```cpp
class DSU {
    int *parent;
    int *rank;

public:
    DSU(int n){
        parent = new int[n];
        rank = new int[n];

        for(int i=0;i<n;i++){
            parent[i] = -1; //initialize array
            rank[i] = 1;
        }
    }

    int findset(int x){
        if(parent[x] == -1){
            return x;
        }
        return parent[x] = findset(parent[x]);
    }

    void unionset(int x,int y){
        int setx = findset(x);
        int sety = findset(y);

        if(setx != sety){
            if(rank[setx] < rank[sety]){
                parent[setx] = sety;
            }else if(rank[setx] > rank[sety]){
                parent[sety] = setx;
            }else{
                parent[sety] = setx;
                rank[setx]++; //increase high
            }
        }
    }
};
```

สร้าง class disjoint set เฉยๆ

```cpp
int kruskalminW(int vertex ,  set<tuple<int,int,int> > &edge){
    DSU dsu(vertex+5);
    int cost =0 ;
    for(auto e : edge){
        int w,a,b;
        w = get<0>(e);
        a = get<1>(e);
        b = get<2>(e);
        if(dsu.findset(a) != dsu.findset(b)){
            dsu.unionset(a,b);
            cost += w;
        }
    }
    return cost;
}
```

อันนี้ใช้ set แต่แรกมันเรียงละ จากการใช้ tuple เราจึงใช้ get เรียกข้อมูลแต่ละตัว

```cpp
for(int i=0;i<m;i++){
        int w,a,b;
        cin >> w >> a >> b;
        a--,b--; //ให้เริ่มที่ 0
        //พวกโจทย์ให้เริ่มที่ 1 ทำให้ต้อง"จอง"อาเรย์ n+1 ช่อง
        //แต่เราทำงี้ทำให้จองแค่ n ช่องได้
        edge.insert({w,a,b});
    }
int bellman(int vertex,vector<vector<int> > &v,int start){
    vector<int> distance(vertex,INT_MAX);
    vector<int> prev(vertex,-1);
    distance[start] = 0;
    for(int i=0;i<vertex-1;i++){
        for(int j=0;j<vertex;j++){
            for(int k=0;k<vertex;k++){
                if(v[j][k] != -1 && distance[j] != INT_MAX){
                    if(distance[k] > distance[j] + v[j][k]){
                        distance[k] = distance[j] + v[j][k];
                        prev[k] = j;
                    }
                }
            }
        }
    }
    for(int i=0;i<vertex;i++){
        for(int j=0;j<vertex;j++){
            if(v[i][j] != -1 && distance[i] != INT_MAX ){
                if(distance[j] > distance[i] + v[i][j]){
                return -1;
                }
            }
        }
    }
}
```

```cpp
int minkey(int vertex,vector<int> &cost,vector<bool> &visited){
    int min = INT_MAX;
    int min_index = -1;
    for(int i=0;i<vertex;i++){
        if(!visited[i] && cost[i] < min){
            min = cost[i];
            min_index = i;
        }
    }
    return min_index;
}
```

```cpp
ios_base::sync_with_stdio(false);cin.tie(0);
```

```cpp
#include <bits//stdc++.h>
```

```cpp
int primMST(int vertex ,vector<vector<int> > &edge, vector<tuple<int, int, int>> &mst_edges){
    vector<int> cost(vertex,INT_MAX);
    vector<bool> visited(vertex,false);
    vector<int> parent(vertex,-1);

    cost[0] = 0;
    parent[0] = -1;//

    int totalcost = 0;

    for(int i=0;i<vertex;i++){
        int u = minkey(vertex,cost,visited);
        visited[u] = true;
        totalcost += cost[u];
        for(int v=0;v<vertex;v++){
            if(edge[u][v] && !visited[v] && edge[u][v] < cost[v]){
                cost[v] = edge[u][v];
                parent[v] = u; //ก่อนจะ v คือ u
            }
        }
    }
                            //สร้าง edge ที่ทำให้ mst แล้วใส่เข้ามา
    return totalcost;           for (int v = 1; v < vertex; v++) {
}                               if (parent[v] != -1) {
                                    mst_edges.push_back({parent[v]+1, v+1, edge[parent[v]][v]}); // Adjust to 1-based indexing
                                }
                            }
```

```cpp
vector<vector<int> > adj(n,vector<int>(n,INT_MAX));
    for(int i=0;i<m;i++){
        int w,a,b;
        cin >> w >> a >> b;
        a--,b--;
        adj[a][b] = w;
        adj[b][a] = w;
    }
```

