图论基础

___debug

January 3, 2016

前言

这次主要讲图论的基础算法。
知识点会比较多，不过不怎么难。
建议每个算法都要自己实现几次。

定义

我们定义：

强连通图 每一个点皆可以抵达其他点的有向图

强连通分量 有向图的极大强连通子图

为了叙述方便，对于在一个有向图上运行 DFS 生成的深度优先森林，我们还定义：

树边 深度优先森林中的边

后向边 将 u 连接到其祖先的非树边

前向边 将 u 连接到其后代的非树边

横向边 其他所有的边，可以在同一个树中但没有祖先关系，也可以不在同一个树中

Tarjan 算法

强连通分量可以通过 Tarjan 算法在 $O(n+m)$ 内求出。

Tarjan 算法主要使用了 DFS，并充分利用了 DFS 过程的特性。

我们定义：

$dfn(u)$ 节点 u 的搜索次序编号（DFS 时间戳）

$low(u)$ 沿着零条或多条树边最后跟上至多一条后向边或横向边能到达的与 u 属于同一个强连通分量的点的最小 dfn。

Tarjan 算法

可以证明：

- 一个强连通分量的所有节点都在同一棵搜索树中
- u 是一个强连通分量的根当且仅当 $low(u) = dfn(u)$

从而问题转化为求出每个节点的 low 值。

Tarjan 算法

其实基本的方式就是边 DFS 边计算 *low* 值，而且把已经找出的强连通分量分离出来，那么剩下的也是一个强连通分量。

具体实现上，对于一条边 (u, v)：

- 若 v 未被访问过（即 (u, v) 为树边），则递归计算 $low(v)$ 并用结果更新 $low(u)$
- 否则，若 v 还没有属于任何一个强连通分量，则用 $dfn(v)$ 更新 $low(u)$

注意 DFS 的时候要维护一个栈以记录当前强连通分量。

实现的具体细节参见 scc_tarjan.cpp。

例题：[HDU2767]Proving Equivalences（白书 P322）

定义

我们定义：

割点 无向图中删除后连通分量增加的点

桥 无向图中删除后连通分量增加的边

点 -双连通图 删除任意一个点后仍连通的无向图

边 -双连通图 删除任意一条边后仍连通的无向图

点 -双连通分量 无向图的极大点 -双连通子图

边 -双连通分量 无向图的极大边 -双连通子图

Tarjan 算法

还是 Tarjan 算法……
同样是定义 $dfn(u)$ 与 $low(u)$。定义同有向图的强连通分量非常相似，不再赘述。

在深度优先森林中，当 u 为 v 的父亲时，显然有：

- 若 $low(v) \geq dfn(u)$，则 u 为割点（根节点需要特判）
- 若 $low(v) > dfn(u)$，则 (u, v) 为桥

从而，我们能很容易求出割点和桥。（求桥的时候一定要考虑重边）

Tarjan 算法

我们求出了割点和桥，那么怎样求出点 -双连通分量和边 -双连通分量呢？

其实点 -双连通分量和有向图的强连通分量的求法是一样的。两者都是用栈维护当前的连通分量。只不过有一点需要特别注意：由于割点会属于多个点 -双连通分量，故栈中应保存边而不是点。

实现的具体细节参见 bcc_tarjan.cpp。

对于边 -双连通分量也可以用类似的做法。
但更为直观的做法是先求出桥再 DFS 一遍，不经过桥即可。

例题：[HDU3844]Mining Your Own Business (白书 P318)

定义

给定 $x_1 \ldots x_n$ 个布尔变量，有若干形如 $x_i, x_i \vee x_j, \neg x_i \vee x_j$ 等的约束条件，求一组可行解。

构图

我们可以构造一张有向图 G，将变量 x_i 拆成两个点 u_i 和 u'_i，分别表示 x_i 为真或 x_i 为假。

同时，我们可以用边来表示逻辑蕴含 ("$\rightarrow$") 的关系。
具体来说，对于 $x_i \vee x_j$，我们连边 $u'_i \rightarrow u_j$ 与 $u'_j \rightarrow u_i$，意为 "x_i 为假蕴含 x_j 为真"、"x_j 为假蕴含 x_i 为真"。其他情况同理。
特别的，对于 x_i，我们连边 $u'_i \rightarrow u_i$。

