C	Vacation	Chọn 1 trong 3 việc (A,B,C) mỗi ngày, không trùng việc hôm qua. Max điểm.	$dp[i][j]$: Hạnh phúc ngày i làm việc j	$dp[i][j] = \max_{k \neq j}(dp[i-1][k]) + cost[i][j]$
D	Knapsack 1	Cái túi: Chọn vật $\sum w \le W$ để max $\sum v$. ($W \le 10^5$)	$dp[w]$: Giá trị max với trọng lượng w	$dp[w] = \max(dp[w], dp[w-w_i] + v_i)$
E	Knapsack 2	Giống bài D nhưng $W \le 10^9, v \le 10^3$.	$dp[v]$: Trọng lượng min để đạt giá trị v	$dp[v] = \min(dp[v], dp[v-v_i] + w_i)$
F	LCS	Tìm xâu con chung dài nhất của 2 xâu s và t.	$dp[i][j]$: Độ dài xâu chung $s[1..i], t[1..j]$	$s[i] == t[j] \to +1$; else $\max(dp[i-1][j], dp[i][j-1])$
G	Longest Path	Tìm đường đi dài nhất trong đồ thị có hướng (DAG).	$dp[u]$: Đường dài nhất bắt đầu từ u	$dp[u] = \max_{(u,v)\in E}(dp[v]+1)$ (DFS + Memo)
H	Grid 1	Đếm cách đi từ $(1,1) \to (H,W)$, tránh tường #.	$dp[i][j]$: Số cách đến (i,j)	$dp[i][j] = dp[i-1][j] + dp[i][j-1]$
I	Coins	N xu với xác suất ngửa p_i. Tính xác suất số ngửa > sấp.	$dp[i][j]$: Tung i xu, j mặt ngửa	$dp[i][j] = dp[i-1][j](1-p) + dp[i-1][j-1]p$
J	Sushi	N đĩa có 1-3 miếng. Chọn ngẫu nhiên đĩa để ăn. Tính kỳ vọng số lượt.	$dp[i][j][k]$: Kỳ vọng khi còn i, j, k đĩa loại 1,2,3	$E = \frac{N+iE_{i-1}+jE_{j-1}+kE_{k-1}}{i+j+k}$
K	Stones	Đống đá K. Bốc a_i viên. Ai không bốc được thì thua.	$dp[k]$: Thắng/Thua khi còn k đá	Win nếu tồn tại nước đi dẫn đến thế Thua của đối thủ
L	Deque	Game lấy số từ 2 đầu. Max hóa (Điểm mình - Điểm địch).	$dp[i][j]$: Hiệu điểm max đoạn $[i,j]$	$\max(a[i] - dp[i+1][j], a[j] - dp[i][j-1])$
M	Candies	Chia K kẹo cho N trẻ, trẻ i nhận tối đa a_i. Đếm số cách.	$dp[i][j]$: Chia cho i người, tổng j	Dùng Prefix Sum: $dp[i][j] = \sum dp[i-1][\dots]$
N	Slimes	Gộp 2 cục slime kề nhau, phí = tổng kích thước. Min tổng phí.	$dp[i][j]$: Phí gộp đoạn $[i,j]$	$\min_k(dp[i][k] + dp[k+1][j]) + \sum(i,j)$
O	Matching	Đếm số cách ghép cặp N nam - N nữ theo bảng tương thích.	$dp[mask]$: Số cách ghép cặp (Bitmask)	$\sum dp[mask \setminus \{j\}]$ nếu bit j match với bit cao nhất
P	Independent	Đếm cách tô màu Trắng/Đen lên cây sao cho ko có 2 Đen kề nhau.	$dp[u][0/1]$: Cây con u, u trắng/đen	Nhân quy tắc: Trắng (con tùy ý), Đen (con phải trắng)
Q	Flowers	Tìm dãy con có chiều cao tăng dần sao cho tổng giá trị a max.	$dp[i]$: Max trọng số kết thúc tại i	$\max_{h_j<h_i} dp[j] + a_i$. Dùng Segment Tree lấy Max.
R	Walk	Đếm số đường đi độ dài đúng K trong đồ thị hướng.	Ma trận M^K	Tổng các phần tử của M^K (Nhân ma trận)
S	Digit Sum	Đếm số lượng số $x \in [1,K]$ có tổng chữ số chia hết cho D.	$dp[pos][tight][rem]$	Digit DP đếm số thỏa mãn mod $D = 0$
T	Permutation	Đếm hoán vị độ dài N thỏa mãn chuỗi dấu < và >.	$dp[i][j]$: Hoán vị dài i, cuối là số lớn thứ j	Tối ưu chuyển trạng thái bằng Prefix Sum $O(N^2)$
U	Grouping	Chia N vật thành các nhóm. Max tổng điểm của các nhóm.	$dp[mask]$: Điểm max chia nhóm tập mask	$dp[mask] = \max(dp[mask \setminus S] + score(S))$ ($O(3^N)$)
V	Subtree	Với mỗi đỉnh, đếm số cách tô màu cây con liên thông chứa nó (mod M).	$dp[u]$: Kết quả đỉnh u	Rerooting DP: Tính Down xong tính Up.
W	Intervals	Xâu nhị phân. Các đoạn $[l,r]$ cho điểm a_i nếu toàn số 1. Max điểm.	$dp[i]$: Điểm max tại i	SegTree + Lazy (Range Add, Range Max).
X	Tower	Xếp chồng khối (nặng w, cứng s, giá trị v). Max tổng giá trị.	Knapsack 1 chiều	Sort theo $w+s$, sau đó DP cái túi như bài D/E.
Y	Grid 2	Lưới $10^9 \times 10^9$, có N vật cản. Đếm cách đi tránh vật cản.	$dp[i]$: Cách đến vật cản i	Bao hàm loại trừ: Tổng đường - đường đi qua vật cản khác.
Z	Frog 3	Nhảy $i \to j$, phí $(h_i-h_j)^2 + C$. Tìm phí min. $N = 2 \cdot 10^5$.	$dp[i]$: Chi phí tối thiểu	Dạng $y = mx + c$. Dù Tree).

