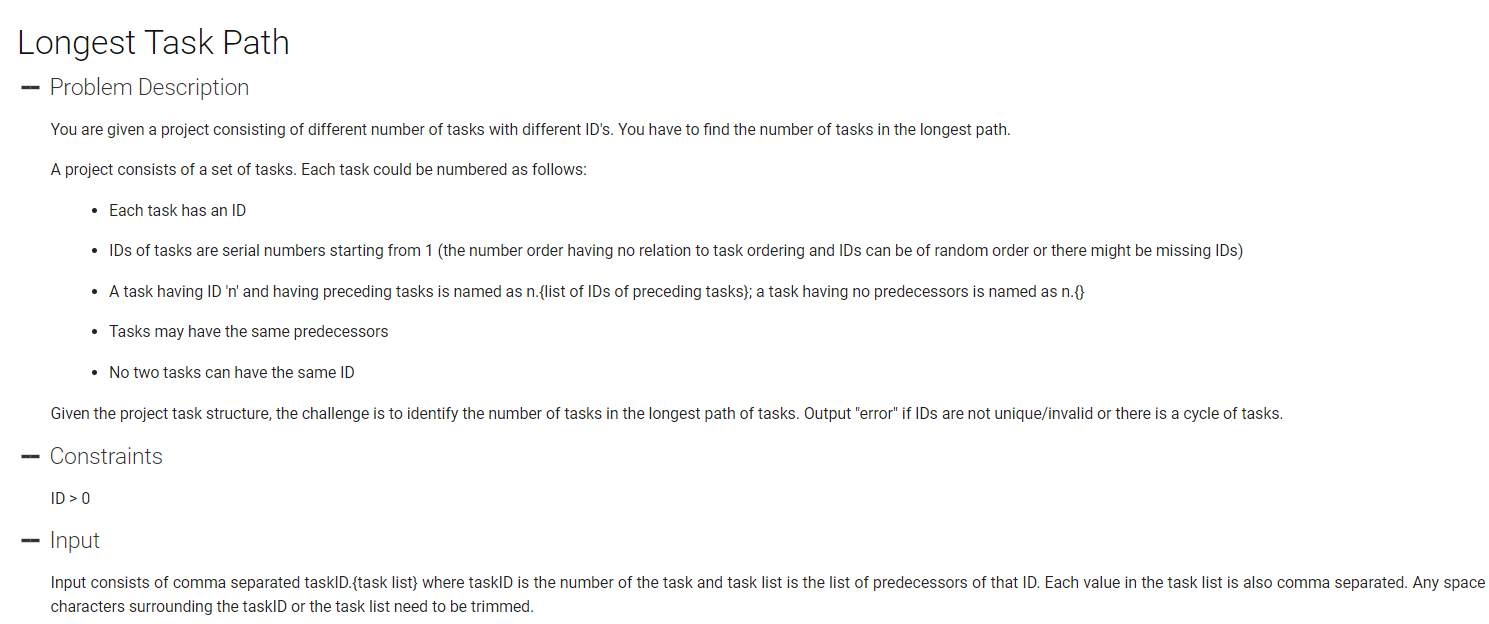
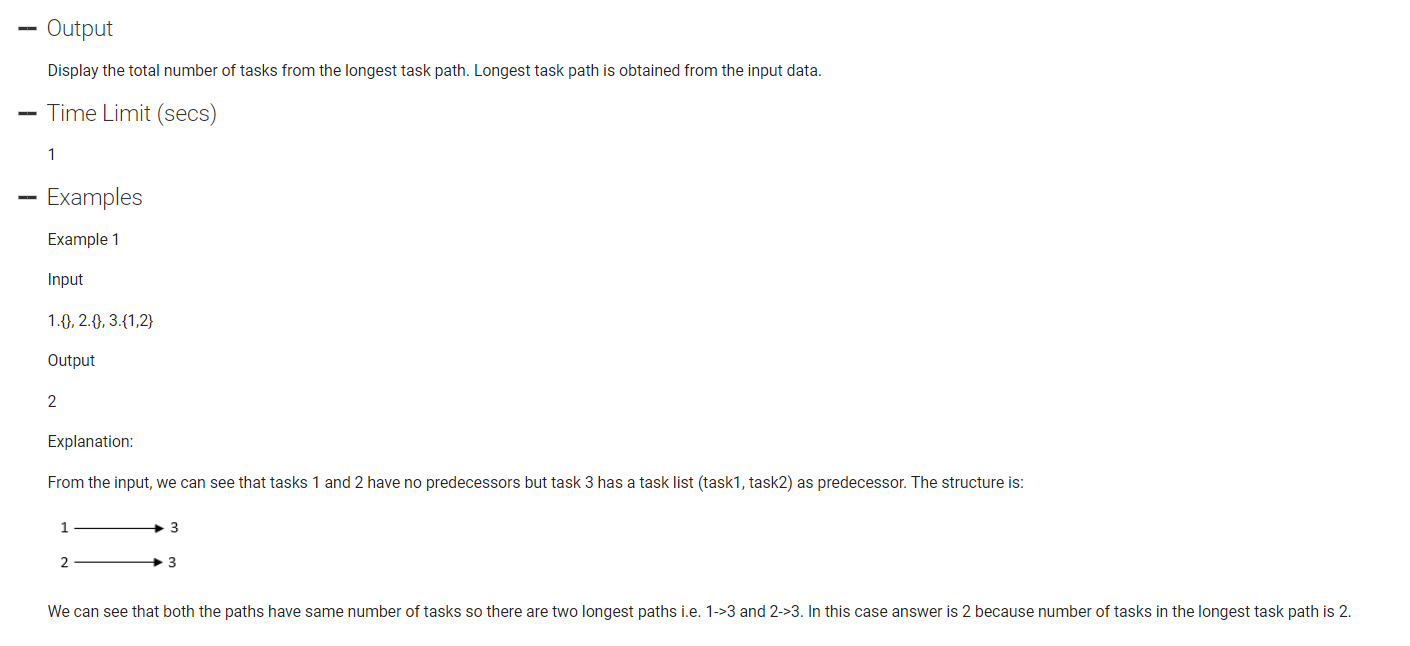
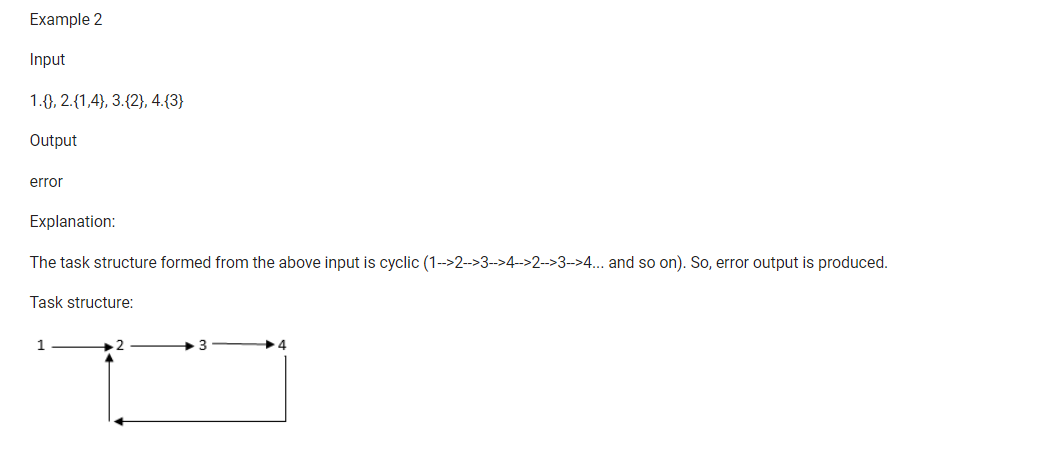
1. LONGEST TASK PATH







#include <bits/stdc++.h>

using namespace std;

unordered\_map<int, vector<int>> mp;

unordered\_set<int> st;

int len;

vector<int> vis;

void printMap()

{

for(auto p : mp)

{

cout<<p.first<<": ";

for(auto x : p.second) cout<<x<<" ";

cout<<"\n";

}

}

// class dsu

// {

// vector<int> pa;

// dsu()

// {

// for(int i=0; i<=n; i++) pa.push\_back(i);

// int findRoot(int u)

// {

// if(u == pa[u]) return u;

// pa[u] = findRoot(pa[u]);

// return pa[u]

// }

// bool doUnion(int u, int v)

// {

// int ru = findRoot(u);

// int rv = findRoot(v);

// if(ru == rv) return true;

// pa[rv] = ru;

// return false;

// }

// }

// }

bool findLongestPath(int s, int l)

{

if(vis[s])

{

return true;

}

if(st.count(s)) return true;

st.insert(s);

vis[s] = 1;

len = max(len, l);

for(auto x : mp[s])

{

if(findLongestPath(x, l+1)) return true;

}

return false;

}

int main() {

int flag = 0, n, req;

vector<string> v;

string s, x;

getline(cin, s);

// cout<<s<<"\n";

x = "";

for(int i=0; i<s.length(); i++)

{

if(s[i] == ' ') continue;

if(flag == 0 && s[i] == ',')

{

v.push\_back(x);

x = "";

continue;

}

if(s[i] == '{') flag = 1;

if(s[i] == '}') flag = 0;

x += s[i];

}

if(x != "") v.push\_back(x);

int i;

for(auto k : v)

{

n = 0;

req = 0;

for(i=0; i<k.length(); i++)

{

if(k[i] == '.')

