


常用代码模板3——搜索与图论

作者:  YXC (<https://www.acwing.com/user/myspace/index/1/>), 2019-07-31 21:51:16, 阅读 12047

瀛

算法基础课相关代码模板

114

- 活动链接 —— 算法基础课 (<https://www.acwing.com/activity/content/11/>)

瀛

树与图的存储

→

168

树是一种特殊的图，与图的存储方式相同。

对于无向图中的边 ab ，存储两条有向边 $a \rightarrow b, b \rightarrow a$ 。

因此我们可以只考虑有向图的存储。

(1) 邻接矩阵: $g[a][b]$ 存储边 $a \rightarrow b$

(2) 邻接表:

```
// 对于每个点k，开一个单链表，存储k所有可以走到的点。h[k]存储这个单链表的头结点
int h[N], e[N], ne[N], idx;
```

```
// 添加一条边a->b
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}
```

```
// 初始化
idx = 0;
memset(h, -1, sizeof h);
```

树与图的遍历

时间复杂度 $O(n + m)$, n 表示点数, m 表示边数

(1) 深度优先遍历 —— 模板题 AcWing 846. 树的重心 (<https://www.acwing.com/problem/content/848/>)

```
int dfs(int u)
{
    st[u] = true; // st[u] 表示点u已经被遍历过

    for (int i = h[u]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (!st[j]) dfs(j);
    }
}
```

(2) 宽度优先遍历 —— 模板题 AcWing 847. 图中点的层次 (<https://www.acwing.com/problem/content/849/>)

```

queue<int> q;
st[1] = true; // 表示1号点已经被遍历过
q.push(1);

while (q.size())
{
    int t = q.front();
    q.pop();

    for (int i = h[t]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (!st[j])
        {
            st[j] = true; // 表示点j已经被遍历过
            q.push(j);
        }
    }
}

```

拓扑排序 —— 模板题 AcWing 848. 有向图的拓扑序列 (<https://www.acwing.com/problem/content/850/>)

时间复杂度 $O(n + m)$, n 表示点数, m 表示边数

```

bool topsort()
{
    int hh = 0, tt = -1;

    // d[i] 存储点i的入度
    for (int i = 1; i <= n; i++)
        if (!d[i])
            q[++tt] = i;

    while (hh <= tt)
    {
        int t = q[hh++];

        for (int i = h[t]; i != -1; i = ne[i])
        {
            int j = e[i];
            if (--d[j] == 0)
                q[++tt] = j;
        }
    }

    // 如果所有点都入队了, 说明存在拓扑序列; 否则不存在拓扑序列。
    return tt == n - 1;
}

```

朴素dijkstra算法 —— 模板题 AcWing 849. Dijkstra求最短路径 I (<https://www.acwing.com/problem/content/851/>)

时间复杂度是 $O(n^2 + m)$, n 表示点数, m 表示边数

```

int g[N][N]; // 存储每条边
int dist[N]; // 存储1号点到每个点的最短距离
bool st[N]; // 存储每个点的最短路是否已经确定

// 求1号点到n号点的最短路，如果不存在则返回-1
int dijkstra()
{
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    for (int i = 0; i < n - 1; i++)
    {
        int t = -1; // 在还未确定最短路的点中，寻找距离最小的点
        for (int j = 1; j <= n; j++)
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;

        // 用t更新其他点的距离
        for (int j = 1; j <= n; j++)
            dist[j] = min(dist[j], dist[t] + g[t][j]);

        st[t] = true;
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}

```

堆优化版dijkstra —— 模板题 AcWing 850. Dijkstra求最短路 II
[\(https://www.acwing.com/problem/content/852/\)](https://www.acwing.com/problem/content/852/)

时间复杂度 $O(m \log n)$, n 表示点数, m 表示边数

```

typedef pair<int, int> PII;

int n; // 点的数量
int h[N], w[N], e[N], ne[N], idx; // 邻接表存储所有边
int dist[N]; // 存储所有点到1号点的距离
bool st[N]; // 存储每个点的最短距离是否已确定

// 求1号点到n号点的最短距离，如果不存在，则返回-1
int dijkstra()
{
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, 1}); // first存储距离，second存储节点编号

    while (heap.size())
    {
        auto t = heap.top();
        heap.pop();

        int ver = t.second, distance = t.first;

        if (st[ver]) continue;
        st[ver] = true;

        for (int i = h[ver]; i != -1; i = ne[i])
        {
            int j = e[i];
            if (dist[j] > distance + w[i])
            {
                dist[j] = distance + w[i];
                heap.push({dist[j], j});
            }
        }
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}

