**Problem Name : Flood Fill**

**Topic : DP**

**Difficulty : Hard**

**EXPLANATION**

Intention: Give you an array, you can change all adjacent and identical numbers to another number each time, and finally make all the numbers equal, and ask how many times you want to change at least.

IDEA: I just started to use memory search, but I thought it would become the interval dp, because its label is dp, so the state behind it can be transferred from the previous state, then this question is the smallest of a range. The value can be transferred from the minimum value of another interval. We assume that the current interval has all become a number. Then the next interval is either expanded to the left or expanded to the right because the outer interval is unknown, so we Be sure to change the value inside to the same value as the outermost value of the interval, but if… 2 3 2 3 … this does not know which side has changed, we need to maintain the two ends, count dp[i][j ][0] is the minimum required for this interval to change with the left side as the credential. dp[i][j][1] is the minimum required for the interval from i to j to change from the right endpoint. Then the previous 2 3 2 3 dp[1][3][0] can be directly transferred by dp[2][3][1], or dp[2][3][0]+1 is transferred.

**Time Complexity : O(N\*NN∗N).**

#include<bits/stdc++.h>

using namespace std;

const int N=5e3+5;

int dp[N][N][2],a[N];

int main()

{

int n;

scanf("%d",&n);

for(int i=1;i<=n;i++)

scanf("%d",&a[i]);

memset(dp,125,sizeof(dp));

for(int i=1;i<=n;i++)

dp[i][i][0]=dp[i][i][1]=0;

for(int len=1;len<n;len++)

{

for(int i=1;i+len-1<=n;i++)

{

if(i>1)

{

if(a[i+len-1]==a[i-1])

dp[i-1][i+len-1][0]=min(dp[i-1][i+len-1][0],dp[i][i+len-1][1]);

if(a[i]==a[i-1])

dp[i-1][i+len-1][0]=min(dp[i-1][i+len-1][0],dp[i][i+len-1][0]);

dp[i-1][i+len-1][0]=min(dp[i-1][i+len-1][0],min(dp[i][i+len-1][1]+1,dp[i][i+len-1][0]+1));

}

if(i+len<=n)

{

if(a[i]==a[i+len])

dp[i][i+len][1]=min(dp[i][i+len][1],dp[i][i+len-1][0]);

if(a[i+len-1]==a[i+len])

dp[i][i+len][1]=min(dp[i][i+len][1],dp[i][i+len-1][1]);

dp[i][i+len][1]=min(dp[i][i+len][1],min(dp[i][i+len-1][0]+1,dp[i][i+len-1][1]+1));

}

}

}

printf("%d\n",min(dp[1][n][0],dp[1][n][1]));

return 0;

}

**Approach 2**

#include<bits/stdc++.h>

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

#define ll long long int

#define F first

#define S second

#define pb push\_back

using namespace std;

using namespace \_\_gnu\_pbds;

typedef tree<ll, null\_type, less<ll>, rb\_tree\_tag,

tree\_order\_statistics\_node\_update>

new\_data\_set;

const ll N = 5005;

ll dp[N][N];

int main()

{

ios\_base:: sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

// ll t;

// cin>>t;

// while(t--)

{

ll i, j, k, n, m, ans = 0, cnt = 0, sum = 0;

cin >> n;

vector<ll> v(n);

for (i = 0; i < n; i++) {

cin >> v[i];

}

v.resize(unique(v.begin(), v.end()) - v.begin());

n = v.size();

for (j = 0; j < n; j++) {

for (i = j; i >= 0; i--) {

dp[i][j] = 1e9;

if (i == j) {

dp[i][j] = 0;

continue;

}

if (v[i] == v[j] && j > i + 1) {

dp[i][j] = 1 + dp[i + 1][j - 1];

}

else {

dp[i][j] = min({dp[i][j], dp[i + 1][j] + 1, dp[i][j - 1] + 1});

}

}

}

cout << dp[0][n - 1];

}

}

💡 Flood Fill



Course Progress