CBUS

```cpp
int n, k;
vector<vector<int>> c;
vector<int> visited, path;       You, now • Uncommitted cha
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

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

Bridges and articulations

```cpp
int n, m;
vector<vector<int>> adj;
vector<int> num;
vector<int> low;
vector<int> articulations;
vector<pair<int, int>> bridges;
int current_num = 0;       You, yesterday • update for finalterm …
void dfs(int u, int parent)
{
    current_num++;
    num[u] = current_num;
    low[u] = current_num;
    int childs = 0;
    for (auto v : adj[u])
    {
        if (v == parent) continue;
        if (num[v] == 0)
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            childs++;
            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.enc
            {
                articulations.push_back(u);
            }
        }
        else
        {
            low[u] = min(low[u], num[v]);
        }
    }
    if (parent == 0 && childs >= 2 &&
    find(articulations.begin(), articulations.end(), u) == articulations.end())
    {
        articulations.push_back(u);
    }
}
```

```cpp
int main()
{
    cin >> n >> m;
    adj.resize(n + 1, vector<int>()
    num.resize(n + 1, 0);
    low.resize(n + 1, 0);

    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++)
    {
        if (num[i] == 0) dfs(i, 0);
    }

    cout << articulations.size() << " " << bridges.size() << "\n";
    return 0;
}
```

CVRP

```cpp
using ll = long long;
using ii = pair<int, int>;
int n, K, Q;
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] > Q) return false;
  return true;
}
void TryX(int u, int k) {
  if (u == 0) {
    if (k < K) TryX(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(y[k + 1], 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]];
    }
  }
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> K >> Q;
  for (int i = 1; i <= n; ++i) cin >> d[i];
  for (int i = 0; i <= n; ++i) {
    for (int j = 0; j <= n; ++j) {
      cin >> c[i][j];
      if (c[i][j]) cmin = min(cmin, c[i][j]);
    }
  }
  TryY(1);
  cout << fopt;
  return 0;
}
```

```cpp
sort(times.begin(), times.begin() + n,
[](int a, int b){
    return a > b;
}); //Job planning (Greedy)
for (int i = 0; i < n; i++)
{
    T = max(T, timepoint + times[i]);
    timepoint++;
}
cout << T << "\n";
```

Max-distance-subarray

```cpp
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;
}
```

```cpp
res.resize(2, vector<int>(n + 1, 0));
res[1][k1] = 1; //NURSE
res[0][1] = 1;
res[0][0] = 1;
res[1][0] = 1;
for (int i = k1 + 1; i <= n; i++)
{
    res[0][i] = res[1][i - 1];
    for (int k = k1; k <= k2; k++)
    {
        if (i - k >= 0)
            res[1][i] += res[0][i - k];
    }
}
cout << res[0][n] + res[1][n] << "\n";
```

Cut materials

```cpp
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) > H) || ((startCol + w - 1) > W)) return 0;
    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;
}
```

```cpp
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++)
    {
        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];
    });

    cout << backtrack(1) << "\n";

    return 0;
```

Balance courses

```cpp
int m, n;
int MIN = INT_MAX;
vector<vector<int>> course_lists;
vector<vector<int>> conflicts;
vector<int> assigns;
vector<int> loads;
int check(int course, int teacher)
{
```

