

C	Vacation	Chọn 1 trong 3 việc (A,B,C) mỗi ngày, không trùng việc hôm qua. Max. điểm.	$dp[i][j]:$ Hanh phuc ngay i lam viec j $dp[i][j] = \max_{k \neq j} (dp[i-1][k]) + cost[j][i]$	L	Deque	Game lây số từ đầu. Max (Điểm minh - Điểm đích).	$dp[i][j]:$ Hiệu điểm max đoạn [i, j]	$\max(a[i] - dp[i+1][j], a[j] - dp[i][j-1])$	T	Permutation	Đếm hoán vị dãy N thỏa mãn chia đầu < và cuối >.	$dp[i][j]:$ Hoán vị dãy i, cuối là số lớn thứ j	Tối ưu chuyển trạng thái bằng Prefix Sum $O(N^2)$	
D	Knapsack 1	Cá nhỏ Chon vật W ≤ W' max $\sum_i:$ ( trọng lượng w_i)	$dp[w]:$ Giá trị max với trọng lượng w	M	Candies	Chia K' kẹo cho N trẻ, trẻ i nhận đó là a_i. Đếm số.	$dp[i][j]:$ Chia cho i người, tổng j	Dùng Prefix Sum: $dp[i][j] = \sum dp[i-1][\dots]$	U	Grouping	Chia N vật thành các nhóm. Max tổng điểm của các nhóm.	$dp[mask]:$ Điểm max chia nhóm tệp mask	$dp[mask] = \max(dp[mask \setminus S] + score(S))$ ( $O(3^N)$ )	
E	Knapsack 2	Gióng bài D nhưng $W \leq 10^3$ , $v_i \leq 10^3$ .	$dp[v]:$ Trọng lượng min để đạt giá trị v	N	Slimes	Gộp 2 cốc slime kế nhau, phi + tổng kích thước. Min tổng tông.	$dp[i][j]:$ Phi gộp đoạn [i, j]	$\min_k (dp[i][k] + dp[k+1][j]) + \sum(i, j)$	V	Subtree	Với mỗi đỉnh, đếm số cách tách mảng cây con liên thông chưa nò (mod M).	$dp[u]:$ Kết quả đinh u	Rerooting DP: Tinh Down xong tính Up.	
F	LCS	Tìm sub chuỗi dài nhất của 2 câu x và t.	$dp[i][j]:$ Độ dài sub chuỗi [1..i], t[1..j]	O	Matching	Đếm số cách ghép cặp N nam - N' nêu theo bảng tương thích.	$dp[mask]:$ Số cách ghép cặp (Bitmap)	$\sum dp[mask \setminus \{j\}]$ nếu bit j match với bit cao nhất	W	Intervals	Xử lý phân số. Các đoạn [l, r] cho điểm q, nếu toàn bộ 1 Max điểm.	$dp[i]:$ Điểm max tại i	SegTree + Lazy (Range Add, Range Max).	
G	Longest Path	Tìm đường đi dài nhất trong đồ thị hướng (DAG).	$dp[v]:$ Đường dài nhất bắt đầu từ v	P	Independent	Đếm cách tóm màu Trắng/Đen lên cây sao cho có 2 Đen kế nhau.	$dp[u][0..1]:$ Cây con u, u trắng/đen	Nhận quy tắc: Trắng (con tủy y), Đen (con phải trắng)	X	Tower	Xếp chồng khói (như w, cùng x, giá trị x). Max tổng giá trị.	Knapsack 1 chiều	Sort theo w + s, sau đó DP cải túi như bài D/E.	
H	Grid 1	Đếm cách từ (1,1) → (H, W), tránh tường #.	$dp[i][j]:$ Số cách đến (i, j)	Q	Flowers	Tim dãy con có chiều cao tăng dần sao cho tổng giá trị là max.	$dp[i]:$ Max trọng số kết thúc tại i	$\max_{0 \leq h_i \leq h_j} dp[j] + a_i$ ; Dùng Segment Tree lấy Max.	Y	Grid 2	Lưới $10^3 \times 10^3$ , có N vật cần. Đếm cách di chuyển vật.	$dp[i]:$ Các int s, n; int MIN = INT_MAX; vector<vector<int>> course_lists;		
