

NTUT_Kn1ghts ICPC Team Notebook

Contents

1 Advanced algorithms

1.1 Iterative deepening A* (IDA*)

2 Dynamic programming algorithms

2.1 0-1 knapsack

2.2 Longest common subsequence (LCS)

2.3 Max 2D range sum

2.4 Traveling salesman problem (TSP)

3 Graph algorithms

3.1 All-pairs shortest paths (APSP)

3.2 Bipartite matching BFS by David

3.3 Centroid decomposition

3.4 Detect negative weight cycle

3.5 DFS

3.6 DFS ICPC 2019 Russia problem E

3.7 Dijkstra by Bill

3.8 Dijkstra by David

3.9 Print Euler tour

3.10 Find articulation points and bridges for undirected graph

3.11 Floyd Warshall by David

3.12 Graph edges property check

3.13 Kruskal by David

3.14 Max flow

3.15 Max cardinality bipartite matching (MCBM)

3.16 Minimum spanning tree (MST)

3.17 Strongly connected component (SCC)

4 Greedy algorithms

4.1 Interval covering

4.2 Longest increasing subsequence (LIS)

4.3 Max 1D range sum

5 Math algorithms

5.1 Chinese remainder theorem

5.2 Extended greatest common divisor (Ext-GCD)

5.3 Greatest common divisor (GCD) and least common multiple (LCM)

5.4 Generate list of prime numbers

5.5 N choose R combination (nCr)

5.6 Stirling's approximation

6 String algorithms

6.1 Knuth–Morris–Pratt algorithm

6.2 Longest palindromic substring

6.3 Minimum edit distance

6.4 Z-algorithm

7 Data structures

7.1 Union-find disjoint sets (UFDS) by David

7.2 Binary indexed/fenwick tree (BIT)

7.3 Rope

7.4 Segment tree

7.5 Union-find disjoint sets (UFDS) by Bill

8 Utilities

8.1 Bit manipulation

8.2 C++ input output

8.3 C++ STL

8.4 Dates

8.5 Prime numbers

8.6 Theorems

1 Advanced algorithms

1.1 Iterative deepening A* (IDA*)

```
// Uva 10181 - 15-Puzzle Problem solved with Iterative Deepening A* (IDA*).
#include <bits/stdc++.h>

using namespace std;

// #define LOCAL
#define N 4 // #rows/columns
#define B 15 // [B]lank tile id
#define PUZZLE (N*N)
#define MAX_STEPS 45 // given by the problem description
#define DIR 4 // 4 [DIR]ections

int dr[DIR] = {0, -1, 0, 1}; // must be right, up, left, down
int dc[DIR] = {1, 0, -1, 0}; // for the XOR operation to work
char dm[] = "RULD"; // [d]irection [m]ove

int p[PUZZLE];
int b_init_pos; // [b]lank [init]ial [pos]ition
int lim; // current [lim]it of the Iterative Deepening Search (IDS)
int pred[MAX_STEPS]; // [pre]viously used [d]irection to go to the current state

bool isViable()
{
    int sum;
    for (int i = 0; i < PUZZLE; ++i)
        for (int j = 0; j < i; ++j)
            if (p[j] > p[i]) ++sum;
    sum += b_init_pos / N + b_init_pos % N;
    sum -= B / N + B % N;
    return sum % 2 == 0;
}

int H()
{
    int h = 0;
    for (int pos = 0; pos < PUZZLE; ++pos) // for all tile 'p[pos]'
    { // compute Manhattan distance to goal state
        if (p[pos] == B) continue;
        h += abs(p[pos] / N - pos / N) // position of 'p[pos]' in goal state is 'p[pos]'
            + abs(p[pos] % N - pos % N); // position of 'p[pos]' in current state is 'pos'
    }
    return h;
}

bool isValid(int r, int c)
{
    return 0 <= r && r < N && 0 <= c && c < N;
}

int Delta_H(int cur_r, int cur_c, int next_r, int next_c)
{
    int val = p[cur_r * N + cur_c]; // [value of the tile being moved into the blank tile position]
    int goal_r = val / N; // position of 'val' in goal state is 'val'
    int goal_c = val % N; // get row & column representation of the position
    return - ( abs(goal_r - cur_r) + abs(goal_c - cur_c) )
        + ( abs(goal_r - next_r) + abs(goal_c - next_c) );
}

bool dfs(int g, int h, int b_pos)
{
    if (g + h > lim) return false;
    if (h == 0) return true; // found a solution!
    int r = b_pos / N;
    int c = b_pos % N;
    for (int d = 0; d < DIR; ++d)
    {
        if (g != 0 && d == (pred[g] ^ 2)) continue; // this direction gets us back to parent state
        int next_r = r + dr[d];
        int next_c = c + dc[d];
        if (!isValid(next_r, next_c)) continue;
        int next_h = h + Delta_H(next_r, next_c, r, c); // O(1)
        int b_next_pos = next_r * N + next_c;
        swap(p[b_pos], p[b_next_pos]);
        pred[g+1] = d;

        if (dfs(g + 1, next_h, b_next_pos)) return true;

        swap(p[b_pos], p[b_next_pos]);
    }
    return false;
}

int ida_star()
{
    {
```

```

int init_h = H();
lim = init_h;
while (lim <= MAX_STEPS)
{
    if ( dfs(0, init_h, b_init_pos) ) return lim;
    ++lim;
}
return -1;
}

void output(int steps)
{
    for (int i = 1; i <= steps; ++i)
        printf("%c", dm[ pred[i] ]);
}

int main()
{
    #ifdef LOCAL
    freopen("in.txt", "r", stdin);
    #endif // LOCAL
    int T;
    scanf("%d", &T);
    while (T--)
    {
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
            {
                int pos = i * N + j;
                scanf("%d", &p[pos]);
                if (p[pos] == 0) p[pos] = B, b_init_pos = pos; // goal state 'p' is 0, 1, 2..14, 15
                else --p[pos]; // blank tile as 15
            }
        if ( !isViable() ) // must-consider condition otherwise TLE
        {
            printf("This puzzle is not solvable.\n");
            continue;
        }
        int ret = ida_star();
        if (ret == -1)
        {
            printf("This puzzle is not solvable.\n");
            continue;
        }
        output(ret), printf("\n");
    }
    return 0;
}

```

```

    if ( j >= WEIGHT ) {
        bag[j][1] = max( bag[j][1] , bag[j][0] );
        bag[j][1] = max( bag[j][1] , bag[j-WEIGHT][0] + VALUE );
    }
}

for(int j = 0 ; j < W ; j++ )
    bag[j][0] = bag[j][1];
}

// mulipte knapsack
int limit[N]; // item limit
void Multiple() {
    for(int i = 0 ; i < N ; i++ ){
        int tmp = 1;
        while( tmp <= weight[i] ){
            for(int j = 0 ; j < W ; j++)
                if( j >= weight[i]+tmp )
                    bag[j][1] = max( bag[j-weight[i]+tmp][0] + value[i]*tmp
                                     , bag[j][0] );
            tmp = tmp*2;
        }
        for(int j = 0 ; j < W ; j++ )
            bag[j][0] = bag[j][1];

        weight[i] = weight[i]-tmp;
        tmp = tmp*2;
    }
    if( weight[i] > 0 ){
        for(int j = 0 ; j < W ; j++)
            if( j >= weight[i]+tmp )
                bag[j][1] = max( bag[j-weight[i]+tmp][0] + value[i]*tmp , bag[j][0] );
        for(int j = 0 ; j < W ; j++ )
            bag[j][0] = bag[j][1];
    }
}

// inf
void Unlimited(){
    memset(bag,0,sizeof(bag));
    for(int i = 0 ; i < N ; i++ ){
        for(int j = 0 ; j < W ; j++ )
            if( j >= weight[i] )
                bag[j][1] = max( bag[j][0] , bag[j-weight[i]][1] + value[i] );
    }
    for(int j = 0 ; j < W ; j++ )
        bag[j][0] = bag[j][1];
}
}

```

2 Dynamic programming algorithms

2.1 0-1 knapsack

```

#define W 1000 // Knapsack weight
#define N 100 // n item
int weight[N]; //item weight
int value[N]; //item value
int bag[W][2];

// 0/1 Knapsack
void ZeroOne(){
    memset(bag,0,sizeof(bag));
    for(int i = 0 ; i < N ; i++ ){
        for(int j = 0 ; j < W ; j++ )
            if( j >= weight[i] )
                bag[j][1] = max( bag[j][0] , bag[j-weight[i]][0] + value[i] );

        for(int j = 0 ; j < W ; j++ )
            bag[j][0] = bag[j][1];
    }
}

// group knapsack
int group; // hou much groups?
int how_many; // one group has many items?
int WEIGHT,VALUE;

void Grouping(){
    memset(bag,0,sizeof(bag));
    for(int i = 0 ; i < group ; i++ ){
        for(int j = 0 ; j < how_many ; j++ ){
            scanf("%d %d",&WEIGHT,&VALUE);

            for(int k = 0 ; k < W ; k++ ){