{

break;

}

req = req\*10 + (k[i]-'0');

}

for(int j = i+1; j<k.length(); j++)

{

if(k[j] == '{' || k[j] == '}' || k[j] == ' ') continue;

if(k[j] == ',')

{

mp[n].push\_back(req);

n = 0;

continue;

}

n = n\*10 + (k[j] - '0');

}

if(n != 0) mp[n].push\_back(req);

}

len = 0;

flag = 0;

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

{

vis.assign(n+2, 0);

st.clear();

if(findLongestPath(i, 1)) flag = 1;

}

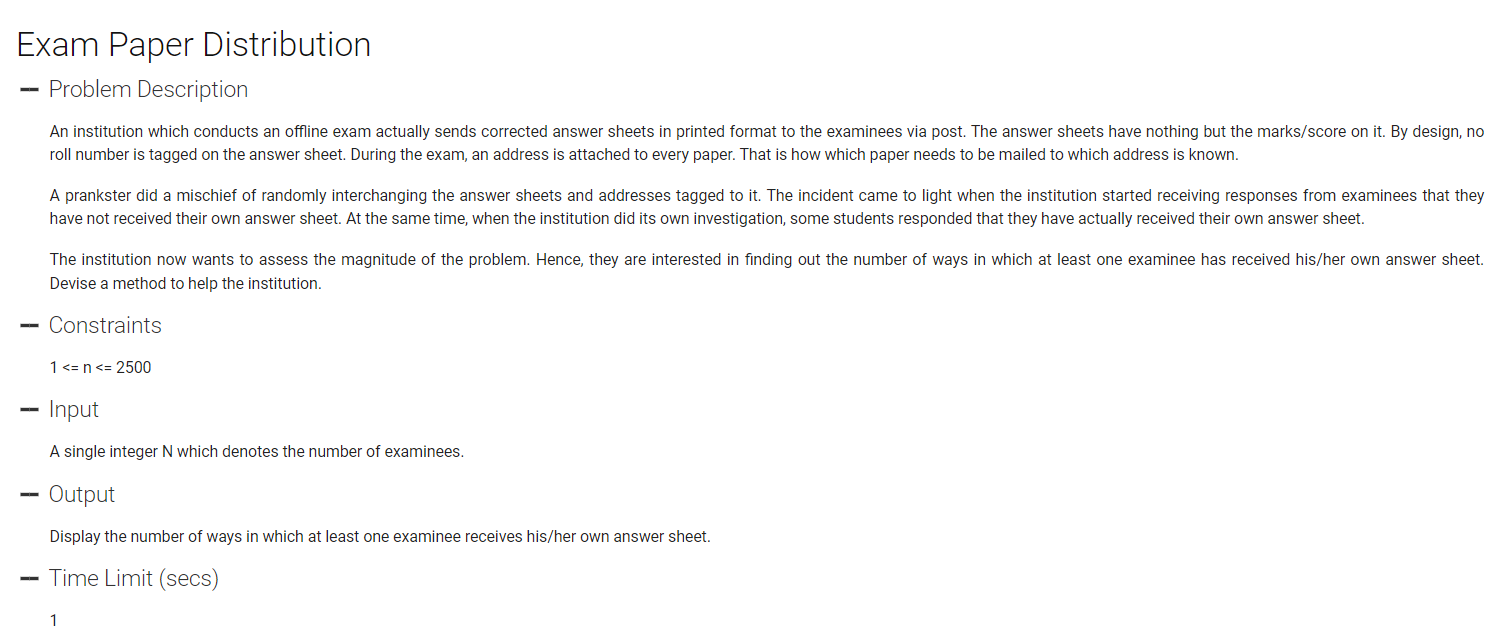
if(flag) cout<<"error\n";

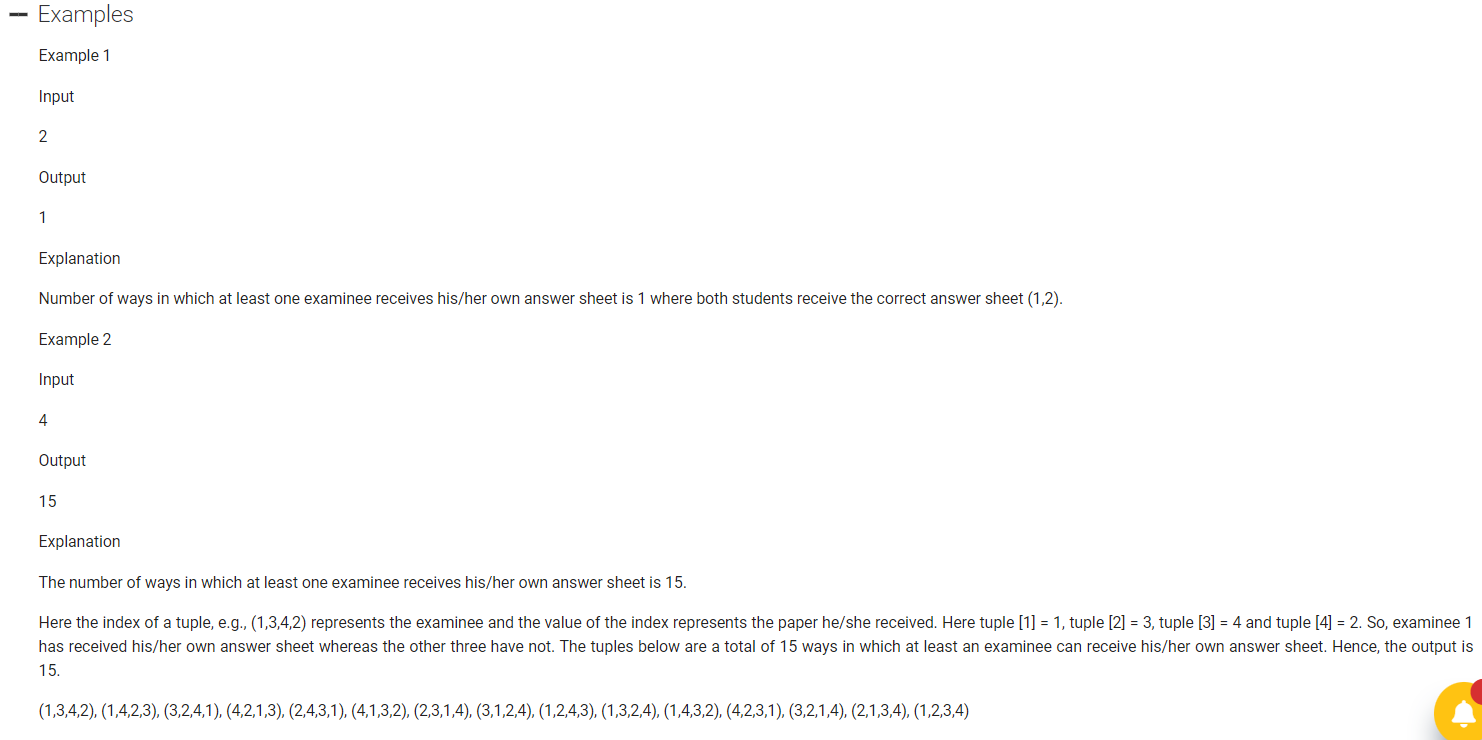
else cout<<len<<"\n";

return 0;

}

1. EXAM PAPER DISTRIBUTION





#include<bits/stdc++.h>

using namespace std;

int n, ans;

vector<int> visited;

vector<int> res;

void countValidSets(int valid)

{

if(res.size() == n)

{

if(valid != 0)

{

ans++;

}

return;

}

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

{

if(!visited[i])

{

visited[i] = 1;

res.push\_back(i);

if(i == res.size()) countValidSets(valid+1);

else countValidSets(valid);

res.pop\_back();

visited[i] = 0;

}

}

}

int main()

{

cin>>n;

visited.assign(n+1, 0);

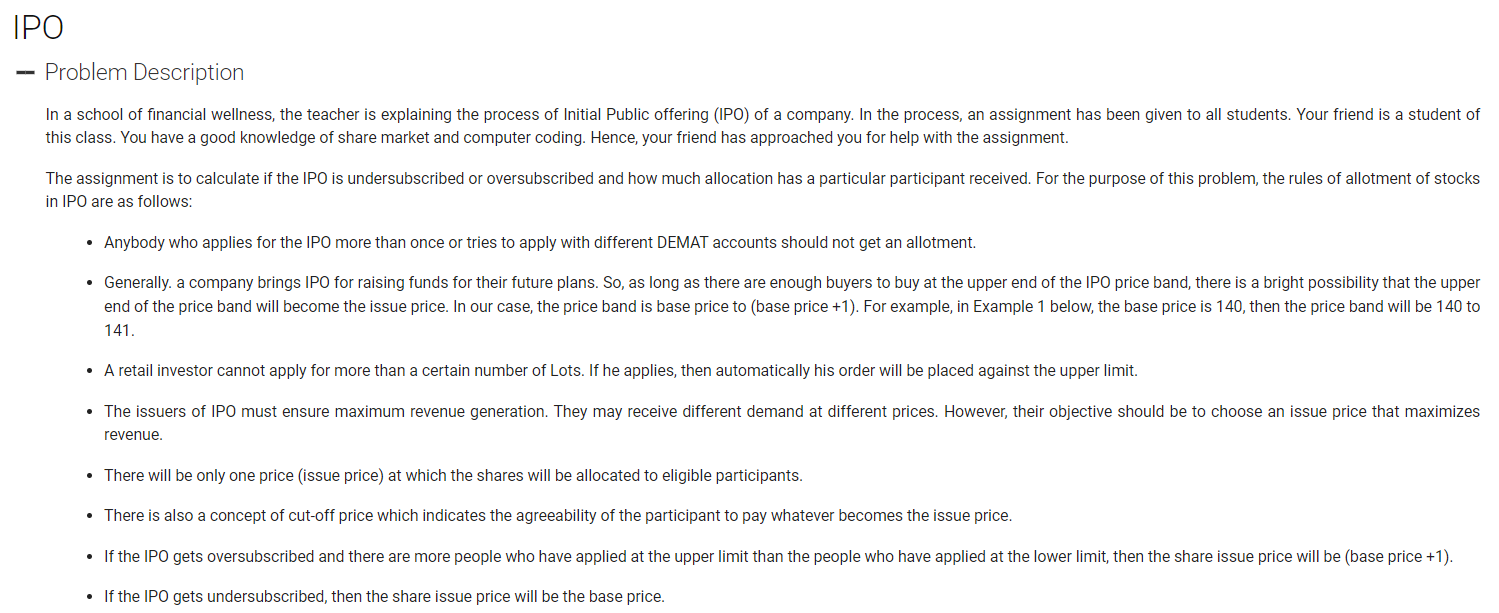
ans = 0;

