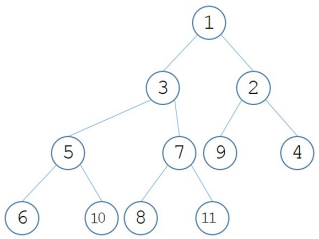
# Assignment 2

Luigi Sacco (h2y9a) 21139143 | André Tertzakian (y610b) 17682148

CPSC 221 | November 7, 2016

**1.**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 11 | 9 | 6 | 7 | 2 | 4 | 5 | 10 | 1 | 3 |
| 8 | 11 | 9 | 6 | 1 | 2 | 4 | 5 | 10 | 7 | 3 |
| 8 | 11 | 9 | 5 | 1 | 2 | 4 | 6 | 10 | 7 | 3 |
| 8 | 11 | 2 | 5 | 1 | 9 | 4 | 6 | 10 | 7 | 3 |
| 8 | 1 | 2 | 5 | 3 | 9 | 4 | 6 | 10 | 7 | 11 |
| 1 | 3 | 2 | 5 | 7 | 9 | 4 | 6 | 10 | 8 | 11 |




**2.**

```
int UCP(string s, int n, int num_opened)
{
    if (n >= s.size()) return 0;
    if (s[n] == '(') return UCP(s, n+1, num_opened+1);
    int c = s[n] == ')' ? 1 : 0;
    if (num_opened > 0) return UCP(s, n+1, num_opened - c);
    else return c + UCP(s, n+1, num_opened);
}

int unmatchedClosingParenthesis(string s)
{
    return UCP(s, 0, 0);
}
```

**3.**

(a)
```
// Sort an array A of 0s and 1s so the 0s come before the 1s
// using swapping. The size of the array is n.
int bitsort(int *A, int n) {
    int i=0, j=n-1;
    while( i <= j ) {
        if( A[i] == 0 )
            i++;
        else if ( A[j] == 1 )
            j--;
        else
            swap( A[i], A[j] );
    }
    return j;
}
```

(b) Loop invariant: $A[0, i-1] = 0$ and $A[j+1, n-1] = 1$
"Every element to the left of $i$ is a 0 and every element to the right of $j$ is a 1."
*proof.* Just before entering the loop, the variables $i$ and $j$ are at the ends of the array, and so there are no elements to the left of $i$ and no elements to the right of $j$, making the invariant trivially correct. For any particular iteration of the loop, and assuming the invariant has held for all previous iterations, it remains to show that the next iteration will preserve the invariant.
Case 1: $A[i] = 0$. The algorithm increases $i$ by exactly one. The invariant is preserved.
Case 2: $A[j] = 1$. The algorithm decreases $j$ by one, which preserves the invariant.
Case 3: *else*. If none of the above cases apply, it means that $A[i] = 1$ and $A[j] = 0$. This violates the loop invariant, and so the algorithm swaps them. By swapping them, note that the elements from 0 to $i$ are all zero (by the previous loop and by this swap) and that the elements from $j$ to $n-1$ are all ones (by the previous loop and by this swap), preserving the invariant. □

(c) The loop continues if $i < y$. Since the variables can only be changed by one, this means that the loop will exit when $i = j$ By the loop invariant, all the elements to the left of $i$ are 0 and to the right of $j$ are 1, as required.

(d) The algorithm will always return the number of 0's in a bit array (where the elements are either 1 or 0) except when the passed array is empty or when it is equal to $\{0\}$ (i.e. contains only one element and that element's value is 0).

**4.**

(a) **The idea of this inductive proof is to show that after each of Alice's turns she can leave the necklace with** $3k$ $(\equiv 0 \mod 3)$ **links if the number of links at the start of her turn is of the form** $3k+1$ **or** $3k+2$ **where** $k \in \mathbb{N}_0$**. If this happens after every turn, eventually the number of links will be** 0 **(which is obviously** $\equiv 0 \mod 3$**). This also happens to be precisely the invariant Alice wants to preserve, which cannot be done when the number of links at the start of her turn is** $= 3k$**.**

$$\text{The inductive proof bases on the fact that } P(n) \implies P(n+3)$$
$$\text{noting that } n+3 \equiv n \mod 3.$$

Base case $n = 1 \implies$ Alice wins because she can pick up one link and win immediately.
Base case $n = 2 \implies$ Similarly, Alice can win because she can pick up the two links.
Base case $n = 3 \implies$ Bob can win. If Alice picks up 1 link, Bob can pick up 2 and wins.
If Alice picks up 2 links, Bob picks up the remaining link and wins.

So for the inductive step, the assumption is:

$$P(n) \equiv \forall n \in \mathbb{N},\ n \not\equiv 0 \mod 3 \implies \text{Alice can win}$$

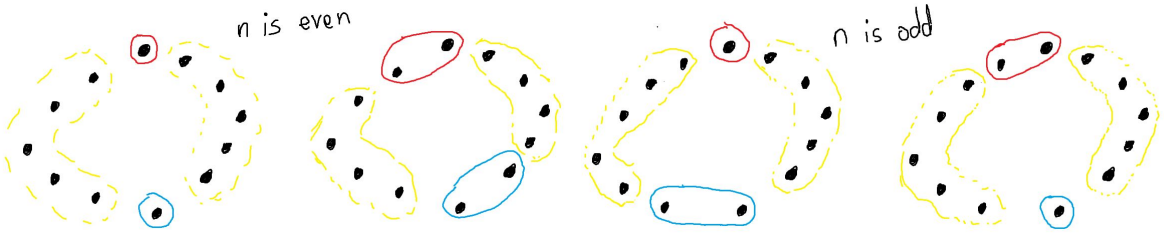
For any of the three cases of what $n \bmod 3$ could be, the inductive step resolves itself due to Alice preserving the invariant and the fact that $n+3 \equiv n \bmod 3$. By this stipulation, for any number of initial links, you can add 3 more links and the outcome will be the same: $P(n) \equiv P(n+3)$. (Provided Alice and Bob play perfectly.) □

