Bill Davis
Final 605.421

Problem 1. Tangoans.
Assuming that 2 tangoans will match only if they are of a different gender, for n male and n female tangoans, we can pair them up as follows. We need to have each male Tangoan pair once with each female Tangoan, we could do this by having the entire group line up, and at time step t removing successful matches and shifting the females over by 1 moving one female from the end of the line to the front. So this requires, in the worst case a total of n^2 comparisons. However in the average case, one pair of matching Tangoans will be found at each time step, thereby reducing the average number to $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ total pairings. In the best case this only requires n total pairings.

Bill Davis

Problem 2. Composition Problem.

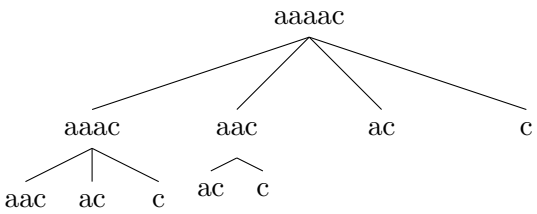
We can solve the Composition Problem Recursively.

```
global stack // for list of substrings in order
global shortcut // for pruning the tree

SOLVE-COMPOSITION(W[0..n], k)
if k = Empty String
  return TRUE // We have found a sequence of nodes which concatenates to k
foreach string s ∈ W
  if match(s, k) AND shortCircuit[k.length - s.length] == TRUE
    push s
    if SOLVE-COMPOSITION(W[0..n], substring(k, s.length to end of k))
      return TRUE
    else pop s

// There are no paths with k.length which equal k
shortCircuit[k.length] = FALSE
return FALSE
```

This is a modification of a naive algorithm. The recursion works as if there was a tree with a root labeled k and each node having at most n children each labeled (parent - W[i]) $\forall i \in n$. This tree may potentially have as many levels as there are letters in k. If we try and search the whole tree effectively looking for a path which when concatenated equals k, this requires examining n^k possible nodes.

The shortcut, then, is neccessary to reduce the amount of work. Once we have determined that there is no paths which, when concatenated, equals the last x digits of k, we never need to examine a branch of the tree which starts with the last x digits. This way we have effectively pruned the tree.

For Example W=[a,aa,aaa,aaaa] k = aaaac



The first leaf we reach, with path [a,a,a,a], will terminate on a string of length 1, namely 'c'. This tells us that we no longer need examine subtrees from nodes labeled with strings of length 1. We will then move up a level and try to match a subtree with path [a,a,a] and substring 'ak'. Since there is no match for this, we can prune subtrees labeled with strings of length 2.

The work required for this is approximately $\mathbf{O}(n \times k)$, since we need to prune off at most k subtrees, and at each level we may have to make n comparisons.

Java source code which implements this algorithm is included.

Bill Davis

Problem 3. Moving on a checkerboard

For this problem we want to find the maxiumum amount of money we can collect while making a series of moves over a checkerboard. In order to do this we need to calculate the maximum amount we can collect at each square along the way.

Therefore assume we are moving up the checkerboard from $y = 0$ to $y = n-1$. At square (x, y), $f(x, y)$ equals the most money we could have accrued when we arrive at (x, y).

$$f(x,y) = \begin{cases} p(x,y) & \text{if } y = 0 \\ p(x,y) + max(f(x-1,y-1), f(x,y-1), f(x+1,y-1)) & \text{if } y > 0 \end{cases}$$

Where $(x-1, y-1), (x, y-1), (x+1, y-1)$ are the only three squares that are legal moves away from (x,y). Either (x-1,y-1) or (x+1,y-1) may not exist if (x,y) is on an edge, in this case we can set its value to $-\infty$.

Then the solution to the problem is MAX($\{f(x, y) : y = n - 1\}$) An algorithm can be constructed as follows for a $n \times n$ checkerboard with dollar values $P(x, y)$

MAX-CHECKERBOARD($P(x, y), n$)
$\pi = NIL$ // Predecessor Array
for $x \in 0$ to n-1
    for $y \in 0$ to n-1
        Calculate f(x,y) and $\pi(y)$ to be max(f(x-1,y-1), f(x,y-1), f(x+1, y-1))

Once we have filled out the dollar values, and the predecessor array, all we need to do is find the max value at $y = n - 1$ and then backtrack using the predecessor array to determine the path.

The Cost of this algorithm is the cost to fill in $f(x, y)$ and the predecessor array, which can be done in $\mathbf{O}(n^2)$.

The correctness of this algorithm can be determined inductively. At the first level the optimal solution is the square which provides the most money, since there is only 1 move. Assuming that we have found the optimal solution at level n, we can determine the optimal solution at n+1, determining the optimal solution at each square using the formula above, and then find-

ing the maximum on the row. This works because we can never legally move to the side or backwards.

Bill Davis

Problem 4. Prove that K-Clique is NPC $\leftrightarrow$ Independant Set is NPC.

We can carry out a reduction as follows. For input graph G define G^c to be the complement of G, i.e. $G^c = \{v : v \in G, (u,v) : (u,v) \notin G\}$. If G^c contains a K-Clique then G contains an independant set of size k. To see this assume that A $\subset V$ is a K-Clique in G^c. For any $(u,v) \in G$ both u and v cannot be in A, otherwise $(u,v) \notin G^c$, and u, v could not form part of a Clique.

Alternatively, if G^c contains an Independant Set of size k then G contains a K-Clique. Assume A $\subset$ V is an Independant set in G^c. This means that for each $u, v \in$A, $(u,v) \in G$, which is the definition of a Clique.

We also can compute these reductions in polynomial time. For example if the graph G is stored as an adjacency matrix, we can compute G^c in $\mathbf{O}(n^2)$ by

```
CONSTRUCT-COMPLEMENT(G)
for i in 1 to n
    for j in 1 to n
        if G(i,j) = 0 then set G^c(i,j) ◁ 1
          else G^c(i,j) ◁ 0
```

So we have shown a reduction that happens in polynomial time between K-Clique and Independant Set and vice versa, therefore K-Clique is NPC $\leftrightarrow$ Independant Set is NPC.