这样，当我们标记点 u 时，我们也必须标记点 u 可以到达的点。如果在标记过程中出现 u_i 和 u'_i 同时被标记的情况，则产生了矛盾。

算法 1

非常显然的暴力算法。
枚举每一对尚未确定的 u_i, u_i'，先标记 u_i，若推出矛盾则重新标记 u_i'，若仍然矛盾则整个问题无解。

注意此算法并不需要回溯，因为 u_i, u_i' 尚未确定，故不与之前的组关联，不会产生影响。
时间复杂度 $O(nm)$，不过一般远远达不到上界。

注意这个算法得出的结果是字典序最小的。

实现的具体细节参见 two-sat.cpp。

例题：[UVa1146]Proving Equivalences（白书 P324）

算法 2

可以发现，如果我们找出所有的强连通分量，则同一个分量中的点要么全部标记，要么全不标记。

若一个强连通分量中同时包括 u_i, u_i'，则一定无解。反过来我们可以证明，若前面的情况没有出现，则一定有解。

具体地，我们先对原图缩点，然后进行拓扑排序，自底向上地标记、删除即可。

时间复杂度 $O(n+m)$。有兴趣的同学可以自行实现这个算法。不过算法 1 已经足够快了。

前言

单源最短路问题可以说是图最基础的一类问题了，不过还是有些地方值得注意的。

关于实现

Dijkstra 算法相信大家已经掌握的很熟练了。

需要注意的是，最好使用手写堆而非std::priority_queue。原因是后者不支持小根堆的 DECREASE_KEY操作，Dijkstra 的复杂度将会从 $O((n+m)\log n)$ 变成 $O((n+m)\log m)$。

实现的具体细节参见 dijkstra-with-heap.cpp。

注意

这个算法加上队列优化就是我们常说的 SPFA。

只是有几个事情要注意一下：

- 虽说 SPFA 很好打也很好用，但是也很容易被卡（网格图什么的）；Dijkstra 则要保险得多
- 判负环时最好把队列改成栈

定义

有 n 个变量，给你 m 个形如

$$x_j - x_i \leq b_k (i, j \in [1, n], k \in [1, m])$$

的约束条件，则称之为差分约束系统。

现在要求出满足条件的一组解。

算法

注意到三角不等式

$$d(u) + w(u, v) \geq d(v)$$

变形得

$$d(v) - d(u) \leq w(u, v)$$

这与之前的约束形式一模一样，故可以用 Bellman-Ford 算法求解。(若没有负权边也可用 Dijkstra 算法求解)

构图

对于最短路求得的一组解，有一个很好的性质：在 x_s 确定的情况下，其他的 x_i 均为最大值。（s 为图中对应的源点）

所以，当某个变量为定值时，要求其他变量的最大值（当然是满足约束的前提下），则是以那个确定的变量为源点求单源最短路。
反之，若是求最小值，则是求最长路。
更一般的，若要求每个变量都取最小/最大值，则可以新建一个超级源点连向所有的点，边权为 0。

注意到差分约束系统中的不等号方向必须一致并且与最短路/最长路的三角不等式相适应，我们可以有如下转化：

- $a \leq b \iff -a \geq -b$
- $a = b \iff a \leq b \wedge a \geq b$
- a 为整数时，$a < b \iff a \leq b - 1$

例题

[POJ3159]Candies（要用 Dijkstra）
[POJ1201]Intervals（Bellman-Ford 求最长路）

相关问题

- 增量最小生成树
- 最小瓶颈生成树

简单算法

首先预处理出最小生成树上每对点路径的最大值。
然后枚举每条不在最小生成上的边，这样产生了一个环，并且这条边不会比环上其他边小。我们删去环上原来最大的边，再加上这条边。
可以证明，次小生成树一定可以通过这样的方式得到。

时间复杂度 $O(n^2 + m)$。

定义

给定一个带权有向图 G 和一个节点 u，找出一个以 u 为根节点的全职最小的树形图，满足除 u 之外的节点入度为 1，且 u 可以到达所有节点。

朱刘算法

首先删除自环。
接下来，我们对每一个结点 u 求出连向它的边权最小的边 $in(u)$，然后在这些边中找环。如果找不到环，那么说明已经得到最小树形图；否则将环缩成点，将每条连向环的边 (u, v) 的权值改成 $w(u, v) - in(v)$。重复上述操作直至找不到环为止。