```

Bellman-Ford算法 —— 模板题 AcWing 853. 有边数限制的最短路 (<https://www.acwing.com/problem/content/855/>)

时间复杂度 $O(nm)$, n 表示点数, m 表示边数

注意在模板题中需要对下面的模板稍作修改，加上备份数组，详情见模板题。

```

int n, m;    // n表示点数，m表示边数
int dist[N];    // dist[x]存储1到x的最短路距离

struct Edge    // 边，a表示出点，b表示入点，w表示边的权重
{
    int a, b, w;
}edges[M];

// 求1到n的最短路距离，如果无法从1走到n，则返回-1。
int bellman_ford()
{
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    // 如果第n次迭代仍然会松弛三角不等式，就说明存在一条长度是n+1的最短路径，由抽屉原理，路径中至少存在两个相同的点，说明图中存在负权回路。
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            int a = edges[j].a, b = edges[j].b, w = edges[j].w;
            if (dist[b] > dist[a] + w)
                dist[b] = dist[a] + w;
        }
    }

    if (dist[n] > 0x3f3f3f3f / 2) return -1;
    return dist[n];
}

```

spfa 算法（队列优化的Bellman-Ford算法）—— 模板题 AcWing 851. spfa求最短路
<https://www.acwing.com/problem/content/853/>

时间复杂度 平均情况下 $O(m)$ ，最坏情况下 $O(nm)$ ， n 表示点数， m 表示边数

```

int n;    // 总点数
int h[N], w[N], e[N], ne[N], idx;    // 邻接表存储所有边
int dist[N];    // 存储每个点到1号点的最短距离
bool st[N];    // 存储每个点是否在队列中

// 求1号点到n号点的最短路距离，如果从1号点无法走到n号点则返回-1
int spfa()
{
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    queue<int> q;
    q.push(1);
    st[1] = true;

    while (q.size())
    {
        auto t = q.front();
        q.pop();

        st[t] = false;

        for (int i = h[t]; i != -1; i = ne[i])
        {
            int j = e[i];
            if (dist[j] > dist[t] + w[i])
            {
                dist[j] = dist[t] + w[i];
                if (!st[j])    // 如果队列中已存在j，则不需要将j重复插入
                {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}

```

spfa判断图中是否存在负环 —— 模板题 AcWing 852. spfa判断负环
<https://www.acwing.com/problem/content/854/>

时间复杂度是 $O(nm)$, n 表示点数, m 表示边数

```

int n; // 总点数
int h[N], w[N], e[N], ne[N], idx; // 邻接表存储所有边
int dist[N], cnt[N]; // dist[x]存储1号点到x的最短距离, cnt[x]存储1到x的最短路中经过的点数
bool st[N]; // 存储每个点是否在队列中

// 如果存在负环, 则返回true, 否则返回false。
bool spfa()
{
    // 不需要初始化dist数组
    // 原理: 如果某条最短路径上有n个点(除了自己), 那么加上自己之后一共有n+1个点, 由抽屉原理一定有两个点相同, 所以存在环。

    queue<int> q;
    for (int i = 1; i <= n; i++)
    {
        q.push(i);
        st[i] = true;
    }

    while (q.size())
    {
        auto t = q.front();
        q.pop();

        st[t] = false;

        for (int i = h[t]; i != -1; i = ne[i])
        {
            int j = e[i];
            if (dist[j] > dist[t] + w[i])
            {
                dist[j] = dist[t] + w[i];
                cnt[j] = cnt[t] + 1;
                if (cnt[j] >= n) return true; // 如果从1号点到x的最短路中包含至少n个点(不包括自己), 则说明存在环
                if (!st[j])
                {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }

    return false;
}

```

floyd算法 —— 模板题 AcWing 854. Floyd求最短路 (<https://www.acwing.com/problem/content/856/>)

时间复杂度是 $O(n^3)$, n 表示点数

```

初始化:
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        if (i == j) d[i][j] = 0;
        else d[i][j] = INF;

// 算法结束后, d[a][b]表示a到b的最短距离
void floyd()
{
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}

