图连通性
若干拓展问题探讨

北京大学　李煜东

引言

- Black History ……
- 去年冬令营营员交流时，我和王启圣上来讲了Robert.E.Tarjan参与发明的斐波那契堆……
- Why this topic > _ < ?
- 今年有幸再次来WC，本着继续膜拜Tarjan的一种精神，于是决定把Tarjan主要研究的方向之一 —— 图连通性作为主题。
- 这部分内容难度本身就不大，所以希望大家度过一个愉悦的上午。。。（什么？你说后缀仙人掌和函数式LCT也愉♂悦？别闹了那不是一种愉悦好么……）

安排

- 第一节课 - 知识普及 - 8:00~9:00
- 无向图连通性（面向NOIP选手）
- 第二节课 - 主要内容 - 9:00~9:15、9:30~10:45
- 有向图连通性、Dominator Tree、Shannon Switching Game、Disjoint Spanning Tree
- 第三节课 - 拓展讨论 - 11:00~12:00
- 图灵机与计算复杂性理论（知识扩充）

无向图连通性

无向图的割点、桥、双连通分量；经典Tarjan算法

需要解决的问题

- 无向图的割点、割点集合与点连通度
- 无向图的桥、割边集合与边连通度
- 无向图的割点与点双连通分量的求法
- 无向图的桥与边双连通分量的求法、边双连通分量的构造
- 使用Tarjan算法求最近公共祖先
- 相关例题讨论

割点

- 在无向连通图G上进行如下定义：
- 割点：若删掉某点P后，G分裂为两个或两个以上的子图，则称P为G的割点。
- 割点集合：在无向连通图G中，如果有一个顶点集合，删除这个顶点集合以及与该点集中的顶点相关联的边以后，原图分成多于一个连通块，则称这个点集为G的割点集合。
- 点连通度：最小割点集合的大小称为无向图G的点连通度。

割边

- 类似地，在无向连通图G上进行如下定义：
- 桥(割边)：若删掉某条边B后，G分裂为两个或两个以上的子图，则称B为G的桥(割边)。
- 割边集合：如果有一个边集合，删除这个边集以后，原图分成多于一个连通块，则称这个边集为割边集合。
- 边连通度：最小割边集合的大小称为无向图G的边连通度。

双连通分量

- 点双连通图：点连通度大于1的图称为点双连通图（没有割点）。
- 边双连通图：边连通度大于1的图称为边双连通图（没有割边）。
- 无向图G的极大(点/边)双连通子图称为(点/边)双连通分量。
- 缩点：把一个双连通分量缩为一个点的过程，就是删除与该双连通分量相关的所有点和边，然后新建一个点，向所有与双连通分量中的点有边相连的点连边。

无向图上的经典Tarjan算法

- Tarjan基于对图的深度优先搜索，并对每个节点引入两个值：
- dfn[u]：节点u的时间戳，记录点u是DFS过程中第几个访问的节点。
- low[u]：记录节点u或u的子树不经过搜索树上的边能够到达的时间戳最小的节点。
- 对于每一条与u相连的边(u,v)：
- 若在搜索树上v是u的子节点，则更新low[u]= min(low[u], low[v])；
- 若(u,v)不是搜索树上的边，则更新low[u]= min(low[u], dfn[v])；

求割点

- 对于一条搜索树上的边(u,v)，其中u是v的父节点，**若low[v]>=dfn[u]，则u为割点。**
- low[v]表示v和v的子树不经过搜索树上的边能够到达的时间戳最小的节点；
- low[v]>=dfn[u]说明从v到以u为根的子树之外的点必须要经过点u；
- 因此u是图中的一个割点。
- 在DFS的过程中，首先递归u的子节点v。从v回溯至u后，检查上述不等式是否成立。若成立，则找到了一个割点u。

求点双连通分量

- 可以在求割点的过程中维护一个栈求出每个点双连通分量。
- 建立一个栈，存储DFS过程中访问的节点，初次访问一个点时把该点入栈。
- 如果边(u,v)满足low[v]>=dfn[u]，即满足了u是割点的判断条件，那么把点从栈顶依次取出，直至取出了点v，取出的这些点和点u一起组成一个点双连通分量。
- **割点可能属于多个点双连通分量**，其余点和每条边属于且仅属于一个点双连通分量。因此在从栈中取出节点时，要把u留在栈中。
- 整个DFS结束后，栈中还剩余的节点构成一个点双连通分量。

