Cubeword

In all subtasks we solve the problem independently for each possible length of a word and sum the answers.

To handle palindromes and words that are reverse of each other, we first add the reverse of each word to the dictionary and then remove duplicates. Then, we will fix a direction for each edge and we will ask that the word written along the edge in that direction has to be in the new wordlist. (This is clearly equivalent to the original problem, and it allows us to choose the directions of edges in any way that suits our approach.)

For each pair of letters a, b precompute the number of words that begin with a and end in b.

Subtask 1

We brute-force all combinations of letters in the vertices of the cube and use the precomputed values to find how many cubewords are there with these letters fixed. Complexity $\mathcal{O}(n + |\Sigma|^8)$.

Subtask 2

Label the vertices 1 through 8 with vertices 1 through 4 lying around the bottom face and vertex $i+4$ directly above vertex i. We will assign letters to vertices in the increasing order of their ids. Start by assigning letters to vertices 1 through 4 and count the number of possibilities of assigning letters to the edges on the bottom face. We then try all possibilities of assigning a letter to vertex 5, and account for the edge $\{1, 5\}$. Note that all edges incident to the vertex 1 are already processed, hence we may now drop the value of 1 from our state. More generally, after processing vertex j we can forget the letter in vertex $j - 4$.

This forms a dynamic programming solution with $8 \cdot |\Sigma|^4$ states, where each state has $|\Sigma|$ transitions, each of which can be performed in constant time.

Complexity $\mathcal{O}(n + |\Sigma|^5)$.

Subtask 3

When viewed as a graph, the cube is bipartite with both partitions of size 4. Consider all possibilities of assigning the letters to one partition. We want to process each of the remaining vertices in $\mathcal{O}(1)$.

Precompute $D[a][b][c]$ as the number of triples of words such that they all begin with the same letter and the other end with a,b,c, respectively. This precomputation can be done naively in $\mathcal{O}(|\Sigma|^4)$.

Coming back to our problem, let the letters on one partition be a,b,c,d respectively. Observe that the answer is $D[a][b][c] \cdot D[a][b][d] \cdot D[a][c][d] \cdot D[b][c][d]$.

Total complexity is $\mathcal{O}(n + |\Sigma|^4)$.

Fun fact: Geometrically, this solution can be viewed as follows. A cube can be cut into five tetrahedrons: one that's completely inside and four such that each of them contains one corner of the cube (and thus three edges). In this solution you brute force the letters in the corners of the inner tetrahedron, and once you fix those, each of the remaining tetrahedrons is independent of the other three.

Subtask 4

To reduce the complexity even further, note that a general cube can be rotated and mirrored in 48 different ways: choose bottom face (6 ways), then front face (4 ways) and finally left face (2 ways).

Assume for a moment that the letters a, b, c, d on the vertices of the first partition are guaranteed to be distinct. Then there is a way of rotating and flipping the cube in such a way that $a < b < c < d$. Hence we can calculate the answer only for this case and multiply it by 24.

However, the letters a, b, c, d are not necessarily distinct. To mitigate this, we simply consider all possibilities of the form $a \leq b \leq c \leq d$, and multiply the answer of each of them by the number of permutations of (a, b, c, d). This reduces this step of the solution to $\mathcal{O}(|\Sigma|^4/24)$.

Precomputing the array D can also be optimised – we again consider only cases $a \leq b \leq c$ and use the fact that $D[a][b][c] = D[a][c][b] = \ldots = D[c][b][a]$. The complexity of this step is thus reduced to $\mathcal{O}(|\Sigma|^4/6)$.

To summarise, we are able to reduce the complexity 6-fold by considering symmetries.