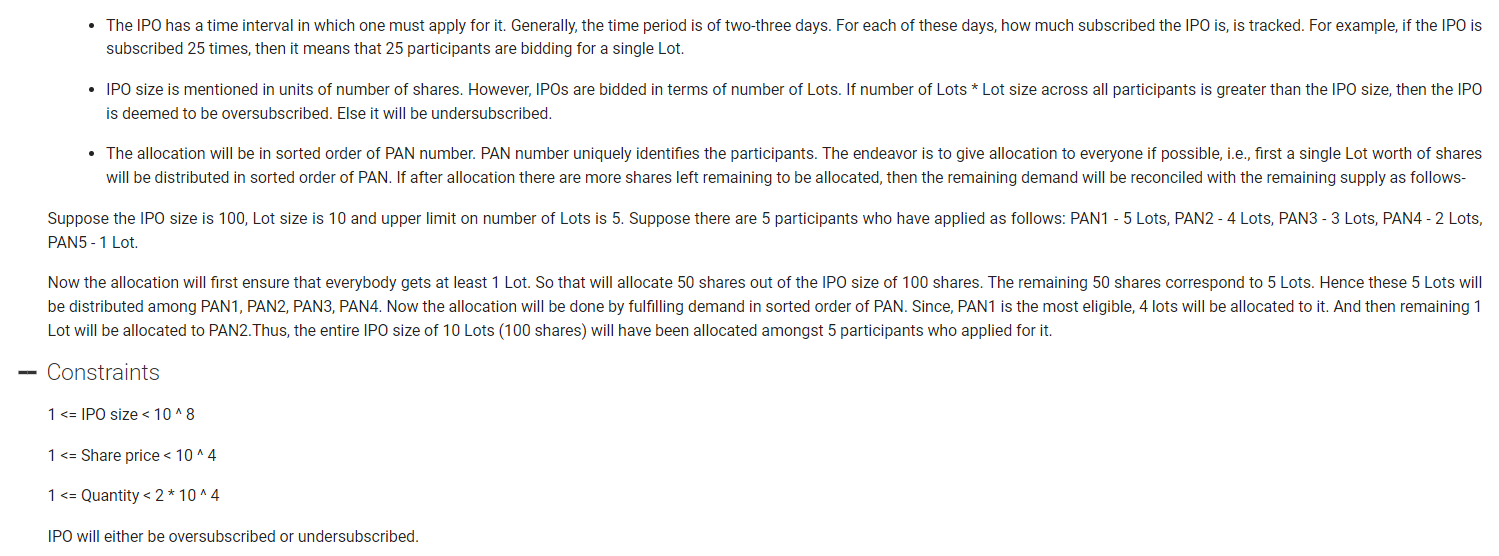
countValidSets(0);

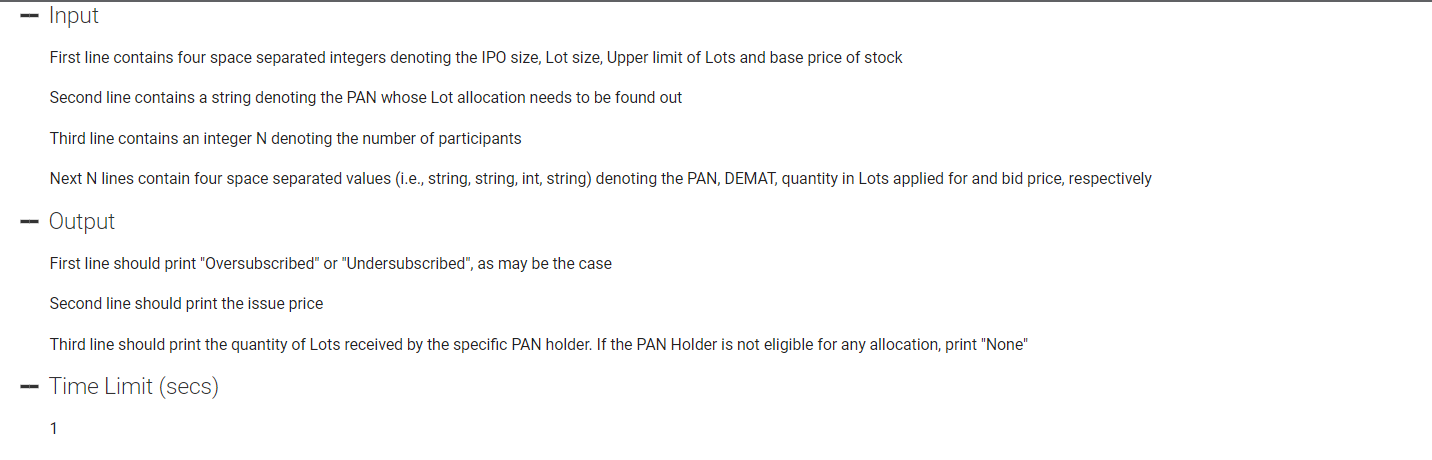
cout<<ans;