实现

```
procedure tarjan(x is a vertex)
{
   visit[x]←true
   dfn[x]←low[x]←num←num+1
   push x into stack s
   for each edge (x,y) from x do
      if not visit[y] then
      {
         tarjan(y)
         low[x]←min(low[x],low[y])
         if low[y]>=dfn[x] then
         {
            mark x as a cut-vertex
            repeat
               z←top of stack s
               pop z from stack s
            until z=y
            mark vertexes just poped and x as a DCC
         }
      } else low[x]←min(low[x],dfn[y])
}
```

Knights of the Round Table

- 国王要在圆桌上召开骑士会议，但是若干对骑士之间互相憎恨。出于各种各样奇♂怪的原因，每次开会前都必须对出席会议的骑士有如下要求：
- ① 相互憎恨的两个骑士不能坐在相邻的2个位置；
- ② 为了让投票表决议题时都能有结果（不平票），出席会议的骑士数必须是奇数。
- 如果有某个骑士无法出席任何会议，则国王会为了Peace of the World把他踢出骑士团。
- 现在给定骑士总数n(n<=1000)，以及m(m<=1000000)对相互憎恨的关系，求至少要踢掉多少个骑士？

Knights of the Round Table

- 题目大意：给定若干骑士和他们之间的仇恨关系，规定召开圆桌会议时，两个有仇恨的骑士不能坐在相邻位置，且召开一次圆桌会议的骑士人数必须为奇数，求有多少骑士永远不可能参加某一次圆桌会议。
- 模型转化：建补图——没有仇恨的骑士间连边。
- 几个骑士可以召开圆桌会议的条件是它们构成一个奇环。
- 问题转化为：求有多少个骑士不包含在任何奇环内。

Knights of the Round Table

- 引理：若某个点双连通分量中存在奇环，则该点双联通分量中的所有点都在某个奇环内。

- 用经典Tarjan算法找出所有点双联通分量。
- 判定每个点双联通分量是不是二分图，就可以知道它有没有奇环。

求桥

- 对于一条搜索树上的边(u,v)，其中u是v的父节点，**若low[v]>dfn[u]，则(u,v)是桥。**
- low[v]表示v和v的子树不经过搜索树上的边能够到达的时间戳最小的节点；
- low[v]>dfn[u]说明从以v为根的子树到子树之外必须要经过边(u,v)，因此(u,v)是桥。
- 可以像求割点一样，当v回溯至u后，判断上述不等式是否成立。
- 另一种判断方法：当**递归v结束时，如果low[v]==dfn[v]**说明v和v的父节点之间的边是桥。
- P.S. 在有重边的图上求桥，需要注意对这些重边加以区分。

求边双连通分量

- 边双连通分量的求法非常简单，只需在**求出所有的桥以后，把桥边删除**。
- 此时原图分成了若干个连通块，每个连通块就是一个边双连通分量。
- 桥不属于任何一个边双连通分量；
- 其余的边和每个顶点都属于且仅属于一个边双连通分量。

实现

```
procedure tarjan(x is a vertex)
{
  visit[x]←true
  dfn[x]←low[x]←num←num+1
  for each edge (x,y) from x do
    if not visit[y] then
    {
      tarjan(y)
      low[x]←min(low[x],low[y])
      if low[y]>dfn[x] then
        mark edge (x,y) as a bridge
    }
    else
      if edge(x,y) not on DFS tree then
        low[x]←min(low[x],dfn[y])
  //the following step can also find bridges
  if dfn[x]=low[x] then
    mark edge into x on DFS tree as a bridge
}
```

边双连通分量的构造

- **任意给定一个无向连通图，最少添加多少条边可以把它变为边双连通图**？
- 求出所有的桥和边双连通分量，把每个双连通分量缩为一个点。
- 此时的图只包含缩点后的双连通分量和桥边，是一棵无根树。
- 统计树中度数为1的节点的个数cnt。把树变为边双连通图，至少需要添加(cnt+1)/2条边。
- 构造方法：每次寻找最近公共祖先最远的两个度数为1的节点，在两点之间连一条边。
- 这样可以使这两个点到LCA的路径上的所有点形成环，环一定是双连通的。

故乡的梦

- 不知每日疲于在城市的水泥森林里奔波的你会不会有时也曾向往过乡村的生活。你会不会幻想过，在夏日一个静谧的午后，你沉睡于乡间路边的树荫里，一片叶子落在了你的肩上，而你正做着一个悠长的梦，一个没有城市的梦。我们的问题，正围绕着这个梦境展开……
- 从 Azure 求学的城市到 Azure 家乡的村庄由若干条路径连接，为了简化起见，我们把这些路径抽象成一个 N 个节点的无向图，每个节点用一个[1,N]内的整数表示，其中城市在S 节点，Azure 的故乡在T 节点。在某些节点对(x,y)之间会有双向道路连接，道路的长度为c。不过最近某些道路可能会在施工，Azure 想知道，假如某条道路被施工而不能通行的话，他到故乡的最短路经的长度为多少（多次询问）。

故乡的梦

- 第一行两个正整数 N 和 M，分别表示结点数和道路数。
- 之后M 行，每行三个正整数x,y,c，表示在x 和 y 之间存在一条长度为c的道路。
- 之后一行包含两个正整数S 和T，表示 Azure 所在位置和故乡的位置。
- 之后一行包含一个整数Q，表示询问的数目。
- 之后Q行，每行两个整数u 和v，表示 Azure 想知道如果连接u 和v之间的道路进入了维修状态，从S 到T 的最短距离会变为多长。
- N,M,Q<=200000.

