COMP20007 Design of Algorithms

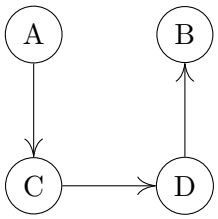
Week 12 Workshop

Tutorial

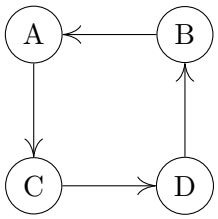
1. Transitive Closure The *transitive closure* of a directed graph G is a new graph H which contains an edge between vertices u and v if and only if there is a path from u to v using one or more edges in the original graph G.

Draw the transitive closure of the following two graphs:

(a)



(b)



2. Warshall's Algorithm Warshall's algorithm is a *dynamic programming* algorithm which computes the transitive closure of a graph defined by an adjacency matrix.

First, we'll assume that the n nodes are labelled $1, 2, \ldots, n$.

We take the problem of *is there a path in G from i to j* (for each pair of nodes $\{i, j\} \subseteq \{1, \ldots, n\}$) and break divide into the sub-problems (for each $k \in \{1, \ldots, n\}$):

> Is there a path from i to j in G using only nodes in $\{1, \ldots, k\}$ as intermediate nodes?

Once we have computed this for $k = n$ we have the answer to the original question for each $\{i, j\}$ and can hence compute the transitive closure.

For each of these sub-problems we will let R^k_{ij} be defined like so:

$$R^k_{ij} := \begin{cases} 1 & \text{if there is a path from } i \text{ to } j \text{ in } G \text{ using only nodes in } \{1, \ldots, k\} \text{ as intermediate nodes} \\ 0 & \text{otherwise} \end{cases}$$

If we have A, the graph G's adjacency matrix we have the following base case and recursive definition of R^k:

$$R^0_{ij} := A_{ij}, \qquad R^k_{ij} := R^{k-1}_{ij} \text{ or } \left(R^{k-1}_{ik} \text{ and } R^{k-1}_{kj}\right)$$

The adjacency matrix of the transitive closure, B, is then defined by $B_{ij} := R^n_{ij}$.

To run Warshall's algorithm, we should compute each of the matrices $R^0, R^1, \ldots, R^n$.

Use Warshall's algorithm to compute the transitive closure of a graph G defined by the following adjacency matrix,

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

3. Floyd's Algorithm So we've seen Warshall's algorithm which computes whether or not there is a path between a pair of vertices $\{i, j\}$ by computing the adjacency matrix of the transitive closure, given the original graph's adjacency matrix.

Floyd's algorithm builds on Warshall's algorithm to solve the *all pairs shortest path* problem. That is, Floyd's algorithm computes the length of the shortest path from each pair of vertices in a graph.

Rather than an adjacency matrix, we will require a weights matrix W, where W_{ij} indicates the weight of the edge from i to j (if there is no edge from i to j then then $W_{ij} = \infty$). We will ultimately find a distance matrix D in which D_{ij} indicates the cost of the shortest path from i to j.

The sub-problems in this case will be answering the following question:

> What's the shortest path from i to j using only nodes in $\{1, \ldots, k\}$ as intermediate nodes?

To perform the algorithm we find D^k for each $k \in \{0, \ldots, n\}$ and set $D := D^n$. The update rule becomes the following:

$$D^0_{ij} := W_{ij}, \qquad D^k_{ij} := \min\left\{D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj}\right\}$$

Perform Floyd's algorithm on the graph given by the following weights matrix:

$$W = \begin{bmatrix} 0 & 3 & \infty & 4 \\ \infty & 0 & 5 & \infty \\ 2 & \infty & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix}.$$

4. Baked Beans Bundles We have bought n cans of baked beans wholesale and are planning to sell bundles of cans at the University of Melbourne's farmers' market.

Our business-savvy friends have done some market research and found out how much students are willing to pay for a bundle of k cans of baked beans, for each $k \in \{1, \ldots, n\}$.

We are tasked with writing a *dynamic programming algorithm* to determine how we should split up our n cans into bundles to maximise the total price we will receive.

(a) Write the pseudocode for such an algorithm.

(b) Using your algorithm determine how to best split up 8 cans of baked beans, if the prices you can sell each bundle for are as follows:

Bundle Size k	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

(c) What's the runtime of your algorithm? What are the space requirements?

5. (Revision) Quicksort & Mergesort

(a) Perform a single *Hoare Partition* on the following array, taking the first element as the pivot.

```
[3, 8, 5, 2, 1, 3, 5, 4, 8]
```

(b) Perform Quicksort on the array from (a). You may use whatever partitioning strategy you like (*i.e.,* you don't need to follow a particular algorithm).

(c) Perform Mergesort on the array from (a).

Computer Lab (Optional)

1. Baked Beans Bundles (Implementation) Write a C program which solves the baked beans problem from the tutorial component of this workshop.

Your program will be given the output as follows:

```
n
p_1
...
p_n
```

Here `p_k` is the price a student is willing to pay for a bundle of k cans of baked beans.

The example from the tutorial would be provided like so:

```
8
1
5
8
9
10
17
17
20
```

Your program should print the maximum price you can sell this collection of cans for, and the sizes of the bundles (in any order). For instance with the above input the output of your program might be:

```
$ ./baked-beans < input.txt
22
2
6
```

This is because the maximum price is 22, which is achieved by selling a bundle of 2, and a bundle of 6.

This problem is often referred to as the *Rod-Cutting Problem*, where you're given an n meter rod and prices for k meters of rod (with $k \in \{1, \ldots, n\}$) and must output the optimal way of splitting the rod to maximise profit.

If you have finished this question you may use the remaining lab time to work on Assignment 2.