```

朴素版prim算法 —— 模板题 AcWing 858. Prim算法求最小生成树

(<https://www.acwing.com/problem/content/860/>)

时间复杂度是 $O(n^2 + m)$, n 表示点数, m 表示边数

```
int n; // n表示点数
int g[N][N]; // 邻接矩阵, 存储所有边
int dist[N]; // 存储其他点到当前最小生成树的距离
bool st[N]; // 存储每个点是否已经在生成树中

// 如果图不连通, 则返回INF(值是0x3f3f3f3f), 否则返回最小生成树的树边权重之和
int prim()
{
    memset(dist, 0x3f, sizeof dist);

    int res = 0;
    for (int i = 0; i < n; i++)
    {
        int t = -1;
        for (int j = 1; j <= n; j++)
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;

        if (i && dist[t] == INF) return INF;

        if (i) res += dist[t];
        st[t] = true;

        for (int j = 1; j <= n; j++) dist[j] = min(dist[j], g[t][j]);
    }

    return res;
}
```

Kruskal算法 —— 模板题 AcWing 859. Kruskal算法求最小生成树
(<https://www.acwing.com/problem/content/861/>)

时间复杂度是 $O(m \log m)$, n 表示点数, m 表示边数


```

int n, m;    // n是点数，m是边数
int p[N];    // 并查集的父节点数组

struct Edge  // 存储边
{
    int a, b, w;

    bool operator< (const Edge &W)const
    {
        return w < W.w;
    }
}edges[M];

int find(int x)  // 并查集核心操作
{
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int kruskal()
{
    sort(edges, edges + m);

    for (int i = 1; i <= n; i++) p[i] = i;  // 初始化并查集

    int res = 0, cnt = 0;
    for (int i = 0; i < m; i++)
    {
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;

        a = find(a), b = find(b);
        if (a != b)  // 如果两个连通块不连通，则将这两个连通块合并
        {
            p[a] = b;
            res += w;
            cnt++;
        }
    }

    if (cnt < n - 1) return INF;
    return res;
}

```

染色法判别二分图 —— 模板题 AcWing 860. 染色法判定二分图
[\(https://www.acwing.com/problem/content/862/\)](https://www.acwing.com/problem/content/862/)

时间复杂度是 $O(n + m)$, n 表示点数, m 表示边数

```
int n;    // n表示点数
int h[N], e[M], ne[M], idx;  // 邻接表存储图
int color[N];    // 表示每个点的颜色, -1表示未染色, 0表示白色, 1表示黑色
```

// 参数: u表示当前节点, c表示当前点的颜色

```
bool dfs(int u, int c)
{
    color[u] = c;
    for (int i = h[u]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (color[j] == -1)
        {
            if (!dfs(j, !c)) return false;
        }
        else if (color[j] == c) return false;
    }

    return true;
}
```

```
bool check()
{
    memset(color, -1, sizeof color);
    bool flag = true;
    for (int i = 1; i <= n; i++)
        if (color[i] == -1)
            if (!dfs(i, 0))
            {
                flag = false;
                break;
            }
    return flag;
}
```

匈牙利算法 —— 模板题 AcWing 861. 二分图的最大匹配 (<https://www.acwing.com/problem/content/863/>)

时间复杂度是 $O(nm)$, n 表示点数, m 表示边数

```

int n1, n2; // n1表示第一个集合中的点数, n2表示第二个集合中的点数
int h[N], e[M], ne[M], idx; // 邻接表存储所有边, 匈牙利算法中只会用到从第一个集合指向第二个集合的边, 所以这里只用存一个方向的边
int match[N]; // 存储第二个集合中的每个点当前匹配的第一个集合中的点是哪个
bool st[N]; // 表示第二个集合中的每个点是否已经被遍历过

bool find(int x)
{
    for (int i = h[x]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (!st[j])
        {
            st[j] = true;
            if (match[j] == 0 || find(match[j]))
            {
                match[j] = x;
                return true;
            }
        }
    }
}

return false;
}


// 求最大匹配数, 依次枚举第一个集合中的每个点能否匹配第二个集合中的点
int res = 0;
for (int i = 1; i <= n1; i++)
{
    memset(st, false, sizeof st);
    if (find(i)) res++;
}


```


评论列表:


在这里写评论... (支持Markdown和Latex语法)