故乡的梦

- 首先用堆优化的Dijkstra算法求出各个点到起点S和终点T的最短路。
- 然后BFS求出最短路径图（所有满足DistS[u]+w(u,v)=DistS[v]的边构成的图）。
- 从图中去掉无用节点（把满足上面条件的边(u,v)看做有向边时不能到达T的点）。
- 使用经典Tarjan算法求出图中的桥，并找出所有的边双连通分量。
- 由最短路径图的性质，这些双连通分量可以一字排开，相邻两个之间由桥边连接。

故乡的梦

- 对于不删除桥边的询问，最短路不变，答案就是DistS[T]。
- 如果删除了桥边，就离线采用以下做法求出所有此类询问的答案：
- 建立一个小根堆，在堆中存储一些边，边(u,v)对应的值为DistS[u]+w(u,v)+DistT[v].
- 依次扫描到每个双连通分量。到达一个DCC后，首先在堆中删除前面的DCC出发到这个DCC的边，然后在堆中添加这个DCC出发到后边DCC的边。
- 堆中的最小值就是"删除掉当前连通块和下一个连通块之间的关键边"时的答案。
- 整个算法的复杂度为O((N+M+Q)logN)。

最近公共祖先

- 离线处理：读入所有的询问（可使用邻接表存储）。
- 给每个节点添加一个颜色标记，尚未访问的标记为0，进入递归而未回溯的节点标记为1，已经访问过的标记为2。
- 维护一个并查集，访问完某个节点时，就合并该节点所在的集合与其父节点所在的集合。
- 正在处理点u时，u和u的所有祖先的标记均为1，已经访问过的节点均与父节点相连；因此遍历子树u后，考虑所有与u相关的询问lca(u,v)，那么lca(u,v)就是v所在并查集的根。

经典Tarjan算法求LCA

```
procedure tarjan(x is a vertex)
{
    fa[x]←x,  color[x]←1
    for each edge (x,y) from x do
        if color[y]=0 then
            tarjan(y),  fa[y]←x
    for each query(x,y) from x do
        if color[y]=2 then
            answer of query(x,y) ← get(y);
    color[x]←2;
}
```

Network

- A network administrator manages a large network. The network consists of N computers and M links between pairs of computers. Any pair of computers are connected directly or indirectly by successive links, so data can be transformed between any two computers. The administrator finds that some links are vital to the network, because failure of any one of them can cause that data can't be transformed between some computers. He call such a link a bridge. He is planning to add some new links one by one to eliminate all bridges.
- You are to help the administrator by reporting the number of bridges in the network after each new link is added.

Network

- The input consists of multiple test cases. Each test case starts with a line containing two integers N ($1 \le N \le 100{,}000$) and M ($N - 1 \le M \le 200{,}000$).
  Each of the following M lines contains two integers A and B ($1 \le A \ne B \le N$), which indicates a link between computer A and B. Computers are numbered from 1 to N. It is guaranteed that any two computers are connected in the initial network.
  The next line contains a single integer Q ($1 \le Q \le 1{,}000$), which is the number of new links the administrator plans to add to the network one by one.
  The i-th line of the following Q lines contains two integer A and B ($1 \le A \ne B \le N$), which is the i-th added new link connecting computer A and B.

Network

- 使用Tarjan算法求出图中所有的桥；
- 找到图中的每一个双联通分量，缩成一个点，此时图变成了一棵树；
- 每次加入一条边(x,y)，相当于x到lca(x,y)、y到lca(x,y)和(x,y)这条边构成了一个环；
- 在环上的边都不是割边，于是可以给这些边打上标记，并更新割边个数。
- 当然以前标记过的边就不要重复计算了；
- 处理环的过程中，对于标记过的边可用并查集路径压缩加以优化，不重复经过以保证时间。
- 思考：在仙人掌上多次询问两点之间的最短路径？

Traffic Real Time Query System

- City C is really a nightmare of all drivers for its traffic jams. To solve the traffic problem, the mayor plans to build a RTQS (Real Time Query System) to monitor all traffic situations. City C is made up of N crossings and M roads, and each road connects two crossings. All roads are bidirectional. One of the important tasks of RTQS is to answer some queries about route-choice problem. Specifically, the task is to **find the crossings which a driver MUST pass when he is driving from one given road to another given road**.

