# Homework 5 for EECS 340

Yu Mi,yxm319

April 10, 2018

## 1 Warm-Up: Applying Dynamic Programming Algorithms

Use the dynamic programming algorithms given in class to solve the following problems. Show your work.

### a Coins in a Line

Suppose that in an instance of he coins-in-a-line game the coins have the following values in the line:

$$(9, 1, 7, 3, 2, 8, 9, 3)$$

What is the maximum that the first player, Alice,can win, assuming that the second player, Bob, play optimally?
*Answer* : To solve this problem, let's define $M_{i,j}$ to be the maximum value of coins taken by Alice, for coins numbered $i$ to $j$, and Bob plays optimally. Therefore, the optimal value for Alice is determined by $M_{1,8}$.

Assume the values of each coin are in the array $V[i]$. The principle of Alice to choose is list as follow:

1. If $j = i + 1$, Alice should pick up the larger one in $V[i]$ and $V[j]$;
2. Otherwise, if Alice chooses coin $i$, then she gets a total value of
$$\min\{M_{i+1,j-1}, M_{i+2,j}\} + V[i];$$
3. Otherwise, if Alice chooses coin $j$, then she gets a total value of
$$\min\{M_{i,j-2}, M_{i+1,j-1}\} + V[j].$$

To sum up, for $j > i + 1$, we have

$$M_{i,j} = \max\{\min\{M_{i+1,j-1}, M_{i+2,j}\} + V[i], \min\{M_{i,j-2}, M_{i+1,j-1}\} + V[j]\}$$

As to the initial situation, we have $M_{i,i+1} = \max\{V[i], V[i+1]\}, i \leq (n-1)$, thus, we can fill out the table for $M_{i,j}$ as shown in Table.1.

Table 1: Values for $M$

| $i \backslash j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 9 | 10 | 16 | 13 | 18 | 21 | 27 |
| 2 | 0 | 1 | 7 | 4 | 9 | 12 | 18 | 15 |
| 3 | 0 | 0 | 7 | 7 | 9 | 11 | 18 | 18 |
| 4 | 0 | 0 | 0 | 3 | 3 | 10 | 12 | 14 |
| 5 | 0 | 0 | 0 | 0 | 2 | 8 | 11 | 11 |
| 6 | 0 | 0 | 0 | 0 | 0 | 8 | 9 | 11 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 9 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |

Thus, we can conclude Alice can get the value of 27 to win.

### b 0-1 Knapsack

Let $S = \{a, b, c, d, e, f, g\}$ be a collection of objects with benefit-weight values, $a : (12, 4), b : (10, 6), c : (8, 5), d : (11, 7), e : (14, 3), f : (7, 1), g : (9 : 6)$. What is the optional solution to the 0-1 knapsack problem for $S$ assuming we have a sack that can hold with total weight 18? Show your work.
*Answer*: Let's define $B[k, w]$ as the maximum total value of a subset of $S_k$ from among all those subsets having weight at most $w$. As initial, we have $B[0, w]$ for each $w \leq W$, then we have the following relationship:

$$B[k, w] = \begin{cases} B[k-1, w], & \text{if } w_k > w \\ \max\{B[k-1, w], B[k-1, w-w_k] + b_k\}, & \text{else.} \end{cases}$$

Following the method mentioned above, we have the table for $B[7, w]$ shown as Table.2

Table 2: Values for $B[7, w]$

| $w$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $B[7, w]$ | 7 | 7 | 14 | 21 | 21 | 21 | 26 | 33 | 33 | 33 | 33 | 34 | 41 | 43 | 44 | 44 | 44 | 44 |

As is shown above, we can conclude that the maximum benefit we can have is 44, which contains item $(12, 4), (11, 7), (14, 3), (7, 1)$.

### c Telescope Scheduling Redux

Suppose we are given a set of telescope observation requests, specified by triples of $(s_i, f_i, b_i)$, defining the start times, finish times, and benefits of each observation request as

$$L = \{(1, 2, 5), (1, 3, 4), (2, 4, 7), (3, 5, 2), (1, 6, 3), (4, 7, 5), (6, 8, 7), (7, 9, 4)\}.$$

Solve the telescope scheduling problem for this set of observation requests.
*Answer*: As is discussed in former homeworks, we define pred($i$) as the predecessor of each request $i$, which has the value of following:

$$\text{pred}(i) = \{0, 0, 1, 2, 0, 3, 5, 6\}.$$

Therefore, we can then define $B[i]$ to be the maximum benefit can be achieved with the first $i$ requests. The recursive definition would be:

$$B_i = \max\{B_{i-1}, B_{\text{pred}(i)} + b_i\}.$$

Thus, the $B[i]$ will have the value as shown in Table.3.

