CS 473 ✧ Fall 2024
Conflict Midterm 1 Problem 1 Solution

Prove that every integer (positive, negative, or zero) can be written in the form $\sum_i (-2)^i$, where the exponents i are distinct non-negative integers.

Solution: This representation is sometimes called ***negabinary***. The negabinary representation was used in at least one Polish computer in the late 1950s. We prove the claim by induction on $|n|$.

Let n be an arbitrary integer. Assume that any integer m such that $|m| < |n|$ can be written in negabinary. There are three cases to consider.

- The base case $n = 0$ is trivial—zero is the empty sum.
- If $n = -1$, then $n = (-2)^1 + (-2)^0$. We need to consider this case explicitly, because $-\lfloor -1/2 \rfloor = 1$, so the following inductive argument doesn't work! Alternatively, we could argue by induction on $|3n-1|$ instead of $|n|$.
- For any integer n other than 0 or -1, we have $\big|-\lfloor n/2 \rfloor\big| < |n|$, so the inductive hypothesis implies that we can write $-\lfloor n/2 \rfloor$ in negabinary. Shifting all the exponents up by 1 gives us a negabinary representation of $\tilde{n} = (-2) \cdot (-\lfloor n/2 \rfloor)$, which does not use $(-2)^0$. There are two subcases to consider.
  - If n is even, then $n = \tilde{n}$ and we're done.
  - If n is odd, then $n = \tilde{n} + 1$, so we can get a negabinary representation of n by adding 2^0 to the negabinary representation of $\tilde{n}$.

In all cases, we conclude that n can be written in negabinary. ■

Rubric: 10 points = 2 for valid strong inductive hypothesis + 2 for explicit exhaustive case analysis + 2 for base case(s) + 2 for correctly applying the stated induction hypothesis + 2 for other details of the inductive case(s). This is not the only correct proof.

A proof that only considers only non-zero integers is worth at most 9 points; only non-negative integers is worth at most 7 points; only positive integers is worth at most 6 points.

CS 473 ✧ Fall 2024
Conflict Midterm 1 Problem 2 Solution

Describe an algorithm that finds the length of the longest string that is the label of *at least two different* downward paths in a given rooted tree T.

Solution: For any node v in the input tree T, let $v.\ell$ denote the label of v, and let $v.parent$ denote the parent of v (or NULL if v is the root of T). If the given representation of T does not include explicit parent pointers, we can easily compute them in $O(n)$ time.

For any pair of nodes u, v in T, let $LongEnd(u, v)$ denote the length of the longest string that is both the label of a downward path ending at u and the label of a downward path ending at v. We need to compute $\max\{LongEnd(u, v) \mid u \neq v\}$. This function obeys the following recurrence:

$$LongEnd(u,v) = \begin{cases} 0 & \text{if } u = \text{NULL or } v = \text{NULL} \\ 0 & \text{if } u.\ell \neq v.\ell \\ 1 + LongEnd(u.parent, v.parent) & \text{otherwise} \end{cases}$$

Before we can memoize this recurrence, we first perform a postorder traversal of T, recording the sequence of nodes that appear in this traversal in an array $post[1..n]$. Thus, if node $post[i]$ is an ancestor of node $post[j]$, then $i > j$. In particular, $post[1]$ is one of the leaves of T, and $post[n]$ is the root.

Now we can memoize our recurrence into a 2-dimensional array $LongEndPost[1..n, 1..n]$, where each entry $LongEndPost[i, j]$ stores the function value $LongEnd(post[i], post[j])$. We can fill this array using two nested loops, each of which considers one of the array indices in reverse order. (It doesn't matter how the loops are nested.) As will fill this array, we also separately keep track of its largest off-diagonal element. The resulting algorithm runs in $\boldsymbol{O(n^2)}$ ***time***. ■

Rubric: 10 points: standard dynamic programming rubric ***except*** 2 points for the recursive cases (instead of 3), and 2 points for the memoization structure (instead of 1). Just writing "2d array" is not enough to specify an appropriate memoization structure; a complete solution also requires a mapping between array indices and tree nodes. As usual, this is not the only correct solution.

CS 473 ✧ Fall 2024
Conflict Midterm 1 Problem 3 Solution

Suppose we are given two bit-strings $A[1..n]$ and $B[1..n]$.

(a) Describe an algorithm that finds a ***shift*** A' of A that maximizes the number of indices i where $A'[i] = B[i] = 1$.

