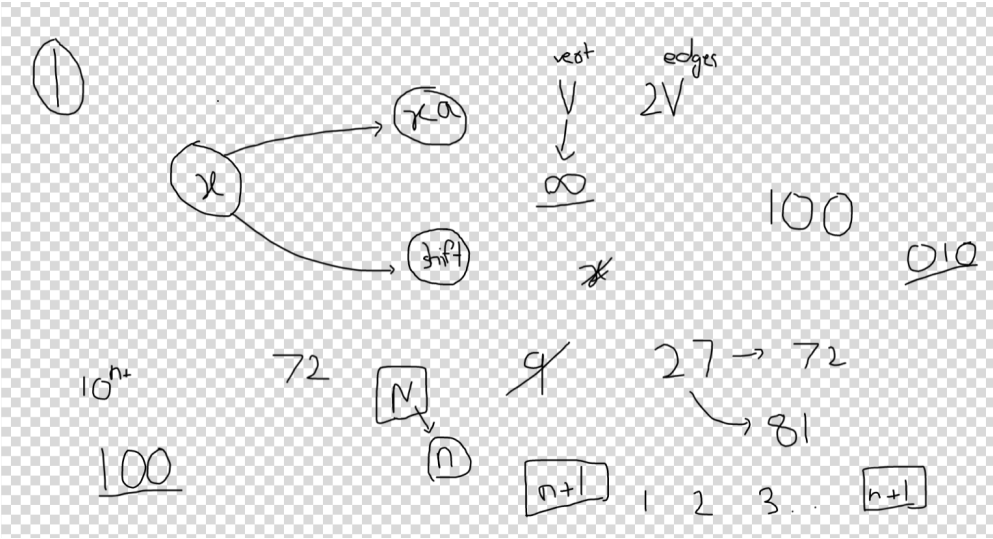
Question: https://atcoder.jp/contests/abc235/tasks/abc235_d
Code:

```cpp
#include<bits/stdc++.h>
using namespace std;

const int lim = 1e7+5;
const int inf = 1e9;

int ans[lim];

//Observation:
//1. Unit weight edges (all are equal)
//Hence Problem is shortened to shortest path problem
//with single source node
//Solve using BFS

int powersOf10[8] = {1, 10, 100, 1000, 10000, 100000, 1000000, 10000000};

int method1(int num, int multiplier){
    if(num*multiplier >= lim)
        return -1;

    return num*multiplier;
}

//12345 -> 50000 + 1234 = 51234
int method2(int num){
    if(num<10 || num%10==0)
        return -1;

    int digs = 0, temp=num;
    while(temp!=0){
        digs++;
        temp/=10;
    }

    int res = (num%10) * powersOf10[digs-1];
    res += (num/10);

    if(res >= lim)
```

```
        return -1;

    return res;
}

int main(){
    for(int i=0; i<lim; i++)
        ans[i] = inf;

    int a, N;
    cin>>a>>N;

    queue<int> q;
    ans[1] = 0;
    q.push(1);
    while(!q.empty()){
        int f = q.front();
        q.pop();

        int nxt1 = method1(f, a);
        if(nxt1!=-1 && ans[nxt1] == inf){
            ans[nxt1] = ans[f]+1;
            q.push(nxt1);
        }

        int nxt2 = method2(f);
        if(nxt1 != -1 && ans[nxt2] == inf){
            ans[nxt2] = ans[f]+1;
            q.push(nxt2);
        }
    }

    cout<<(ans[N]==inf?-1:ans[N]);
}
```



Given n cities and m bidirectional roads. Each road has a cost associated with it. In order to travel you have to pay the required cost of that particular road.
A traveller wants to travel from city 1 tp city n. Find the path such that he/she needs to pay minimum cost.
Given
1<= n <=1e5,
1<= m <=1e5
//-> Use Normal Djikstra For this

Given n cities and m bidirectional roads. Each road has a cost associated with it. In order to travel you have to pay the required cost of that particular road.
A traveller wants to travel from city 1 to city n.
Now Assume that the traveller is
a magician and he/she can remove the cost of any road, but he/she can do this for atmost k times. Find the path such that the travelling cost is minimum.
he/she needs to pay minimum cost.
Given
1<= n <=1e5,
1<= m <=1e5,
1<= k <=1e5
1<= n*k <=1e5

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define INF 1000000000000000000
```

