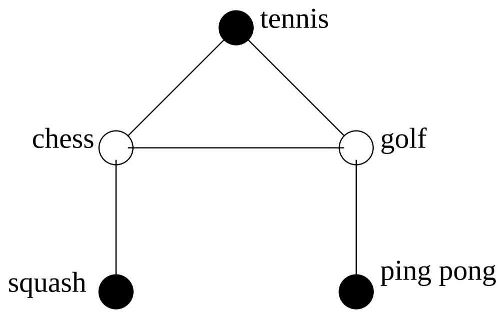
CSI 4105 Fall 2002 **Assignment #2: Solutions**

1. Understanding the application

	tennis	chess	3-person golf	squash	ping pong
John	√	√			
Mary	√		√		
Peter		√	√		
Tracey			√		√
Julian					√
Helen		√		√	
Paul				√	



2. Designing a greedy heuristic algorithm

Part A:

<u>Greedy Heuristic</u>

Algorithm GreedyIndSet($G = (V, E)$):

1. Sort V in non-decreasing order of degree (number of neighbours)
2. $V' \leftarrow V$
3. $I \leftarrow \emptyset$
4. while $V \neq \emptyset$ do
    5. $i \leftarrow$ vertex of smallest index in V'
    6. $I \leftarrow I \cup \{i\}$
    7. $V' \leftarrow V' \setminus \{i\}$
    8. Remove from V' all neighbours of i
9. endwhile
10. return I

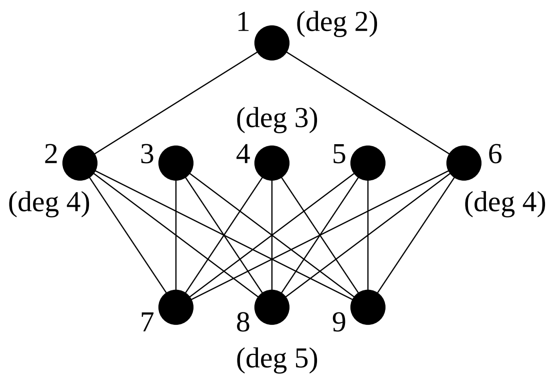
Intuition Behind the Heuristic:

By selecting vertices of smaller degree first, we hope to allow for more vertices to be placed in the independent set.

Polynomial Running Time:

Let $n = |V|$ and $m = |E|$.
Using adjacency lists for representing the graph, and linked lists to represent V' and I, the time complexity for each step is as follows:
Step 1: $O(n \log n)$ for sorting V
Step 2: $O(n)$
Step 3: $O(1)$
The loop is executed at most n times.
Total over all times step 5 is executed: $O(n)$
Total over all times step 6 is executed: $O(n)$
Total over all times steps 7 and 8 are executed: At most n elements are removed from V'. Each removal is done with a search on a list of size at most n, so the total time is $O(n^2)$.
The total running time is $O(n^2)$.

Part B:



The algorithm will always return $I = \{1, 3, 4, 5\}$.
The maximum independent set is $M = \{2, 3, 4, 5, 6\}$.

3. Polytime decision implies polytime optimization for this problem

```
Algorithm ComputeMaxIndSet(G = (V, E)):
    max ← n
    while (DecisionIndSet(G, max) = 0) do
        max ← max − 1
    V′ ← V
    E′ ← E
    for each v ∈ V do
        V″ ← V′ \ {v}
        E″ ← E′ \ {e|v is incident to e}
        Let G″ = (V″, E″)
        if (DecisionIndSet(G″, max) = 1) then
            V′ ← V″
            E′ ← E″
        endif
    endfor
    return V′
```

<u>Polynomial Running Time</u>

Let $n = |V|$.

- The first `while` loop will run for at most n iterations, thus taking $O(n)$ steps.
- The `for` loop will run for n steps. Each step consists of:
    1. the deletion of a vertex and its incident edges, which can be done in $O(n^2)$ steps
    2. in some iterations, copying V'' and E'', which can be done in $O(m + n)$ steps

  Thus, the `for` loop runs in $O(n^3)$ steps.

The total running time is therefore $O(n^3)$.

4. Polynomial-time algorithm for a special case

If every vertex in G has degree 2, then G consists of a collection of cycles.

```
Algorithm MaxIndSet(G = (V, E))
    Input: G, a graph that is a collection of cycles
    I ← ∅
    Mark all vertices ''unvisited''
```

```
(explore each cycle in turn)
while there are unvisited nodes do
    Let v be an unvisited node
    Mark v visited
    I ← I ∪ {v}
    Let x, y, be the two neighbours of v
    Mark x, y visited
    prev ← v
    do
        Let next be the neighbour of x that is not prev
        Mark next visited
        if next ≠ y then
            I ← I ∪ {next}
            prev ← x
            x ← next
        endif
    until (next = y)
endwhile
return I
```

The above algorithm will explore cycle by cycle, placing alternating vertices of a cycle in the set I.
Note that if the cycle has even length, say $2k$, then k vertices of the cycle are in the independent set. If the cycle has odd length, say $2k+1$ for some k, then k vertices are placed in the independent set.