I	Coins	$N$ xu với xác suất $p_i$ , Tính xác suất số xu > sáp.	$dp[i][j]:$ Tung i xu, j mặt	R	Walk	Đếm số đường đi dài nhất K' trong đồ thị hướng.	Ma trận $M^K$	Tổng các phần tử của $M^K$ (Nhân ma trận)	vector<vector<int>> conflicts;	Z	Frog 3	Nhảy i → j, phi $(h_i - h_j)^2 + C$ . Tím phi min. N	$dp[i]:$ Chi vector<int> assigns; vector<vector<int>> loads; int check(int course, int teacher)	Balafice courses {
J	Sushi	N đĩa có 1-3 miếng. Chọn cón i, j, k, đĩa loại 1,2,3	$dp[i][j][k]:$ Kỳ vọng khi ngẫu nhiên đĩa đén. Tình kỳ vọng số lượt.	S	Digit Sum	Đếm số lượng số i có k chữ số.	$dp[pos][tight][rem]$	Digit DP đếm số thỏa mãn mod D = 0						

```

    Wrong code due to this.    con da
CBUS          You, now + Uncommitted chi

int n, k;
vector<vector<int>> c;
vector<int> visited, path;
int load = 0, cost = 0, MIN = INT_MAX, cmin = INT_MAX;
int check(int i)
{
    if (i <= n && visited[i] == 0 && load < k)
    {
        return 1;
    }
    if (i > n && visited[i] == 0 && visited[i - n] == 1)
    {
        return 1;
    }
    return 0;
}
void travel(int step)
{
    for (int i = 1; i <= 2 * n; i++)
    {
        if (check(i) == 1)
        {
            cost += c[path.back()][i];
            path.push_back(i);
            visited[i] = 1;
            if (i <= n) load++;
            else load--;
            if (step == (2 * n))
            {
                MIN = min(MIN, cost + c[i][0]);
            }
            else
            {
                int g = cost + cmin * (2 * n + 1 - step);
                if (g <= MIN) travel(step + 1);
            }
            if (i <= n) load--;
            else load++;
            visited[i] = 0;
            path.pop_back();
            cost -= c[path.back()][i];
        }
    }
}
int main()
{
    cin >> n >> k;
    c.resize(2 * n + 1, vector<int>(2 * n + 1, 0));
    for (int i = 0; i <= 2 * n; i++)
    {
        for (int j = 0; j <= 2 * n; j++)
        {
            cin >> c[i][j];
            if (c[i][j] != 0) cmin = min(cmin, c[i][j]);
        }
    }
    path.push_back(0);
    visited.resize(2 * n + 1, 0);
    visited[0] = 1;
    travel(1);
    cout << MIN << "\n";
    return 0;
}