Table 3: $B_i$ values

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $B_i$ | 0 | 5 | 4 | 12 | 6 | 3 | 17 | 13 | 21 |

As is shown above, we can have the maximum benefit of 21, the corresponding tasks are:

$$(1, 2, 5), (2, 4, 7), (4, 7, 5), (7, 9, 4).$$

## 2 Warm-Up: Applying Greedy Algorithms

Use the greedy algorithms given in class to solve the following problems. Show your work.

### a Fractional Knapsack

Let $S = \{a, b, c, d, e, f, g\}$ be a collection of objects with benefit-weight values, $a : (12, 4), b : (10, 6), c : (8, 5), d : (11, 7), e : (14, 3), f : (7, 1), g : (9, 6)$. What is an optimal solution to the fractional knapsack problem for $S$ assuming we have a sack that can hold objects with total weight 18? Show your work. *Answer*: Since this question asks for fractional knapsack, we can calculate the benefit-per-weight values as follows: $a : 3, b : 1.667, c : 1.4, d : 1.571, e : 4.667, f : 7, g : 1.5$. Thus we can pick up the objects according to their benefit-per-weight. The weight for objects to take is :$f : 1, e : 3, a : 4, b : 6, d : 4$, the total value would be $7 + 14 + 12 + 10 + 4 \times \frac{11}{7} = 49.286$.

### b Task Scheduling

Suppose we are given a set of tasks specified by pairs of the start times and finish time as

$$T = \{(1, 2), (1, 3), (1, 4), (2, 5), (3, 7), (4, 9), (5, 6), (6, 8), (7, 9)\}.$$

Solve the task scheduling problem for this set of tasks. *Answer*: To solve the task scheduling problem with greedy algorithm, we can append the new tasks right behind the end of a prior task. Thus we have:

1. $\{(1, 2), (2, 5), (5, 6), (6, 8)\}$
2. $\{(1, 3), (3, 7), (7, 9)\}$
3. $\{(1, 4), (4, 9)\}$

All the tasks can be solved by three workers.

## 3 Warm-Up: Coins in a Line and the Failings of Greed

Show that, in the coins-in-a-line game, a greedy-denial strategy of having the first player, Alice, always choose the available coin that minimizes the maximum value of the coin available to Bob will not necessarily result in an optimal solution for her.
*Answer*: When we have the coins in the line are $(1, 9, 8, 7, 1)$, Alice may take the leftmost (1) in the first place, Bob will take (7), Alice then take (8), Bob then take (9), Alice will finally take (1). Thus Bob will have a total value of $7 + 9 = 16$, and Alice will have 10.

## 4 The Cleveland Pothole Problem

*Answer*: Since each of the metal plates can cover a length of $24'$, we can first sort the potholes in a increasing order and then group the potholes that can be covered by a single plate. The algorithm is shown as follow:

```
Algorithm CoverHoles($S$)
Input: $n$ positions of the potholes.
Output: The number of plates needed.
$P \leftarrow$ sorted $S$ in increasing order
$Plates \leftarrow 0, Previous \leftarrow -\infty$
for $i \leftarrow 1$ to $n$ do
    if $Previous + 24 < P[i]$ then                ▷ This hole cannot be covered by the previous plate
        $Plates \leftarrow Plates + 1$
        $Previous \leftarrow P[i]$
    end if
end for
return $Plates$
```

## 5 Zombie Apocalypse: Cooking Edition

*Answer*: In mathematics, the **rearrangement inequality** states that

$$x_n y_1 + \cdots + x_1 y_n \le x_{\sigma(1)} y_1 + \cdots + x_{\sigma(n)} y_n \le x_1 y_1 + \cdots + x_n y_n$$

for every choice of real numbers

$$x_1 \le \cdots \le x_n \quad \text{and} \quad y_1 \le \cdots \le y_n.$$

The proof is as follows:

Observe first that $x_1 > x_2$ and $y_1 > y_2$, implies $(x_1 - x_2)(y_1 - y_2) > 0$ or $x_1 y_1 + x_2 y_2 > x_2 y_1 + x_1 y_2$,, hence the result is true if $n = 2$.

