Cqo

Cây khung nhỏ nhất

```
/*input
4 4
1 2 1
2 3 2
3 4 3
4 1 4
*/
#include "bits/stdc++.h"
using namespace std;
#define fi first
#define se second

const int N = 1e5 + 5;
const int INF = 1e9;

// khai báo đồ thị. g[u] chứa các cạnh nối với đỉnh u. Các cạnh sẽ được lưu dưới dạng
pair<v,c>
int n, m;
vector <pair<int, int>> g[N];

int dis[N]; // mảng d lưu khoảng cách của toàn bộ đỉnh

int prim(int s) { // thuật toán Prim bắt đầu chạy từ đỉnh nguồn s
    int ret = 0;
    // Sử dụng priority_queue lưu khoảng cách của các đỉnh tăng dần đối với cây khung
    // Vì priority_queue.top luôn là phần tử lớn nhất, ta sẽ phải sử dụng
greater<pair<int,int>>
    // để priority_queue.top là phần tử nhỏ nhất
    // các phần tử lưu trong priority queue sẽ có dạng pair<dis[u],u>
    priority_queue<pair<int, int>, vector<pair<int,int>>, greater<pair<int,int>>> q;

    // khởi tạo khoảng cách của các đỉnh là vô cùng lớn
    for (int i = 1; i <= n; i++) dis[i] = INF;

    // khởi tạo đỉnh nguồn có khoảng cách là 0 và push đỉnh này vào
    dis[s] = 0;
    q.push({0, s});

    while (!q.empty()) {
        // lấy đỉnh có khoảng cách nhỏ nhất chưa được kết nạp
        auto top = q.top(); q.pop();
        int curDis = top.fi; int u = top.se;

        if (curDis != dis[u]) continue;

        // kết nạp đỉnh u vào cây khung
```

```
            ret += dis[u]; dis[u] = -INF;

            // cập nhất khoảng cách cho các đỉnh kề u
            for (auto &e : g[u]) {
                int v = e.fi; int c = e.se;
                if (dis[v] > c) {
                    dis[v] = c;
                    q.push({ dis[v], v});
                }
            }
        }
        return ret;
    }
    int main() {
        ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

        cin >> n >> m;

        for (int i = 1; i <= m; i++) {
            int u, v, c;
            cin >> u >> v >> c;
            g[u].push_back({v, c});
            g[v].push_back({u, c});
        }

        cout << prim(1) << '\n';
    }
```

Phân tích thừa số nguyên tố

1. Giải thuật $O(\sqrt{(N)})$

```
vector < int > extract(int N)
{
    vector < int > prime_factor;

    for (int i = 2; i * i <= N; ++i)
        while (N % i == 0)
        {
            prime_factor.push_back(i);
            N /= i;
        }

    if (N > 1)
        prime_factor.push_back(i);

    return prime_factor;
}
```

2. Phân tích thừa số nguyên tố bằng sàng Eratosthenes:
   Từ hai giải thuật trên ta thấy: Ở mỗi bước phân tích cần tìm ra ước nguyên tố nhỏ nhất của N rồi chia N cho ước đó. Ta sẽ thay đổi sàng Eratosthenes đi một chút để lấy được ngay ước nguyên tố nhỏ nhất của N trong $O(1)$ ở mỗi bước phân tích, điều này sẽ giúp giảm thời gian chạy đi đáng kể.

Cài đặt

```
void eratosthenes_sieve(int N)
{
    fill(smallest_divisor + 1, smallest_divisor + 1 + N, 0); // Ước nguyên tố nhỏ r
    fill(is_prime, is_prime + 1 + N, true);
    is_prime[0] = is_prime[1] = false;

    for (int i = 2; i * i <= N; ++i)
        if (is_prime[i]) // Nếu i là số nguyên tố
            for (int j = i * i; j <= N; j += i)
            {
                is_prime[j] = false;

                if (smallest_divisor[j] == 0)
                    smallest_divisor[j] = i; // Ước nguyên tố nhỏ nhất của j là i.
            }

    // Xét riêng các trường hợp i là số nguyên tố, ước nguyên tố nhỏ nhất là chính
    for (int i = 2; i <= N; ++i)
        if (is_prime[i])
            smallest_divisor[i] = i;
}

vector < int > extract(int N)
{
    eratosthenes_sieve(maxN); // Sàng số nguyên tố tới một giá trị nào đó.

    vector < int > prime_factor;

    while (N > 1)
    {
        int p = smallest_divisor[N];
        prime_factor.push_back(p);
        N /= p;
    }

    return prime_factor;
}

