#include<bits/stdc++.h>

#define ll int64\_t

#define mod 1000000007

#define ima INT\_MAX

#define imi INT\_MIN

#define f(i, in, n) for (int64\_t i=in; i<n; i++)

#define fl(i, in, n) for (int64\_t i=n-1; i>=in; i--)

#define lim(i, a, b) for (int64\_t i=a; i<b; i++)

#define siz 1000001

using namespace std;

vector<bool> prime(siz, true);

void SieveOfEratosthenes() {

    for (ll p=2; p\*p<=siz; p++) {

        if (prime[p]) {

            for (ll i=p\*p; i<=siz; i+=p) prime[i]=false;

        }

    }

}

ll power(ll x, uint64\_t y, uint64\_t m) {

    if (y == 0)

        return 1;

    ll p = power(x, y / 2, m) % m;

    p = (p \* p) % m;

    return (y % 2 == 0) ? p : (x \* p) % m;

}

// Function to return gcd of a and b

ll gcd(ll a, ll b) {

    if (a == 0)

        return b;

    return gcd(b % a, a);

}

ll modInverse(ll a, ll m) {

    ll g = gcd(a, m);

    if (g != 1)

        return -1;

    else {

        return power(a, m - 2, m);

    }

}

ll getPairsCount(vector<ll> arr, ll n, ll sum) {

    unordered\_map<ll, ll> m;

    // Store counts of all elements in map m

    f(i, 0, n)

        m[arr[i]]++;

    ll twice\_count = 0;

    f(i, 0, n) {

        twice\_count += m[sum - arr[i]];

        if (sum - arr[i] == arr[i])

            twice\_count--;

    }

    // return the half of twice\_count

    return twice\_count / 2;

}

ll power(ll x, ll y, ll p) {

    ll res = 1;     // Initialize result

    x = x % p; // Update x if it is more than or

                // equal to p

    if (x == 0) return 0; // In case x is divisible by p;

    while (y > 0) {

        // If y is odd, multiply x with result

        if (y & 1)

            res = (res\*x) % p;

        // y must be even now

        y = y>>1; // y = y/2

        x = (x\*x) % p;

    }

    return res;

}

ll findComplement(ll n, ll bit\_count) {

   //ll bit\_count = floor(log2(n))+(ll)1;

   ll ones = (((ll)1 << bit\_count) - 1);

   return ones ^ n;

}

bool pal(string s) {

    string s1=s;

    reverse(s.begin(), s.end());

    string s2=s;

    return s1==s2;

}

ll fact(ll n) {

    ll res=1;

    f(i, 2, n+1) {

        res\*=i;

        //res=(res%mod+mod)%mod;

    }

    return res;

}

// Disjoint Sets

ll find(ll n, vector<ll> &parent, vector<ll> &rank) {

    if (parent[n]==n) return n;

    return parent[n]=find(parent[n], parent, rank);

}

void unionByRank(ll a, ll b, vector<ll> &parent, vector<ll> &rank) {

    ll x=find(a, parent, rank);

    ll y=find(b, parent, rank);

    if (x==y) return;

    if (rank[x]>=rank[y]) {

        parent[y]=x;

        rank[x]+=rank[y];

    }

    else {

        parent[x]=y;

        rank[y]+=rank[x];

    }

}

// Graph Algorithms

void dfs(ll i, vector<vector<ll>> &graph, vector<bool> &visited, ll &ct) {

    visited[i]=true;

    ct++;

    for (auto x : graph[i]) {

        if (!visited[x]) dfs(x, graph, visited, ct);

    }

}

vector<ll> Dijkstra(vector<vector<pair<ll, ll>>> &graph, ll S) {

    ll n=graph.size();

    vector<ll> distance(n, INT\_MAX);

    vector<bool> visited(n, false);

    priority\_queue< pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > pq;

    pq.push({0, S});

    distance[S]=0;

    while (!pq.empty()) {

        pair<ll, ll> top=pq.top();

        pq.pop();

        for (auto x : graph[top.second]) {

            if (!visited[x.first] and distance[top.second]+x.second<distance[x.first]) {

                distance[x.first]=distance[top.second]+x.second;

                pq.push({distance[x.first], x.first});

            }

        }

        visited[top.second]=true;

    }

    return distance;

}

// void dfs(ll i, vector<vector<pair<ll, ll>>> &graph, vector<bool> &visited, ll ct, ll &ans, ll &val, ll des) {

//     if (i==des and ct<=val) ans++;

//     visited[i]=true;

//     for (auto x : graph[i]) {

//         if (!visited[x.first]) dfs(x.first, graph, visited, ct+1, ans, val, des);

//     }

//     visited[i]=false;

// }

int getMaxSquare(vector<int> heights){

    stack<int> st;

    int i = 0, best\_n = 0;

    while(i < heights.size()){

        while(!st.empty() && heights[i] < heights[st.top()]){

            int n = st.top();

            st.pop();

            int w = st.empty() ? i  : i - st.top() - 1;

            int edge = min(w, heights[n]);

            best\_n = max(best\_n, edge);

        }

        st.push(i++);

    }

    while(!st.empty()){

        int n = st.top();

        st.pop();

        int w = st.empty() ? i  : i - st.top() - 1;

        int edge = min(w, heights[n]);

        best\_n = max(best\_n, edge);

    }

    return best\_n;

}

int main() {

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

    cin.tie(NULL);

    //SieveOfEratosthenes();

    const ll mo=998244353;

    ll t=1;

    cin>>t;

    while (t--) {

        ll n;

        cin>>n;

        string s;

        cin>>s;

        f(i, 0, n) {

            if (i%2!=0) s[i]='0';

        }

        ll value=0, ans=0;

        f(i, 0, n) {

            if (s[i]=='1') {

                value=((2\*value)+1)%mo;

            }

            else value=(2\*value)%mo;

            ans=ans xor value;

            ans=ans%mo;

        }

        cout<<ans<<"\n";

    }

    return 0;

}