提交评论

 fedfan (<https://www.acwing.com/user/myspace/index/12024/>) 2个月前 回复
朴素版prim算法和dijkstra算法时间复杂度是 $O(n^2+m)$, 为什么基础课讲的是 $O(n^2)$?
(<https://www.acwing.com/user/myspace/index/12024/>)


 yxc (<https://www.acwing.com/user/myspace/index/1/>) 1个月前 回复
(<https://www.acwing.com/user/myspace/index/1/>)

 zdw (<https://www.acwing.com/user/myspace/index/16448/>) 2个月前 回复
赞, 果断收藏
(<https://www.acwing.com/user/myspace/index/16448/>)

 yxc (<https://www.acwing.com/user/myspace/index/1/>) 2个月前 回复
(<https://www.acwing.com/user/myspace/index/1/>)

 Leo_Jose (<https://www.acwing.com/user/myspace/index/3536/>) 3个月前 回复
老师, 宽度优先遍历那个代码块的13行应该是
(<https://www.acwing.com/user/myspace/index/3536/>)
if(!st[j])而不是if(!s[j])把

 yxc (<https://www.acwing.com/user/myspace/index/1/>) 3个月前 回复
(<https://www.acwing.com/user/myspace/index/1/>)

 stfst (<https://www.acwing.com/user/myspace/index/1577/>) 3个月前 回复了 yxc 的评论 回
(<https://www.acwing.com/user/myspace/index/1577/>)
不一样啊,st和s

 yxc (<https://www.acwing.com/user/myspace/index/1/>) 3个月前 回复了 stfst 的评论 回复

(<https://www.acwing.com/user/myspace/index/1/>)



沧笙 (<https://www.acwing.com/user/myspace/index/24652/>) 3个月前 回复

为啥prim和dijkstra循环次数不一样。???
(<https://www.acwing.com/user/myspace/index/24652/>)



yxc (<https://www.acwing.com/user/myspace/index/1/>) 3个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 算法很灵活，只要能保证正确，Dijkstra不用太纠结循环n次还是n-1一次。



RobotOlivaw (<https://www.acwing.com/user/myspace/index/8294/>) 2个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 是因为dijkstra最初将起点加进了集合中的原因吧。



yxc (<https://www.acwing.com/user/myspace/index/1/>) 2个月前 回复了 RobotOlivaw 的评论

(<https://www.acwing.com/user/myspace/index/1/>)

对滴。



牙疼 (<https://www.acwing.com/user/myspace/index/11683/>) 4个月前 回复

bellman-ford算法里面少了给backup数组
(<https://www.acwing.com/user/myspace/index/11683/>)



yxc (<https://www.acwing.com/user/myspace/index/1/>) 4个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 一般情况下不需要使用backup数组，不影响结果的正确性。



牙疼 (<https://www.acwing.com/user/myspace/index/11683/>) 5个月前 回复

匈牙利算法里面有句注释好像错了，应该是从第一个集合指向第二个集合吧？
(<https://www.acwing.com/user/myspace/index/11683/>)

// 邻接表存储所有边，匈牙利算法中只会用到从第二个集合指向第一个集合的边，所以这里只用存一个方向的边



yxc (<https://www.acwing.com/user/myspace/index/1/>) 4个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>)



qym2008 (<https://www.acwing.com/user/myspace/index/15881/>) 5个月前 回复

邻接表中ne[], e[] 是什么意思呀，蒟蒻刚学，麻烦了
(<https://www.acwing.com/user/myspace/index/15881/>)



yxc (<https://www.acwing.com/user/myspace/index/1/>) 5个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 邻接表用邻接链表表示，ne是链表中的下一个点的下标，e是邻点的编号。



yxc (<https://www.acwing.com/user/myspace/index/1/>) 5个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 可以参考AcWing 826. 单链表(<https://www.acwing.com/video/253/>)。



greg666 (<https://www.acwing.com/user/myspace/index/1768/>) 7个月前 回复

猛然发现，模板总结跟我代码打卡的风格类似呀。
(<https://www.acwing.com/user/myspace/index/1768/>)



yxc (<https://www.acwing.com/user/myspace/index/1/>) 7个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 很棒呀。