```

2.2 Longest common subsequence (LCS)

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
using namespace std;

struct LCS{
    int step , max_len ;
}Dp[5000][5000];

int main()
{
    #ifdef LOCAL
    freopen("inl.txt", "r", stdin );
    #endif // LOCAL
    int intX, intY, Min_step , Max_len ;
    string strX, strY ;
    while(cin >> intX >> strX >> intY >> strY ){
        //init
        for(int i = 0 ; i <= intY ; i++){
            Dp[0][i].max_len = 0 ;
            Dp[0][i].step = i ;
        }
        for(int i = 0 ; i <= intX ; i++){
            Dp[i][0].max_len = 0 ;
            Dp[i][0].step = i ;
        }
        Max_len = 0 ;
        Min_step = 0 ;

        //lcs

```

```

for(int i = 1 ; i <= intX ; i++){
    for(int j = 1 ; j <= intY ; j++){
        if(strX[i-1] == strY[j-1]){
            Dp[i][j].max_len = Dp[i-1][j-1].max_len + 1 ;
            Dp[i][j].step = Dp[i-1][j-1].step ;

            //debug
            //cout << strX[i-1] << ' ' << strY[j-1] << ' ' << Dp[i][j].max_len << '\n' ;
            //cout << strX[i-1] << ' ' << strY[j-1] << ' ' << Dp[i][j].step << '\n' ;

        }
        else{
            Dp[i][j].max_len = max(Dp[i-1][j].max_len , Dp[i][j-1].max_len ) ;
            Dp[i][j].step = min( min(Dp[i-1][j-1].step , Dp[i][j-1].step ) , Dp[i-1][j].step )
                        +1 ;
        }
    }
    cout << Dp[intX][intY].step << '\n' ;
}
return 0;
}

```

2.3 Max 2D range sum

```

// Max 2D Range Sum - UVa 108 - solved with DP O(n^4).
// Abridged problem statement: Given an n x n square matrix of integers A where
// each integer ranges from [-127..127], find a sub-matrix of A with the maximum
// sum.
#include <bits/stdc++.h>
using namespace std;
int A[200][200];
int main() {
    int n; scanf("%d", &n); // square matrix size
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) {
            scanf("%d", &A[i][j]);
            if (i > 0) A[i][j] += A[i-1][j]; // add from top
            if (j > 0) A[i][j] += A[i][j-1]; // add from left
            if (i > 0 && j > 0) A[i][j] -= A[i-1][j-1]; // avoid double count
        } // inclusion-exclusion
    int maxSubRect = -127*100*100; // the lowest possible val
    for (int i = 0; i < n; ++i) // start coordinate
        for (int j = 0; j < n; ++j)
            for (int k = i; k < n; ++k)
                for (int l = j; l < n; ++l) { // end coord
                    int subRect = A[k][l]; // from (0, 0) to (k, l)
                    if (i > 0) subRect -= A[i-1][l]; // O(1)
                    if (j > 0) subRect -= A[k][j-1]; // O(1)
                    if (i > 0 && j > 0) subRect += A[i-1][j-1]; // O(1)
                    maxSubRect = max(maxSubRect, subRect); // the answer is here
                }
    printf("%d\n", maxSubRect);
    return 0;
}

```

2.4 Traveling salesman problem (TSP)

```

// This is a solution for UVa 10496 - Collecting Beepers. The problem is a
// variant of the Traveling Salesman Problem (TSP): Given n cities and their
// pairwise distances in the form of a matrix 'dist' of size n * n, compute the
// minimum cost of making a tour that starts from any city s, goes through all
// the other n - 1 cities exactly once, and finally returns to the city s. In
// this case, the salesman is Karel in a 2D world who can only move along the
// x and y axis. The cities are beepers whose coordinates are given, from which
// pairwise distances can be calculated. Algorithm takes time O(2^n * n^2).
// INPUT: The first line is the number of test cases. The first line of each
// test case is world's size (x-size and y-size). Next is the starting position
// of Karel. Next is the number of beepers. Next are the beepers' x- and y-
// coordinates.
// OUTPUT: For each test case, output the minimum distance to move from Karel's
// starting position to each of the beepers and back to the starting position.

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

#define LSOne(S) ((S) & -(S))

const int MAX_n = 11;

int dist[MAX_n][MAX_n], memo[MAX_n][1<<(MAX_n-1)]; // Karel + max 10 beepers

int dp(int u, int mask) {
    if (mask == 0) return dist[u][0]; // mask = free coordinates
    // close the loop
}

```

```

int &ans = memo[u][mask];
if (ans != -1) return ans; // computed before
ans = 2000000000;
int m = mask;
while (m) { // up to O(n)
    int two_pow_v = LSOne(m); // but this is fast
    int v = __builtin_ctz(two_pow_v)+1; // offset v by +1
    ans = min(ans, dist[u][v] + dp(v, mask^two_pow_v)); // keep the min
    m -= two_pow_v;
}
return ans;
}

int main() {
    int TC; scanf("%d", &TC);
    while (TC--) {
        int xsize, ysize; scanf("%d %d", &xsize, &ysize); // these two values are not used
        int x[MAX_n], y[MAX_n];
        scanf("%d %d", &x[0], &y[0]);
        int n; scanf("%d", &n); ++n; // include Karel
        for (int i = 1; i < n; ++i) // Karel is at index 0
            scanf("%d %d", &x[i], &y[i]);
        for (int i = 0; i < n; ++i) // build distance table
            for (int j = i; j < n; ++j)
                dist[i][j] = dist[j][i] = abs(x[i]-x[j]) + abs(y[i]-y[j]); // Manhattan distance
        memset(memo, -1, sizeof memo);
        printf("The shortest path has length %d\n", dp(0, (1<<(n-1))-1)); // DP-TSP
    }
    return 0;
}

```

3 Graph algorithms

3.1 All-pairs shortest paths (APSP)

```

// All-Pairs Shortest Paths (APSP) solved with Floyd Warshall O(V^3).
// inside int main()
// Precondition: AdjMat[i][j] contains the weight of edge (i, j) or INF (1B)
// if there is no such edge ('AdjMat' is a 32-bit signed integer array).
// Let 'p' be 2D parent matrix, where p[i][j] is the last vertex before j on
// a shortest path from i to j, i.e. i -> ... -> p[i][j] -> j.
for (int i = 0; i < V; ++i)
    for (int j = 0; j < V; ++j)
        p[i][j] = i; // initialize the parent matrix
for (int k = 0; k < V; ++k) // remember that loop order is k->i->j
    for (int i = 0; i < V; ++i)
        for (int j = 0; j < V; ++j)
            if (AdjMat[i][k] + AdjMat[k][j] < AdjMat[i][j])
                AdjMat[i][j] = AdjMat[i][k] + AdjMat[k][j];
                p[i][j] = p[k][j];
}

// print shortest paths
void printPath(int i, int j)
{
    if (i != j) printPath(i, p[i][j]);
    printf("%d ", j);
}

```

3.2 Bipartite matching BFS by David

```

#include <iostream>
#include <string>
#include <cstring>
#include <vector>
using namespace std;

int fp[100010], fq[100010];
int vfp[100010], vfq[100010];
int turn = 0;
vector<int> cp[100010], cq[100010];

int BFSBMfp(int n) {
    vfp[n] = turn;
    for (int i = 0; i < cp[n].size(); ++i) {
        if (vfp[cp[n][i]] != turn) {
            vfp[cp[n][i]] = turn;
            if (fq[cp[n][i]] == -1 || BFSBMfp(fq[cp[n][i]])) {
                fp[n] = cp[n][i];
            }
        }
    }
}

```

```

        fq[cp[n][i]] = n ;
        return 1 ;
    }
}
return 0 ;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    int n , p , q , k , x , y ;
    cin >> n ;
    while(n--){
        cin >> p >> q >> k ;
        int MaxnPQ = max(p,q);
        for(int i = 1 ; i <= MaxnPQ ; i++){
            cp[i].clear();
            fp[i] = -1;
            cq[i].clear();
            fq[i] = -1;
        }
        int cnt = 0;
        for(int i = 0 ; i < k ; i++){
            cin >> x >> y ;
            cp[x].push_back(y) ;
            cq[y].push_back(x) ;
            if(fp[x] == -1 && fq[y] == -1 ){
                fp[x] = y ;
                fq[y] = x ;
                cnt++ ;
            }
        }
        for(int i = 1 ; i <= p ; i++){
            if(fp[i] == -1 ){
                turn++;
                if(BFSBMfp(i))
                    cnt++;
            }
        }
        cout << cnt << '\n' ;
    }
    return 0;
}

```

3.3 Centroid decomposition

```

#include<iostream>
#include<bits/stdc++.h>
#define LOCAL
#define MAXN 50005
using namespace std;

int n , k , a , b ;
int ans , cnt ;
int Max[MAXN] , sz[MAXN] , rt ;
int head[MAXN] , dis[MAXN];
bool vis[MAXN] ;
struct node{
    int v , nx ;
}Edge[MAXN*2];

void init(int n ){
    Max[0] = n ;
    ans = cnt = 0 ;
    for(int i = 0 ; i <= n ; i++){
        head[i] = -1 ;
        vis[i] = 0 ;
    }
}

void add(int u , int v){
    Edge[cnt].v = v ;
    Edge[cnt].nx = head[u] ;
    head[u] = cnt++ ;
}

void get_rt(int u , int fa ){
    sz[u] = 1 ; Max[u] = 0 ;
    for(int i = head[u] ; ~i ; i=Edge[i].nx){
        int v = Edge[i].v ;
        if(vis[v] || v == fa ) continue ;
        get_rt(v,u);
        sz[u] += sz[v] ;
    }
}

```