เราสามารถเรียกรับค่าแบบนี้ได้เลย เพราะเราประกาศเป็นเวกเตอ 2 มิติ มันจะเป็นการใช้ adj matrix

อันนี้คือเอาไว้เก็บว่า edge ที่ทำให้เกิด mst คือเส้นไหนบ้าง ถ้าโจทย์อยากให้ทำก็ใช้ละใส่ไว้นอกลูปเลย

```cpp
for (const auto &[u, v, weight] : mst_edges) {
        cout << u << " - " << v << " : " << weight << endl;
    }
```

สั่ง print ตัวที่เก็บใน mst_edge ที่เก็บเป็น tuple ไว้อยู่

```cpp
struct pikatcost
{
    int i,j,k;
};
```

```cpp
queue<pikatcost> q;
q.push({0,0,0});
vector<vector<bool> > visited(r,vector<bool>(c,false));
int ans = 0;
while(!q.empty()){
    int x = q.front().i;
    int y = q.front().j;
    int cost = q.front().k;
    q.pop();

    if(x < 0 || y < 0 || x >= r || y >= c || visited[x][y] || v[x][y] == 0){
        continue;
    }
    visited[x][y] = true;
    if(v[x][y] == 2){ //เจอสมบัติต้องวิ่งไปกลับ
        ans += cost*2;
        treasure--;
    }
    if(treasure == 0){ //สมบัติหมดละ
        break;
    }
    q.push({x+1,y,cost+1});
    q.push({x-1,y,cost+1});
    q.push({x,y+1,cost+1});
    q.push({x,y-1,cost+1});
}
```

อันนี้มีการใช้งาน struct นิดหน่อย ให้เขียนง่ายๆ แต่จริงๆใช้ tuple ได้นะ ละ get เอา

หา shortest path แต่ว่าแต่ละเส้น w = 1 ดังนั้นใช้ bfs หาได้เลย

จริงๆตอนเดินในตารางเขียนแบบนี้ดีกว่า เงื่อนไข continue นอกลูปนี้ลบออก เหลือแค่ if(visited[x][y]) { continue; }

```cpp
vector<pii> direction =
    {
        {1,0},{-1,0},
        {0,1},{0,-1}
    };
```

```cpp
for(int dir =0;dir<4;dir++){
        int newx = x + direction[dir].first; //move row
        int newy = y + direction[dir].second; //move col
        if(newx < 0 || newx >= r || newy < 0 || newy >= c || visited[newx][newy] || v[newx][newy] == 0){
            continue;
        }
        q.push({newx,newy,cost+1});
    }
```

```cpp
//dijkstra
    int vertex,edge,start;
    cin >> vertex >> edge >> start;
    vector<vector<pii>> adjgraph(vertex);
    for(int i=0;i<edge;i++){
        int u,v,w;
        cin >> u >> v >> w;
        adjgraph[u].push_back({v,w});
        adjgraph[v].push_back({u,w});
    }


    vector<int> dist(vertex,INT_MAX);
    vector<int> prev(vertex,-1);
    vector<bool> visited(vertex,false);

    dist[start] = 0; //สมมติว่าโหนดแรกไม่เริ่มที่ 0 (มีให้ใส่ start) ถ้าไม่มี ก็ dist[0] = 0;
    priority_queue<pii,vector<pii>,greater<pii> > pq;
    pq.push({0,start});
    while(!pq.empty()){
        int u = pq.top().second;
        pq.pop();
        if(visited[u]){
            continue;
        }
        visited[u] = true;
        for(auto e : adjgraph[u]){
            int v = e.first;
            int w = e.second;
            if(dist[v] > dist[u] + w){
                dist[v] = dist[u] + w;
                prev[v] = u;
                pq.push({dist[v],v});
            }
        }
    }
```