```

## Inversion

```
while (i < leftSize && j < rightSize)
{
    You, 2 days ago • Create inversion
    // merge sort
    if (l[i] > r[j])
    {
        a[left++] = r[j++];
```

## Bridges and articulations

```
int n, m;
vector<vector<int>> adj;
vector<int> num;
vector<int> low;
vector<int> articulations;
vector<int> intos, intos_bridges;
int current_num = 0;
void dfs(int u, int parent)
{
    current_num++;
    num[u] = current_num;
    low[u] = current_num;
    int children = 0;
    for (auto v : adj[u])
    {
        if (v == parent) continue;
        if (num[v] == 0)
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            children++;
            if (low[v] > num[u]) bridges.push_back(make_pair(u, v));
            if (parent == 0 && low[v] == num[u] && find(articulations.begin(), articulations.end(), u) == articulations.end())
            {
                articulations.push_back(u);
            }
        }
        else
        {
            low[u] = min(low[u], num[v]);
        }
    }
}
if (parent == 0 && children == 2 && find(articulations.begin(), articulations.end(), u) == articulations.end())
{
    articulations.push_back(u);
}
```

CVRP China Net Cup 2024

```

using ll = long long;
using ii = pair<int, int>;
int n, K, O;
int d[1005], load[1005];
int c[1005][1005], cmin = INT_MAX;
int y[1005], x[1005], visited[1005];
int nbr, segments;
int f, fopt = INT_MAX;
bool check(int custom, int bus) {
    if (custom == 0) return true;
    if (visited[custom]) return false;
    if (load[bus] + d[custom] > O) return false;
    return true;
}
void TryX(int u, int k) {
    if (u == 0) {
        if (k < K) TryY(y[k + 1], k + 1);
        return ;
    }
    for (int i = 0; i <= n; ++i) {
        if (check(i, k)) {
            x[u] = i;
            visited[i] = true;
            load[k] += d[i];
            f += c[u][i];
            ++segments;
            if (i == 0) {
                if (n + nbr == segments) fopt = min(fopt, f)
                else if (f + (n + nbr - segments) * cmin < fopt)
                    TryX(i, k + 1);
            }
            else {
                if (f + (n + nbr - segments) * cmin < fopt)
                    TryX(i, k);
            }
            visited[i] = false;
            load[k] -= d[i];
            f -= c[u][i];
            --segments;
        }
    }
}
void TryY(int k) {
    int st = 0;
    if (y[k - 1]) st = y[k - 1] + 1;
    for (int i = st; i <= n; ++i) {
        if (check(i, k)) {
            y[k] = i;
            visited[i] = true;
            if (i) ++segments;
            load[k] += d[i];
            f += c[0][y[k]];
            if (k == K) {
                nbr = segments;
                TryX(y[1], 1);
            }
            else TryY(k + 1);
            visited[i] = false;
            if (i) --segments;
            load[k] -= d[i];
            f -= c[0][y[k]];
        }
    }
}

```

```
Max-distance-subarray  
int n, c, res;
```

```

vector<int> a;
int check(int d)
{
    int last = 0;
    int cnt = 1;
    for (int i = 1; i < n; i++)
    {
        if (a[i] - a[last] >= d)
        {
            cnt++;
            last = i;
        }
        if (cnt == c)
        {
            return 1;
        }
    }
    return 0;
}

void findD(int left, int right)
{
    if (left <= right)
    {
        int mid = left + (right - left) / 2;

        if (check(mid))
        {
            res = mid;
            findD(mid + 1, right);
        }
        else
        {
            findD(left, mid - 1);
        }
    }
}

```

You, 2 days ago • Create Max\_distance.

```

int main()
{
    int T;
    cin >> T;
    while (T--)
    {
        cin >> n >> c;
        a.resize(n, 0);
        for (int i = 0; i < n; i++)
        {
            cin >> a[i];
        }
        sort(a.begin(), a.end());
        int d = (a[n - 1] - a[0]) / (c - 1);

        findD(1, d);

        cout << res << "\n";
    }
    return 0;
}

```

### Cut materials

```

int h, w, n;
vector<vector<int>> recs(MAX + 1, vector<int>(2));
vector<vector<int>> grid(MAX + 1, vector<int>(MAX, 0));
void place(int h, int w, int startRow, int startCol, int value)
{
    for (int i = startRow; i < startRow + h; i++)
    {
        for (int j = startCol; j < startCol + w; j++)
        {
            grid[i][j] = value;
        }
    }
}
int canPlace(int h, int w, int startRow, int startCol)
{
    if (((startRow + h - 1) > MAX) || ((startCol + w - 1) > MAX)) return
    for (int i = startRow; i < startRow + h; i++)
    {
        for (int j = startCol; j < startCol + w; j++)
        {
            if (grid[i][j] == 1) return 0;
        }
    }
    return 1;
}

