COMP20007 Design of Algorithms

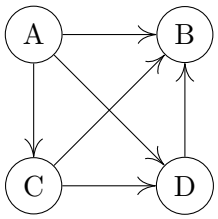
Week 12 Workshop Solutions

Tutorial

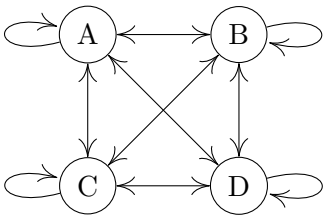
1. Transitive Closure The transitive closures of the graphs have edges between each pair of nodes which have some path connecting them.

We draw a *self-loop* (i.e., an edge from a vertex back to itself) if there is a cycle containing that vertex in the original graph (in other words if there's a path which starts and ends at that vertex).

(a)



(b)



2. Warshall's Algorithm Recall the update rule:

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

We'll start by setting R^0 to A, and then follow the update rule for each k from 1 to n (4 in this case).

In practice we run down the columns from left to right, stopping when we meet a 1. This first happens when we are in row 3, column 1. At that point, 'or' row 1 onto row 3 (and so on):

$$R^0 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^1 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^3 = \begin{bmatrix} \mathbf{1} & 0 & 1 & 1 \\ \mathbf{1} & 0 & 1 & \mathbf{1} \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B := R^4 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

3. Floyd's Algorithm Again, applying the following update rule for $k \in \{0, \ldots, 4\}$.

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

We can think about this as "selecting" the kth row and column at each step, and for all other elements of the matrix D^k we check whether the current distance can be improved by taking the sum of the corresponding elements in the "selected" row and column.

The "selected" rows and columns are shown in red. Updates are shown in bold.

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

$$D^1 = \begin{bmatrix} 0 & 3 & \infty & 4 \\ \infty & 0 & 5 & \infty \\ 2 & \mathbf{5} & 0 & \mathbf{6} \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

$$D^2 = \begin{bmatrix} 0 & 3 & \mathbf{8} & 4 \\ \infty & 0 & 5 & \infty \\ 2 & 5 & 0 & 6 \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

$$D^3 = \begin{bmatrix} 0 & 3 & 8 & 4 \\ \mathbf{7} & 0 & 5 & \mathbf{11} \\ 2 & 5 & 0 & 6 \\ \mathbf{3} & \mathbf{6} & 1 & 0 \end{bmatrix}$$

$$D := D^4 = \begin{bmatrix} 0 & 3 & \mathbf{5} & 4 \\ 7 & 0 & 5 & 11 \\ 2 & 5 & 0 & 6 \\ 3 & 6 & 1 & 0 \end{bmatrix}$$

4. Baked Beans Bundles

(a) Write the pseudocode for such an algorithm.

The subproblems we will try to solve will be $P[i]$, which will indicate the best price we can get if we have i cans.

Note that if we have 0 cans then we don't get anything, so $P[0] = 0$.

We'll let price_k indicate the price of a bundle with k cans.

The update rule then becomes:

$$P[i] = \max_{k \in \{1,\ldots,i\}} \left\{ \text{price}_k + P[i-k] \right\}$$

To perform our algorithm we compute these subproblems for $i = 1 \ldots n$, and the solution will be the answer to $P[n]$.

If we also want to keep track of the sizes of the bundles we select we should also keep track of which k gave the maximum value for each i. We can denote this mathematically using the argmax function:

$$B[i] = \text{argmax}_{k \in \{1,\ldots,i\}} \left\{ \text{price}_k + P[i-k] \right\}$$

The pseudocode for this algorithm is as follows:

function BAKEDBEANS($prices[1 \ldots n]$)
  $P \leftarrow$ new array of 0s with indices 0 through n
  $B \leftarrow$ new array of 0s with indices 0 through n
  for $i = 1 \ldots n$ **do**
    $max\ price \leftarrow 0$

```
    max k ← 0
    for k = 1 . . . i do
        if prices[k] + P[i − k] > max price then
            max price ← prices[k] + P[i − k]
            max k ← k
    P[i] ← max price
    B[i] ← max k
// print the maximum price for all n
output P[n]
// print each k used
j ← n
while B[j] > 0 do
    output B[j]
    j ← j − B[j]
```

(b) Running the algorithm above on the example given yields the following arrays:

$$P = \Big[0, 1, 5, 8, 10, 13, 17, 18, 22\Big]$$
$$B = \Big[0, 1, 2, 3, 2, 2, 6, 1, 2\Big]$$

So the maximum price we can get is 22 and we use a bundle of 2 ($B[8]$) and then 6 ($B[8-2] = B[6]$).

(c) The runtime of this algorithm is $O(n^2)$. We can reason about this like so: we're updating n subproblems, and each update takes up to n iterations, so the runtime complexity is $O(n^2)$.

Each subproblem requires $O(1)$ space and we have $n+1$ subproblems, so the space complexity of this algorithm is $O(n)$.

5. (Revision) Quicksort & Mergesort

(a)

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

[3, 8, 5, 2, 1, 3, 5, 4, 8]
 i                        j

[3, 8, 5, 2, 1, 3, 5, 4, 8]
    i           j

Swap:
[3, 3, 5, 2, 1, 8, 5, 4, 8]
    i           j

[3, 3, 5, 2, 1, 8, 5, 4, 8]
       i     j

Swap:
[3, 3, 1, 2, 5, 8, 5, 4, 8]
       i     j

[3, 3, 1, 2, 5, 8, 5, 4, 8]
          j  i

Crossed Over => Stop
Swap A[l] with A[j]:
[2, 3, 1, 3, 5, 8, 5, 4, 8]

Done!
```

(b)

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

[1]     [3]           [5, 5, 4]      [8]
                       p
                      [5, 4, 5]
                             p

                     [5, 4]
                      p
                        [4]
=> [1, 2, 3, 3, 5, 3, 5, 8, 8]
```

(c)

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