Claim: For a cycle of length $2k$, the maximum independent set has size k and for a cycle of length $2k+1$, the maximum independent set has size k.

Proof: Let I be an independent set.
Since, for each edge, at most one of its incident vertices can be in I, if we count the number of edges incident to some vertex of I, we have

$$\sum_{v \in I} 2 \leq |E|$$

which is $2|I| \leq |E|$. Therefore, $|I| \leq \frac{|E|}{2}$. Since $k = \lfloor \frac{|E|}{2} \rfloor$, in either case we conclude $|I| \leq k$.
For each connected component (each cycle), the algorithm selects its maximum independent set to be part of I. Therefore, the algorithm finds the maximum independent set for the graph.

Running Time

- Let's assume we use the adjacency list representation for the graph, and we use an array indexed by vertices to record visited / unvisited information.
- Checking for who is the next unvisited node takes $O(n)$ in total (where $n = |V|$), since we can try the indices in order.
- Each time a vertex is visited, a constant number of steps are executed inside the `while` loop. Each vertex is visited at most once, so the total number of steps executed over all iterations of the `while` and `do-until` loops is $O(n)$.

The algorithm runs in $O(n)$ steps.

5. NP-Completeness Proof

1. HALFCLIQUE $\in$ NP

Certificate: A set of vertices S

```
Verification Algorithm:  A(G, S)
    Input: G = (V, E),  S
    if |S| < n/2 then return 0
    for each x, y ∈ S,  x ≠ y,  do
        if {x, y} ∉ E then return 0
    endfor
    return 1
```

- This algorithm runs in $O(n^2)$, since $|S| \leq n$.
- If $G \in$ HALFCLIQUE, then there exists S' which is a clique with at least $\frac{n}{2}$ vertices, so $A(G, S') = 1$.
- If $G \notin$ HALFCLIQUE, then for all $S \subseteq V$, S is not a clique with at least $\frac{n}{2}$ vertices, so $A(G, S) = 0$.

2. Select reduction problem

We will prove HALFCLIQUE is NP-Hard by showing CLIQUE $\leq_p$ HALFCLIQUE.

3. Reduction algorithm

```
Algorithm F
    Input: G = (V, E), k
    Output: G' = (V', E')
    n ← |V|
    if k > n then return an arbitrary G' ∉ HalfClique (for example, G' =
    (V', E') with V' = {1, 2, 3} and E' = ∅)
    if k ≥ ⌈n/2⌉ then
        L ← 2k − n
        add L new vertices x_1, x_2, ..., x_L to V:
            V' ← V ∪ {x_1, x_2, ..., x_L}
        E' ← E
    else
        L = n − 2k
        add L new vertices x_1, x_2, ..., x_L to V:
            V' ← V ∪ {x_1, x_2, ..., x_L}
        E' ← E ∪ {{x_i, x} | 1 ≤ i ≤ L, x ∈ V, x ≠ x_i}
    endif
    return G' = (V', E')
```

4. Show correctness

We show that $(G, k) \in \textsc{Clique} \Leftrightarrow G' \in \textsc{HalfClique}$.

Case 1: $k \geq \lceil \frac{n}{2} \rceil$
In this case, $n' = |V'| = n + 2k - n = 2k$.
$(G, k) \in \textsc{Clique}$
$\Leftrightarrow G$ has a clique of at least size k
$\Leftrightarrow G'$ has a clique of size at least k, where $n' = |V'| = 2k$
$\Leftrightarrow G' \in \textsc{HalfClique}$

Case2: $k < \lceil \frac{n}{2} \rceil$
In this case, $n' = |V'| = 2n - 2k$. Note that the L vertices added to V will increase the size of any clique by L vertices (since these L vertices are connected to every vertex in V').
$(G, k) \in \textsc{Clique}$
$\Leftrightarrow G$ has a clique of size at least k
$\Leftrightarrow G'$ has a clique of size at least $k + L = n - k$, where $|V'| = n' = 2n - 2k = 2(n - k)$
$\Leftrightarrow G' \in \textsc{HalfClique}$

5. The reduction runs in polynomial time

- If the algorithm goes beyond the first step, it is because $k \leq n$.
- If $k \geq \lceil \frac{n}{2} \rceil$, then the algorithm only adds $2k - n$ isolated vertices to G. This can be done in $O(n^2)$ if using adjacency matrices to represent the graphs or $O(n)$ if using adjacency lists.
- If $k < \lceil \frac{n}{2} \rceil$, then the algorithm only adds $n - 2k$ vertices and $O(n^2)$ edges. This can be done in $O(n^2)$ in either graph representation (both for adjacency matrices and adjacency lists).

Therefore, the overall running time of the algorithm is $O(n^2)$.