(b) Describe an algorithm that finds a ***rotation*** A' of A that maximizes the number of indices i where $A'[i] = B[i] = 1$.

Solution (part (a)): We compute the convolution of the reversal of A with B, and return the largest element of the resulting vector.

```
BestShiftScore($A[1..n], B[1..n]$):
    for $i \leftarrow 1$ to $n$
        $Arev[i] = A[n-i+1]$
    $C \leftarrow$ Convolution($Arev, B$)
    $best \leftarrow 0$
    for $k \leftarrow 2$ to $2n$
        $best \leftarrow \max\{best, C[k]\}$
    return $best$
```

The algorithm runs in ***$O(n \log n)$ time***. ■

To prove the algorithm correct, we can express each entry $C[k]$ as follows:

$$\begin{aligned} C[k] &= (Arev * B)[k] \\ &= \sum_{i+j=k} Arev[i] \cdot B[j] && [\text{def. } *] \\ &= \sum_{i+j=k} A[n+1-i] \cdot B[j] && [\text{def. } Arev] \\ &= \sum_{j} A[j+(n+1-k)] \cdot B[j] && [i = k-j] \end{aligned}$$

The last sum is the number of matching 1s when A is shifted $n+1-k$ steps to the left (or $-(n-1+k)$ steps to the right if $n-1+k < 0$).

Rubric: 5 points = 1 for using FFT/convolution + 2 for correctly setting up the convolution + 1 for extracting the correct output from the convolution + 1 for time analysis. This is not the only correct solution. A correct $O(n^2)$-time algorithm is worth at most 2 points.

Solution (part (b)): We essentially call the previous algorithm on the strings A and the concatenation of B with itself.

```
BestRotateScore($A[1..n], B[1..n]$):
    for $i \gets 1$ to $n$
        $Arev[i] = A[n-i+1]$
        $BB[i] \gets B[i]$
        $BB[n+1] \gets B[i]$
    $C \gets$ Convolution($Arev, BB$)
    $best \gets 0$
    for $k \gets 2$ to $3n$
        $best \gets \max\{best, C[k]\}$
    return $best$
```

Let $A \circlearrowright k$ denote the result of rotating A by k steps to the right, or $-k$ steps to the left if $k < 0$. Similarly, let $A \Rsh k$ denote the result of shifting A by k steps to the right. The number of common 1s between $A \circlearrowright k$ and B is exactly the same as the number of common 1s between $A \Rsh k$ and $B \bullet B$. ■

Rubric: 5 points = 1 for using FFT/convolution + 2 for correctly setting up the convolution + 1 for extracting the correct output from the convolution + 1 for time analysis. Alternatively: 4 for doubling either string and invoking part (a) + 1 for time analysis. This is not the only correct solution. A correct $O(n^2)$-time algorithm is worth at most 2 points.

CS 473 ✧ Fall 2024
Conflict Midterm 1 Problem 4 Solution

Describe and analyze an algorithm to find the maximum number of points you can earn from a given Number Blast puzzle. (See the question handout for a detailed description of the rules.) The input to your algorithm is an array $A[1 \,..\, n]$ of positive integers.

Solution: The following algorithm considers the moves in order from left to right. Choosing the same intervals in a different order does not change either the total score for those intervals or the set of available moves.

For any index i, let $\mathit{MaxBlast}(i)$ denote the maximum score we can achieve using only squares in the interval $A[i \,..\, n]$. Considering all possible *leftmost* moves gives us the following recurrence:

$$\mathit{MaxBlast}(i) = \begin{cases} 0 & \text{if } i > n \\ \max \left\{ \begin{array}{c} \max \left\{ A[i] \cdot A[j] + \mathit{MaxBlast}(j+1) \;\middle|\; i+1 \le j \le n \right\} \\ \max \left\{ A[i+1] \cdot A[j] + \mathit{MaxBlast}(j+1) \;\middle|\; i+2 \le j \le n \right\} \end{array} \right\} & \text{otherwise} \end{cases}$$

The left end of the leftmost chosen interval must be either $A[i]$ or $A[i+1]$, because otherwise we would leave two untouched squares in a row. The right end of the leftmost chosen interval must be some other square $A[j]$ further to the right.

We can memoize this function into a one-dimensional array, which we can fill in reverse order in $\boldsymbol{O(n^2)}$ ***time***. ■

Rubric: 10 points: standard dynamic programming rubric. This is not the only correct solution. An algorithm that runs in $O(n^3)$ time is worth at most 8 points; scale partial credit.