ABC 061 解説

writer: Hec

2017/05/13

A: Between Two Integers

3 つの整数 A, B, C を入力として受け取ります。問題文の通り、$A \leq C$ と $C \leq B$ の 2 つの条件を同時に満たしているか判定を行います。最後に、条件を満たしている場合は「Yes」、そうでない場合は「No」を出力します。

C++のコード例

```
int main(void) {
    int A, B, C;
    cin >> A >> B >> C;
    if (A <= C and C <= B)
        cout << "Yes" << endl;
    else
        cout << "No" << endl;
    return 0;
}
```

B: Counting Roads

各都市から他の都市に何本の道路が伸びているかを調べます。まず、ループを使いて $i(1 \leq i \leq N)$ 番目の都市に注目します。次に、ループを用いて全ての道路を調べていき、道路の両端に i 番目の都市が含まれているかを判定し、数えていきます。その後に、数えた道路の本数を $i(1 \leq i \leq N)$ 番目の都市から伸びている道路の本数として出力します。これらの操作は、2 重ループを用いることで実装ができます。時間計算量は $O(NM)$ となり、これは間に合います。

配列を利用すると、より高速に答えを求めることが可能です。まず、各都市から何本の道路が伸びているかを管理する長さ N の配列 road を用意し、全ての要素を 0 に初期化します。次に、ループを使って全ての道路について調べていきます。この時、ある道路の両端が都市 a と 都市 b だったときに、$\mathrm{road}[a-1]$ と $\mathrm{road}[b-1]$ の値を 1 増やします。最後に、各都市から何本の道路が伸びているかを road を用いて出力します。この解法の時間計算量は、$O(N+M)$ となります。

C++のコード例 (時間計算量 $O(N+M)$)

```cpp
int main(void) {
    int N, M;
    cin >> N >> M;

    const int NMMAX = 50;
    int A[NMMAX], B[NMMAX];

    for (int i = 0; i < M; ++i) {
        cin >> A[i] >> B[i];
    }

    int road[NMMAX];
    for (int i = 0; i < N; ++i) {
        road[i] = 0;
    }

    for (int i = 0; i < M; ++i) {
        road[A[i] - 1] += 1;
        road[B[i] - 1] += 1;
    }

    for (int i = 0; i < N; ++i) {
        cout << road[i] << endl;
    }

    return 0;
}
```

C: Big Array

まず、この問題で求めたい答えは、入力によって生成される配列の小さい方から K 番目の値です。圧縮された入力を展開して元の配列を求めると、その要素数は最大で 10^{10} となるため MLE となります。

そこで、入力を展開せずに K 番目の値を求める方法を考えます。ここでは、バケツソートを用いた解法について説明します。この解法では、配列の値である a_i の範囲は 1 から 10^5 までと小さいことに注目します。まず、長さ 10^5 の配列 num を用意し、全ての要素を 0 に初期化します。次に、ループを使って $\mathrm{num}[a_i]$ に b_i を加算します。最後に、ループを用いて 1 から 10^5 まで調べていき、K 番目の値を求めます。この解法の時間計算量は $O(N + \max A_i)$ となるため間に合います。

また、配列の代わりに pair を用いたソートでも同じようにで解くことができます。その場合の時間計算量は $O(N \log N)$ となるため間に合います。なお、展開後の配列の最大要素数は 10^{10} であるため、32bit 整数型によるオーバーフローに注意してください。

C++のコード例 (バケツソート)

```cpp
using ll = long long;
const int AMAX = 100000;
ll cnt[AMAX + 1];

int main(void) {
    int N;
    ll K;
    cin >> N >> K;

    for (int i = 0; i < N; ++i) {
        int A, B;
        cin >> A >> B;
        cnt[A] += B;
    }

    for (int ans = 1; ans <= AMAX; ++ans){
        if (K <= cnt[ans]) {
            cout << ans << endl;
            break;
        }
        K -= cnt[ans];
    }

    return 0;
}
```