รับค่าแบบนี้คือการให้เก็บเป็น adj list

```cpp
vector<vector<int> > v(n,vector<int>(n));
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            int w;
            cin >> w;
            v[i][j] = w;
        }
    }
```

รับค่าโดยกราฟนี้คือ directed และเก็บ weight เป็น adj matrix

```cpp
int maxdis = 0;
    for(auto d : distance){
        if(d != INT_MAX){
            maxdis = max(maxdis,d);
        }else{
            return -1;
        }
    }
    return maxdis;
```

ใน distance แต่ละช่องก็จะเก็บระยะทางที่สั้นสุดจาก start ไปยังเมืองต่างๆซึ่ง ถ้าloop หาเมืองไกลสุดก้ max เมืองใกล้สุดก้ min

```cpp
//bfs
    queue<int> q;
    q.push(startnode);
    vector<bool> visitedBFS(vertex,false);
    visitedBFS[startnode] = true;
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(auto e : adjgraph[u]){
            int v = e.first;
            if(!visitedBFS[v]){
                visitedBFS[v] = true;
                q.push(v);
            }
        }
    }

//dfs
    stack<int> s;
    int startnode = 0;
    s.push(startnode);
    vector<bool> visitedDFS(vertex,false)
    visitedDFS[startnode] = true;
    while(!s.empty()){
        int u = s.top();
        s.pop();
        for(auto e : adjgraph[u]){
            int v = e.first;
            if(!visitedDFS[v]){
                visitedDFS[v] = true;
                s.push(v);
            }
        }
    }
```

```cpp
//binarysearch
    int left = 0 ,right = n-1;
    while(left < right){
        int mid = (left+right)/2;
        if(arr[mid] < target){
            left = mid+1;
        }else if(arr[mid] > target){
            right = mid-1;
        }else {
            left = right = mid;
        }
    }
```

ถ้ามัน bug ก็ลองเพิ่มลด mid-1 mid+1 ไรดู ไม่ก้ วาดตารางทดเลย

genpermu basic ใช้ used บอกว่าเคยใช้ไปแล้วยังงัจะเจน แล้วใส่คำต้องด้วย i เพราะไม่ได้เจนเลข 0 1 แต่เป็น 1 2 3....

```cpp
vector<int> c,a;
vector<int> dpjum;
int n,k;

int calc(int n){
    if(n<k){
        return a[n];
    }

    //lookup with vector
    if(dpjum[n] != -1){
        return dpjum[n];
    }

    int sum =0;
    for(int i=1;i<=k;i++){
        sum += calc(n-i) * c[i];
        sum %= mod;
    }

    dpjum[n] = sum;//save subproblem
    return dpjum[n];
}
```

ทำ topdown ก็แค่ทำแบบตรงๆ แต่เดิมการยัดลงตัวmem แล้วมาเปิดดูว่าถ้ามีก็เอาตัวนั้นออกมาเลยไม่เรียกซ้ำ

```cpp
int main(){
    ios_base::sync_with_stdio(false);cin.tie(0);
    cin >> n;

    ll prevprev = 1;
    ll prev = 3;

    ll a_n = 0;
    for(int i=3;i<=n;i++){
        a_n = (prevprev*2 + prev) % mod;
        prevprev = prev;
        prev = a_n;
    }
    cout << a_n;
}
```