Assume it is true at rank $n - 1$, and let $x_1 > \cdots > x_n$, and $y_1 > \cdots > y_n$. Choose a permutation $\sigma$ for which the arrangement gives rise a maximal result.

If $\sigma(n)$ were different from $n$, say $\sigma(n) = k$, there would exist $j < n$ such that $\sigma(j) = n$.

But $x_n > x_k$ and $y_n > y_k$, hence $x_n y_n + x_k y_k > x_k y_n + x_n y_k$, by what has just been proved. Consequently, it would follow that the permutation $\tau$ coinciding with $\sigma$, except at $j$ and $n$, where $\tau(j) = k$ and $\tau(n) = n$, gives rise a better result. This contradicts the choice of $\sigma$. Hence $\sigma(n) = n$, and from the induction hypothesis, $\sigma(i) = i$ for every $i < n$.

The same proof holds if one replace strict inequalities by non strict ones. ■

According to this, we can simply select the largest $c_i$ to pair with largest $b_i$, then the second largest pair and so forth. The algorithm is described as follow:

```
**Algorithm** Pairing$(C, B)$
**Input**: a list of calories $C$ and a list of benefits $B$, both have a size of $n$.
**Output**: The maximum total benefit.
$C' \leftarrow$ sorted $C$ in increasing order
$B' \leftarrow$ sorted $B$ in increasing order
$sum \leftarrow 0$
**for** $i \leftarrow 1$ to $n$ **do**
    $sum \leftarrow C'[i] \times B'[i]$
**end for**
**return** $sum$
```

## 6 Checker-Sum Redux

*Answer*: Since the checker-sum can be computed based on the former checker-sum, we can then use dynamic programming to calculate these latter values.

Specifically, we have:

$$\begin{aligned}
\text{checker-sum}(A)_{i,j} &= (-1)^{i+j} \sum_{k=1}^{i} \sum_{l=1}^{j} (-1)^{k+l} A_{k,l} \\
&= (-1)^{i+j} \sum_{k=1}^{i-1} \sum_{l=1}^{j-1} (-1)^{k+l} A_{k,l} + (-1)^{i+j} \sum_{l=1}^{j-1} (-1)^{i+l} A_{i,l} + (-1)^{i+j} \sum_{k=1}^{i-1} (-1)^{k+j} A_{k,j} + (-1)^{i+j} A_{i,j} \\
&= \text{checker-sum}(A)_{i-1,j-1} - (\text{checker-sum}(A)_{i-1,j} - \text{checker-sum}(A)_{i-1,j-1}) - \\
&\quad (\text{checker-sum}(A)_{i,j-1} - \text{checker-sum}(A)_{i-1,j-1}) + (-1)^{i+j} A_{i,j} \\
&= 3 \times \text{checker-sum}(A)_{i-1,j-1} - \text{checker-sum}(A)_{i,j-1} - \text{checker-sum}(A)_{i-1,j} + (-1)^{i+j} A_{i,j}
\end{aligned}$$

Thus, we have the algorithm:

```
Algorithm Checker-Sum($A$)
Input: a matrix $A \in \mathbb{R}^{n \times n}$
Output: The checker-sum of A.
$Sum \leftarrow$ an all-zero $n \times n$ matrix          ▷ Includes initial conditions: all sum are zero
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $n$ do
        $Sum[i,j] \leftarrow 3 \times Sum[i-1,j-1] - Sum[i,j-1] - Sum[i-1,j] + (-1)^{i+j}A_{i,j}$
    end for
end for
return $Sum$
```

## 7 On The Grid

*Answer*: Since each of the input $n, m$ will be related to the direct next indice of $n, m$ we can do dynamic programming from the first one to the last one:

```
Algorithm F'($L, n, m$)
Input: the topological ordering $L$, and the dimensions $n, m$.
Output: The same value of F$(n, m)$
$Results \leftarrow$ an empty $n \times m$ matrix
for $k \leftarrow 1$ to $L.length$ do
    $i \leftarrow L[k, 0]$          ▷ get the $k$-th element of L
    $j \leftarrow L[k, 1]$
    if $Test(i, j)$ then
        $Results[i,j].append(Base(i, j))$
    else
        $I \leftarrow$ NextIndices$(i, j)$
        for $(i, j) \in I$ do
            $Results[i,j].append($Combiner$(Results[i,j]))$
        end for
    end if
end for
return $Results[n, m]$
```