}

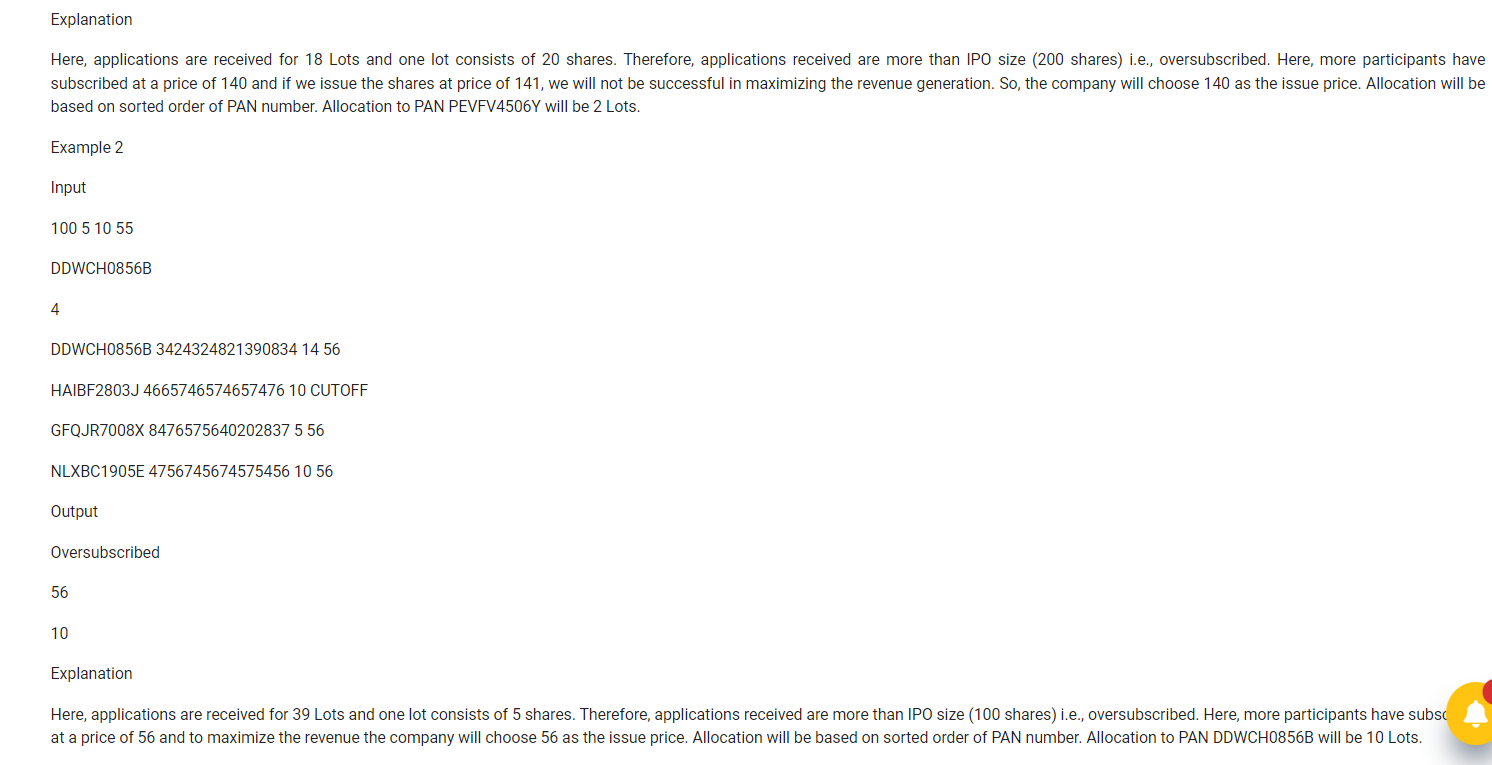
1. IPO



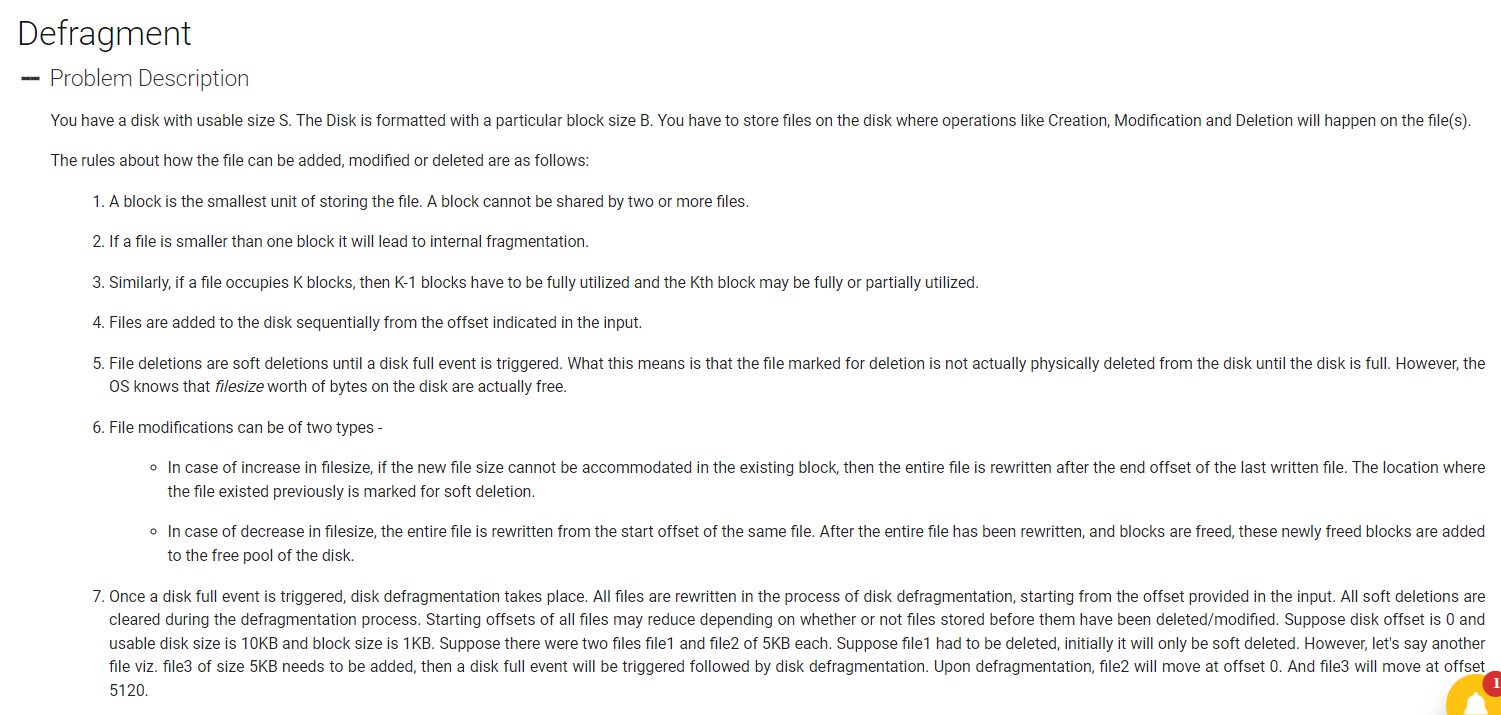


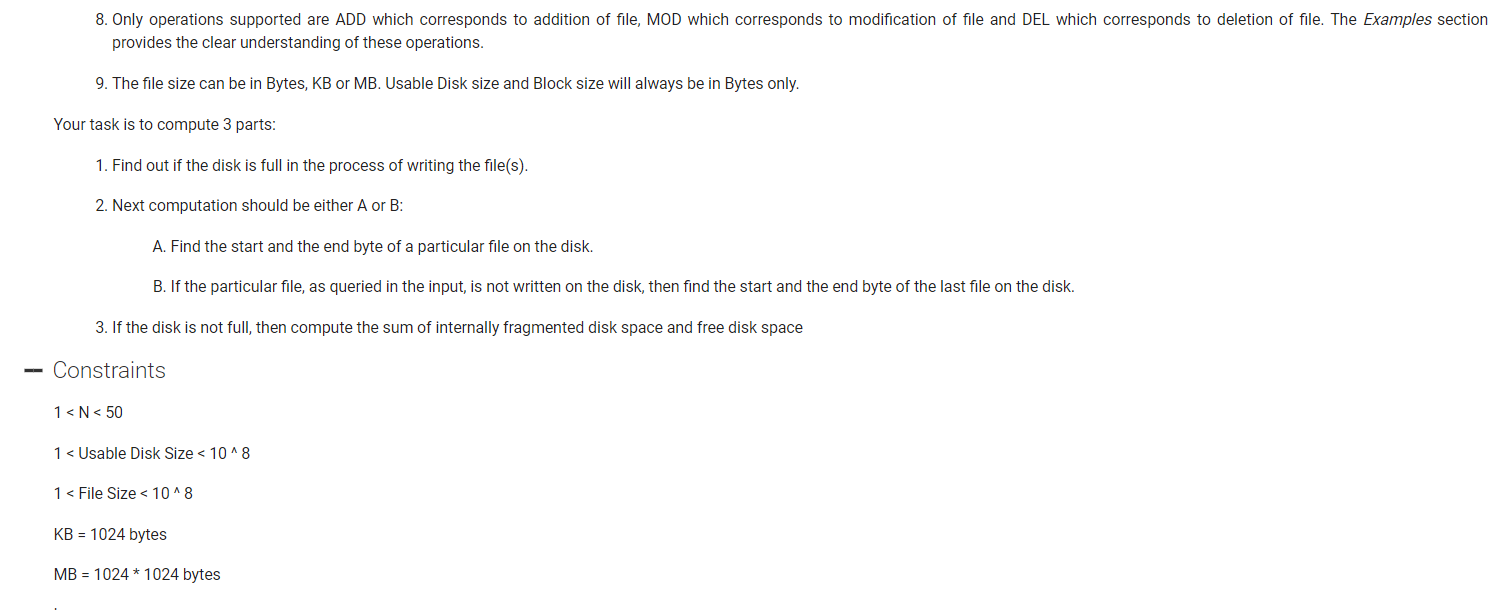


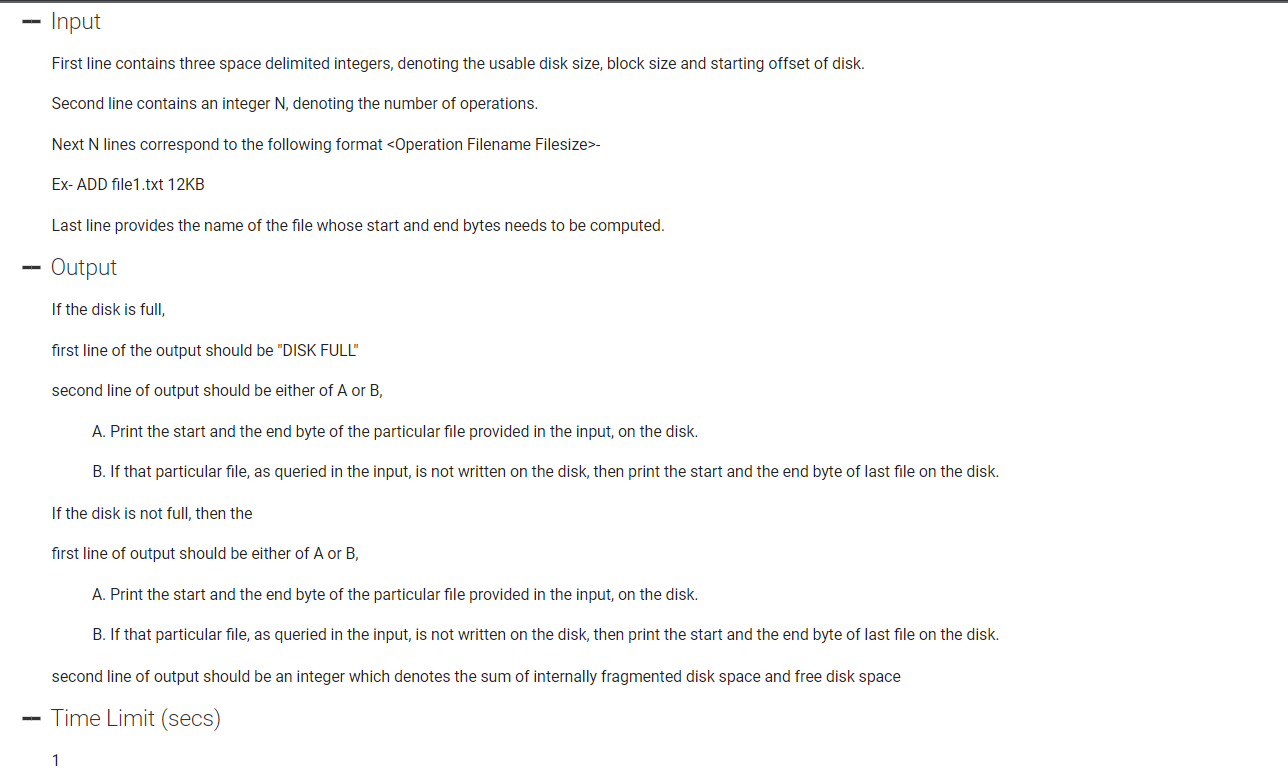




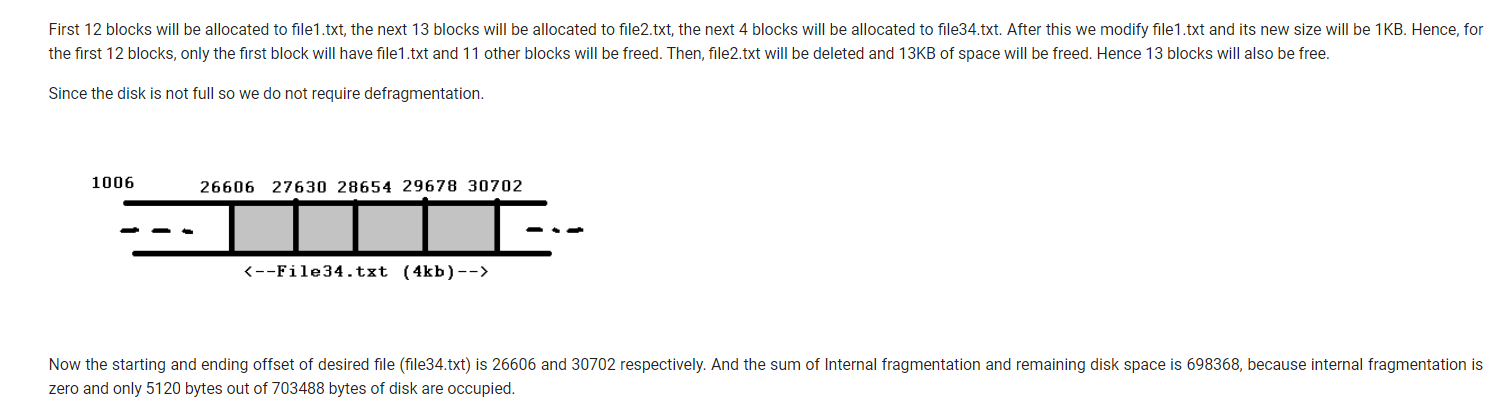
1. DEFRAGMENT



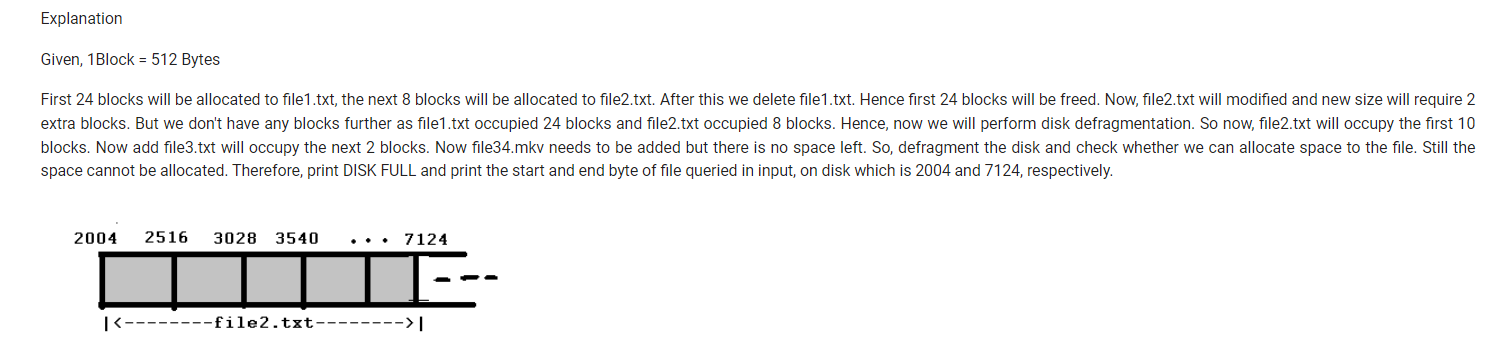


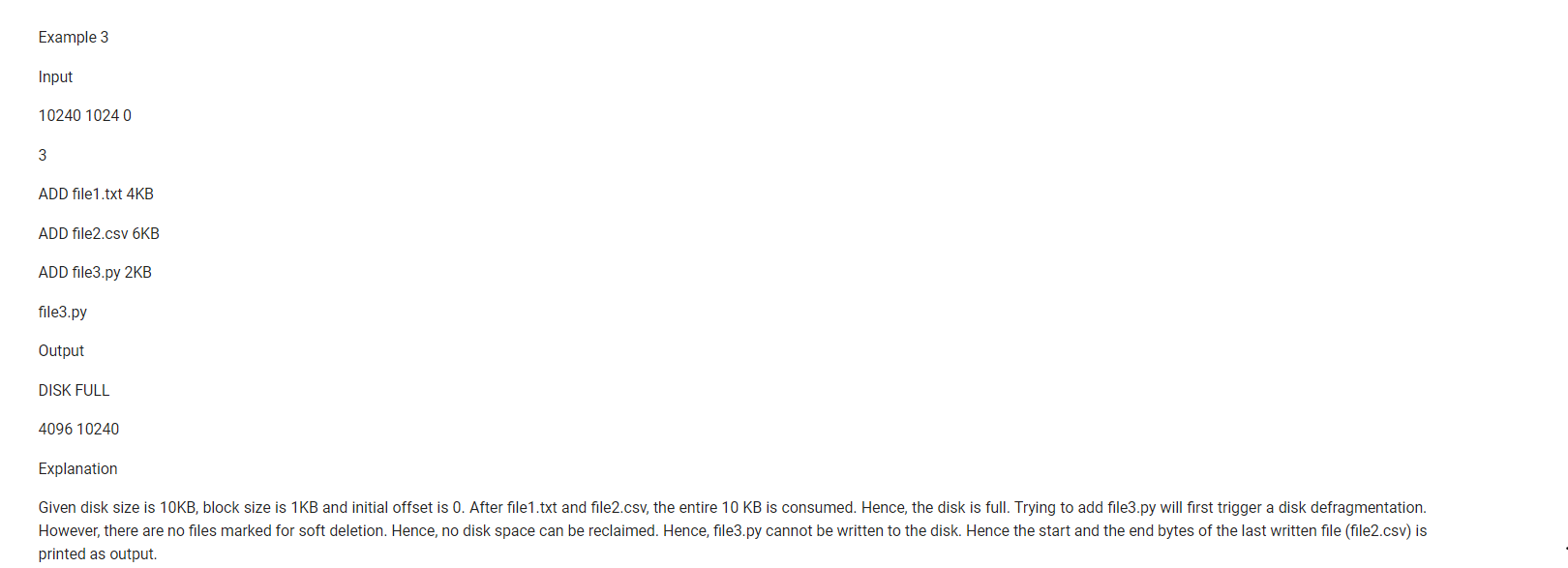




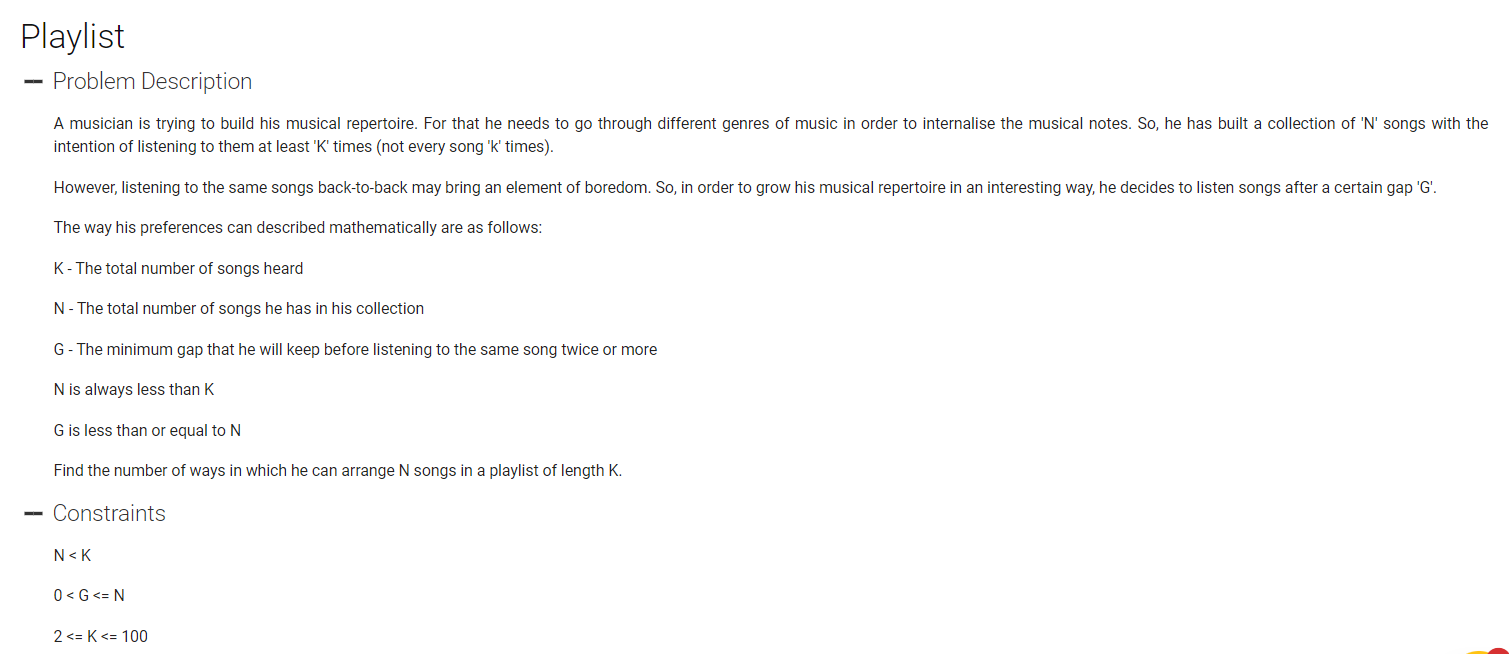


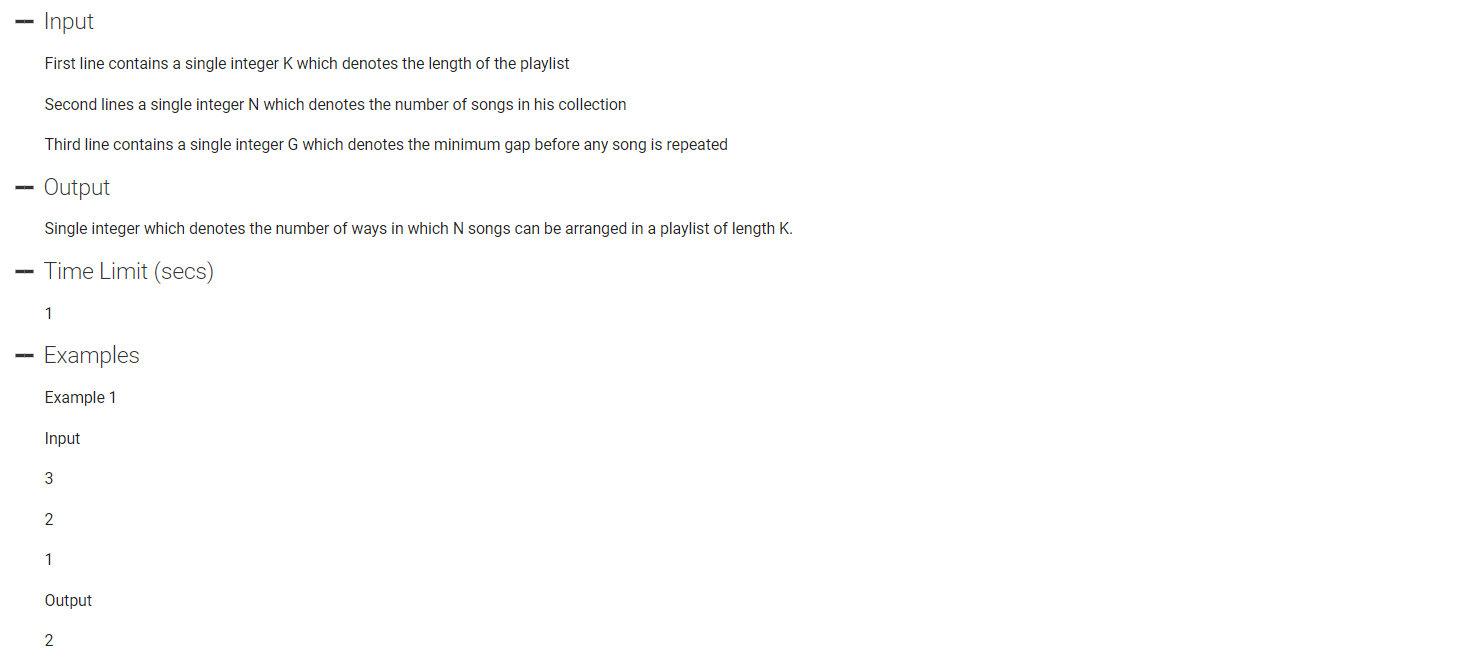


