October 19, 2025

DP FORM 1

Ques 1. Given a string that consists of A/B/C/D but some of the places are missing and denoted by '?'
Find :

a. Find no. of ways to fill the missing places (?) with A/B/C/D such that no two neighbours are same.

b. What if the string is circular. What if the no. of A's has to be odd.

c. Print lexicographically smallest solution.

S = "? ? A ? B ? D ?"

Restrictions :

→ no 2 neighbours are same.
→ the string is circular
→ the # of A's has to be odd

- This is a Form 1 type problem.

- State

DP(i, prev, first, (#A)%2) → [# of ways to fill '?' in [i---N]

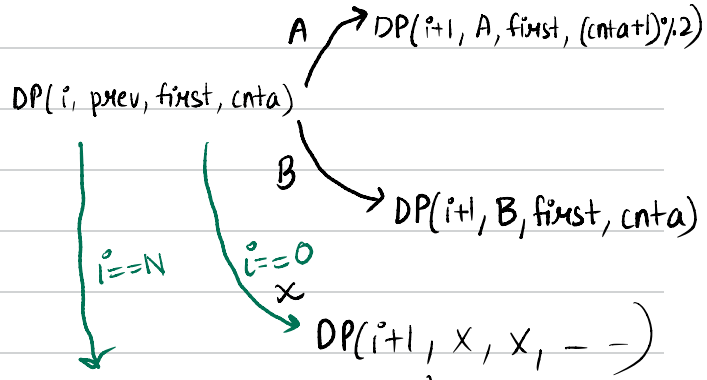
prev : Storing the previous value here as if we ensure that incoming isn't same as prev then we ensure that no 2 neighbours are same completing our first restriction.

first : This is storing the value at 0th index because when we reach (N-1)th index, in order to ensure our string is circular, the val at 0th idx and (N-1)th idx should NOT be same (2nd restriction)

(#A%2) : ensures the count of no. of A's is even (3rd restriction)

Here, we can see how restrictions define state.

- Transition



if (prev != first && cnta==1)
    return 1
else return 0

- Time complexity

DP(i, prev, first, cnta%2)
N, 4 (A/B/C/D), 4 (A/B/C/D), 2 (0,1)

# of Transition = 4

AlgoZenith

# of state = N * 4 * 4 * 2

T.C = O(32N(1+4))

⇒ O(N)

```
int n;
string s;

int dp[1001][4][4][2];

int rec(int i,int prev,int fst, int cnta){
    // pruning
    // basecase
    if(i==n){
        if(prev!=fst && cnta==1){
            return 1;
        }else return 0;
    }
    // cache check
    if(prev!=-1 && dp[i][prev][fst][cnta]!=-1)
        return dp[i][prev][fst][cnta];

    // transition
    int ans = 0;
    // Build choices
    set<int> choices;
    if(s[i]=='?')choices = {0,1,2,3};
    else choices = {s[i]-'A'};

    if(prev!=-1)choices.erase(prev);

    for(auto v:choices){
        int nfst = fst;
        if(i==0) nfst = v;
        if(v==0){// A case
            ans += rec(i+1, v, nfst, (cnta^1));
        }else{
            ans += rec(i+1, v, nfst, cnta);
        }
    }
    // save and return
    if(prev!=-1) dp[i][prev][fst][cnta] = ans;
    return ans;
}
```

```
int n;
string s;

int rec(int i,int prev,int fst, int cnta){
    // pruning
    // basecase
    if(i==n){
        if(prev!=fst && cnta==1){
            return 1;
        }else return 0;
    }
    int ans = 0;
    // Build choices
    set<int> choices;
    if(s[i]=='?')choices = {0,1,2,3};
    else choices = {s[i]-'a'};

    if(prev!=-1)choices.erase(prev);

    for(auto v:choices){
        int nfst = fst;
        if(i==0) nfst = v;
        if(v==0){// A case
            ans += rec(i+1, v, nfst, (cnta^1));
        }else{
            ans += rec(i+1, v, nfst, cnta);
        }
    }
    // save and return
    return ans;
}


void solve(){
    cin>>n;
    cin>>s;
    // memset(dp,-1,sizeof(dp));
    cout<<    rec(0,-1,-1,0);
}
```

```
string ans;
void generate(int i,int prev,int fst, int cnta){
    // basecase
    if(i==n){
        return;
    }
    // Build choices
    set<int> choices;
    if(s[i]=='?')choices = {0,1,2,3};
    else choices = {s[i]-'A'};

    if(prev!=-1)choices.erase(prev);

    for(auto v:choices){
        int nfst = fst;
        if(i==0) nfst = v;
        if(v==0){// A case
            if(rec(i+1, v, nfst, (cnta^1))>0){
                ans += char('A'+v);
                generate(i+1, v, nfst, (cnta^1));
                return;
            }
        }else{
            if(rec(i+1, v, nfst, cnta)>0){
                ans += char('A'+v);
                generate(i+1, v, nfst, cnta);
                return;
            }
        }
    }
}
```

```
void solve(){
    cin>>n;
    cin>>s;
    memset(dp,-1,sizeof(dp));
    cout<< rec(0,-1,-1,0) << endl;
    generate(0,-1,-1,0);
    cout<<ans<<endl;
}
```

Full code using DP:

AlgoZenith