```cpp
#define ff first
#define ss second


int djikstra(int n, int k, vector <pair<int,int>> adj[])
{
    int dist[n+1][k+1];
    for(int i=0;i<=n;i++)
    for(int j=0;j<=k;j++)
    dist[i][j] = INF;

    dist[1][0] = 0;

    // set <pair<dist, state>>
    //set <pair<dist, pair<node,no_of_moves_till_now>>>
    set <pair<int,pair<int,int>>> s;
    s.insert({0,{1,0}});

    while(!s.empty())
    {
        pair <int,pair<int,int>> p = *s.begin();
        s.erase(s.begin());
        // p.ss.ff->node
        // p.ss.ss->moves done till now
        // p.ff->dist

        //edges->{nextNode, weight}
        for(auto e:adj[p.ss.ff])
        {
            // [p.ss.ff][p.ss.ss]
            // [e.ff][p.ss.ss]
            // [e.ff][p.ss.ss+1]

            if(dist[e.ff][p.ss.ss]>p.ff+e.ss)
            {
                if(dist[e.ff][p.ss.ss]!=INF)
                s.erase({dist[e.ff][p.ss.ss] , {e.ff,p.ss.ss}});
                dist[e.ff][p.ss.ss] = p.ff+e.ss;
                s.insert({dist[e.ff][p.ss.ss] , {e.ff,p.ss.ss}});
            }
```

```
            if(p.ss.ss < k && dist[e.ff][p.ss.ss+1]>p.ff)
            {
                if(dist[e.ff][p.ss.ss+1]!=INF)
                s.erase({dist[e.ff][p.ss.ss+1] , {e.ff,p.ss.ss+1}});
                dist[e.ff][p.ss.ss+1] = p.ff;
                s.insert({dist[e.ff][p.ss.ss+1] , {e.ff,p.ss.ss+1}});
            }

        }

    }
    // return dist[n][k] -> false, because it's not always possible to
reach nth node using
    // k maginc moves
    int ans = INF;
    for(int i=0;i<=k;i++)
    ans = min(ans,dist[n][i]);
    if(ans==INF)
    return -1;
    return ans;
}

int32_t main(){
    int n;
    cin>>n;
    int m;
    cin>>m;
    int k;
    cin>>k;
    vector <pair<int,int>> adj[n+1];
    for(int i=0;i<m;i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        adj[u].push_back({v,w});
        adj[v].push_back({u,w});
    }
    cout<<djikstra(n,k,adj);
    return 0;
}
```

The land of Champakvan has n cities connected by m bidirectional ropeways.
Each of the ropeways has a travel time associated with it.
Each city has 0 or more different kinds of flowers available. The types of flowers are numbered from 1 to k.
All the cities are numbered from 1 to n. You start at city 1 and want to reach the capital of Champakvan, city n , collecting at least 1 flower of each of the different types of flowers available in Champakvan. These flowers are required for the king to celebrate flower's day, very urgently. So, you need to find the minimum time to reach the capital with at least 1 flower of each of the types of flowers available in Champakvan.

1<= n <=1e3
1<= m <=2e3
1<= k <=10 -> {imply you can think of bitmasking}->2^k
1<= no of flowers in a city <=k
1<= cost of roadways <=1e4

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define INF 1000000000000000000
#define ff first
#define ss second


int dijkstra(int n, int f, vector <pair<int,int>> adj[], vector <int> bit)
{
    int k = (1ll<<f); // Total Combination of flowers
    int dist[n+1][k];
    for(int i=0;i<=n;i++)
    for(int j=0;j<k;j++)
    dist[i][j]=INF;
    dist[1][bit[1]] = 0;// Start with takin all flowers on city 1;
    // set <pair<dist, state>>
    // set <pair<dist,pair<node,flowers_taken>>>
    set <pair<int,pair<int,int>>> s;
    s.insert({0,{1,bit[1]}});
    while(!s.empty())
    {
        pair <int,pair<int,int>> p =*s.begin();
        s.erase(s.begin());
```

```
        for(auto e:adj[p.ss.ff]){
            if(dist[e.ff][p.ss.ss|bit[e.ff]]>p.ff+e.ss)
            {
                //p.ss.ss|bit[e.ff]->total flowers taken till e.ff
                if(dist[e.ff][p.ss.ss|bit[e.ff]]!=INF)

s.erase({dist[e.ff][p.ss.ss|bit[e.ff]],{e.ff,p.ss.ss|bit[e.ff]}});
                dist[e.ff][p.ss.ss|bit[e.ff]] = p.ff+e.ss;

s.insert({dist[e.ff][p.ss.ss|bit[e.ff]],{e.ff,p.ss.ss|bit[e.ff]}});
            }
        }
    }
    return dist[n][k-1];
}

int32_t main(){
    int n;
    cin>>n;
    int m;
    cin>>m;
    int k;
    cin>>k;// no of flowers
    vector <pair<int,int>> adj[n+1];
    for(int i=0;i<m;i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        adj[u].push_back({v,w});
        adj[v].push_back({u,w});
    }
    int q; // how many cities have flowers
    cin>>q;
    vector <int> bt(n+1,0);//represets flowers available in city i
    while(q--)
    {
        int city;
        cin>>city;
        int tot;
```

```
        cin>>tot;
        for(int i=0;i<tot;i++)
        {
            int flower_type;
            cin>>flower_type;
            bt[city] = bt[city]|(1ll<<(flower_type-1));// storing types of
flower using bitmasking
        }
    }
    cout<<dijkstra(n,k,adj,bt);
    return 0;
}
```