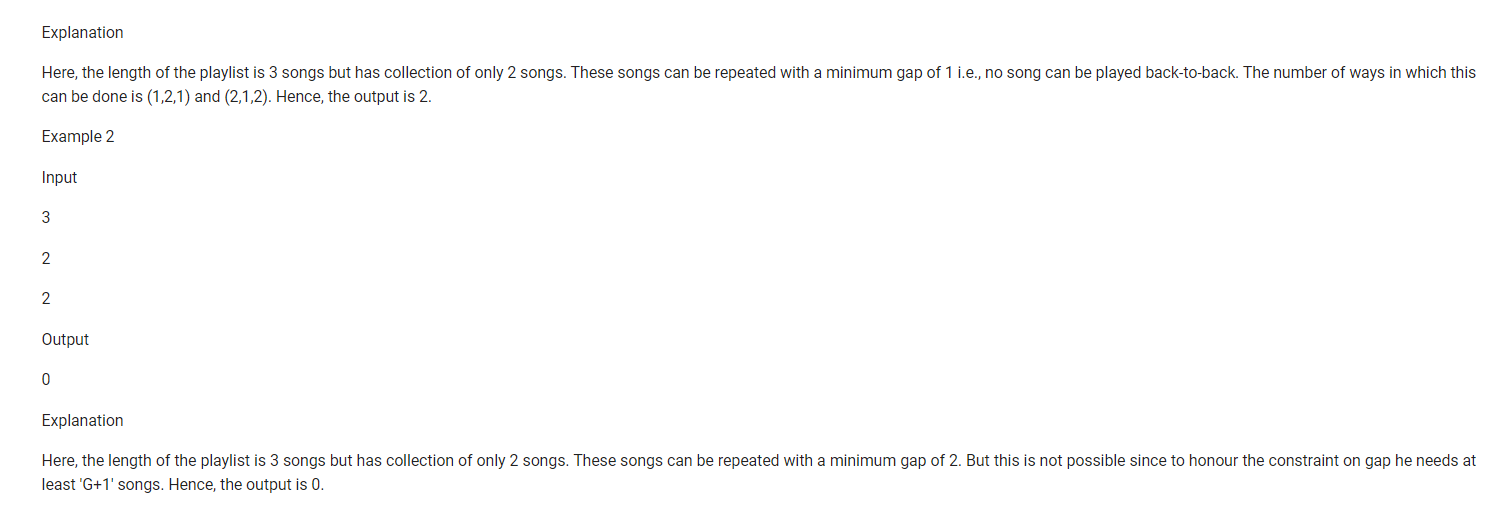




1. PLAYLIST







#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll k, n, g, ans;

void tolArrangements(ll vertex, ll time, unordered\_map<ll, ll> mp)

{

if(mp.count(vertex) && mp[vertex] + g >= time) return;

if(time == k)

{

ans++;

return;

}

int k = mp[vertex];

mp[vertex] = time;

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