```

        Max[u] = max(Max[u] , sz[v]);
    }
    Max[u] = max(Max[u] , n - sz[u]);
    if(Max[rt] > Max[u])
        rt = u ;
}

void get_dis(int u , int fa , int d){
    for(int i = head[u] ; ~i ; i=Edge[i].nx){
        int v = Edge[i].v ;
        if(vis[v] || v == fa ) continue ;
        dis[++cnt] = d + 1 ;
        get_dis(v,u,dis[cnt]);
    }
}

int get_ans(int u , int d ){
    dis[cnt=1] = d ;
    get_dis(u,0,d) ;
    sort(dis+1 , dis+cnt+1) ;
    int l = 1 , ans = 0 ;

    while(l < cnt && dis[l] + dis[cnt] < k ) l++ ;
    while(l < cnt && dis[l] <= k - dis[l]){
        ans += upper_bound(dis + 1 + 1 , dis + cnt + 1 , k - dis[l]) - lower_bound(dis+1+1 , dis+cnt+1 , k-dis[l]);
        l++ ;
    }
    return ans ;
}

void dfs(int u ){
    vis[u] = 1 ;
    //cout << rt << ' ' << u << '\n' ;
    ans += get_ans(u , 0);
    for(int i = head[u] ; ~i ; i = Edge[i].nx){
        int v = Edge[i].v ;
        if(vis[v]) continue ;
        ans -= get_ans(v , 1) ;
        n = sz[v] , rt = 0 , get_rt(v,u);
        dfs(rt);
    }
}

int main(){
    // #ifdef LOCAL
    // freopen("in1.txt" , "r" , stdin);
    // #endif // LOCAL

    cin >> n >> k ;
    init(n);
    for(int i =1; i < n ; i++){
        cin >> a >> b ;
        add(a,b);
        add(b,a);
    }
    rt = 0 ; get_rt(1,0);
    dfs(rt);
    cout << ans << '\n' ;
}

```

3.4 Detect negative weight cycle

```

// Bellman Ford's O(VE)
vi dist(V, INF); dist[s] = 0;
for (int i = 0; i < V - 1; ++i) // relax all E edges V - 1 times
    for (int u = 0; u < V; ++u) // these two loops = O(E)
        for (int j = 0; j < (int)AL[u].size(); ++j) // [A]djacency [L]ist
            {
                ii vw = AL[u][j];
                dist[vw.first] = min( dist[vw.first], dist[u] + vw.second ) ; // relax
            }
}

```

3.5 DFS

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
using namespace std;
int m , n , flag=1;
int Maxn_city = 0 , Maxn_path = 0 ;

```

```

vector<int> tree[200020] ;
int city[200020] = {} ;
int visit[200020] = {} ;
vector<int> travel ;

void BFS_to_large_path(int root ){
    visit[root] = 1 ;
    travel.push_back(root);
    for(int i = 0 ; i < tree[root].size() ; i++){
        int node = tree[root][i] ;
        if(!visit[node]){
            BFS_to_large_path(node);
            travel.pop_back();
            visit[root] = 0 ;
        }
    }
    //debug to check large path
    //if (root == 1)
    //    cout << "l=" << travel.size() << ' ' << Maxn_path << ' ' << city[root] << '\n' ;

    if(city[root] && travel.size() > Maxn_path){
        Maxn_city = travel[travel.size()/2];
        Maxn_path = travel.size();
    }
}

void BFS_to_other_path(int root ,int path){
    visit[root] = 1 ;
    for(int i = 0 ; i < tree[root].size() ; i++){
        int node = tree[root][i] ;
        if(!visit[node]){
            BFS_to_other_path(node , path+1);
            visit[root] = 0 ;
        }
    }
    //debug
    if(root == 1 )
        cout << "city=" << root << " path= " << path << '\n' ;

    if(city[root] && path != Maxn_path)
        flag = 0 ;
}

int main(){
#ifdef LOCAL
    freopen("in1.txt" , "r" , stdin);
#endif // LOCAL
    cin >> n >> m ;
    int a , b ;
    for(int i = 0 ; i < n-1 ; i++){
        cin >> a >> b ;
        tree[a].push_back(b) ;
        tree[b].push_back(a) ;
    }

    for(int i = 0 ; i < m ; i++){
        cin >> a ;
        city[a] = 1 ;
    }

    BFS_to_large_path(a);
    //visit[a] = 0 ;
    BFS_to_other_path(Maxn_city , 1 );
    if(flag)
        cout << "YES\n" << Maxn_city ;
    else
        cout << "NO" ;

    //debug
    cout << "Maxn_path= " << Maxn_path << " Maxn_city= " << Maxn_city << '\n' ;
}

```

3.6 DFS ICPC 2019 Russia problem E

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
using namespace std;
int m , n , flag=1;
int Maxn_city = 0 , Maxn_path = 0 ;
vector<int> tree[200020] ;
int city[200020] = {} ;
int visit[200020] = {} ;
vector<int> travel ;

void BFS_to_large_path(int root ){
    visit[root] = 1 ;

```

```

    travel.push_back(root);
    for(int i = 0 ; i < tree[root].size() ; i++){
        int node = tree[root][i] ;
        if(!visit[node]){
            BFS_to_large_path(node);
            travel.pop_back();
            visit[root] = 0 ;
        }
    }
    //debug to check large path
    //if (root == 1)
    //    cout << "l=" << travel.size() << ' ' << Maxn_path << ' ' << city[root] << '\n' ;

    if(city[root] && travel.size() > Maxn_path){
        Maxn_city = travel[travel.size()/2];
        Maxn_path = travel.size();
    }
}

void BFS_to_other_path(int root ,int path){
    visit[root] = 1 ;
    for(int i = 0 ; i < tree[root].size() ; i++){
        int node = tree[root][i] ;
        if(!visit[node]){
            BFS_to_other_path(node , path+1);
            visit[root] = 0 ;
        }
    }
    //debug
    if(root == 1 )
        cout << "city=" << root << " path= " << path << '\n' ;

    if(city[root] && path != Maxn_path)
        flag = 0 ;
}

int main(){
#ifdef LOCAL
    freopen("in1.txt" , "r" , stdin);
#endif // LOCAL
    cin >> n >> m ;
    int a , b ;
    for(int i = 0 ; i < n-1 ; i++){
        cin >> a >> b ;
        tree[a].push_back(b) ;
        tree[b].push_back(a) ;
    }

    for(int i = 0 ; i < m ; i++){
        cin >> a ;
        city[a] = 1 ;
    }

    BFS_to_large_path(a);
    //visit[a] = 0 ;
    BFS_to_other_path(Maxn_city , 1 );
    if(flag)
        cout << "YES\n" << Maxn_city ;
    else
        cout << "NO" ;

    //debug
    cout << "Maxn_path= " << Maxn_path << " Maxn_city= " << Maxn_city << '\n' ;
}

```

3.7 Dijkstra by Bill

```

// Dijkstra implementation for negative weight edges O((V + E) log V)
vi dist(V, INF); dist[s] = 0;
priority_queue<ii, vii, greater<ii> > pq;
pq.push( ii(0, s) );
while (!pq.empty())
{
    ii front = pq.top(); pq.pop();
    int d = front.first;
    int u = front.second;
    if (d > dist[u]) continue;
    for (int i = 0; i < (int)AL[u].size(); ++i) // [A]dacency [L]ist
    {
        ii vw = AL[u][i];
        int v = vw.first;
        int w = vw.second;
        if (dist[u] + w < dist[v])
        {
            dist[v] = dist[u] + w;    // relax operation
            pq.push( ii(dist[v], v) );
        }
    }
}

```