算法时间复杂度 $O(nm)$。
实现的具体细节参见 zhuliu.cpp。

朱刘算法口胡起来很简单，但是实现有很多需要注意的地方，一定要看看代码。

例题：[POJ3164]Command Network

匈牙利算法

二分图最大匹配的定义大家应该很熟悉吧。

我们定义：

- 未盖点 未匹配的点
- 交错路 满足任意相邻的边一定是一个匹配边和一个非匹配边的路径
- 增广路 两个端点都是未盖点的交错路

容易发现，只要图中存在增广路，最大匹配数便可以增加 1。

故只需从左边每个未盖点出发找增广路即可。找增广路可以用 DFS 实现，非常简单。时间复杂度 $O(n^3)$。
实现的具体细节参见 hungary.cpp。

例题：[HDU2444]The Accomodation of Students

定义

对于二分图，我们定义：

完美匹配 包括了图中所有点的匹配（必有 $|X| = |Y|$）

最佳完美匹配 匹配边权值和最大的完美匹配

后面我们可以发现，即使 $|X| < |Y|$，只要 X 方点全部匹配，那么也是可以使用 KM 算法的。

KM 算法

我们定义可行点标 $l(u)$，对于任意边 (u, v)，满足

$$l(u) + l(v) \geq w(u, v)$$

定义相等子图 G' 为 G 中所有的点和满足 $l(u) + l(v) = w(u, v)$ 的边组成的一个子图。

容易证明，若 G' 有完美匹配，则该匹配是 G 的最佳完美匹配。

从而，我们要做的就是找到一个足够好的顶标，在 G' 中找到完美匹配。

KM 算法

我们不妨先构造一个合法的顶标（比如 X 方为出最大值，Y 方为 0），然后在此基础上修改，不断加入新边，直到找到完美匹配。
问题就在与如何修改顶标。

仍然沿用匈牙利算法的思路。从 X 方点的任一未盖点出发寻找增广路，其中在匈牙利树中出现了的点分别是 X'、Y'。

如果我们对 X' 中点的顶标都减小 a，Y' 中点的顶标都增加 a，会出现什么情况呢？

KM 算法

对边 (u, v) 考虑如下几种情况：

- $u \in X', v \in Y'$ 或 $u \notin X', v \notin Y'$，此时 $l(u) + l(v)$ 不变
- $u \in X', v \notin Y'$，则这条边之前一定不在 G' 中；此时 $l(u) + l(v)$ 减小，有可能进入 G'
- $u \notin X', v \in Y'$，则这条边之前一定是非匹配边；此时 $l(u) + l(v)$ 虽然增大，但没有影响

所以，上述修改顶标的方法是可行的。

但注意到最初的约束 $l(u) + l(v) \geq w(u, v)$，我们只能选取

$$a = \min\{l(u) + l(v) - w(u, v) | u \in X', v \notin Y'\}$$

这样，不断地修改顶标，直到找到从这个点出发的增广路，然后换一个未盖点重复该过程即可。

KM 算法

上述算法时间复杂度 $O(n^4)$，其实还可以优化。
对于 Y 方点，考虑 DFS 过程中维护

$$slack(v) = \min\{l(u) + l(v) - w(u, v) | u \in X\}$$

则有

$$a = \min\{slack(v) | v \notin Y'\}$$

时间复杂度降为 $O(n^3)$。
实现的具体细节参见 km.cpp。

例题：[POJ3565]Ants（白书 P351）
[HDU2853]Assignment（构图比较巧妙）

最小覆盖

定义图的最小覆盖为选择尽量少的点让每条边都至少有一个端点被选择。

容易证明，二分图的最小覆盖数等于其最大匹配数。（不妨利用最小割最大流定理思考）

至于构造解，如果用最小割最大流定理思考，则是很简单的事了：先从 X 中所有未盖点出发扩展匈牙利树，标记所有点。则 X 中的未标记点和 Y 中的已标记点构成了图的最小覆盖。

例题：[UVa11419]SAM I AM（白书 P355）

最大独立集

定义图的最大独立集为选择尽量多的点使任意两点均不相邻。

容易看出二分图的最大独立集与最小覆盖互补（不仅是数量上的）。

例题：[POJ2771]Guardian of Decency（白书 P356）

最小路径覆盖

定义图的最小路径覆盖为尽量少的路径使得每个节点恰好在一条路径上。注意路径可以退化为一个点。
现在要你求 DAG 上的最小路径覆盖。

我们可以把原图中每个点 u 拆成 u_x 和 u_y；如果原图有边 (u, v)，则连边 (u_x, v_y)。
则原图的最小路径覆盖等于点数减去新图的最大匹配。

注意只有 DAG 才可以这么做。
如果路径可以相交，则要先对图用 Floyd 求一次传递闭包。

例题：[POJ2060]Taxi Cab Scheme（白书 P357）

前言

终于讲到这里来了.....
网络流内容非常多，特别是构图、模型转化方面。这里只会介绍比较基础的内容。

Dinic 算法

在残量网络中，我们定义：

- $dis(u)$ 源点 s 到 u 的距离
- 层次图 由所有的点和满足 $dis(u)+1=dis(v)$ 的边 (u,v) 构成的子图
- 阻塞流 在层次图中的最大流

Dinic 算法即是不断构造层次图，然后用阻塞流增广。由于每次增广都至少会使 $dis(t)$ 增加 1，故最多增广 $n-1$ 次。
算法复杂度 $O(n^2m)$。
注意这个复杂度其实是非常宽松的，实际效率可能要好得多。

实现的具体细节嘛.....参考白书。
注意不要过度依赖 STL 中的容器。(算法即便不开-O2还是很快的)

建模

- 多源多汇的最大流
- 节点有容量限制的最大流
- 无源无汇有上下界的可行流
- 有源有汇有上下界的最小流/最大流
- 费用与流量平方成正比的最小费用流
- ……

建模

- 二分图带权最大独立集
- 公平分配问题
- 区间 k 覆盖问题
- 最大闭合子图
- 最大密度子图
- ……