```

```

(assigns[course] == -1)

for (int teacher = 0; teacher < m; teacher++)
{
    if (check(course, teacher))
    {
        assigns[course] = teacher;
        loads[teacher]++;
        if (course == (n - 1))
        {
            MIN = min(MIN,
                       *max_element(loads.begin(),
                                    loads.end()));
        }
        else
        {
            int MAX = *max_element(loads.begin(),
                                   loads.end());
            if (MAX <= MIN) assign(course + 1);
        }
        loads[teacher]--;
        assigns[course] = -1;
    }
}

You, 3 days ago • Create balanced_courses_assignment()
in()

n >> m >> n;
course_lists.resize(m, vector<int>());
for (int i = 0; i < m; i++)
{
    int k;
    cin >> k;
    for (int j = 0; j < k; j++)
    {
        int course;
        cin >> course;
        course_lists[i].push_back(course - 1);
    }
}

t k;
n >> k;
conflicts.resize(n, vector<int>(n, 0));
ile (k--)

int i, j;
cin >> i >> j;
conflicts[i - 1][j - 1] = 1;
conflicts[j - 1][i - 1] = 1;

signs.resize(n, -1);
ads.resize(m, 0);
sign(0);
cout << MIN << "\n";
turn a.
int backtrack(int k)

if (k == n + 1) return 1;

int h = recs[k][0];
int w = recs[k][1];

for (int i = 1; i <= H; i++)
{
    if (check(w, i))
    {
        ads[w] = 1;
        signs[w] = 1;
        if (backtrack(k + 1))
        {
            return 1;
        }
        ads[w] = 0;
        signs[w] = -1;
    }
}

```

```

for (int j = 1; j <= W; j++)
{
    if (canPlace(h, w, i, j))
    {
        place(h, w, i, j, 1);
        if (backtrack(k + 1)) return 1;
        place(h, w, i, j, 0);
    }
    if (h != w && canPlace(w, h, i, j))
    {
        place(w, h, i, j, 1);
        if (backtrack(k + 1)) return 1;
        place(w, h, i, j, 0);
    }
}
return 0;
int main()
{
    cin >> H >> W;
    cin >> n;
    int area = 0;
    for (int i = 1; i <= n; i++)
    {
        cin >> recs[i][0] >> recs[i][1];
        area += (recs[i][0] * recs[i][1]);
    }

    if (area > H * W)
    {
        cout << 0 << "\n";
        return 0;
    }

    sort(recs.begin() + 1, recs.end(),
         [] (vector<int> a, vector<int> b) {
            return a[0] * a[1] > b[0] * b[1];
        });
}

```

### Bus-inter-city (BFS to build weighted directed graph and dijkstra)

```

int n, m;
vector<int> c, d, visited, dp;
vector<vector<int>> adj;
vector<long long> dis;
vector<vector<pair<int, int>> adj1;
void bfs(int i, int limit)
{
    fill(visited.begin(), visited.end(), 0);
    queue<pair<int, int>> q;
    q.push({i, 0});
    visited[i] = 1;
    while (!q.empty())
    {
        int u = q.front().first;
        int depth = q.front().second;
        q.pop();
        if (u != i)
        {
            adj1[i].push_back({u, c[i]});
        }
        if (depth >= limit) continue;

        for (auto v : adj[u])
        {
            if (!visited[v])
            {
                visited[v] = 1;
                q.push({v, depth + 1});
            }
        }
    }
}

void dijkstra(int s)
{
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, pq;
    pq.push({0, s});

    for (int i = 0; i < n; i++)
    {
        int u = -1;
        while ((pq.empty()) || (u == -1))
        {
            u = pq.top().second;
            pq.pop();
            if (u == -1) break;
            visited[u] = 1;
            for (auto edge : adj1[u])
            {
                int v = edge.first;
                int w = edge.second;
                if (dis[u] + w <= dis[v])
                {
                    dis[v] = dis[u] + w;
                    pq.push({dis[v], v});
                }
            }
        }
    }
}