## 8 A State of Balance

Suppose we are given a collection $A = \{a_1, a_2, ..., a_n\}$ of $n$ positive integers and add up to $N$. Design an $O(nN)$-time algorithm for determining whether there is a subset $B \subset A$ such that $\sum_{a_i \in B} a_i = \sum_{a_i \in A-B} a_i$.
*Answer*: We can use dynamic programming(DP) to solve this problem. First, we define our DP table $Table[i,j]$ where $0 \leq i \leq n$ and $0 \leq j \leq N$which picks value from between $True$ and $False$ saying that there is a subset of items in $A_i = \{a_1, a_2, ..., a_n\}$ that adds to exactly $j$. Thus our solution will be $Table[n, N/2]$.

As to the initial conditions, $Table[0,0] = 1$, and for any $i > 0$, $Table[0,i] = 0$ and $Table[i,0] = 1$(picking nothing). For any $i \geq 1$ and $j \geq 1$, we have:

$$Table[i,j] = \max\{Table[i-1,j], Table[i-1,j-a_i]\}$$

when $j >= a_i$. Such that the running time would be $O(nN)$. The algorithm is shown as follow:

```
Algorithm CanBalance($A, n, N$)
Input: A list of positive number $A$, and its length $n$, its sum $N$
Output: Whether there is a $B$ that $\sum_{a_i \in B} a_i = \sum_{a_i \in A-B} a_i$.
```

```
if $N$ is odd number then
    return $False$                                    ▷ When $N$ is odd, there can't be possible balance.
end if
$Table \leftarrow$ an $n \times N$ matrix with default value all $False$.
$Table[:, 0] \leftarrow True$                                    ▷ Assign all $Table[i, 0] = 1$
for $j \leftarrow 1$ to $N$ do
    for $i \leftarrow 1$ to $n$ do
        if $j \geq a_i$ then
            $Table[i, j] \leftarrow Table[i-1, j] + Table[i-1, j-a_i]$      ▷ Logical plus, return $True$ or $False$
        else
            $Table[i, j] \leftarrow Table[i-1, j]$
        end if
    end for
end for
return $Table[n, N/2]$
```

## 9 Monotonically Decreasing Subsequences

Describe an $O(n^2)$ algorithm to find the length of the longest monotonically (strictly) decreasing subsequence(s) of $n$ integers. For example, given the sequence $(2, 4, 1, 3)(n = 4)$, the decreasing subsequences are $(4, 1), (2, 1), (4, 3), (2), (4), (1), (3)$, so your algorithm should return 2 in this case.
*Answer*: To do this, we can define our DP table as $B[i]$, which stands for the longest decreasing subsequences from $i$ to $n$. Thus, for each of $i$, we only need to find if there is any $B[j], j \in [i, n]$ which $B[j]$is smaller than $B[i]$, then $B[i] = \max\{B[j] + 1\}$.

As to the initial condition, we only need to set all the $B[i]$ to 1, since each of the elements in the list can be a single subsequence with length 1. The algorithm can be shown as follows:

```
Algorithm MDS($S, n$)
Input: A list of positive number $S$, and its length $n$.
Output: The length of the longest monotonically decreasing subsequence.
$B \leftarrow$ a list with size $n$ and all values initially set to 1
$max \leftarrow 1$                                    ▷ Marks the max length
for $i \leftarrow n$ to 1 do
    for $j \leftarrow i$ to $n$ do
        if $S[i] > S[j]$ and $B[j] + 1 > B[i]$ then
            $B[i] \leftarrow B[j] + 1$
        end if
    end for
    if $max < B[i]$ then
        $max \leftarrow B[i]$
    end if
end for
return $max$
```

## 10 Let's Do the Time Warp Again

Speech recognition systems need to match audio streams that represent the same words spoken at different speeds. Suppose, therefore, that you are given two sequences of numbers,$X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_m)$, representing two different audio streams that need to be matched. A mapping between $X$ and $Y$ is a list, $M$, of distinct pairs, $(i, j)$, that is ordered lexicographically, such that, for each $i \in [1, n]$, there is at least one pair, $(i, j)$, in $M$, and for each $j \in [1, m]$, there is a least one pair, $(i, j)$, in $M$. Such a mapping is monotonic if, for any $(i, j)$ and $(k, l)$ in $M$, with $(i, j)$ coming before $(k, l)$ in $M$, we have $i \le k$ and $j \le l$. For example, given