D: Score Attack

まず、問題で与えられるスコアの正負を逆にして、ゲームの最終的なスコアを最小化すると考えてみます。そうすると、この問題は、スコアを距離とみなした頂点 1 から頂点 N への最短経路問題とみなすことができます。最短経路問題を解くための有名なアルゴリズムには、ダイクストラ法、ベルマンフォード法、ワーシャルフロイド法が存在します。今回の問題では、負のコストの辺が存在するために、ダイクストラ法を適用できません。また、ワーシャルフロイド法の時間計算量は $O(N^3)$ であるため厳しいです（C++なら通る可能性があります）。そこで、時間計算量 $O(NM)$ であるベルマンフォード法をもとに解法を考えていきます。

ここで、最短距離を表す長さ N の配列 dist を用意して、最短距離の 1 回の更新を次のように定義します。

> 全ての辺に注目して、頂点 a_i の最短距離（$\mathrm{dist}[a_i]$）とコスト c_i から頂点 b_i の最短距離（$\mathrm{dist}[b_i]$）を更新する。

負閉路（辺のコストの総和が負となる閉路）がない場合には、最短距離の更新を $N-1$ 回繰り返すことで最短経路を求めることができます。なぜなら、この最短経路において各頂点は高々 1 回しか登場しないからです（2 回以上登場したら閉路ができる）。次に、負閉路の検出について考えてみます。N 回目以降の更新でも最短距離をより短くできれば、その経路上には同じ頂点が 2 回以上登場しているので閉路があると言えます。そして、閉路の存在と最短距離を更新できたことから、負閉路があると言えます。

これらの事実を利用して、次のような解法が考えられます。

1. 頂点 1 の最短距離を $\mathrm{dist}[1]=0$、その他の頂点 v の最短距離を $\mathrm{dist}[v]=\infty$ と初期化します。
2. 最短距離の更新を $N-1$ 回繰り返します（経路の長さは最大で $N-1$ であるため）。
3. 頂点 N の最短距離を表す変数として ans=dist[N] とします。
4. 次に、負閉路を検出するための長さ N の配列 negative を用意して、false で初期化します。
5. 最短距離の更新を N 回繰り返す（負閉路の長さは最大で N であるため）。ただし、このとき更新された頂点 b_i について、negative[b_i]=true とします。また、negative[a_i] が true の場合には、negative[b_i] を true にします。

そして、negative[N] が true になっている場合には「inf」、そうでない場合には $-$ans を出力します。この問題の場合には、$\infty > -N \min c_i$ となるように設定すれば十分です。この解法の時間計算量は $O(NM)$ となり、十分間に合います。

C++のコード例（頂点番号を 0-indexed とする）

```
using ll = long long;
const ll INF = 1LL << 50;


int main(void) {
    int N, M;
    cin >>  N >> M;

    const int NMAX = 1000;
    const int MMAX = 2000;
    int a[MMAX], b[MMAX];
    ll c[MMAX];

    for (int i = 0; i < M; ++i) {
        cin >> a[i] >> b[i] >> c[i];
        c[i] = -c[i];
    }

    ll dist[NMAX];

    for (int i = 0; i < N; ++i) {
        dist[i] = INF;
    }

    dist[0] = 0;


    for (int loop = 0; loop < N - 1; ++ loop) {
        for (int i = 0; i < M; ++i) {
```

```
            if (dist[a[i] - 1] == INF) continue;

            if (dist[b[i] - 1] > dist[a[i] - 1] + c[i]) {
                dist[b[i] - 1] = dist[a[i] - 1] + c[i];
            }
        }
    }

    ll ans = dist[N - 1];


    bool negative[NMAX];

    for (int i = 0; i < N; ++i) {
        negative[i] = false;
    }

    for (int loop = 0; loop < N ; ++ loop) {
        for (int i = 0; i < M; ++i) {
            if (dist[a[i] - 1] == INF) continue;

            if (dist[b[i] - 1] > dist[a[i] - 1] + c[i]) {
                dist[b[i] - 1] = dist[a[i] - 1] + c[i];
                negative[b[i] - 1] = true;
            }

            if (negative[a[i] - 1] == true) {
                negative[b[i] - 1] = true;
            }
        }
    }


    if (negative[N - 1])
        cout << "inf" << endl;
    else
        cout << -ans << endl;

    return 0;
}
```