```

Mặc dù việc thực hiện sàng Eratosthenes vẫn mất $O\big(N.\log(N)\big), O(N.log(N))$, tuy nhiên thao tác phân tích một số PP thành thừa số nguyên tố chỉ mất độ phức tạp $O(logP)$. Điều này sẽ rất có lợi trong các bài toán phải phân tích thừa số nguyên tố nhiều lần.

Đếm ước của 1 số nguyên dương

```
int count_divisors(int N)
{
    int total_divisor = 1; // Chắc chắn N có ước là 1.
    for (int i = 2; i * i <= N; ++i)
    {
        int cnt = 0; // Đếm số lượng thừa số nguyên tố i trong phân tích của N.
        while (N % i == 0)
        {
            ++cnt;
            N /= i;
        }

        total_divisors *= (cnt + 1);
    }

    if (N > 1)
        total_divisors *= 2;

    return total_divisors;
}
```

```
long long exponentiation(long long A, long long B)
{
    if (B == 0)
        return 1LL;

    long long half = exponentiation(A, B / 2LL);

    if (B & 1)
        return half * half * A;
    else
        return half * half;
}

long long get_sum_divisors(long long N)
{
    if (N == 1)
        return 1;

    long long x = 1, y = 1;
    for (int i = 2; i * i <= N; ++i)
    {
        long long cnt = 0; // Đếm số lượng thừa số nguyên tố i trong phân tích của N.
        while (N % i == 0)
        {
            ++cnt;
            N /= i;
        }

        if (cnt != 0)
        {
            x *= (exponentiation(i, cnt + 1) - 1);
            y *= (i - 1);
        }
    }

    if (N > 1)
        x *= N * N - 1, y *= (N - 1);

    return x / y;
}
```

Quy hoạch động

Cây Khế

Truyện cổ tích Cây khế là một câu chuyện rất quen thuộc với tuổi thơ của mỗi người. Trong chuyện, chúng ta đã biết người em được chim thần yêu cầu mang theo túi ba gang để lấy vàng trả cho những quả khế. Tuy nhiên, chiếc túi của người em chỉ có trọng lượng tối đa là mm. Trên hòn đảo chứa vàng có nn thỏi vàng lớn, mỗi thỏi vàng có giá trị là aa và khối lượng là bb. Vì số lượng vàng quá nhiều nên người em không biết phải lấy những thỏi vàng nào để có thể bán được nhiều tiền nhất.

Yêu cầu: Hãy giúp người em chọn ra những thỏi vàng sao cho tổng khối lượng của chúng không vượt quá tải trọng mm của chiếc túi, và tổng giá trị của chúng là lớn nhất?

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define endl "\n"
const int ll = 1e7, oo = 1e9;
int dp[1001][1001], n, m;
vector<pair<int , int>> v;
void enter()
{
    cin >> n >> m;
    v.push_back({oo,oo});
    for(int  i = 1; i <=n; i++)
    {
        int a, b;
        cin >> a >> b;
        v.push_back({b, a});
    }