```

    }
} // this variant can cause duplicate items in the priority queue

```

3.8 Dijkstra by David

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
#define INF 999999999
using namespace std;
int intMap[1010][1010] = {} , intValue[1010][1010] = {};
int m , n ;

struct Node{
    int x , y , v ;
    void read( int _x , int _y , int _v){
        x = _x ; y = _y ; v = _v ;
    }
    bool operator < (const Node &a) const{
        return v > a.v ;
    }
}nodNode;

void print_map(){
    for(int i = 1 ; i <= n ; i++){
        for(int j = 1 ; j <= m ; j++){
            if(intValue[i][j] == 999999999)
                cout << 'x' << ' ' ;
            else
                cout << intValue[i][j] << ' ' ;
        }
        cout << '\n' ;
    }
    cout << '\n' ;
}

void bfs(){
    int x , y , intDirection[4][2] = {-1,0 , 0,1 , 1,0 , 0,-1};
    int intDx , intDy ;
    Node nodTemp ;
    priority_queue<Node> deqNode ;
    nodTemp.read(1,1,0);
    deqNode.push(nodTemp);
    while(deqNode.size()){
        x = deqNode.top().x ;
        y = deqNode.top().y ;
        deqNode.pop() ;

        for(int i = 0 ; i < 4 ; i++){
            intDx = intDirection[i][0] + x ;
            intDy = intDirection[i][1] + y ;

            //debug
            //cout << intDx << ' ' << intDy << ' ' << intValue[x][y] + intMap[intDx][intDy] << ' ' <<
                i << '\n' ;

            if(intValue[x][y] + intMap[intDx][intDy] < intValue[intDx][intDy] ){
                intValue[intDx][intDy] = intValue[x][y] + intMap[intDx][intDy] ;
                nodTemp.read(intDx , intDy , intValue[intDx][intDy]);
                deqNode.push(nodTemp) ;
            }
        }
        //print_map() ;
    }
}

int main() {
    #ifdef LOCAL
        freopen("in1.txt" , "r" , stdin );
        freopen("out.txt" , "w" , stdout );
    #endif
    ios::sync_with_stdio(false);
    int intCase ;
    cin >> intCase ;
    while(intCase --){
        cin >> n >> m ;
        for(int i = 1 ; i <= n ; i++){
            for(int j = 1 ; j <= m ; j++){
                cin >> intMap[i][j] ;
                intValue[i][j] = INF ;
            }
        }
    }
}

```

```

for(int i = 1 ; i <= n ; i++){
    intValue[i][0] = 0 ;
    intValue[i][m+1] = 0 ;
    intMap[i][0] = INF +1 ;
    intMap[i][m+1] = INF +1 ;
}
for(int i = 1 ; i <= m ; i++){
    intValue[0][i] = 0 ;
    intValue[n+1][i] = 0 ;
    intMap[0][i] = INF +1 ;
    intMap[n+1][i] = INF +1 ;
}
intValue[1][1] = intMap[1][1] ;

//debug
//cout << intValue[1][1] << '\n' ;

bfs();
cout << intValue[n][m] << '\n' ;

return 0;
}

```

3.9 Print Euler tour

```

// Given an Eulerian-tour graph - a connected undirected graph whose vertices a-
// ll have even degrees, produce its Euler tour. The graph is unweighted, stored
// in an adjacency list where the second attribute in edge info pair is a boole-
// an '1' (edge can still be used) or '0' (edge can no longer be used).
list<int> cyc; // we need list for fast insertion in the middle

void EulerTour(list<int>::iterator i , int u)
{
    for (int j = 0; j < (int)AL[u].size(); ++j) // [A]dacency [L]ist
    {
        ii & vw = AL[u][j];
        int v = vw.first;
        if (vw.second) // if this edge can still be used
        {
            vw.second = 0; // remove this edge
            // remove bi-directional edge
            for (int k = 0; k < (int)AL[v].size(); ++k)
            {
                ii & uw = AL[v][k];
                if (uw.first == u && uw.second)
                {
                    uw.second = 0;
                    break;
                }
            }
            // continue the tour
            EulerTour(cyc.insert(i , u) , v);
        }
    }
}

// inside int main()
cyc.clear();
EulerTour(cyc.end(), 0); // 'cyc' contains an Euler tour starting at vertex '0'
for (list<int>::iterator i = cyc.begin(); i != cyc.end(); ++i)
    printf("%d\n", *i);

```

3.10 Find articulation points and bridges for undirected graph

```

// Find articulation points & bridges for undirected graph solved with DFS O(V + E).
void articulationPointAndBridge(int u)
{
    dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u] <= dfs_num[u]
    for (int i = 0; i < (int)AL[u].size(); ++i) // [A]dacency [L]ist
    {
        int v = AL[u][i].first;
        if (dfs_num[v] == UNVISITED) // a tree edge
        {
            dfs_parent[v] = u;
            if (u == dfsRoot) ++rootChildren; // special case if 'u' is a root

            articulationPointAndBridge(v);

            if (dfs_low[v] >= dfs_num[u]) articulation_vertex[u] = true;
            if (dfs_low[v] > dfs_num[u]) printf("Edge (%d, %d) is a bridge\n", u , v);
        }
    }
}

```

```

        dfs_low[u] = min( dfs_low[u], dfs_low[v] );    // update dfs_low[u]
    }
    else if ( v != dfs_parent[u] ) dfs_low[u] = min( dfs_low[u], dfs_num[v] ); // update dfs_low[u]
}

// inside int main()
dfsNumberCounter = 0;
dfs_num.assign(V, UNVISITED);
dfs_low.assign(V, 0);
dfs_parent.assign(V, 0);
articulation_vertex.assign(V, 0);
printf("Bridges:\n");
for (int u = 0; u < V; ++u)
    if (dfs_num[u] == UNVISITED)
    {
        dfsRoot = u;
        rootChildren = 0;
        articulationPointAndBridge(u);
        articulation_vertex[dfsRoot] = (rootChildren > 1);    // special case
    }
printf("Articulation Points:\n");
for (int u = 0; u < V; ++u)
    if (articulation_vertex[u]) printf(" Vertex %d\n", u);

```

3.11 Floyd Warshall by David

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
using namespace std;
char before[520][520] = {} ;
int after[520][520] = {} ;

int main()
{
#ifdef LOCAL
    freopen("in1.txt", "r", stdin);
#endif // LOCAL

    int n ;
    cin >> n ;
    for(int i = 0 ; i < n ; i++){
        for(int j = 0 ; j < n ; j++){
            cin >> before[i][j] ;
        }
    }

    for(int i = 0 ; i < n ; i++){
        for(int j = i+1 ; j < n ; j++){
            int sum = 0 ;
            for(int k = i + 1 ; k < j ; k++){
                if(after[i][k])
                    sum += before[k][j] - '0' ;
            }

            if( (sum + 1) % 10 == before[i][j] - '0'){
                after[i][j] = 1 ;
            }
        }
    }

    for(int i = 0 ; i < n ; i++){
        for(int j = 0; j < n ; j++){
            cout << after[i][j] ;
            cout << '\n' ;
        }
    }

    return 0;
}

```

3.12 Graph edges property check

```

// Graph Edges Property Check solved with DFS O(V + E).
void graphCheck(int u)    // DFS for checking graph edge properties
{
    dfs_num[u] = EXPLORED;
    for (int i = 0; i < (int)AL[u].size; ++i) // [A]djacency [L]ist
    {
        int v = AL[u][i].first;
        if (dfs_num[v] == UNVISITED)    // Tree Edge, EXPLORED->UNVISITED
        {

```

```

            dfs_parent[v] = u;    // parent of this child is me
            graphCheck(v);
        }
        else if (dfs_num[v] == EXPLORED)    // EXPLORED->EXPLORED
        {
            if (v == dfs_parent[u]) printf(" Two ways (%d, %d)-(%d, %d)\n", u, v, v, u);
            else printf(" Back Edge (%d, %d) (Cycle)\n", u, v); // can check if graph is cyclic
        }
        else if (dfs_num[v] == VISITED)    // EXPLORED->VISITED
            printf(" Forward/Cross Edge (%d, %d)\n", u, v);
    }
    dfs_num[u] = VISITED;
}

// inside int main()
dfs_num.assign(V, UNVISITED);
dfs_parent.assign(V, 0);
for (int u = 0; u < V; ++u)
    if (dfs_num[u] == UNVISITED)
        printf("Component %d:\n", ++numComp), graphCheck(u);

```

3.13 Kruskal by David

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
#define ll long long
using namespace std;
int parent[1020] ;

struct edge{
    ll n1 , n2 , w ;
}node[25020];

int compare(edge A , edge B ){
    return A.w < B.w ;
}

int find_root(int a){
    if(a != parent[a] )
        return parent[a] = find_root(parent[a]) ;
    return a ;
}

int main()
{
#ifdef LOCAL
    freopen("in1.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif // LOCAL
    int n , m , p_n1 , p_n2 ; // parent_n1 , parent_n2
    vector<int> hce ; //heavy edge circle
    while(cin >> n >> m && n + m != 0 ){
        for(int i = 0 ; i < m ; i++){
            cin >> node[i].n1 >> node[i].n2 >> node[i].w ;
        }

        for(int i = 0 ; i < n ; i++){
            parent[i] = i ;
            sort(node , node + m , compare ) ;
            hce.clear() ;

            //kruskal
            for(int i = 0 ; i < m ; i++){
                p_n1 = find_root(node[i].n1) ;
                p_n2 = find_root(node[i].n2) ;
                if(p_n1 != p_n2 )
                    parent[p_n2] = p_n1 ;
                else
                    hce.push_back(node[i].w) ;

                //debug
                /*<
                for(int i = 0 ; i < n ; i++){
                    cout << parent[i] << ' ' ;
                    cout << '\n' ;
                }
                */
            }
            sort(hce.begin() , hce.end()) ;
            if(hce.size()){
                for(int i = 0 ; i < hce.size()-1 ; i++){
                    cout << hce[i] << ' ' ;
                    cout << hce[hce.size()-1] ;
                }
            }

```

```

    }
    else
        cout << "forest" ;
        cout << '\n' ;
    }
    return 0;
}