(b) If $n = 1$ Bob loses because Alice can just take the one link.
If $n = 2$ Bob loses because Alice can take both links.
If $n = 3$ Bob can win. If Alice picks 1, Bob picks 2; if Alice picks 2, Bob picks 1.
If $n > 3$ Bob can win. To show that he can win, we will give an invariant of what Bob can preserve right after his turn and prove that after every round he is indeed able to execute a move that preserves the invariant and and why he wins if he preserves it.

**Invariant: After Bob's turn, the necklace will be broken up into some *segments* such that any segment's size is equal to that of at least one other segment.**

· Segments are defined to be disjoint subsets of the set of necklace links such that its links were originally adjacent. As a corollary, the size of the union of all segments at any given point in the game is exactly the number of links left.
· Note that we don't care how we define segments before Alice's first turn. After her first turn, she *must* 'break' the necklace so that the set of links is comprised of one segment.

To give an argument of why the evenness of $n$ does not matter, consider the following example where not matter what Alice (in red) does in her first turn, Bob can execute a move such that the invariant is preserved. (Note the size of the remaining yellow segments.)




On the next turn, either Alice divides a segment in two or she doesn't.

· Case 1: If she doesn't break up any segment, then Bob will simply "mirror" her move by choosing links on another untouched segment equal to the number of links Alice chose. Note that the invariant is preserved even if there are more than two segments because at any point at the end of a round there must be an even number of segments (This is true because there are two segments two start with, and if Alice doesn't break one, then Bob won't break one, but if Alice does, then Bob will too, as described in the next case).

· Case 2: If Alice breaks a segment $i$ of size $l$ and makes two subsegments of sizes $0 < l_1, l_2 < l$ then Bob can break another segment $j \neq i$ of size $l$ the same way. More concretely, if Alice breaks up a segment of size $l$ that makes two smaller segments, the invariant states there will be at least another segment with size $l$, and so Bob simply breaks that one up the same way Alice did. And if he does so, the invariant is preserved.
If Alice completely deletes a segment (if the segment's size was 1 or 2 and she picked up all its links), then Bob will again remove the same number of links to another segment of the same size and hence also delete it.

It is clear that from the continuous execution of Bob's strategy to keep the invariant that he will win because in all steps he has "the last word." If Alice breaks a segment, they keep going. Eventually the segments are going to reach size 1 or 2, which can be deleted. And Alice cannot delete a segment without there being another deleteable one for Bob. □

**5.**

(a)

$$\text{Fixing } s_1 = 2^8 - 1 \implies \begin{aligned} h(s_0 s_1 s_2) &= s_0 + 2^8 s_1 + 2^{16} s_2 && \mod 2^{16} - 1 \\ &= s_0 + 2^8 s_1 + s_2 && \mod 2^{16} - 1 \\ &= s_0 + 2^8(2^8 - 1) + s_2 && \mod 2^{16} - 1 \\ &= s_0 + s_2 + (1 - 2^8) && \mod 2^{16} - 1 \end{aligned}$$

Now let's say that we want to fix our hash function to get exactly 0.
With this construction we get that the possible values of the 8-bit characters must satisfy $s_0 + s_2 = 2^8 - 1$, described in the following table:

| $\mathbf{s_0}$ | $\mathbf{s_1}$ | $\mathbf{s_2}$ |
|---|---|---|
| 0 | | $2^8 - 1$ |
| 1 | | $2^8 - 2$ |
| 2 | | $2^8 - 3$ |
| $\vdots$ | $2^8 - 1$ | $\vdots$ |
| $2^8 - 3$ | | 2 |
| $2^8 - 2$ | | 1 |
| $2^8 - 1$ | | 0 |
| 0 | 0 | 0 |

So there are 256 possibilites when $s_1 = 2^8 - 1$ and one extra case when $s_0 = s_1 = s_2 = 0$, totaling 257 three-character strings that hash to the same spot (in this case 0). In fact, for the hashes ranging from 0 to 255, there are 257 three-character strings that satisfy that condition.

(b)

$$\begin{aligned} h(s_0 s_1 \dots s_n) = \sum_{k=0}^{n} 2^{8k} s_k &\equiv s_0 + 2^8 s_1 + 2^{16} s_2 + (2^8)^3 s_3 + (2^{16})^2 s_4 + (2^8)^5 s_5 + \dots && \mod 2^{16} - 1 \\ &\equiv s_0 + 2^8 s_1 + s_2 + 2^8 s_3 + s_4 + 2^8 s_5 + \dots && \mod 2^{16} - 1 \end{aligned}$$

As expressed by the equation above, using properties of modular arithmetic, a string will hash to the same value as all of its permutations where the even or odd numbered indices are swapped with each other. (i.e. swapping any blue's or any red's with each other.)
As an example, it is clear to see that $h(s_0 s_1 s_2 s_3) = h(s_2 s_1 s_0 s_3) = h(s_0 s_3 s_2 s_1) = h(s_2 s_3 s_0 s_1)$.