```

```
int main()
{
    cin >> n >> m;
```

```

c.resize(n + 1, 0);
d.resize(n + 1, 0);
dp.resize(n + 1, 0);
visited.resize(n + 1, 0);
adj.resize(n + 1, vector<int>());
adj1.resize(n + 1, vector<pair<int, int>>());
dis.resize(n + 1, INT_MAX);
for (int i = 1; i < n; i++)
{
    cin >> c[i] >> d[i];
}

```

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

```
for (int i = 1; i <= n; i++)
{
    bfs(i, d[i]);
}
```

```
fill(visited.begin(), visited.end(), 0);
```

```
dijkstra(1);
```

```
cout << dis[n] << "\n";
return 0;
}
```

```

amount.resize(n, vector<int>(X + 1, INT_MAX - 1));
for (int i = 0; i < n; i++) amount[i][0] = 0;
//base case i = 0, Money-exchange
//knapsack init
for (int j = 0; j <= X; j++)
{
    if ((j / d[0]) * d[0] == j)
    {
        amount[0][j] = j / d[0];
    }
}
for (int i = 1; i < n; i++)
{
    for (int j = 1; j <= X; j++)
    {
        amount[i][j] = amount[i - 1][j];
        if (j >= d[i])
        {
            amount[i][j] = min(amount[i][j],
                                amount[i][j - d[i]] + 1);
        }
    }
}
cout << amount[n - 1][X] != INT_MAX - 1 ?
    amount[n - 1][X] : -1 << "\n";
return 0;
}
```

### Make-span-schedule (topo sort and DP)

```

int n, m;
vector<vector<int>> adj;
vector<int> d;
vector<int> order;
int hasCycle = 0;
int ans = 0;
vector<int> f;

void topo(int u)
{
    if (visited[u] == 2) return; // done

    if (visited[u] == 1)
    {
        hasCycle = 1;
        // cycle
        return;
    }

    visited[u] = 1;

    for (auto v : adj[u])
    {
        topo(v);
        if (hasCycle) return;
    }

    visited[u] = 2;
    order.push_back(u);
}

int main()
{
    cin >> n >> m;
    adj.resize(n + 1, vector<int>());
    f.resize(m + 1, 0);
    for (int i = 1; i <= n; i++)
    {
        cin >> d[i];
    }

    adj.resize(n + 1, vector<int>());
    visited.resize(n + 1, 0);
    for (int i = 1; i <= m; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }

    for (int i = 1; i <= n; i++)
    {
        if (visited[i] == 0) topo(i);
    }

    reverse(order.begin(), order.end());
    for (auto u : order)
    {
        f[u] += d[u];
        for (auto v : adj[u])
        {
            f[v] = max(f[v], f[u]);
        }
        ans = max(ans, f[u]);
    }

    cout << ans << "\n";
    return 0;
}

```

### Max-matching-bipartite

```

int n, m;
vector<vector<int>> adj;
// adj[task] = list of staffs
// that can do this task
vector<int> matchTask;
// matchTask[staff] = task
// assigned to this staff (0 if none)
vector<bool> visited;
// Kuhn's algorithm
// (DFS to find augmenting path)
bool dfs(int task)
{
    for (int staff : adj[task])
    {
        if (visited[staff]) continue;
        visited[staff] = true;
        // If staff is free or we can
        // reassign their current task
        if (matchTask[staff] == 0
            || dfs(matchTask[staff]))
        {
            matchTask[staff] = task;
            return true;
        }
    }
    return false;
}

```

```
int main()
{
    cin >> n >> m;

    adj.resize(n + 1);
    matchTask.resize(m + 1, 0);

    for (int i = 1; i <= n; i++)
    {
        int k;
        cin >> k;
        for (int j = 0; j < k; j++)
        {
            int staff;
            cin >> staff;
            adj[i].push_back(staff);
        }
    }
}

