 1

```
Suppose there are five person with this direct friendship information

0----1----2----4
     |         |
     3----------

Then the diameter of this friendship graph is 3: 0-1-2-4 or 0-1-3-4
```

Example 2

```
Suppose there are nine person with this direct friendship information

0----1----2----4----5
|    |         |    |
|    3----------    6
|                   |
8-------------------7

Then the diameter of this friendship graph is 4: 0-1-2-4-5, 0-1-3-4-5, or 0-8-7-6-5
```

[2]See: http://en.wikipedia.org/wiki/Six_degrees_of_separation