```

3.14 Max flow

```

int res[MAX_V][MAX_V], mf, f, s, t;
vi p; // p stores the BFS spanning tree from s

void augment(int v, int minEdge)
{
    if (v == s) { f = minEdge; return; }
    else if ( p[v] != -1 )
    {
        augment( p[v], min(minEdge, res[ p[v] ][ v ] ) );
        res[ p[v] ][ v ] -= f;
        res[ v ][ p[v] ] += f;
    }
}

// inside int main(): set up 'AL', 'res', 's', and 't' with appropriate values
// remember to add backward edges to 'AL'
mf = 0;
while (true) // O(V * E^2) Edmonds Karp's algorithm
{
    f = 0;
    vi dist(MAX_V, INF); dist[s] = 0;
    queue<int> q; q.push(s);
    p.assign(MAX_V, -1);
    while (!q.empty())
    {
        int u = q.front(); q.pop();
        if (u == t) break; // immediately stop BFS if we already reach sink t
        for (int i = 0; i < (int)AL[u].size(); ++i)
        {
            int v = AL[u][i]; // vector<vi> [A]djacency [L]ist
            if (res[u][v] > 0 && dist[v] == INF)
                dist[v] = dist[u] + 1, q.push(v), p[v] = u;
        }
    }
    augment(t, INF); // find the min edge weight 'f' in this path, if any
    if (f == 0) break; // we cannot send any more flow ('f' = 0), terminate
    mf += f; // we can still send a flow, increase the max flow!
}
printf("%d\n", mf);

```

3.15 Max cardinality bipartite matching (MCBM)

```

// Max Cardinality Bipartite Matching (MCBM) solved with augmenting path algorithm O(VE).
vi match, vis;

int Aug(int l) // return 1 if an augmenting path is found & 0 otherwise
{
    if (vis[l]) return 0;
    vis[l] = 1;
    for (int i = 0; i < (int)AL[l].size(); ++i) // [A]djacency [L]ist
    {
        int r = AL[l][i]; // edge weight not needed -> vector<vi> AL
        if ( match[r] == -1 || Aug(match[r]) )
        {
            match[r] = l;
            return 1; // found 1 matching
        }
    }
    return 0; // no matchings
}

// inside int main()
// build unweighted bipartite graph with directed edge left->right set
// left vertices [0..N-1], right vertices [N..V-1]
int MCBM = 0;
match.assign(V, -1); // V is the number of vertices in bipartite graph
for (int l = 0; l < N; ++l) // N = size of the left set
{
    vis.assign(N, 0); // reset before each recursion
    MCBM += Aug(l);
}
printf("Found %d matchings\n", MCBM);

```

3.16 Minimum spanning tree (MST)

```

// Minimum Spanning Tree (MST) solved with Kruskal O(E log V)
// inside int main()
vector<pair<int, ii>> EdgeList; // (weight, two vertices) of the edge
for (int i = 0; i < E; ++i)
{
    scanf("%d %d %d", &u, &v, &w);
    EdgeList.push_back( make_pair( w, ii(u, v) ) );
}
sort(EdgeList.begin(), EdgeList.end()); // sort by edge weight O(E log E)
int mst_cost = 0;
UnionFind UF(V); // all V are disjoint sets initially
for (int i = 0; i < E; ++i)
{
    pair<int, ii> front = EdgeList[i];
    if (!UF.isSameSet(front.second.first, front.second.second))
    {
        mst_cost += front.first;
        UF.unionSet(front.second.first, front.second.second);
    }
}
printf("MST cost = %d\n", mst_cost);

```

3.17 Strongly connected component (SCC)

```

// Tarjan O(V + E)
vi dfs_num, dfs_low, visited;
int dfsNumberCounter, numSCC;
vi S;

void tarjanSCC(int u)
{
    dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u] <= dfs_num[u]
    S.push_back(u); // stores 'u' in a vector based on order of visitation
    visited[u] = 1;
    for (int i = 0; i < (int)AL[u].size(); ++i) // [A]djacency [L]ist
    {
        int v = AL[u][i].first;
        if (dfs_num[v] == UNVISITED) tarjanSCC(v);
        if (visited[v]) dfs_low[u] = min( dfs_low[u], dfs_low[v] ); // condition for update
    }

    if (dfs_low[u] == dfs_num[u]) // if this is a root (start) of an SCC
    {
        printf("SCC %d:", ++numSCC); // this part is done after recursion
        while (true)
        {
            int v = S.back(); S.pop_back();
            visited[v] = 0;
            printf(" %d", v);
            if (u == v) break;
        }
        printf("\n");
    }
}

// inside int main()
dfs_num.assign(V, UNVISITED);
dfs_low.assign(V, 0);
visited.assign(V, 0);
dfsNumberCounter = numSCC = 0;
for (int u = 0; u < V; ++u)
    if (dfs_num[u] == UNVISITED)
        tarjanSCC(u);

```

4 Greedy algorithms

4.1 Interval covering

```

// This is a solution for UVa 10382 - Watering Grass. The problem is a variant
// of Interval Covering problem, which is solved by O(n) Greedy algorithm.

#include <bits/stdc++.h>

#define pb push_back
#define not_set -1

```



```

using namespace std;

typedef pair<double, double> dd;
typedef vector<dd> vdd;
typedef enum { STOP = 0,
              CONTINUE } status;

int n, l, w;
vdd spinklers;
int answer;
double pivot;

struct sort_compare_t {
    bool operator()(dd a, dd b) const {
        return a.first < b.first || (a.first == b.first && a.second > b.second);
    }
} sort_compare;

void InputSpinklers() {
    for (int i = 0; i < n; i++) {
        double x, r; // must be double otherwise WA.
        scanf("%lf %lf", &x, &r);
        if (w > 2 * r) // ignore spinklers that cannot cover the width of the strip.
        {
            continue;
        }
        if (w == 2 * r) // ignore spinklers that produce no intervals.
        {
            continue;
        }
        double dx = sqrt(r * r - w * w / 4.0);
        spinklers.pb(dd(x - dx, x + dx));
    }
}

status Check(int& j) {
    if (j == not_set) // there is an interval after pivot that cannot be covered.
    {
        return STOP;
    }
    // record j.
    answer++;
    pivot = spinklers[j].second;
    if (pivot >= l) // solution found!
    {
        return STOP;
    }
    j = not_set;
    return CONTINUE;
}

void SolveIntervalCovering() {
    sort(spinklers.begin(), spinklers.end(), sort_compare);
    answer = 0;
    pivot = 0.0;
    int j = not_set;
    int iter = 0;
    while (true) {
        if (iter == spinklers.size()) // iterated through all spinklers/intervals.
        {
            Check(j);
            break;
        }

        if (spinklers[iter].first <= pivot) {
            if (pivot < spinklers[iter].second) // note the next candidate down!
            {
                if (j == not_set || spinklers[iter].second > spinklers[j].second) // note down the
                    // most right candidate.
                {
                    j = iter;
                }
                iter++;
            }
            else // skip intervals that are completely covered by the previously selected ones.
            {
                iter++;
            }
        }
        else // out bound.
        {
            if (Check(j) == STOP) {
                break;
            }
        }
    }

    if (pivot >= l) {
        printf("%d\n", answer);
    }
    else {
        printf("-1\n");
    }
}

```

```

int main() {
    while (scanf("%d %d %d", &n, &l, &w) != EOF) {
        spinklers.clear();
        InputSpinklers();
        SolveIntervalCovering();
    }
}

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

typedef vector<int> vi;

int n;
vi A;

void print_array(const char *s, vi &L, int n) {
    for (int i = 0; i < n; ++i) {
        if (i) printf(" ");
        else printf("%s: [", s);
        printf("%d", L[i]);
    }
    printf("]\n");
}

vi p; // predecessor array

void print_LIS(int i) {
    if (p[i] == -1) { printf("%d", A[i]); return; } // backtracking routine
    print_LIS(p[i]); // base case
    printf(" %d", A[i]); // backtrack
}

int memo[10010]; // old limit: up to 10^4

int LIS(int i) { // O(n^2) overall
    if (i == 0) return 1;
    int &ans = memo[i];
    if (ans != -1) return ans; // was computed before
    ans = 1; // LIS can start anywhere
    for (int j = 0; j < i; ++j) // O(n) here
        if (A[j] < A[i]) // increasing condition
            ans = max(ans, LIS(j)+1); // pick the max
    return ans;
}

int main() {
    // note: A[n-1] must be set as the largest value ("INF")
    // so that all LIS (that can start anywhere) will end at n-1
    srand(time(NULL));
    int n = 10+rand()%11; // [10..20]
    A.assign(n, 0);
    A[n-1] = 99; // set A[n-1] = INF
    for (int i = 0; i < n-1; ++i)
        A[i] = rand()%101-50; // [-50..50]

    n = 12;
    vi sample({-7, 10, 9, 2, 3, 8, 8, 1, 2, 3, 4, 99});
    A = sample;

    printf("n = %d:", n);
    for (int i = 0; i < n; ++i)
        printf(" %d", A[i]);
    printf("\n");

    // early 2000 problems usually accept O(n^2) solution
    memset(memo, -1, sizeof memo);
    printf("LIS length is %d\n", LIS(n-1)); // with O(n^2) DP

    // 2020s problems will likely only accept O(n log k) solution
    // new limit: n can be up to 200K
    int k = 0, lis_end = 0;
    vi L(n, 0), L_id(n, 0);
    p.assign(n, -1);

    for (int i = 0; i < n; ++i) { // O(n)
        int pos = lower_bound(L.begin(), L.begin()+k, A[i]) - L.begin();
        L[pos] = A[i];
        L_id[pos] = i; // greedily overwrite this
        // remember the index too
        p[i] = pos ? L_id[pos-1] : -1; // predecessor info
        if (pos == k) { // can extend LIS?
            k = pos+1; // k = longer LIS by +1
            lis_end = i; // keep best ending i
        }
    }
}