```

```
fill(visited.begin(), visited.end(), 0);
dijkstra(1);
cout << dis[n] << "\n";
return 0;
}
```

### SCC

```

int n, m;
vector<vector<int>> adj;
vector<int> num;
vector<int> low;
stack<int> SCC;
vector<int> onStack;
int current_num = 0, cnt = 0;
vector<vector<int>> res;
void dfs(int u)
{
    current_num++;
    num[u] = current_num;
    low[u] = current_num;
    SCC.push(u);
    onStack[u] = 1;
    for (auto v : adj[u])
    {
        if (num[v] == 0)
        {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else if (onStack[v] == 1)
        {
            low[u] = min(low[u], num[v]);
        }
    }
    if (low[u] == num[u])
    {
        int v;
        for (v = u; adj[v].size() > 0
            && adj[v][0] == u; v = adj[v].back());
        res.push_back({u, v});
        onStack[v] = 0;
    }
}

```

```
int main()
{
    cin >> n >> m;
    adj.resize(n + 1, vector<int>());
    f.resize(m + 1, 0);
    for (int i = 1; i <= n; i++)
    {
        cin >> d[i];
    }

    adj.resize(n + 1, vector<int>());
    visited.resize(n + 1, 0);
    for (int i = 1; i <= m; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }

    for (int i = 1; i <= n; i++)
    {
        if (visited[i] == 0) topo(i);
    }

    reverse(order.begin(), order.end());
    for (auto u : order)
    {
        f[u] += d[u];
        for (auto v : adj[u])
        {
            f[v] = max(f[v], f[u]);
        }
        ans = max(ans, f[u]);
    }

    cout << ans << "\n";
    return 0;
}

```

### Total-path-lengths

```

int n;
vector<vector<pair<int, int>> a(MAX + 1);
vector<int> subtree_size(MAX + 1, 1);
vector<int> visited(MAX + 1, 0);
vector<long long> answer(MAX + 1, 0);

```

```
void dfs(int u, int depth)
{
    visited[u] = 1;
    answer[u] += depth;
    for (auto path : a[u])
    {
        int v = path.first;
        int w = path.second;
        if (visited[v] == 0)
        {
            dfs(v, depth + subtree_size[v] * w);
            subtree_size[u] += subtree_size[v];
        }
    }
}

```

```
void dfs2(int u)
{
    visited[u] = 1;
    for (auto path : a[u])
    {
        int v = path.first;
        int w = path.second;
        if (visited[v] == 0)
        {
            dfs(v, depth + subtree_size[v] * w);
            subtree_size[u] += subtree_size[v];
        }
    }
}

```

```
int main()
{
    cin >> n;

    for (int i = 1; i < n; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        a[u].push_back({v, w});
        a[v].push_back({u, w});
    }
}

```

```
dfs(1, 0);
for (int i = 1; i <= n; i++) visited[i] = 0;
dfs2(1);