{

tolArrangements(i, time+1, mp);

}

}

int main() {

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

cin.tie(NULL);

cout.tie(NULL);

cin>>k>>n>>g;

unordered\_map<ll, ll> mp;

ans = 0;

// for(ll i=1; i<=n; i++)

// {

// mp.clear();

// tolArrangements(i, 0);

// }

tolArrangements(1, 1, mp);

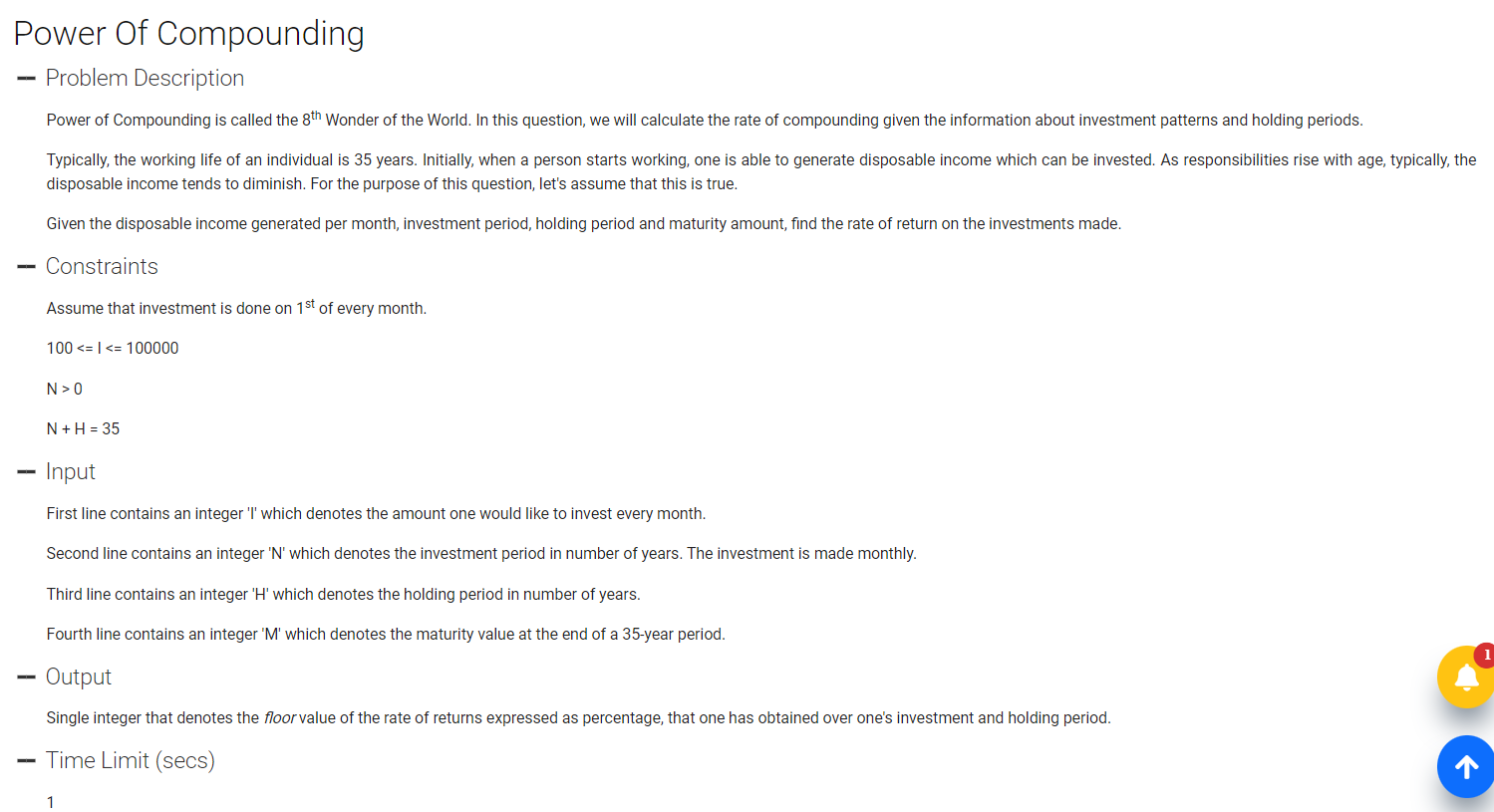
if(g >= n) cout<<0<<"\n";