$$X = (3, 9, 9, 5) \text{ and } Y = (3, 3, 9, 5, 5),$$

one possible monotonic mapping between $X$ and $Y$ would be

$$M = [(1, 1), (1, 2), (2, 3), (3, 3), (4, 4), (4, 5)].$$

The dynamic time warping problem is to find a monotonic mapping, $M$, between $X$ and $Y$, that minimizes the distance $D(X, Y)$, between $X$ and $Y$, subject to $M$, which is defined as

$$D(X, Y) = \sum_{(i,j)\in M} |x_i - y_i|,$$

where this minimization is taken over all possible monotonic mappings between $X$ and $Y$. For instance, in the example $X$,$Y$, and $M$, given above, we have $D(X, Y) = 0$. Describe an efficient algorithm for solving the dynamic time warping problem. What is the running time of your algorithm? *Answer*: Since each of the $x_i$ and $y_i$ can be computed from 1 to $n$ in an incremental way and only have 3 possibilities of transform: insertion, deletion and match, we can simply select the smallest way of cost to determine the optimal $D(X, Y)$. As to the initial condition, we can simply set all the boundaries to $\infty$. The algorithm is shown as follow:

```
Algorithm DTWDistance($X, Y, n, m$)
Input: Array $XY$ with size $nm$
Output: The smallest possible $D(X, Y)$
$DTW \leftarrow$ an $n \times m$ matrix with initial value all zero
$DTW[:, 0] \leftarrow \infty$                    ▷ Set all boundary cost to $\infty$
$DTW[0, :] \leftarrow \infty$                    ▷ Set all boundary cost to $\infty$
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $m$ do
        $cost \leftarrow |X[i] - Y[j]|$
        $DTW \leftarrow cost + \min(DTW[i-1, j], DTW[i, j-1], DTW[i-1, j-1])$
    end for
end for
return $DTW[n, m]$
```

## 11 EECS454 only

### a R-17.9

Given $B = (x_1 + \overline{x_2} + x_3) \cdot (x_4 + x_5 + \overline{x_6}) \cdot (x_1 + \overline{x_4} + \overline{x_5}) \cdot (x_3 + x_4 + x_6)$, draw the instance of VERTEX-COVER that is constructed by the reduction from 3SAT of the Boolean formula $B$.
*Answer*: The figure is shown as Fig.1:

### b R-17.10

Draw an example of a graph with 10 vertices and 15 edges that has a vertex cover of size 2.
*Answer*: The figure is shown as Fig.2:

One vertex cover is the set which contains only the two white nodes in the center.