```

```

printf("Considering element A[%d] = %d\n", i, A[i]);
printf("LIS ending at A[%d] is of length %d: ", i, pos+1);
printf("[");
print_LIS(i);
printf("]\n");
printf("L is now", L, k);
printf_array("L is now", L, k);
printf("\n");
}

printf("Final LIS is of length %d: ", k);
print_LIS(lis_end); printf("\n");

assert(LIS(n-1) == k); // both must be identical
return 0;
}

```

4.3 Max 1D range sum

```

// Max 1D Range Sum solved with Jay Kadane O(n).
// inside int main()
int n = 9;
int A[] = { 4, -5, 4, -3, 4, 4, -4, 4, -5 }; // a sample array A
int sum = 0;
int ans = 0; // important, 'ans' must be initialized to 0
for (int i = 0; i < n; ++i)
{
    sum += A[i];
    ans = max(ans, sum);
    if (sum < 0) sum = 0;
}
printf("Max 1D Range Sum = %d\n", ans);

```

5 Math algorithms

5.1 Chinese remainder theorem

```

#include <bits/stdc++.h>
#define qtr ios::sync_with_stdio(0); cin.tie(0);
#define endl '\n'
#define int long long
#define MOD 1000000
using namespace std;

int inv(int a, int m){
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;
    if(m == 1) {
        return 0;
    }
    while(a > 1){
        q = a/m;
        t = m;
        m = a%m, a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }
    if(x1 < 0){
        x1 += m0;
    }
    return x1;
}

int findMinX(vector<int> num, vector<int> rem, int k){
    int prod = 1;
    for(int i = 0; i < k; i++) prod *= num[i];
    int result = 0;
    for(int i = 0; i < k; i++){
        int pp = prod / num[i];
        result += rem[i] * inv(pp, num[i]) * pp;
    }
    return result % prod;
}

int32_t main() { //qtr
    int n = 3;
    vector<int> rem, factor;
    rem.resize(n);
    factor.resize(n);

```

```

for(int i = 0; i < n; i++){
    cin >> factor[i];
}
for(int i = 0; i < n; i++){
    cin >> rem[i];
}
cout << findMinX(factor, rem, n) << endl;
}

```

5.2 Extended greatest common divisor (Ext-GCD)

```

// ax mod b = 1
// ax + by = 1, x=y=0
// a,b Relatively Prime
LL exgcd(LL a, LL b, LL &x, LL &y){
    if(b){
        LL tmd=exgcd(b, a%b, y, x);
        y=-a/b*x;
        return tmd;
    }
    x=1, y=0;
    return a;
}

```

5.3 Greatest common divisor (GCD) and least common multiple (LCM)

```

// or __gcd(a, b) in gcc
int gcd(int a, int b){
    return a%b?gcd(b, a%b):b;
}
int lcm(int a, int b){
    return a*b/gcd(a, b);
}

```

5.4 Generate list of prime numbers

```

// Generate list of prime numbers using Sieve of Eratosthenes.
ll _sieve_size;
bitset<10000010> bs; // [b]it [s]et 10^7 should be enough for most cases
vi primes; // compact list of primes

void sieve(ll upperbound) // create list of primes in [0..upperbound)
{
    _sieve_size = upperbound + 1; // add 1 to include upperbound
    bs.set(); // set all bits to 1
    bs[0] = bs[1] = 0; // exception index 0 and 1
    for (ll i = 2; i <= _sieve_size; ++i)
        if (bs[i])
        {
            // cross out multiples of i starting from i * i!
            for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
            primes.push_back( (int)i );
        }
}

bool isPrime(ll N) // a good enough deterministic prime tester
{
    if (N <= _sieve_size) return bs[N]; // O(1) for small primes
    for (int i = 0; i < (int)primes.size(); ++i)
        if (N % primes[i] == 0) return false;
    return true; // it takes longer if N is a large prime!
} // note: only work for N <= (last prime in vi 'primes')^2

// inside int main()
sieve(10000000); // can go up to 10^7 (need few seconds)
printf("%d\n", isPrime(2147483647)); // 10-digit prime
printf("%d\n", isPrime(136117223861LL)); // not a prime, 104729 * 1299709

```

5.5 N choose R combination (nCr)

```

#define MAXN 100
long long nCr[MAXN+5][MAXN+5];
// nCr[i][j] = \\(C_n^j) * r\\
void build_nCr() {
    for(int i = 1; i < MAXN+5; i++) {
        for(int j = 1; j < MAXN+5; j++) {
            if(i == j)
                nCr[i][j] = 1;
            else if(i > j)
                nCr[i][j] = nCr[i-1][j] * i / (i-j);
        }
    }
}

```

5.6 Stirling's approximation

```

double Stirling(int n){
    return (0.5*log(2.0*acos(-1.0)*n)+n*log(n+0.0)-n)/log(10.0);
} // n! Digits

```

6 String algorithms

6.1 Knuth–Morris–Pratt algorithm

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
#define MAXN 100020
using namespace std;
string strA , strB ;
int b[MAXN] , p[MAXN] ;

void kmp_process(){
    int n = strB.length() , i = 0 , j = -1 ;
    b[0] = -1 ;
    while(i < n){
        while(j >= 0 && strB[i] != strB[j]) j = b[j] ;
        i++; j++;
        b[i] = j ;
    }

    //debug
    // for(int k = 0 ; k <= n ; k++)
    //     cout << b[k] << ' ' ;
    //     cout << '\n' ;
}

int kmp(){
    int n = strA.length() , m = strB.length() , i = 0 , j = 0 ;
    while(i < n){
        while(j >= 0 && strA[i] != strB[j]) j = b[j] ;
        i++; j++;
    }
    return j ;
}

int main()
{
    #ifdef LOCAL
        freopen("in1.txt" , "r" , stdin );
    #endif // LOCAL

    while(cin >> strA){
        strB = strA;
        reverse(strB.begin() , strB.end());
        kmp_process();
        int n = kmp() ;
        cout << strA << strB.substr(n) << '\n' ;
    }
    return 0;
}

```

6.2 Longest palindromic substring

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
#define MAXN 1010
using namespace std;
int dp[MAXN][MAXN] = {} ;
string strA , strB ;
int n , m ;

int lcs(){
    n = strA.length();
    m = strB.length();
    for(int i = 0 ; i <= n ; i++) dp[i][0] = 0 ;
    for(int j = 0 ; j <= m ; j++) dp[0][j] = 0 ;
    for(int i = 1 ; i <= n ; i++){
        for(int j = 1 ; j <= m ; j++){
            if(strA[i-1] == strB[j-1]) dp[i][j] = dp[i-1][j-1]+1 ;
            else dp[i][j] = max(dp[i-1][j] , dp[i][j-1]);
        }
    }
    return dp[n][m] ;
}

int main()
{
    #ifdef LOCAL
        freopen("in1.txt" , "r" , stdin );
    #endif // LOCAL
    int t ;
    cin >> t ;
    cin.ignore();
    while(t--){
        getline(cin, strA);
        strB = strA ;
        reverse(strB.begin() , strB.end());
        cout << lcs() << '\n' ;
    }

    return 0;
}

```

6.3 Minimum edit distance

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
#define MAXN 100
using namespace std;
string strA , strB ;
int dis[MAXN][MAXN] , back_table[MAXN][MAXN] ;
int cnt , m , n ;

void backtracking(int i , int j){
    if(i==0 || j==0){
        while(i > 0){
            cout << cnt++ << " Delete " << i << '\n' ;
            i--;
        }
        while(j > 0){
            cout << cnt++ << " Insert " << i+1 << " , " << strB[j-1] << '\n' ;
            j--;
        }
        return ;
    }

    if(strA[i-1] == strB[j-1])
        backtracking(i-1, j-1);
    else{
        if(dis[i][j] == dis[i-1][j-1]+1){
            cout << cnt++ << " Replace " << i << " , " << strB[j-1] << '\n' ;
            backtracking(i-1, j-1);
        }
        else if(dis[i][j] == dis[i-1][j]+1){
            cout << cnt++ << " Delete " << i << '\n' ;
            backtracking(i-1, j) ;
        }
        else if(dis[i][j] == dis[i][j-1]+1){
            cout << cnt++ << " Insert " << i+1 << " , " << strB[j-1] << '\n' ;
            backtracking(i, j-1);
        }
    }
}

void med(){ //Minimum Edit Distance
    for(int i = 0 ; i <= n ; i++) dis[i][0] = i ;
    for(int j = 0 ; j <= m ; j++) dis[0][j] = j ;
}

```