}
void build()
{
    for(int i = 1; i <= n; i++)
    {
        for(int j = 0; j <= m; j++)
        {
            dp[i][j] = dp[i - 1][j];
            if(j >= v[i].first)
                dp[i][j] = max(dp[i][j], dp[i - 1][j - v[i].first] + v[i].second);
        }
    }
    cout << dp[n][m];
}
main()
{
    enter();
    build();
}
```

Đếm Cách Trả Tiền

Một cửa hàng có nn đồng xu mệnh giá lần lượt là $c_1, c_2, \ldots, c_n$; số lượng mỗi loại coi như không giới hạn. Cửa hàng cần phải trả lại cho một vị khách số tiền đúng bằng m. Tuy nhiên, do trong ngày còn rất nhiều khách hàng, nên chủ cửa hàng cần tính toán số cách trả tiền có thể để lựa chọn một cách phù hợp nhất.

Yêu cầu: Hãy giúp chủ cửa hàng đếm số lượng cách trả tiền cho vị khách nói trên số tiền đúng bằng m? Biết rằng, hai cách trả tiền là hoán vị của nhau chỉ tính là một cách.

Mô tả đầu vào

- Dòng đầu tiên chứa hai số nguyên dương n, m
- Dòng thứ hai chứa nn số nguyên dương $c_1, c_2, \ldots, c_n$
  Các số trên cùng một dòng phân tách nhau bởi dấu cách.

```
#include <bits/stdc++.h>
using namespace std;
#define endl "\n"
const int ll = 1e6, oo = 1e9, mod = 1e9 + 7;
int dp[101][ll + 1],t[ll], n, m;
void enter()
{
    cin >> n >> m;
    for(int  i = 1; i <=n; i++)
    {
        cin >> t[i];
    }
}

void build()
{
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++){
        for (int j = 0; j <= m; j++)
        {
            dp[i][j] = dp[i - 1][j];
            if (j >= t[i])
                dp[i][j] = (dp[i][j] + dp[i][j - t[i]]) % mod;
        }
    }

    cout << dp[n][m];
}
main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    enter();
    build();

}
```

Xâu Con Chung

Cho hai xâu kí tự A và B. Một xâu con C được gọi là xâu con chung của hai xâu A và B nếu như có thể xóa đi một số kí tự của A và B, giữ nguyên thứ tự các kí tự còn lại thì thu được C.

Yêu cầu: Hãy tìm xâu con chung dài nhất của hai xâu A và B?

Mô tả đầu vào

- Gồm hai dòng chứa hai xâu A và B chỉ bao gồm các kí tự latin in thường.

```
#include<bits/stdc++.h>
#define float double
using namespace std;
string a, b;
int dp[1001][1001];
int ma, mb;
void enter()
{
    cin >> a;
    cin >> b;
    ma = a.length();
    mb = b.length();
}
void build()
{
    int res = 0;
    for(int i = 1; i <= ma; i++)
    {
        for(int j = 1; j <= mb; j++)
        {
            if(a[i - 1] == b[j - 1 ]) dp[i][j] = dp[i - 1][ j - 1] + 1;
            else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            res = max(dp[i][j], res);
        }
    }
    cout << res;
}
main()
{
    enter();
    build();
}
```

Dãy Con Chia Hết Cho K

Cho một dãy số nguyên dương gồm nn phần tử $a_1, a_2, \ldots$,Một dãy con của dãy đã cho là một cách chọn ra một số phần tử trong dãy và giữ nguyên thứ tự ban đầu của chúng.

Yêu cầu: Với số nguyên dương K cho trước, hãy xác định dãy con gồm nhiều phần tử nhất của dãy ban đầu sao cho tổng của chúng chia hết cho k?

Mô tả đầu vào

- Dòng đầu tiên chứa hai số nguyên dương n và k
  Dòng tiếp theo chứa n số nguyên dương $a_1, a_2, \ldots$ phân tách nhau bởi dấu cách.

Mô tả đầu ra

- Đưa ra số nguyên duy nhất là số lượng phần tử trong dãy số được chọn. Nếu không tìm được dãy con nào thì in ra -1.

