Genetic Reconstruction

For each creature, we already know one of its alleles, which corresponds to its eye color. This allele is the alphabetically first one. Additionally, the other allele must be greater than or equal to the eye color (because the eye color is the minimum of the two alleles).

For each creature, we must decide which parent passes down the eye color and which one passes down the other allele. This gives us 2^n possible configurations since we need to make this choice for each creature.

Assume we know which parent gives the eye color and which gives the other allele for each creature. For simplicity, let's say the eye color comes from p1 (parent 1), and the other allele comes from p2 (parent 2).

Define c[i] as the eye color (the first allele) of creature i and g[i] as the second allele. We will iterate over all creatures to assign values for g[i], initializing them to -1 to indicate they are not yet set.

Now, consider the following:

1. If c[i] matches either c[p1[i]] or g[p1[i]] (alleles from parent 1), there are no additional constraints.
2. Otherwise, we need to set g[p1[i]] to c[i]. If g[p1[i]] is already set to a different value, this configuration is invalid, and we skip it.

Next, if g[i] is already set:

1. If g[i] matches either c[p2[i]] or g[p2[i]] (alleles from parent 2), there are no additional constraints.
2. Otherwise, we set g[p2[i]] to g[i]. If g[p2[i]] is already set to a different value, this configuration is invalid, and we skip it.

If g[i] is not yet set, it must satisfy a minimum requirement: it must be at least c[i]. This means we need to ensure that one of the alleles from p2[i] is at least c[i]. We update this minimum constraint for p2[i] as h[p2[i]] = max(h[p2[i]], h[i]).

Since creatures place constraints on their parents, we should process them in reverse order (from creature n down to 1).

In a second pass, we iterate through the creatures in increasing order (from 1 to n), now that parent information is fixed. We assign the smallest possible value for g[i] that is compatible with their parent p2.

Here is a sample c++ code:

```
string ans = "z";

void check(int msk) {
    memset(g, -1, sizeof g);
    memset(h, 0, sizeof h);
    bool bad = false;

    // Flip parents for creatures based on the bitmask.
    for (int i = 0; i < n; i++) {
        if ((1 << i) & msk) {
            swap(p1[i], p2[i]);
        }
    }

    // Reverse order to propagate constraints from creatures to parents.
    for (int i = n - 1; i >= 0; i--) {
        h[i] = max(h[i], c[i]);

        // Skip if no parents.
        if (p1[i] == -1) continue;

        // First allele constraint: c[i] comes from p1.
        if (c[p1[i]] != c[i]) {
            if (g[p1[i]] != -1 && g[p1[i]] != c[i]) {
                bad = true;
                break;
            }
            g[p1[i]] = c[i];
        }

        // Second allele constraint: g[i] comes from p2.
        if (g[i] != -1) {
            if (c[p2[i]] != g[i]) {
                if (g[p2[i]] != -1 && g[p2[i]] != g[i]) {
                    bad = true;
                    break;
                }
                g[p2[i]] = g[i];
            }
        } else if (h[i]) {
            h[p2[i]] = max(h[p2[i]], h[i]);
        }
    }

    // Now assign values for g[i] in increasing order.
    string tmp;
    for (int i = 0; i < n; i++) {
        if (p2[i] == -1) {
            if (g[i] == -1) {
                g[i] = h[i];
            }
        } else {
            if (g[i] == -1) {
```

```
                if (c[p2[i]] >= h[i]) {
                    g[i] = c[p2[i]];
                } else {
                    g[i] = g[p2[i]];
                }
            }
        }

        // Check if the assignment is invalid.
        if (g[i] < h[i]) {
            bad = true;
            break;
        }

        // Build the solution string.
        tmp += char('a' + c[i]);
        tmp += char('a' + g[i]);
        tmp += '\n';
    }

    // Revert parent swaps after checking.
    for (int i = 0; i < n; i++) {
        if ((1 << i) & msk) {
            swap(p1[i], p2[i]);
        }
    }

    // Update answer if valid.
    if (!bad) {
        ans = min(ans, tmp);
    }
}

// Iterate through all possible configurations.
for (int msk = 0; msk < (1 << n); msk++) {
    check(msk);
}
```