T-SHLoRk (<https://www.acwing.com/user/myspace/index/6314/>) 7个月前 回复

为什么SPFA判断负环需要加入所有的节点进入队列呢？是为了防止负环和源点不在一个连通块么？
(<https://www.acwing.com/user/myspace/index/6314/>)



yxc (<https://www.acwing.com/user/myspace/index/1/>) 7个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 对滴。



Darron (<https://www.acwing.com/user/myspace/index/12323/>) 7个月前 回复

总，bellman-ford算法中模板是不是有问题啊，少了memcpy(backup, dist, sizeof dist);
(<https://www.acwing.com/user/myspace/index/12323/>)



yxc (<https://www.acwing.com/user/myspace/index/1/>) 7个月前 回复

(<https://www.acwing.com/user/myspace/index/1/>) 这里是没有问题滴，一般的最短路问题也不需要备份距离数组，只有当有边数限制时才需要。



T-SHLoRk (<https://www.acwing.com/user/myspace/index/6314/>) 7个月前 回复了 yxc 的评论

(<https://www.acwing.com/user/myspace/index/6314/>)

BellFord感觉本质上就是一个滚动数组的动态规划问题呀。不过模板没写backup但是模板题有个k，觉得这个还是提一句比较好。



T-SHLoRk (<https://www.acwing.com/user/myspace/index/6314/>) 7个月前 回复了 T-SHLoRk

的评论。(<https://www.acwing.com/user/myspace/index/6314/>)

BTW，想问一下 BellFord算法里面为什么是判断dist[n] > 0x3f/2 ?而不是dist[n] = 0x3f



yxc (<https://www.acwing.com/user/myspace/index/1/>) 7个月前 回复了 T-SHLoRk 的评论 回

(<https://www.acwing.com/user/myspace/index/1/>)

已在模板中提示。



yxc (<https://www.acwing.com/user/myspace/index/1/>) 7个月前 回复了 T-SHLoRk 的评论 回

(<https://www.acwing.com/user/myspace/index/1/>)

因为存在负权边，所以当无解时 $\text{dist}[n]$ 可能等于 $0x3f3f3f3f$ 加上一个负权。



一苇 (<https://www.acwing.com/user/myspace/index/9927/>) 8个月前 回复

Y神。prim与dijkstra 我有些混乱了
(<https://www.acwing.com/user/myspace/index/9927/>)

yxc (<https://www.acwing.com/user/myspace/index/1/>) 8个月前 回复

好好理解，加油
(<https://www.acwing.com/user/myspace/index/1/>)

一苇 (<https://www.acwing.com/user/myspace/index/9927/>) 8个月前 回复了 yxc 的评论 回复

今天再看了然
(<https://www.acwing.com/user/myspace/index/9927/>)

yxc (<https://www.acwing.com/user/myspace/index/1/>) 8个月前 回复了一苇 的评论 回复

不错
(<https://www.acwing.com/user/myspace/index/1/>)

一苇 (<https://www.acwing.com/user/myspace/index/9927/>) 8个月前 回复了 yxc 的评论 回复

神啊，直接表那个成功，初始 $\text{dist}[n]$ 吧，9927 本渣渣没留意到这一点，一直超时，排查了好久 --!!
(<https://www.acwing.com/user/myspace/index/9927/>)

yxc (<https://www.acwing.com/user/myspace/index/1/>) 7个月前 回复了一苇 的评论 回复

加油
(<https://www.acwing.com/user/myspace/index/1/>)



在线白给 (<https://www.acwing.com/user/myspace/index/7412/>) 8个月前 回复

y总，染色法判断二分图模板里面的dfs函数里的”int father”这个形参好像没起到作用啊。
(<https://www.acwing.com/user/myspace/index/7412/>)

yxc (<https://www.acwing.com/user/myspace/index/1/>) 8个月前 回复

是的，已删除
(<https://www.acwing.com/user/myspace/index/1/>)



烛之武 (<https://www.acwing.com/user/myspace/index/1099/>) 10个月前 回复

y总，时间复杂度应该加上
(<https://www.acwing.com/user/myspace/index/1099/>)

yxc (<https://www.acwing.com/user/myspace/index/1/>) 10个月前 回复

加油
(<https://www.acwing.com/user/myspace/index/1/>)



ZYzzz (<https://www.acwing.com/user/myspace/index/6048/>) 10个月前 回复

赞赞赞赞赞赞赞！！！！！
(<https://www.acwing.com/user/myspace/index/6048/>)