**Template**

#include <bits/stdc++.h>

#define ENDL '\n'

#define deb(u) cout << #u " : " << (u) << ENDL;

#define deba(alias, u) cout << alias << ": " << (u) << ENDL;

#define debp(u, v) cout << u << " : " << v << ENDL;

#define pb push\_back

#define F first

#define S second

#define lli long long

#define ld long double

#define pii pair<int, int>

#define pll pair<lli, lli>

#define ALL(a) (a).begin(), (a).end()

#define ALLR(a) (a).rbegin(), (a).rend()

#define FOR(i, a, n) for (int i = (a); i < (n); ++i)

#define FORN(i, a, n) for (int i = (a - 1); i >= n; --i)

#define IO \

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

cin.tie(0); \

cout.tie(0)

using namespace std;

int main()

{

IO;

return 0;

}

**Build and run the code**

g++ problem.cpp && ./a.out < input.txt > output.txt

**Convex hull**

Graphical user interface, text

Description automatically generated

**Sprague-Grundy**

int getMex(unordered\_set<int> x){

int mex=0;

while(x.find(mex)!=x.end()){

mex++;

}

return mex;

}

int getGroundy(int n,vector<int> &results){

if(results[n]!=-1){

return results[n];

}

unordered\_set<int> aux;

if(n>0){

aux.insert(getGroundy(n-1,results));

}

if(n>1){

aux.insert(getGroundy(n-2,results));

}

if(n>2){

aux.insert(getGroundy(0,results));

}

results[n]=getMex(aux);

return results[n];

}

int main()

{

IO;

string s;

cin>>s;

unordered\_map<char,int> piles;

for(auto x:s){

piles[x]++;

}

int n=0;

for(auto x:piles){

n=max(n,x.second);

}

vector<int> results(n+1,-1);

results[0]=0;

for(auto x:piles){

getGroundy(x.second,results);

}

int xorvalue=-1;

for(auto x:piles){

if(xorvalue==-1){

xorvalue=results[x.second];

}else{

xorvalue^=results[x.second];

}

}

cout<<(xorvalue!=0?"Alice":"Bob")<<ENDL;

return 0;

}

**Segment Tree**

void build(int node,int start,int end){

if(start==end){

st[node]=numbers[start];

}else{

int mid=(start+end)/2;

build(node\*2,start,mid);

build(node\*2+1,mid+1,end);

st[node]=st[node\*2]+st[node\*2+1];

}

}

void update(int node,int start,int end, int idx,int val){

if(start==end){

st[node]=val;

numbers[idx]=val;

}else{

int mid=(start+end)/2;

if(start<=idx && idx<=mid){

update(node\*2,start,mid,idx,val);

}else{

update(node\*2+1,mid+1,end,idx,val);

}

st[node]=st[node\*2]+st[node\*2+1];

}

}

int query(int node, int start, int end ,int l ,int r){

if(r<start || end<l){

return 0;

}

if(l<=start && end<=r){

return st[node];

}

int mid=(start+end)/2;

return query(node\*2,start,mid,l,r)+query(node\*2+1,mid+1,end,l,r);

}