# 图论

## bfs/dfs 版 SPFA 判负环（用来卡常）

```
ll dis[maxn];
int cnt[maxn];
bool spfa(double k)
{
    memset(dis, 0x3f, sizeof(dis));
    queue<int> q;
    for(int i=1;i<=n;i++) q.push(i);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        for(auto&& [v,w] : G[u])
        {
            auto t = dis[u]+w;
            if(t<dis[v])
            {
                dis[v] = t;
                cnt[v] = cnt[u]+1;
                if(cnt[v]==n) return true;
                q.push(v);
            }
        }
    }
    return false;
}
bool spfa(int u)
{
    vis[u] = true;
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(dis[v]>dis[u]+Edge[i].w)
        {
            dis[v] = dis[u]+Edge[i].w;
            if(vis[v]||spfa(v)) return true;
        }
    }
    vis[u] = false;
    return false;
}
```

### 二分图最大匹配

- 一个图没有奇环当且仅当是一个二分图

### 匈牙利算法

- 枚举每一个左部结点 $u$;
  - 枚举 $u$ 结点的邻接结点 $v$, 对于每个结点 $v$
    - 如果 $v$ 没有被匹配过, 那么直接让 $u, v$ 匹配;
    - 否则让原来匹配 $v$ 的结点 $u'$ 去尝试匹配其他右部结点;
      - 如果 $u'$ 匹配到了其他结点 $v'$, 那么 $u, v$ 匹配, $u', v'$ 匹配;
      - 否则 $u$ 失配。

```
vector<int> G[maxn];
int vis[maxn], match[maxn];
bool dfs(int u,int tag)
{
    if(vis[u]==tag) return false;
    vis[u] = tag;
    for(auto v : G[u])
    {
        if(!match[v]||dfs(match[v],tag))
        {
            match[v] = u;
            return true;
        }
    }
    return false;
}
for(int i=1;i<=n;i++) if(dfs(i,i)) ans++;
```

### 强连通分量

```
vector<vector<int>> G(n+1);
int tim = 0;
vector<int> dfn(n+1), low(n+1);
vector<bool> vis(n+1);
stack<int> st;
int cnt = 0;
vector<int> scc(n+1);  // output: {scc: scc id}
function<void(int)> tarjan = [&](int u)
{
    low[u] = dfn[u] = ++tim;
    st.push(u);
    vis[u] = true;
    for(auto v : G[u])
    {
        if(!dfn[v])
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if(vis[v]) low[u] = min(low[u], dfn[v]);
    }
    if(dfn[u]==low[u])
    {
        cnt++;
        int v;
        do
        {
            v = st.top();
```

```
            st.pop();
            scc[v] = cnt;
            vis[v] = false;
        }
        while(v!=u);
    }
};
for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i);
//* 以下为缩点，注意直接这样会有重边
vector<vector<int>> H(cnt+1);
for(int u=1;u<=n;u++)
    for(auto v : G[u])
        if(scc[u]!=scc[v])
            H[scc[u]].push_back(scc[v]);
//! 缩点后结点排列顺序是逆拓扑序，不需要再次拓排
for(int u=cnt;u>=1;u--) // do something...(按拓扑序处理)
```

### 点双连通分量与割点

```
vector<vector<int>> G(n+1);
int tim = 0;
vector<int> dfn(n+1), low(n+1);
stack<int> st;
vector<bool> cut(n+1); // output: {cut: is_cut_vertex}
vector<vector<int>> bcc; // output: {bcc: nodes in bcc_i}
function<void(int,int)> tarjan = [&](int u,int f)
{
    low[u] = dfn[u] = ++tim;
    st.push(u);
    if(!f&&G[u].empty()) // 孤立点
    {
        bcc.emplace_back(1,u);
        return;
    }
    int s = 0;
    for(auto v : G[u])
    {
        if(!dfn[v])
        {
            tarjan(v,u);
            low[u] = min(low[u], low[v]);
            if(low[v]>=dfn[u])
            {
                s++;
                if(f||s>1) cut[u] = true;
                bcc.emplace_back();
                int w;
                do
                {
                    w = st.top();
                    st.pop();
                    bcc.back().push_back(w);
                }
                while(w!=v);
                bcc.back().push_back(u);
            }
```

```
        }
        else low[u] = min(low[u], dfn[v]);
    }
};
//! 注意: 请保证图中不存在自环
for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i,0);
```

### 边双连通分量与桥

```
typedef pair<int,int> pii;
vector<vector<pii>> G(n+1); //! edge: <to, id>
int tim = 0;
vector<int> dfn(n+1), low(n+1);
vector<bool> cut(m+1); // output: {cut: is_cut_edge}
// 求桥
function<void(int,int)> tarjan = [&](int u,int f)
{
    low[u] = dfn[u] = ++tim;
    for(auto [v,id] : G[u])
    {
        if(!dfn[v])
        {
            tarjan(v,u);
            low[u] = min(low[u], low[v]);
            if(low[v]>dfn[u]) cut[id] = true;
        }
        else if(v!=f) low[u] = min(low[u], dfn[v]);
    }
};
// 求边双连通分量
int cnt = 0;
vector<int> bcc(n+1); // output: {bcc: bcc id}
function<void(int)> dfs = [&](int u)
{
    bcc[u] = cnt;
    for(auto [v,id] : G[u])
    {
        if(bcc[v]||cut[id]) continue;
        dfs(v);
    }
};
for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i,0);
for(int i=1;i<=n;i++) if(!bcc[i]) cnt++, dfs(i);
```