ถ้าทำ bottom up แล้วไม่รู้จะจองที่อาเรเท่าไหร่ก็ทำงี้ ได้เลย prev prev บลาๆๆๆ

```cpp
void genPermu(int n,vector<int> &sol,int len,vector<bool> &used , vector<int> &before){
  if(len < n){
    for(int i=0;i<n;i++){
        if(used[i] == false && (before[i] == -1 || used[before[i]])){
            used[i] = true;
            sol[len] = i;
            genPermu(n,sol,len+1,used,before);
            used[i] = false;
        }
    }
  }else{
    for(auto &e : sol){
        cout << e << " ";
    }
    cout << endl;
  }
}
```

```cpp
void combiV2Bynatte (vector<int> &sol,int step,int one){
    if(step == n){
        // int one = 0;
        // for(auto &e : sol) if(e == 1)  one++;
        // if(one != r) return;

        for(auto &e:sol) cout << e;
        cout << endl;
    }else{
        int zero=step-one;
        int zero_quota = n-r;
        if(step - one < n-r){
            sol[step] = 0;
            combiV2Bynatte(sol,step+1,one);
        }
        if(one < r){
            sol[step] = 1;
            combiV2Bynatte(sol,step+1,one+1);
        }
    }
}
```

```cpp
//search time that pig can found truffle equal n
    ll left = 0;
    ll right = INFINITY;
    ll mintime = 0;
    while(left < right){
        ll mid = (left+right)/2;
        ll counttruffle = 0;
        for(int i=0;i<m;i++){
            counttruffle += (mid/pig[i]);
            if(counttruffle >= n){
                break; //prevent overflow
            }
        }
        if(counttruffle >= n){
            mintime = mid;
            right = mid; //ลดขอบขวา
        }else{
            left = mid+1; //เพิ่มขอบซ้าย
        }
    }
    cout << mintime;
```

ทำ bsearch ที่ต้องการหา จำนวน“ของ”อย่างน้อยที่ต้องการที่ เวลาใดๆ จึงใช้ bsearch แล้ว sum ของเพื่อเช็คว่าได้เท่ายัง ถ้ายังก็ลดขอบเขตการค้นเรื่อยๆๆ

```cpp
struct Edge {
    int to, weight;
};
```

```cpp
bool floydWarshallList(int n, vector<vector<Edge>>& graph) {

    vector<vector<int>> dist(n, vector<int>(n, INT_MAX));

    // Initialize distances
    for (int i = 0; i < n; i++) {
        dist[i][i] = 0;
        for (auto& edge : graph[i]) {
            dist[i][edge.to] = min(dist[i][edge.to], edge.weight);
        }
    }
```

ถ้าเป็น adj matrix

```cpp
    // Floyd-Warshall algorithm
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // Check to avoid integer overflow
                if (dist[i][k] != INT_MAX &&
                    dist[k][j] != INT_MAX &&
                    dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    // Check for negative cycles
    for (int i = 0; i < n; i++) {
        if (dist[i][i] < 0) {
            return true; // Negative cycle detected
        }
    }
```

```cpp
bool floydWarshallMatrix(vector<vector<int>>& graph) {
    int n = graph.size();

    // Convert -1 to INT_MAX for unreachable paths
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (graph[i][j] == -1) graph[i][j] = INT_MAX;
            if (i == j) graph[i][j] = 0;
        }
    }
```

ถ้าเป็น adj list

```cpp
    // Floyd-Warshall algorithm
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // Check to avoid integer overflow
                if (graph[i][k] != INT_MAX &&
                    graph[k][j] != INT_MAX &&
                    graph[i][k] + graph[k][j] < graph[i][j]) {
                    graph[i][j] = graph[i][k] + graph[k][j];
                }
            }
        }
    }

    // Check for negative cycles
    for (int i = 0; i < n; i++) {
        if (graph[i][i] < 0) {
            return true; // Negative cycle detected
        }
    }
```