```
#include<bits/stdc++.h>
using namespace std;
const int ll = 1e3 + 1, oo = 1e9;
int n, k, a[ll], dp[ll][ll];

void enter()
{
    cin >> n >> k;
    for(int i = 1; i <= n; i++)
        cin >> a[i];
}
void build()
{
    int res = oo;
    for(int i = 0; i <= n; i++)
        for(int j = 0; j <= k; j++)
            dp[i][j] = oo;
    dp[0][0] = 1;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 0; j <= k - 1; j ++)
        {
            dp[i][j] = max(dp[i - 1][j], abs(dp[i - 1][j - a[i] % k]) + 1);
            res = min(dp[i][j], res);
        }
    }
    cout << n - res;
}

main()
{
    enter();
    build();
}
```

Trò Chơi 1

Thầy giáo của Hanna Truong vừa nghĩ ra một trò chơi cho cả lớp trong giờ học để giải tỏa căng thẳng. Thầy vẽ lên bảng một ma trận $m \times n$, các dòng đánh số từ 1 tới m theo chiều từ trên xuống dưới, các cột đánh số từ 1 tới n theo chiều từ trái sang phải. Ô nằm ở vị trí giao giữa hàng i và cột j gọi là ô (i, j). Trên mỗi ô chứa một số nguyên.

Người chơi bắt đầu xuất phát ở ô $(1, 1)$ và chỉ được phép di chuyển sang ô bên phải hoặc ô bên dưới, và phải tìm cách di chuyển sao cho khi tới ô (m, n) thì tổng số điểm đạt được trên các ô đã đi qua là lớn nhất.

Yêu cầu: Hanna muốn có cách nào đó để luôn chiến thắng trò chơi. Hãy giúp cô bé tìm cách đi trên bảng số để thu được số điểm lớn nhất?

Mô tả đầu vào

- Dòng đầu tiên chứa hai số nguyên dương m, n - kích thước bảng số.
- m dòng tiếp theo, mỗi dòng chứa n số nguyên $a_{i,j}$ $(1 \leq i \leq m, 1 \leq j \leq n)$ thể hiện một hàng của bảng số.

```
void enter()
{
    cin >> m >> n;
    for(int i = 1; i <= m; i ++)
        for(int j = 1; j <= n; j++)
            cin >> a[i][j];
}
void build()
{
    for(int i = 1; i <= m; i++)
        for(int j = 1; j <= n; j++)
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + a[i][j];
    cout << dp[m][n];
}

main()
{
    enter();
    build();
}
```

Mang tong tien to 2 chieu

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

using namespace std;

long long query(int x1, int y1, int x2, int y2,
                vector < vector < long long > > &sum)
{
    return sum[x2][y2] - sum[x1 - 1][y2] - sum[x2][y1 - 1] + sum[x1 - 1][y1 - 1];
}

main()
{
    int m, n, q;
    cin >> m >> n >> q;

    vector < vector < long long > > sum(m + 1, vector < long long >(n + 1, 0));
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j)
        {
            int x;
            cin >> x;

            sum[i][j] = sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1] + x;
        }

    while (q--)
    {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;

        cout << query(x1, y1, x2, y2, sum) << endl;
    }
}
```

Stack

Xóa Chữ Số

Yêu cầu: Bạn Minh là một khách hàng mới của hãng. Hãy giúp bạn Minh xóa đi k chữ số của số nguyên dương n sao cho thu được số lớn nhất (tất nhiên là ai cũng muốn thu được số tiền lớn nhất).

Mô tả đầu vào

- Dòng thứ nhất chứa số nguyên dương n.
- Dòng thứ hai là số nguyên dương k – số chữ số cần xóa.

Mô tả đầu ra

- Đưa ra số lớn nhất nhận được sau khi xóa đi k chữ số của n.

```
n = input()
k = int(input())
n = str(n)
st = []
for i in range(len(n)):
    while len(st) != 0 and st[-1] > n[i] and k > 0:
        st.pop();
        k -= 1
    st.append(n[i])
print(''.join(st))
```

