上下界网络流

2021-04-09 | 2021-04-16 | OI & ACM | 21
3.6k | 3 分钟

简介

无源汇上下界可行流(循环流)

模型：现在有一个无源汇的网络，求出一个流，使得每个点满足流量平衡，并且每条边的流量有上下界

分析：

令 $in[u]$ 表示流入 u 这个点的所有边的下界之和， $out[u]$ 表示 u 这个点流出的所有边的下界之和

首先我们注意到最终每条边的流量一定大于等于下界，那么我们不妨先令每条边的流量都为下界

但是我们注意到如果 $in[u] \neq out[u]$ 的话，那么这张图是不满足流量守恒的，所以我们考虑在残量网络上再求一个流，使得这两个流叠加得到可行流

如果 $in[u] > out[u]$，说明新的流中流入 u 的要小于流出 u 的

如果 $in[u] = out[u]$，说明新的流中流入 u 的要等于流出 u 的

如果 $in[u] < out[u]$，说明新的流中流入 u 的要大于流出 u 的

那么在我们现有的网络流算法的限制下，我们如果求这样一个流量依然不守恒的流呢

或者换句话说，以 $in[u] > out[u]$ 为例，流入 u 的要小于流出 u 的，那么小于的那一部分流量从哪里来呢

这个时候我们能够想到源点和汇点是不受流量守恒限制的，所以我们考虑将这一部分流量连到 ss 和 tt 上

那么我们能够得到这样的建图：

对于原图中的边 $u \to v$，u 连 v，容量为 上界 - 下界

对于每个点 u，如果 $in[u] > out[u]$，则 ss 连 u，容量为 $in[u] - out[u]$

对于每个点 u，如果 $out[u] > in[u]$，则 u 连 tt，容量为 $out[u] - in[u]$

原图存在可行流当且仅当新图满流（从 ss 连出的边都满流

原图中的每条边的流量为容量下界 + 在新图中这条边的流量

```cpp
#include <iostream>
#include <queue>
#define maxn 2010
#define INF 1000000000
using namespace std;

int n, m;

struct Edge {
    int to, next, w;
} e[1000000]; int c1, head[maxn], in[maxn], out[maxn];
inline void add_edge(int u, int v, int w) {
    e[c1].to = v; e[c1].w = w;
    e[c1].next = head[u]; head[u] = c1++;
}

inline void Add_edge(int u, int v, int l, int r) {
    add_edge(u, v, r - l); add_edge(v, u, 0);
    in[v] += l; out[u] += l;
}

int s, t, ss, tt, dep[maxn];
bool bfs() {
    fill(dep, dep + maxn, 0); dep[s] = 1;
    queue<int> Q; Q.push(s);
    while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        for (int i = head[u]; ~i; i = e[i].next) {
```

```
            int v = e[i].to, w = e[i].w;
            if (w > 0 && !dep[v]) {
                dep[v] = dep[u] + 1;
                if (v == t) return 1; Q.push(v);
            }
        }
    }
    return 0;
}

int dfs(int u, int _w) {
    if (u == t || !_w) return _w;
    int s = 0;
    for (int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].to, w = e[i].w;
        if (dep[v] == dep[u] + 1 && w > 0) {
            int di = dfs(v, min(_w - s, w));
            e[i].w -= di; e[i ^ 1].w += di;
            s += di; if (s == _w) break;
        }
    }
    if (s < _w) dep[u] = 0;
    return s;
}

int dinic() {
    int mf = 0;
    while (bfs()) mf += dfs(s, INF);
    return mf;
}

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

    cin >> n >> m; s = 0; t = n + 1;
    for (int i = 1; i <= m; ++i) {
        int x, y, l, r; cin >> x >> y >> l >> r;
        Add_edge(x, y, l, r);
    } int mf = 0;
    for (int i = 1; i <= n; ++i) {
        if (in[i] > out[i]) Add_edge(s, i, 0, in[i] - out[i]);
        else Add_edge(i, t, 0, out[i] - in[i]);
        mf += max(0, in[i] - out[i]);
    }
    cout << (dinic() == mf);
```

```
75      return 0;
76  }
77
```

有源汇上下界可行流

模型：现在有一个有源汇的网络，求出一个流，使得除源汇之外每个点满足流量平衡，并且每条边的流量有上下界

分析：

在无源汇上下界可行流的基础上再连一条 t 到 s 的边，下界 0，上界 ∞

显然，因为源汇不满足流量平衡，所以不能用循环流的方法，但如果让源汇满足流量平衡，就可以使用了

注意：原图的可行流的流量为 t 到 s 这条边的流量，每条边的流量和循环流是一样的

有源汇上下界最大流

模型：现在有一个有源汇的网络，求出一个最大流，使得除源汇之外每个点满足流量平衡，并且每条边的流量有上下界

分析：

在可行流的基础上，只需将 ss 和 tt 的所有边都删掉，并且将 t 到 s 这条边删掉，然后在跑一个 s 到 t 的最大流

我门发现，原图已经是一个可行流了，即已经满足下界，把 ss 和 tt 删掉之后，在图上无论做什么，都不会再影响下界，所以跑一遍最大流是可以出答案的，且不会影响下界

注意：最大流的大小 = 可行流的大小 + s 到 t 的最大流的大小

```
1   int main() {
2       ios::sync_with_stdio(false);
3       cin.tie(nullptr); cout.tie(nullptr);
4
5       cin >> n >> m; s = 0; t = n + 1; ss = n + 2; tt = n + 3;
6       for (int i = 1; i <= m; ++i) {
```

```
        int x, y, l, r; cin >> x >> y >> l >> r;
        Add_edge(x, y, l, r);
    }
    Add_edge(t, s, 0, INF); int mf = 0;
    for (int i = s; i <= t; ++i) {
        if (in[i] > out[i]) Add_edge(ss, i, 0, in[i] - out[i]);
        else Add_edge(i, tt, 0, out[i] - in[i]);
        mf += max(0, in[i] - out[i]);
    } swap(s, ss); swap(t, tt);
    if (dinic() != mf) return cout << "-1\n", 0;
    swap(s, ss); swap(t, tt); int ans = 0;
    for (int i = head[s]; ~i; i = e[i].next)
        if (e[i].to == t) ans = e[i].w, e[i].w = e[i ^ 1].w = 0;
    for (int i = head[ss]; ~i; i = e[i].next) e[i].w = e[i ^ 1].w = 0;
    for (int i = head[tt]; ~i; i = e[i].next) e[i].w = e[i ^ 1].w = 0;
    cout << ans + dinic() << "\n";
    return 0;
}
```

有源汇上下界最小流

按照有源汇上下界可行流建图，先跑一个可行流

然后把 ss 和 tt 以及 t 到 s 的边删掉，然后跑一遍 t 到 s 的最大流

例题

Luogu P4553 80人环游世界 有源汇上下界最小费用最大流

---------- 本文结束 感谢阅读 ----------

Tech # 网络流 # 上下界网络流

‹ CF 1175E Minimal Segment Cover

Luogu P6054 [RC-02] 开门大吉 ›