```

```
cout << *max_element(answer.begin(), answer.end()) << "\n";
return 0;
}

```

### Hungary

```

const long long INF = 1e18; // Giá trị vô cùng lớn cho long long
int n, m, k, sz;
vector<vector<long long>> a; // Ma trận chi phí (đã đảo đổi)
vector<long long> u, v, minv;
stack<int> onStack;
vector<int> p, way; // p: mapping, way: mang truy vết
int main()
{
    if ((cin >> n >> m >> k)) return 0;

    sz = max(n, m);
    a.assign(sz + 1, vector<long long>(sz + 1, 0));

    for (int i = 0; i < k; i++)
    {
        int x, y, w;
        cin >> x >> y >> w;
        if (a[x][y] == 0) a[x][y] = -w;
        else a[x][y] = min(a[x][y], -(long long)w);
    }

    u.assign(sz + 1, 0);
    v.assign(sz + 1, 0);
    p.assign(sz + 1, 0);
    way.assign(sz + 1, 0);
    for (int i = 1; i <= sz; i++)
    {
        p[i] = i;
        int j0 = 0;
        minv.assign(sz + 1, INF);
        vector<bool> used(sz + 1, false);
        do
        {
            used[j0] = true;
            int o = p[j0];
            int new_flow = 0;
            int j1;
            for (int j = 1; j <= sz; j++)
            {
                if (!used[j])
                {
                    long long cur = a[o][j] - u[o] - v[j];
                    if (cur < minv[j])
                    {
                        minv[j] = cur;
                        way[j] = j0;
                    }
                }
            }
            if (minv[j] < delta)
            {
                delta = minv[j];
                j1 = j;
            }
        } while (j0 != o);
        for (int j = 0; j <= sz; j++)
        {
            if (!used[j])
            {
                long long cur = a[p[j]][j] - u[p[j]] - v[j];
                if (cur < minv[j])
                {
                    minv[j] = cur;
                    way[j] = p[j];
                }
            }
        }
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
        while (j0 != o);
    }
    cout << minv[1] << "\n";
    return 0;
}

```

### Edmond-karp (max flow)

```

int n, m;
vector<vector<int>> capacity;
vector<vector<int>> adj;
int bfs(int source, int sink, vector<int> &parent)
{
    fill(parent.begin(), parent.end(), -1);
    parent[source] = source;
    queue<pair<int, int>> q;
    q.push(source, INT_MAX);
    while (!q.empty())
    {
        int u = q.front().first;
        int flow = q.front().second;
        q.pop();
        for (int v : adj[u])
        {
            if (parent[v] == -1 && capacity[u][v] > 0)
            {
                parent[v] = u;
                int new_flow = min(flow, capacity[u][v]);
                if (v == sink)
                {
                    return new_flow;
                }
                q.push(v, new_flow);
            }
        }
    }
    return 0;
}

int edmond_karp(int source, int sink)
{
    int max_flow = 0;
    vector<int> parent(n + 1);
    int new_flow;
    while ((new_flow = bfs(source, sink, parent)) > 0)
    {
        max_flow += new_flow;
        int cur = sink;
        while (cur != source)
        {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }
    return max_flow;
}

int main()
{
    cin >> n >> m;
    capacity.resize(n + 1, vector<int>(n + 1, 0));
    adj.resize(n + 1);
    for (int i = 1; i <= m; i++)
    {
        int u, v, c;
        cin >> u >> v >> c;
        capacity[u][v] += c;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int source, sink;
    cin >> source >> sink;
    cout << edmond_karp(source, sink) << "\n";
    return 0;
}

```

### Longest-path

```

int n;
vector<int> visited(MAX + 1, 0);
vector<vector<pair<int, int>> a(MAX + 1);
vector<int> distances(MAX + 1);

void bfs(int i)
{
    queue<int> q;
    if (visited[i] == 1) return;
    q.push(i);
    visited[i] = 1;
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (auto path : a[u])
        {
            int v = path.first;
            int w = path.second;
            if (visited[v] == 0)
            {
                q.push(v);
                visited[v] = 1;
                distances[v] = distances[u] + w;
            }
        }
    }
}

int main()
{
    cin >> n;

    for (int i = 1; i < n; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        a[u].push_back({v, w});
        a[v].push_back({u, w});
    }
}

int Max = 0;
int start = 1;
for (int i = 1; i <= n; i++)
{
    if (Max < distances[i])
    {
        Max = distances[i];
        start = i;
    }
    distances[i] = 0;
    visited[i] = 0;
}

bfs(start);
Max = 0;
for (int i = 1; i <= n; i++)
{
    if (Max < distances[i])
    {
        Max = distances[i];
    }
}
cout << Max << "\n";
return 0;
}

```