BFS

ĐẾM ĐẢO

```
#include <bits/stdc++.h>
using namespace std;
#define f first
#define s second
int m, n, a[105][105], ans = 0;
const int step[4][2] = {{0,1},{0,-1},{-1,0},{1,0}};
void enter(){
    cin >> m >> n;
    for(int i = 1; i <= m; i++){
        for(int j = 1; j <= n; j++){
            cin >> a[i][j];
        }
    }
}
bool check(int x, int y, vector<vector<int>> &visisted){
    if(x >= 1 and x <= m and y >= 1 and y <= n and visisted[x][y] == 0 and a[x][y] ==
1){
        return true;
    }
    else return false;
}
void bfs(int x, int y, vector<vector<int>> &visisted){
    queue<pair<int, int>> qu;
    qu.push({x, y});
    visisted[x][y] = 1;
    while(!qu.empty()){
        pair<int, int> cur = qu.front();
        qu.pop();
        for(int i = 0; i < 4; i++){
            int next_x = cur.f + step[i][0];
            int next_y = cur.s + step[i][1];
            if(check(next_x, next_y, visisted)){
                visisted[next_x][next_y] = 1;
                qu.push({next_x, next_y});
            }
        }
    }
}
void build(){
    vector<vector<int>> visisted(m + 1, vector<int> (n + 1));
    for(int i = 1; i <= m; i++){
        for(int j = 1; j <= n; j++){
            if(a[i][j] == 1 and visisted[i][j] == 0){
```

```
                bfs(i, j, visisted);
                ans++;
            }
        }
    }
    cout << ans;
}
signed main(){
    //freopen("a.inp", "r", stdin);
    //freopen("a.out", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    enter();
    build();
}
```

dfs

Đường đi dài nhất

```
#include <bits/stdc++.h>
using namespace std;
const int ll = 1e6 + 10;
bool visted[ll];
vector<vector<pair<int, int>>> x(ll);
int dp[ll], n;
void dfs(int u){
    visted[u] = true;
    for(auto h : x[u]){
        if(visted[h.first] == false){
            dp[h.first] = dp[u] + h.second;
            dfs(h.first);
        }
    }

}
signed main(){
    cin >> n;
    for(int i = 1; i <= n - 1; i++){
        int u, v, w;
        cin >> u >> v >> w;
        x[u].push_back({v, w});
        x[v].push_back({u, w});
    }
    dfs(2);
    int dln, ans = 0;
    for(int i = 1; i <= n; i++){
        if(ans < dp[i]){
            ans = dp[i];
            dln = i;
        }
    }
    fill(dp, dp + 1 + n, 0);
    fill(visted, visted + 1 + n, false);
    dfs(dln);
    cout << *max_element(dp, dp + 1 + n);
}
```

Đường Tới Nhà Hàng

Anh bạn Hoàng Hiếu của chúng ta vừa nhận được học bổng tại một trường đại học danh tiếng của Mỹ, và anh quyết định tự thưởng cho bản thân mình một bữa beafsteak thật ngon tại một nhà hàng trong vùng.

Nhà của Hiếu nằm trong một khu đô thị có dạng một đồ thị hình cây với N đỉnh, đỉnh 1 vừa là gốc vừa là nhà của anh. Ở các đỉnh lá của cây, mỗi đỉnh có một nhà hàng. Nhưng không may cho Hiếu là anh rất sợ mèo, mà trong khu đô thị này lại có nhiều mèo. Hiếu đã biết trước đỉnh nào trong khu đô thị có mèo, và anh không thể đi những con đường có quá M con mèo liên tiếp được.

Yêu cầu: Hãy xác định số lượng nhà hàng mà Hiếu có thể lựa chọn để tới ăn, với điều kiện đường đi từ nhà Hiếu tới nhà hàng đó không có quá M con mèo liên tiếp?

Mô tả đầu vào
Dòng đầu tiên chứa hai số nguyên dương N, M - số lượng đỉnh trong khu đô thị của Hiếu và số lượng con mèo tối đa mà Hiếu có thể "chịu đựng được" nếu như gặp phải chúng liên tiếp trên đường đi.