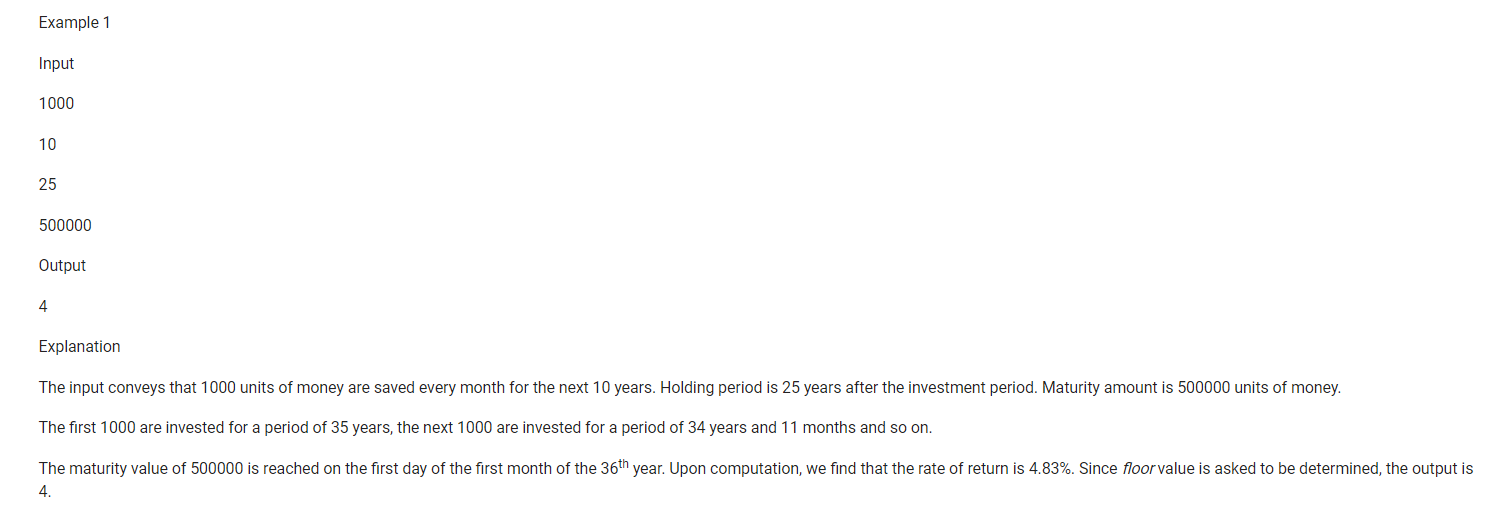
else cout<<ans\*n<<"\n";

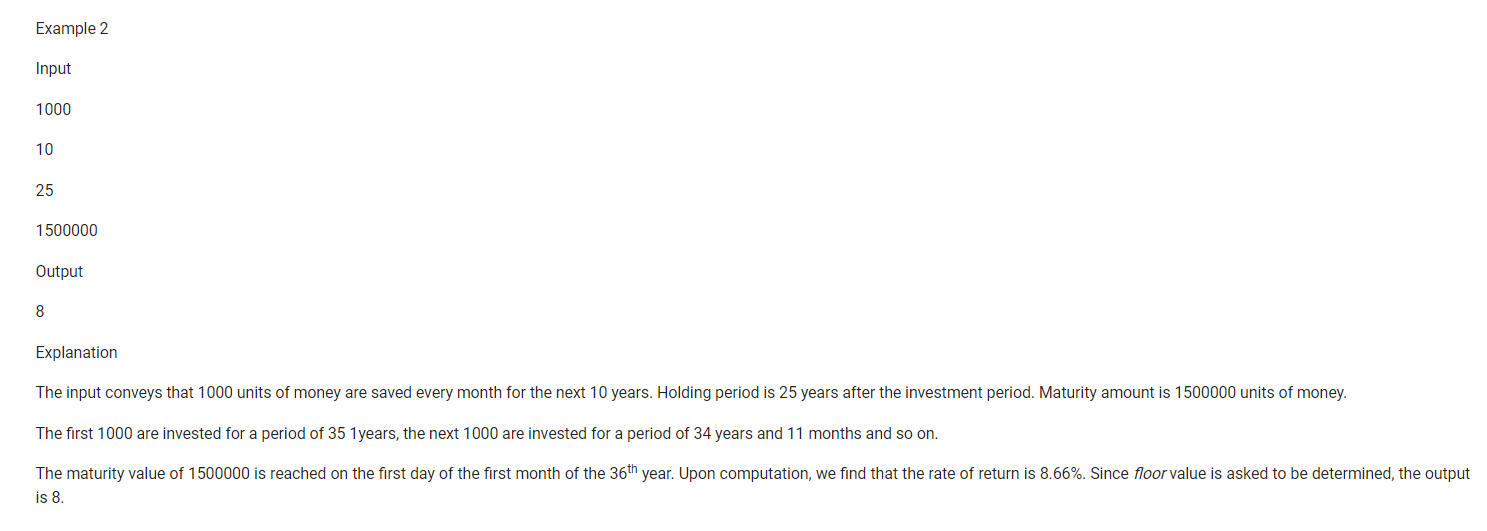
return 0;