```

    for(int i = 1 ; i <= n ; i++){
        for(int j = 1 ; j <= m ; j++){
            if(strA[i-1] == strB[j-1]) dis[i][j] = dis[i-1][j-1] ;
            else dis[i][j] = min(dis[i-1][j-1], min(dis[i-1][j] , dis[i][j-1]))+1;
        }
    }

int main()
{
#ifdef LOCAL
    freopen("in1.txt" , "r" , stdin );
    freopen("out.txt" , "w" , stdout);
#endif // LOCAL
    cin.tie(0);
    cout.tie(0);
    ios::sync_with_stdio(false);
    int flag = 0 ;
    while(getline(cin , strA) && getline(cin , strB)){
        n=strA.length() ;
        m=strB.length() ;
        cnt = 1 ;
        med();
        if(flag) cout << '\n' ;
        flag = 1 ;
        cout << dis[n][m] << '\n' ;
        backtracking(n,m);
    }
    return 0;
}

```

6.4 Z-algorithm

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL
#define MAXN 1000020
using namespace std;
int z[MAXN] = {} ;
int x=0 , y=0 , maxn = 0;
string s ;

int main()
{
#ifdef LOCAL
    freopen("in1.txt","r",stdin);
#endif // LOCAL

    cin >> s ;
    for(int i = 1 ; i < s.length() ; i++){
        z[i] = max(0,min(z[i-x] , y - i + 1));
        while(i + z[i] < s.length() && s[z[i]] == s[i+z[i]] ){
            x = i ;
            y = i + z[i] ;
            z[i]++;
        }
    }

    for(int i = 0 ; i < s.length() ; i++){
        if(z[i] == s.length() - i && maxn >= s.length()-i ){
            cout << s.substr(0,z[i]);
            return 0 ;
        }
        maxn = max(maxn , z[i]);
    }
    cout << "Just a legend" ;
    return 0;
}

```

7 Data structures

7.1 Union-find disjoint sets (UFDS) by David

```

#include <iostream>
#include <bits/stdc++.h>
#define LOCAL

using namespace std;

```

```

int intSum[200080] , intParent[200080] , intSet[200080] ;

int find_root(int intA){
    if(intParent[intA] == intA)
        return intA ;
    intParent[intA] = find_root(intParent[intA]) ;
    return intParent[intA] ;
}

int each_debug(int n ){
    for(int i = 1 ; i <= n ; i++){
        cout << i << ' ' << intParent[i] << ' ' << '\n' << intSet[find_root(i)] << ' ' << intSum[find_root(i)] << '\n' ;
    }
    system("Pause") ;
}

int main()
{
#ifdef LOCAL
    freopen("in1.txt","r",stdin);
    freopen("out.txt","w",stdout) ;
#endif // LOCAL

    int n, m , operation , p , q ;
    while(cin >> n >> m){
        for(int i = 1 ; i <= n ; i++){
            intParent[i] = i+n ;
            intParent[i+n] = i+n ;
            intSum[i+n] = 1;
            intSet[i+n] = 1 ;
        }
        while(m--){
            cin >> operation ;
            if(operation == 1 ){
                cin >> p >> q ;
                int intRoot_p , intRoot_q ;
                intRoot_p = find_root(intParent[p]) ;
                intRoot_q = find_root(intParent[q]) ;
                if(intRoot_p != intRoot_q){
                    intParent[intRoot_q] = intRoot_p ;
                    intSum[intRoot_p] += intSum[intRoot_q] ;
                    intSet[intRoot_p] += intSet[intRoot_q] ;
                }
                //debug
                //each_debug(n) ;
            }
            else if (operation == 2 ){
                cin >> p >> q ;
                int intRoot_p , intRoot_q ;
                intRoot_p = find_root(intParent[p]) ;
                intRoot_q = find_root(intParent[q]) ;
                if(intRoot_p != intRoot_q){
                    intParent[p] = intRoot_q ;
                    intSum[intRoot_q] += p ;
                    intSum[intRoot_p] -= p ;
                    intSet[intRoot_q] ++ ;
                    intSet[intRoot_p] -- ;
                }
                //debug
                //each_debug(n) ;
            }
            else if (operation == 3){
                cin >> p ;
                cout << intSet[find_root(p)] << ' ' << intSum[find_root(p)] << '\n' ;
            }
        }
        return 0;
    }
}

```

7.2 Binary indexed/fenwick tree (BIT)

```

#include <iostream>
using namespace std;

#define LOGSZ 17

int tree[(1<<LOGSZ)+1];
int N = (1<<LOGSZ);

// add v to value at x

```

```

void set(int x, int v) {
    while(x <= N) {
        tree[x] += v;
        x += (x & -x);
    }
}

// get cumulative sum up to and including x
int get(int x) {
    int res = 0;
    while(x) {
        res += tree[x];
        x -= (x & -x);
    }
    return res;
}

// get largest value with cumulative sum less than or equal to x;
// for smallest, pass x-1 and add 1 to result
int getind(int x) {
    int idx = 0, mask = N;
    while(mask && idx < N) {
        int t = idx + mask;
        if(x >= tree[t]) {
            idx = t;
            x -= tree[t];
        }
        mask >>= 1;
    }
    return idx;
}

```

7.3 Rope

```

#include <iostream>
#include <bits/stdc++.h>
#include <ext/rope>
#define LOCAL
#define MAXN 50020
using namespace std;
using namespace __gnu_cxx;

int main()
{
    #ifdef LOCAL
        freopen("in1.txt", "r", stdin);
    #endif // LOCAL
    int n, t, a, b, c, d=0;
    int v = 0;
    string strA;
    rope<char> r[MAXN], rtmp;
    cin >> n;
    while(n--){
        cin >> t;

        if(t==1){
            cin >> a;
            cin >> strA;
            a -= d;
            r[++v] = r[v];
            r[v].insert(a, strA.c_str());
            //debug
            //cout << r[v] << '\n';
        }
        else if(t==2){
            cin >> a >> b;
            a -= d; b -= d;
            r[++v] = r[v];
            r[v].erase(a-1, b);
            //debug
            //cout << r[v] << ' ' << r[v-1] << '\n';
        }
        else if(t==3){
            cin >> a >> b >> c;
            a -= d; b -= d; c -= d;
            rtmp = r[a].substr(b-1, c);
            cout << rtmp << '\n';
            d += count(rtmp.begin(), rtmp.end(), 'c');
        }
    }
    return 0;
}

```

7.4 Segment tree

```

#include <iostream>
#include <bits/stdc++.h>
#include <string>
#define LOCAL
#define Lson(x) ((x << 1) + 1)
#define Rson(x) ((x << 1) + 2)
#define INF 99999999
using namespace std;
const int N = 100005;
int shift[35], num[N], len_shift;
string strLine;

struct Node{
    int left, right, Min_Value;
}node[4 * N];

void build(int left, int right, int x = 0){
    node[x].left = left;
    node[x].right = right;
    if(left == right){
        node[x].Min_Value = num[left];
        return;
    }
    int mid = (left + right) / 2;

    //debug
    //cout << mid << '\n';
    //cout << x << ' ' << node[x].left << ' ' << node[x].right << ' ' << '\n';

    build(left, mid, Lson(x));
    build(mid + 1, right, Rson(x));
    node[x].Min_Value = min(node[Lson(x)].Min_Value, node[Rson(x)].Min_Value);
}

void handle(){
    len_shift = 0;
    shift[len_shift] = 0;
    for(int i = 6; i < strLine.length(); i++){
        if(strLine[i] >= '0' && strLine[i] <= '9'){
            shift[len_shift] = shift[len_shift] * 10 + (int)(strLine[i] - '0');
        }
        else{
            shift[++len_shift] = 0;
        }
    }
    //finally char is ')', so len_shift is right
    sort(shift, shift + len_shift);

    //debug
    //++<
    for(int i = 0; i < len_shift; i++)
        cout << shift[i] << ' ';
    cout << '\n';
    /*
}

int query(int left, int right, int x = 0){
    if(node[x].left >= left && node[x].right <= right)
        return node[x].Min_Value;
    int mid = (node[x].left + node[x].right) / 2;
    int ans = INF;

    //debug
    //cout << x << ' ' << node[x].left << ' ' << node[x].right << ' ' << node[x].Min_Value << '\n';

    if(left <= mid)
        ans = min(ans, query(left, right, Lson(x)));
    if(mid < right)
        ans = min(ans, query(left, right, Rson(x)));
    return ans;
}

void set_num(int position, int value, int x = 0){
    if(node[x].left == position && node[x].right == position){
        node[x].Min_Value = value;
        return;
    }
    int mid = (node[x].left + node[x].right) / 2;
    if(position <= mid)
        set_num(position, value, Lson(x));
    if(mid < position)