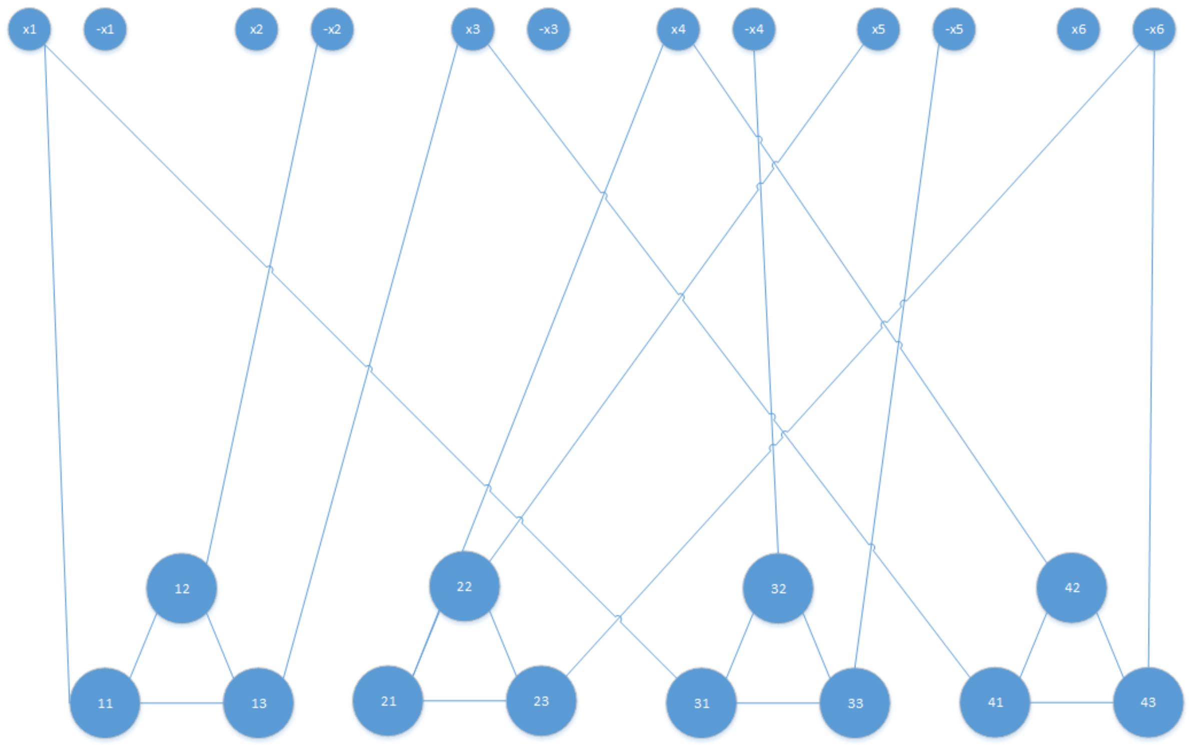

Figure 1: Instance of VERTEX-COVER($x_4$ is connected to 21 and 42)

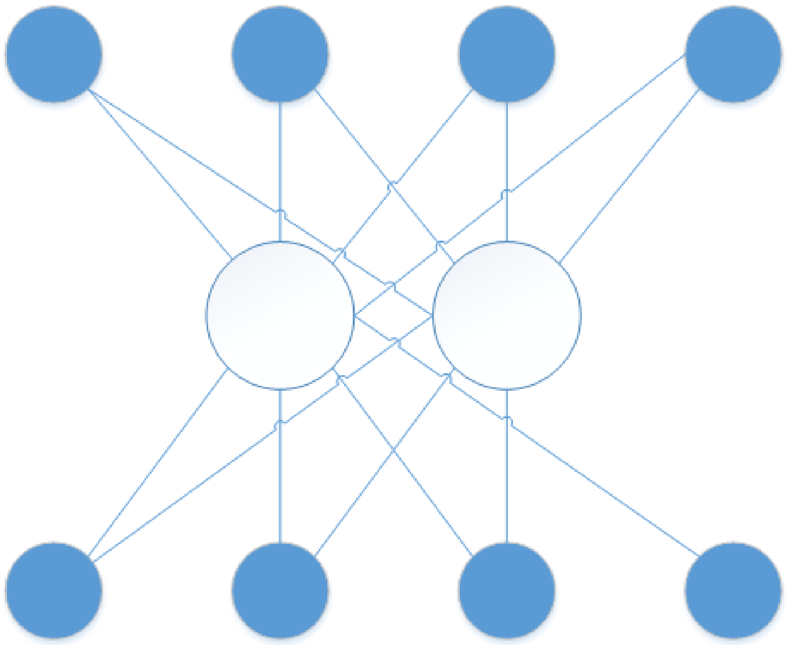

Figure 2: Possible graph

## c R-17.12

Professor Amongus has just designed an algorithm that can take any graph G with n vertices and determine in $O(n^k)$ time whether $G$ contains a clique of size $k$. Does Professor Amongus deserve the Turing Award for having just shown that **P** = **NP**? Why or why not?
*Answer*: Determining whether a graph has a clique of size $k$ in polynomial time can only break its **NP**-

completeness, hence it does not prove $\mathbf{P} = \mathbf{NP}$.

## d C-17.10

Define INDEPENDENT-SET as the problem that takes a graph $G$ and an integer $k$ and asks whether $G$ contains an independent set of vertices of size $k$. That is, $G$ contains a set $I$ of vertices of size $k$ such that, for any $v$ and $w$ in $I$, there is no edge $(v, w)$ in $G$. Show that INDEPENDENT-SET is **NP**-complete.
*Answer*: Assume $G$ is a instance of CLIQUE, which means there exists a CLIQUE $C$ of size $k$ in $G$. Thus, for any $u, v \in C, (u, v) \in E$. Thus, $(u, v) \notin E^c$. Thus, the vertices in $C$ form an INDEPENDENT-SET in $G^c$, and $G^c$ is a instance of INDEPENDENT-SET. Thus, we can reduce INDEPENDENT-SET problem to a CLIQUE problem, which is **NP**-complete, which means INDEPENDENT-SET is also **NP**-complete.