}

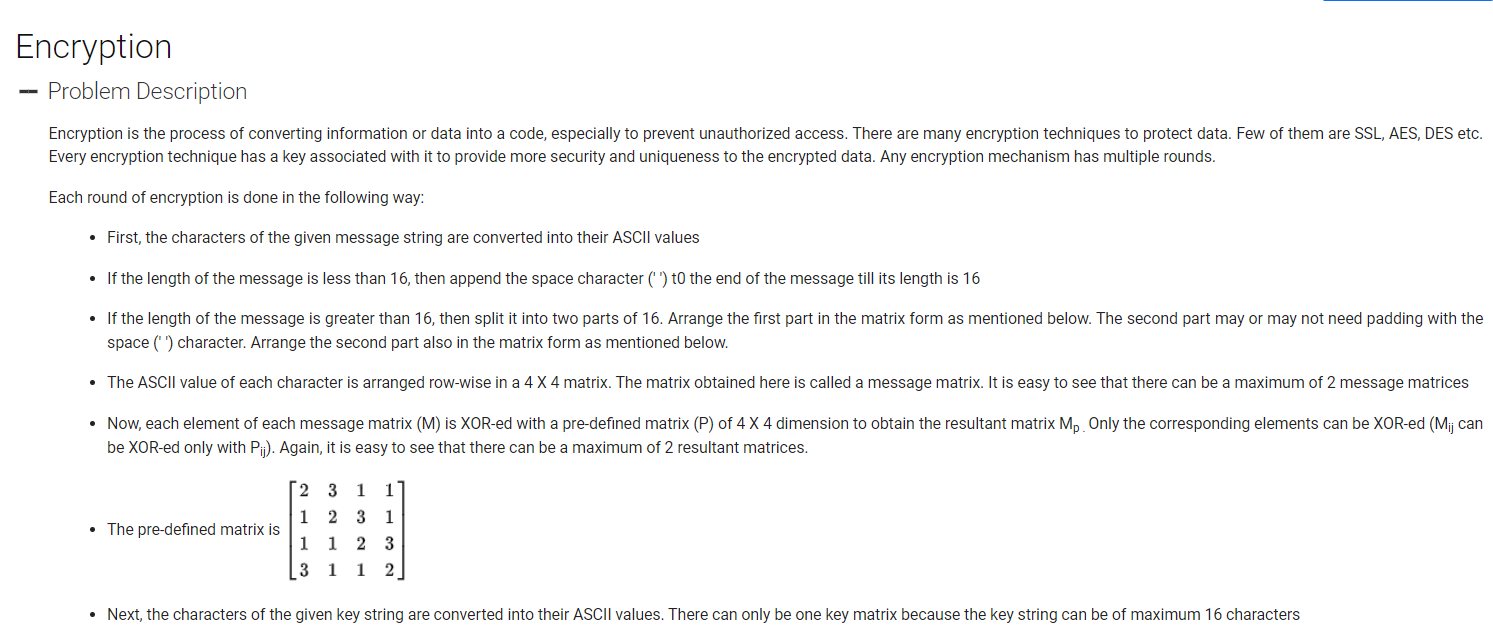
1. POWER OF COMPOUNDING

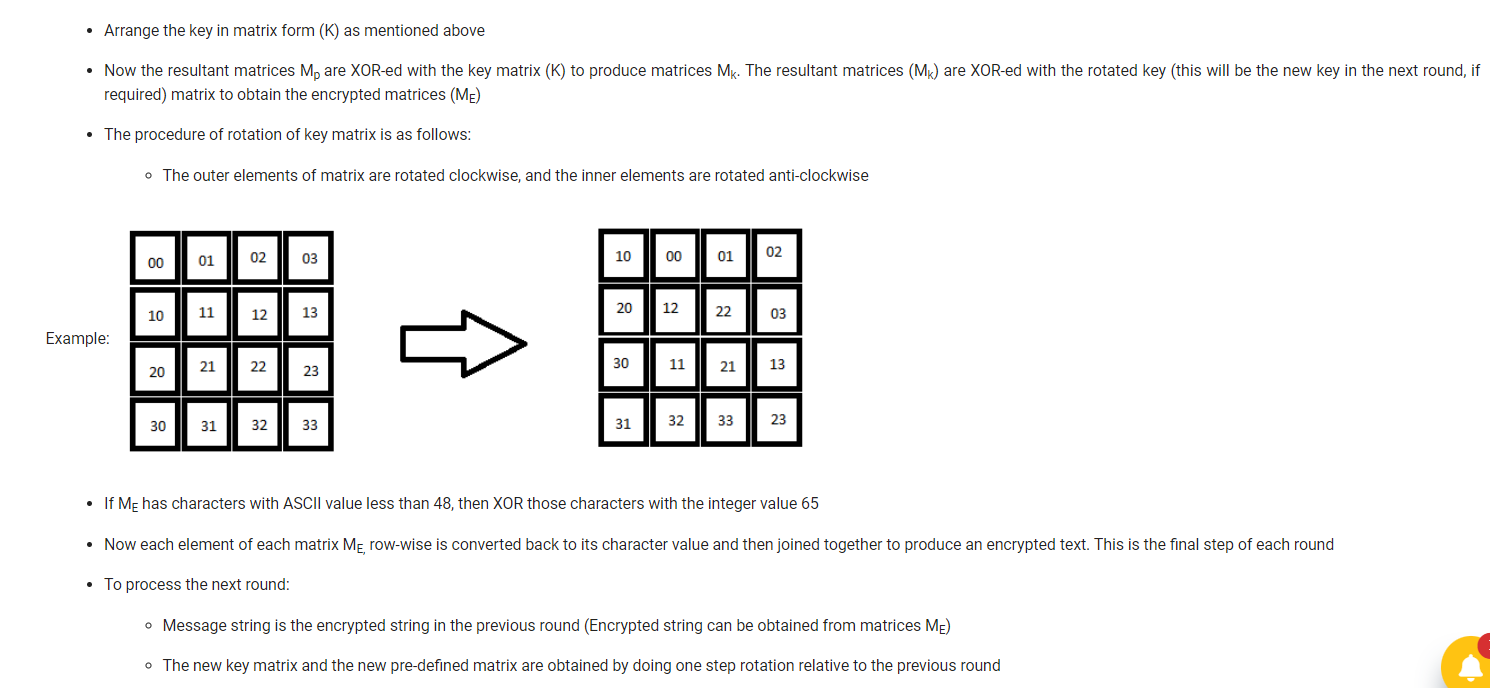


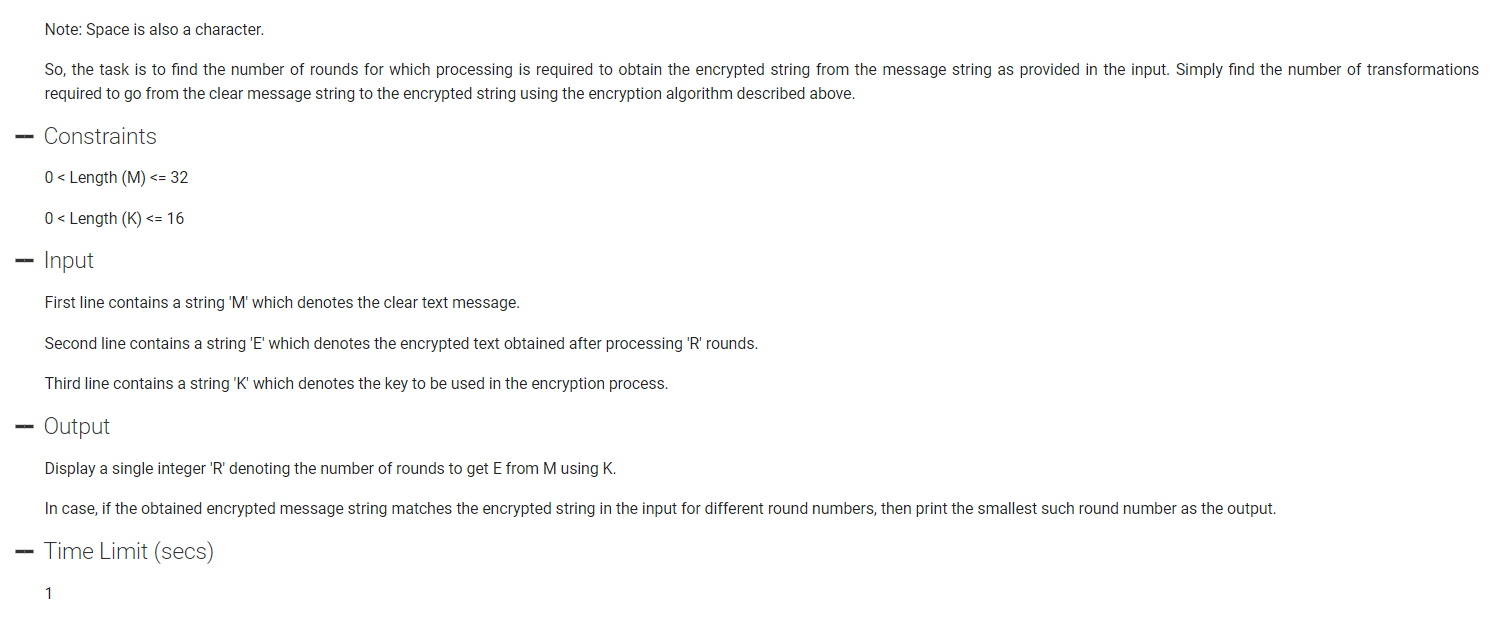


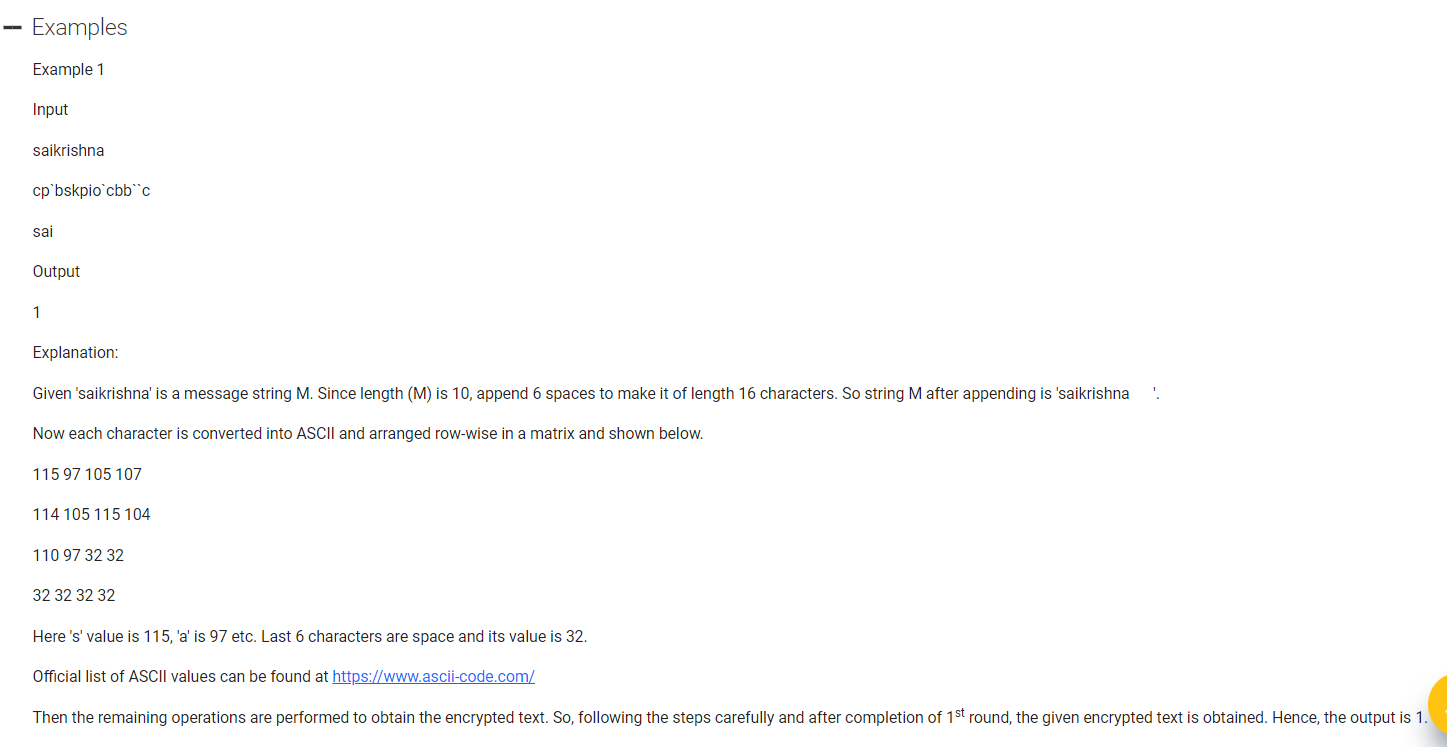


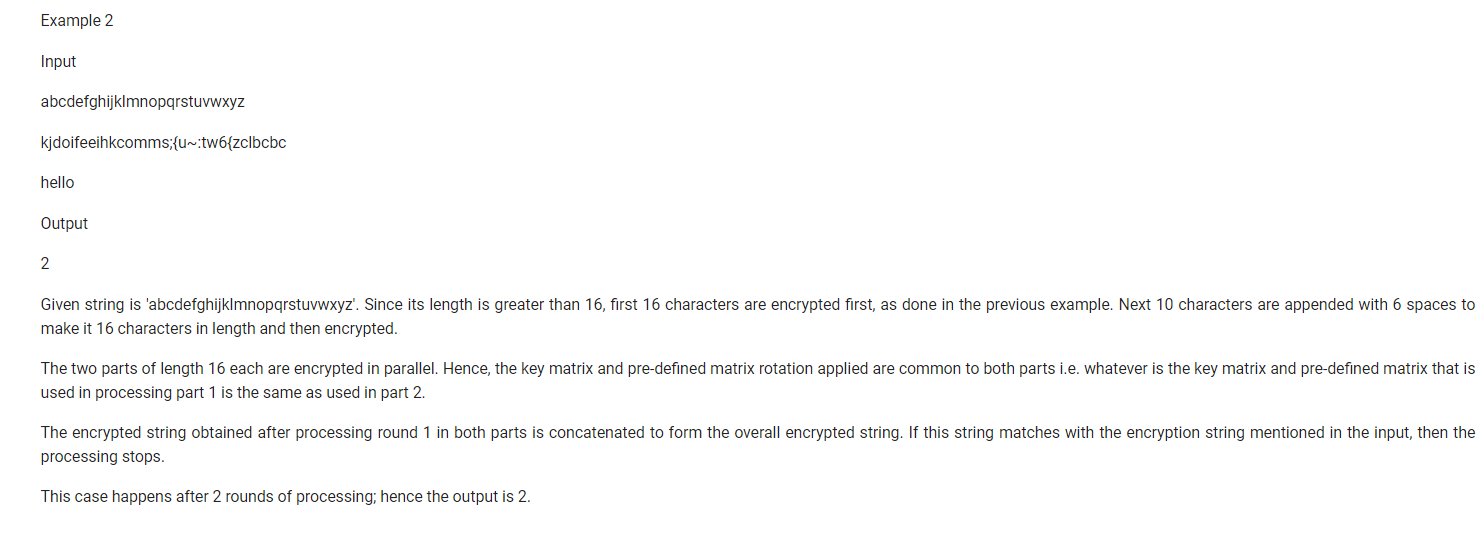
1. ENCRYPTION











1. SYSTEMATIC EQUITY PLAN