- Dòng thứ hai chứa NN số nguyên a_i
  mang giá trị 0 hoặc 1 - với $a_i = 1$ nếu đỉnh thứ i có mèo và ngược lại.
- $N - 1$ dòng cuối cùng, mỗi dòng chứa một cạnh (x, y) $(x \neq y)$ (x,y) của đồ thị hình cây mô tả khu đô thị nhà Hiếu.

```
void dfs(int u, int so_con_meo){
    visited[u] = true;
    int check_la = 1;
    if(so_con_meo > m){
        return;
    }
    for(auto e: adj[u]){
        if(!visited[e]){
            check_la = 0;
            if(a[e] == 1){
                so_con_meo += 1;
                dfs(e, so_con_meo);
            }
            else{
                dfs(e, 0);
            }
        }
    }
    if(check_la == 1 and so_con_meo <= m){
        ans ++;
    }
}
```

Cắt Tỉa Cây

Công ty XYZ cung cấp sản phẩm là những chiếc cây bằng nhựa để trang trí trong gia đình. Những chiếc cây bao gồm n tán lá và $n-1$ cành nối sao cho các tán lá đều liên thông với nhau. Để phục vụ công tác vận chuyển, các cành cây và tán lá đều có thể tháo lắp tùy ý thành các phần có kích thước nhỏ hơn, sau đó khách hàng sẽ tự lắp ráp các phần lại với nhau. Tuy nhiên, các khách hàng khó tính của công ty không muốn nhận được những phần cây có kích thước là số lẻ (số tán lá lẻ). Mặt khác, công ty lại muốn tháo được càng nhiều cành cây càng tốt để các phần cây nhỏ có thể được sắp xếp gọn gàng hơn trong khoang chứa hàng.

Yêu cầu: Hãy tìm cách tháo nhiều cành cây ra nhất có thể sao cho các phần cây còn lại đều có kích thước là số chẵn?

Mô tả đầu vào

- Dòng đầu tiên ghi số nguyên dương n.
- $n-1$ dòng tiếp theo, mỗi dòng ghi một cặp số nguyên $|(u,v)$ – thể hiện một cành cây nối giữa hai tán lá u và v.

Mô tả đầu ra

- Đưa ra số lượng cành nhiều nhất có thể tháo ra sao cho các phần còn lại của cây đều có kích thước chẵn. Nếu không thể tháo cành cây nào thì đưa ra -1.

```
#include<bits/stdc++.h>
#define int long long
#define endl "\n"
using namespace std;
const int ll = 1e6 + 7;
int n, children[ll], res = 0;
vector<int> adj[ll];
bool visited[ll];
void enter(){
    cin >> n;
    for(int i = 1; i <= n - 1; i++){
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
}
void dfs(int u){
    visited[u] = true;
    children[u] = 1;
    for(auto v: adj[u]){
        if(!visited[v]){
            dfs(v);
            children[u] += children[v];
        }
    }
    if(children[u] % 2 == 0)
        res ++;
}
void build(){
    if(n % 2 != 0){
        cout << -1;
    }
    else{
        dfs(1);
        cout << res - 1;
    }
}
signed main(){
    //freopen("a.inp", "r", stdin);
    //freopen("a.out", "w", stdout);
    enter();
    build();
}
```

Thuật toán Dijkstra

Cho một đồ thị có hướng với N đỉnh (được đánh số từ 0 đến $N-1$), M cạnh có hướng, có trọng số, và một đỉnh nguồn S. Trọng số của tất cả các cạnh đều không âm. Yêu cầu tìm ra đường đi ngắn nhất từ đỉnh S tới tất cả các đỉnh còn lại (hoặc cho biết nếu không có đường đi).

```
const long long INF = 2000000000000000000LL;
struct Edge{
    int v;
    long long w;
};
void dijkstra(int n, int S, vector<vector<Edge>> E, vector<long long> &D, vector<int>
&trace) {
    D.resize(n, INF);
    trace.resize(n, -1);

    vector<bool> P(n, 0);
    D[S] = 0;

    for (int i = 0; i < n; i++) {
        int uBest; // tìm đỉnh u chưa dùng, có khoảng cách nhỏ nhất
        long long Max = INF;
        for (int u = 0; u < n; u++) {
            if(D[u] < Max && P[u] == false) {
                uBest = u;
                Max = D[u];
            }
        }

        // cải tiến các đường đi qua u
        int u = uBest;
        P[u] = true;
        for(auto x : E[u]) {
            int v = x.v;
            long long w = x.w;
            if(D[v] > D[u] + w) {
                D[v] = D[u] + w;
                trace[v] = u;
            }
        }
    }
}
```