Disjoint Set Union

Naive Implementation:

```
//Disjoint Set Union:
//Data structure dealing with disjoint sets

//3 functions:
//1. make_dsu() -> prepare the data structure with initial disjoint sets
//2. find_rep(int elem) -> which disjoint set does 'elem' belong to
//3. union_dsu(int elem1, int elem2) -> unify the disjoint sets of elem1 and
elem2

const int N = 100;
int par[N+5];

void make_dsu(){
    for(int i=1; i<=N; i++){
        par[i] = i;
        //i is the representative of the disjoint set to which it belongs
    }
}

int find_rep(int elem){
    if(par[elem]==elem)
        return elem;
    return find_rep(par[elem]);
}
```

```
int union_dsu(int elem1, int elem2){
    int rep1 = find_rep(elem1);
    int rep2 = find_rep(elem2);
    if(rep1 != rep2){
        par[rep1] = rep2;
    }
}
```

Optimizations:
1. Path Compression: Storing the representative as the parent of all the elements on the path to the representative, so that further calls can be sped up.
2. Union By Size: When performing union, the representative of the set having more number of elements (greater size) is to be made the parent.

Optimized Implementation (Using Union By Size and Path Compression):

```
int sz[N+5];    //sz[rep] will give size of D.S.

void make_dsu(){
    for(int i=1; i<=N; i++){
        par[i] = i;
        sz[i] = i;
        //i is the representative of the disjoint set to which it belongs
    }
}

int find_rep(int elem){
    if(par[elem]==elem)
        return elem;
    par[elem] = find_rep(par[elem]);
    return par[elem];
}

int union_dsu(int elem1, int elem2){
    int rep1 = find_rep(elem1);
    int rep2 = find_rep(elem2);
    if(rep1 != rep2){
        //Establish parent-child
        int sz1 = sz[rep1];
        int sz2 = sz[rep2];
```

```
        if(sz1 < sz2){
            //2 will become parent
            par[rep1] = rep2;
            sz[rep2] += sz[rep1];
        }else{
            //1 will become parent
            par[rep2] = rep1;
            sz[rep1] += sz[rep2];
        }
    }
}
```

Problem: https://www.hackerrank.com/challenges/merging-communities/problem
Solution:

```
#include<bits/stdc++.h>
using namespace std;
vector<int> parent(1e5+1), height(1e5+1, 1), djsrank(1e5+1, 0);
int djsfind(int elem){
    if(parent[elem]!=elem)
        parent[elem] = djsfind(parent[elem]);
    return parent[elem];
}
void djsmerge(int a, int b){
    int root_a = djsfind(a), root_b=djsfind(b);
    if(root_a==root_b) return;
    if(djsrank[root_a]<djsrank[root_b]){
        height[root_b]+=height[root_a];
        parent[root_a] = root_b;
    }else{
        height[root_a] += height[root_b];
        parent[root_b] = root_a;
        if(djsrank[root_a]==djsrank[root_b]) djsrank[root_a]++;
    }
}
int main()
{
    int n, q, a, b;
```

```
    char oper;
    cin>>n>>q;
    for(int i=1; i<=n; i++) parent[i]=i;
    while(q--){
        cin>>oper;
        if(oper=='Q'){
            cin>>a;
            cout<<height[djsfind(a)]<<"\n";
        }else{
            cin>>a>>b;
            djsmerge(a, b);
        }
    }
}
```