```cpp
void assign(int course)
{
    if (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_assignmen
int main()
{
    cin >> 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);
        }
    }
    int k;
    cin >> k;
    conflicts.resize(n, vector<int>(n, 0));
    while (k--)
    {
        int i, j;
        cin >> i >> j;
        conflicts[i - 1][j - 1] = 1;
        conflicts[j - 1][i - 1] = 1;
    }
    assigns.resize(n, -1);
    loads.resize(m, 0);
    assign(0);
    cout << MIN << "\n";
    return 0;
}
```

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

```cpp
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>>,
                            greater<pair<long long, int>>> pq;
    dis[s] = 0;
    pq.push({0, s});

    for (int i = 0; i < n; i++)
    {
        int u = -1;

        while (!pq.empty()) {
            u = pq.top().second;
            pq.pop();
            if (!visited[u]) break;
        }
        if (u == -1 || dis[u] == INT_MAX) 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;
}
```

```cpp
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 inf
for (int j = 1; 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)

```cpp
int n, m;
vector<vector<int>> adj;
vector<int> d;
vector<int> visited;
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;
    d.resize(n + 1, 0);
    f.resize(n + 1, 0);
    for (int i = 1; i <= n; i++)
    {
        cin >> d[i];
    }
    adj.resize(n + 1, vector<int>());
    visited.resize(n + 1, 0);
    // 2: done, 1: current, 0: have not visited yet
    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]) 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

```cpp
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);
        }
    }

    int result = 0;
    for (int task = 1; task <= n; task++)
    {
        visited.assign(m + 1, false);
        if (dfs(task))
        {
            result++;
        }
    }

    cout << result << "\n";

    return 0;
}
```

SCC

```cpp
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])
    {
        cnt++;
        vector<int> res1;
        while (!SCC.empty())
        {
            int v = SCC.top();
            SCC.pop();
            onStack[v] = false;

            res1.push_back(v);
            // cout << v << " ";
            if (v == u)
            {
                sort(res1.begin(), res1.end());
                res.push_back(res1);
                // cout << "\n";
                break;
            }
        }
    }
}
int main()
{
    cin >> n >> m;
    adj.resize(n + 1, vector<int>());
    num.resize(n + 1, 0);
    low.resize(n + 1, 0);
    onStack.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 (num[i] == 0) {
            dfs(i);
        }
    }
    cout << cnt << "\n";

    sort(res.begin(), res.end());
    for (auto i : res)
    {
        for (auto j : i)
        {
            cout << j << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

Hungary

```cpp
const long long INF = 1e18; // Giá trị vô cực lớn cho long long

int n, m, k, sz;
vector<vector<long long>> a; // Ma trận chi phí (đã đảo dấu)
vector<long long> u, v, minv;
vector<int> p, way; // p: mảng matching, way: mảng 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++)        You, 15 hours ago • U
    {
        p[0] = i;
        int j0 = 0;
        minv.assign(sz + 1, INF);
        vector<bool> used(sz + 1, false);

        do
        {
            used[j0] = true;
            int i0 = p[j0];
            long long delta = INF;
            int j1;

            for (int j = 1; j <= sz; j++)
            {
                if (!used[j])
                {
                    long long cur = a[i0][j] - u[i0] - v[j];
                    if (cur < minv[j])
                    {
                        minv[j] = cur;
                        way[j] = j0;
                    }
                    if (minv[j] < delta)
                    {
                        delta = minv[j];
                        j1 = j;
                    }
                }
            }
            for (int j = 0; j <= sz; j++)
            {
                if (used[j])
                {
                    u[p[j]] += delta;
                    v[j] -= delta;
                }
                else
                {
                    minv[j] -= delta;
                }
            }
            j0 = j1;
        } while (p[j0] != 0);

        do
        {
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0 != 0);
    }
    cout << v[0] << "\n";

    return 0;
```

Edmond-karp (max flow)

```cpp
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

```cpp
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(make_pair(v, w));
        a[v].push_back(make_pair(u, w));
    }
    for (int i = 1; i <= n; i++)
    {
        distances[i] = 0;
    }
    bfs(1);
    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++)
    {        You, last week • all
        if (Max < distances[i])
        {
            Max = distances[i];
        }
    }
    cout << Max << "\n";
    return 0;
}
```

Total-path-lengths

```cpp
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[1] += 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); // or +1
            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)
        {
            // Cần tính answer[v] TRƯỚC khi gọi đệ quy xuống v
            answer[v] = answer[u] - w * subtree_size[v] + w * (n - subtree_size[v]); // 1 : weight
            dfs2(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;
}
```