```

```

        set_num(position , value , Rson(x)) ;
        node[x].Min_Value = min(node[Lson(x)].Min_Value , node[Rson(x)].Min_Value );
    }

int main()
{
    int n , q , intTemp ;
    ios::sync_with_stdio(0);
#ifdef LOCAL
    freopen("out.txt" , "w" , stdout ) ;
    freopen("in1.txt" , "r" , stdin ) ;
#endif // LOCAL
    cin >> n >> q ;
    for(int i = 1 ; i <= n ; i++)
        cin >> num[i] ;
    build(1,n);

    //debug
    /**<
    for(int i = 0 ; i < 13 ; i++){
        cout << node[i].left << ' ' << node[i].right << ' ' << node[i].Min_Value << '\n' ;
    }
    return 0 ;
    */

    while(q--){
        cin >> strLine ;
        if(strLine[0] == 'q'){
            handle();
            cout << query(shift[0] , shift[1] ) << '\n' ;
        }
        else if (strLine[0] == 's'){
            handle();
            intTemp = num[shift[0]] ;

            for(int i = 1 ; i < len_shift ; i++){
                set_num(shift[i-1] , num[shift[i]]) ;
                num[shift[i-1]] = num[shift[i]] ;
            }
            num[shift[len_shift-1]] = intTemp ;
            set_num(shift[len_shift-1] , intTemp) ;

            //debug
            //cout << intTemp << ' ' << shift[len_shift-1] << '\n' ;
            //for(int i = 1 ; i <= n ; i++)
            //    cout << num[i] << ' ' ;
        }
    }
    return 0;
}

```

7.5 Union-find disjoint sets (UFDS) by Bill

```

class UnionFind
{
public:
    UnionFind(int N)
    {
        rank.assign(N, 0);
        p.assign(N, 0);
        for (int i = 0; i < N; ++i) p[i] = i;
    }
    int findSet(int i) { return (p[i] == i) ? i : ( p[i] = findSet(p[i]) ); }
    bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
    void unionSet(int i, int j)
    {
        if ( !isSameSet(i, j) )
        {
            int x = findSet(i);
            int y = findSet(j);
            if (rank[x] > rank[y]) p[y] = x;    // rank keeps the tree short
            else
            {
                p[x] = y;
                if (rank[x] == rank[y]) ++rank[y];
            }
        }
    }
private:
    vi p, rank;
};

```

8 Utilities

8.1 Bit manipulation

```

#define isOn(S, j) (S & (1<<j))
#define setBit(S, j) (S |= (1<<j))
#define clearBit(S, j) (S &= ~(1<<j))
#define toggleBit(S, j) (S ^= (1<<j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1<<n)-1)

```

8.2 C++ input output

```

#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    // Ouput a specific number of digits past the decimal point,
    // in this case 5
    cout.setf(ios::fixed); cout << setprecision(5);
    cout << 100.0/7.0 << endl;
    cout.unsetf(ios::fixed);

    // Output the decimal point and trailing zeros
    cout.setf(ios::showpoint);
    cout << 100.0 << endl;
    cout.unsetf(ios::showpoint);

    // Output a '+' before positive values
    cout.setf(ios::showpos);
    cout << 100 << " " << -100 << endl;
    cout.unsetf(ios::showpos);

    // Output numerical values in hexadecimal
    cout << hex << 100 << " " << 1000 << " " << 10000 << dec << endl;
}

```

8.3 C++ STL

```

// Example for using stringstream and next_permutation

#include <algorithm>
#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

int main(void) {
    vector<int> v;

    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);

    // Expected output: 1 2 3 4
    //                  1 2 4 3
    //                  ..
    //                  4 3 2 1
    do {
        stringstream oss;
        oss << v[0] << " " << v[1] << " " << v[2] << " " << v[3];

        // for input from a string s,
        // istringstream iss(s);
        //  iss >> variable;

        cout << oss.str() << endl;
    } while (next_permutation (v.begin(), v.end()));

    v.clear();

    v.push_back(1);
}

```

```

v.push_back(2);
v.push_back(1);
v.push_back(3);

// To use unique, first sort numbers. Then call
// unique to place all the unique elements at the beginning
// of the vector, and then use erase to remove the duplicate
// elements.

sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end());

// Expected output: 1 2 3
for (size_t i = 0; i < v.size(); i++)
    cout << v[i] << " ";
cout << endl;
}

```

8.4 Dates

```

// Routines for performing computations on dates. In these routines,
// months are expressed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

#include <iostream>
#include <string>

using namespace std;

string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y){
    return
        1461 * (y + 4800 + (m - 14) / 12) / 4 +
        367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
        3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
        d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year
void intToDate (int jd, int &m, int &d, int &y){
    int x, n, i, j;

    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string intToDay (int jd){
    return dayOfWeek[jd % 7];
}

int main (int argc, char **argv){
    int jd = dateToInt (3, 24, 2004);
    int m, d, y;
    intToDate (jd, m, d, y);
    string day = intToDay (jd);

    // expected output:
    // 2453089
    // 3/24/2004
    // Wed
    cout << jd << endl;
    << m << "/" << d << "/" << y << endl;
    << day << endl;
}

```

8.5 Prime numbers

```

// O(sqrt(x)) Exhaustive Primality Test
#include <cmath>
#define EPS 1e-7
typedef long long LL;

```

```

bool IsPrimeSlow (LL x)
{
    if(x<=1) return false;
    if(x<=3) return true;
    if (!(x%2) || !(x%3)) return false;
    LL s=(LL)(sqrt((double)(x))+EPS);
    for(LL i=5;i<=s;i+=6)
    {
        if (!(x%i) || !(x%(i+2))) return false;
    }
    return true;
}

// Primes less than 1000:
//      2      3      5      7      11     13     17     19     23     29     31     37
//     41     43     47     53     59     61     67     71     73     79     83     89
//     97    101    103    107    109    113    127    131    137    139    149    151
//    157    163    167    173    179    181    191    193    197    199    211    223
//    227    229    233    239    241    251    257    263    269    271    277    281
//    283    293    307    311    313    317    331    337    347    349    353    359
//    367    373    379    383    389    397    401    409    419    421    431    433
//    439    443    449    457    461    463    467    479    487    491    499    503
//    509    521    523    541    547    557    563    569    571    577    587    593
//    599    601    607    613    617    619    631    641    643    647    653    659
//    661    673    677    683    691    701    709    719    727    733    739    743
//    751    757    761    769    773    787    797    809    811    821    823    827
//    829    839    853    857    859    863    877    881    883    887    907    911
//    919    929    937    941    947    953    967    971    977    983    991    997

// Other primes:
// The largest prime smaller than 10 is 7.
// The largest prime smaller than 100 is 97.
// The largest prime smaller than 1000 is 997.
// The largest prime smaller than 10000 is 9973.
// The largest prime smaller than 100000 is 99991.
// The largest prime smaller than 1000000 is 999983.
// The largest prime smaller than 10000000 is 9999991.
// The largest prime smaller than 100000000 is 99999989.
// The largest prime smaller than 1000000000 is 999999937.
// The largest prime smaller than 10000000000 is 9999999967.
// The largest prime smaller than 100000000000 is 99999999977.
// The largest prime smaller than 1000000000000 is 999999999989.
// The largest prime smaller than 10000000000000 is 9999999999971.
// The largest prime smaller than 100000000000000 is 99999999999973.
// The largest prime smaller than 1000000000000000 is 99999999999989.
// The largest prime smaller than 10000000000000000 is 999999999999937.
// The largest prime smaller than 100000000000000000 is 999999999999997.
// The largest prime smaller than 1000000000000000000 is 9999999999999989.

```

8.6 Theorems

Euler path/tour theorems: An Euler path is a path that visits every edge exactly once. An Euler tour is an Euler path that starts and ends at the same vertex. A graph is an Eulerian-tour graph (i.e. it has an Euler tour) iff all of its vertices has even degrees. A graph is an Eulerian-path graph (i.e. it has an Euler path) iff all but 2 of its vertices has even degrees.

Euler's handshaking lemma: A graph does not have an Euler tour iff it has an even number of vertices of odd degrees.

Two conjunctive normal form (2-CNF) theorem: Given the implication graph of a 2-CNF, the formula is satisfiable iff there is no variable that belongs to the same strongly connected component (SCC) as its negation.

Bipartite graph related theorems:

- (1) Min vertex cover (MVC) = Max cardinality bipartite matching (MCBM).
- (2) Max independent set (MIS) = $V - \text{MVC}$.
- (3) The number of spanning tree of a complete bipartite graph $K(n,m)$ is $m^{n-1} \cdot n^{m-1}$.

Cayley's formula: There are n^{n-2} spanning trees of a complete graph with n labeled vertices.

Derangement: A permutation of the elements of a set such that none of the elements appear in their original position. The number of derangements 'der(n)' can be computed as follow: $\text{der}(n) = (n-1) \cdot (\text{der}(n-1) + \text{der}(n-2))$ where $\text{der}(0) = 1$ and $\text{der}(1) = 0$.

Erds Gallai's theorem: A necessary and sufficient condition for a finite sequence of natural numbers is the degree sequence of a simple graph. A sequence of non-negative integers $d_1 \geq d_2 \geq \dots \geq d_n$ can be the degree sequence of a simple graph on n vertices iff

- (1) $\sum_{i=1}^n d_i$ is even, and
- (2) $\sum_{i=1}^k d_i \leq k \cdot (k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for $1 \leq k \leq n$.