Traffic Real Time Query System

- The first line contains two integers N and M. ($0<N<=10000$, $0<M<=100000$)
  The next M lines describe the roads. In those M lines, the i^{th} line (i starts from 1)contains two integers X_i and Y_i, representing that $road_i$ connects crossing X_i and Y_i ($X_i \neq Y_i$).
- The following line contains a single integer Q. ($0<Q<=10000$)
  Then Q lines follows, each describing a RTQ by two integers S and T($S \neq T$) meaning that a driver is now driving on the road s and he wants to reach road t .
- For each RTQ prints a line containing a single integer representing the number of crossings which the driver MUST pass.

Traffic Real Time Query System

- 题目要求在一个无向图上多次询问从一条路到另一条路的必经点。
- Tarjan求出点双连通分量，缩点后建树；
- 对于每个询问通过LCA即可计算；

- 类似的题目还有多次询问一个点到另一个点的必经边（ContestHunter Round #24 C）。
- 改为求边双连通分量即可。

有向图连通性与Dominator Tree

有向图的割点、桥；必经节点树(Dominator Tree)的Lengauer-Tarjan算法

需要解决的问题

- 有向图的强连通分量（大家已经非常熟悉，不再介绍）
- 有向图的必经点
- 有向图的桥边
- 必经节点树 Dominator Tree
- Lengauer-Tarjan算法

Flow Graph的DFS遍历

- Flow Graph
- 若有向图G中存在一点r，从r出发可以到达G中所有的点，则称G是Flow Graph，记为(G,r)。
- 搜索树（Dfs Tree）
- 从r出发对G进行深度优先遍历，递归时经过的有向边构成一棵树，称为G以r为根的搜索树。
- 时间戳（Dfn）
- 遍历过程中，按照节点经过的时间先后顺序给每个节点一个标号——时间戳，记为dfn[x]。时间戳为x的节点的原编号记为id[x]，即id[dfn[x]]=x。

对图中的边进行分类

- 若(G,r)是一个Flow Graph，T是其Dfs Tree，则G中的每条边(x,y)必属于以下四类之一：
- **树枝边 (tree arcs)**：边(x,y)在搜索树T中；
- **前向边 (forward arcs)**：在T中存在一条从x到y的路径；
- **后向边 (cycle arcs)**：在T中存在一条从y到x的路径；
- **横叉边 (cross arcs)**：在T中既没有x到y的路径，也没有y到x的路径，并且dfn[y]<dfn[x]。

有向图的必经点问题

- 例题：Important Sisters
- 冬令营讲义第123页

- 题目要求解决的模型：
- 给定有向图G（可能有环）和图中的一个点r，对于G中的任意一个点x，求从r到x的路径上（可能有很多条）必须经过的点集。

必经点与最近必经点

- **必经点（dom）**
- 若在(G,r)中从r到y的路径一定经过点x，则称x是从r到达y的必经点，记为x dom y。
- 从r出发到达y的所有必经点构成的集合记为dom(y)，即dom(y)={x | x dom y}。

- **最近必经点（idom）**
- 节点y的必经点集合dom(y)中dfn值最大的点x是距离y最近的必经点，称为y的最近必经点。最近必经点是唯一的，因此可以记x=idom(y)。

暴力求解

- $pre(y)=\{x \mid (x,y)\in E\}$ —— y的前驱节点集合
- $suc(x)=\{y \mid (x,y)\in E\}$ —— x的后继节点集合
- $dom(r)=\{r\}$
- $dom(y)=\bigcap_{x\in pre(y)} dom(x)\ \cup\ \{y\}$
- $idom(x)=id[Max\{\,dfn[y] \mid y\in dom(x)\}]$
- DAG上可以按照拓扑序计算，有环图上可以迭代计算，$O(V^2)$。

Dominator Tree

- 设有向图G=(V,E)，(G,r)是一个Flow Graph，则称(G,r)的子图D=(V, { (idom(i),i) | i∈V,i≠r }, r)为(G,r)的一棵Dominator Tree。
- (G,r)的Dominator Tree是一棵有向有根树，从r出发可以到达G中的所有点，并且树上的每条边(u,v)都满足：u=idom(v)，即父节点是子节点的最近必经点。
- ⇒ x=idom(y)，当且仅当有向边(x,y)是Dominator Tree中的一条树枝边。
- x dom y，当且仅当在Dominator Tree中存在一条从x到y的路径。
- ⇔ x的必经点集合dom(x)就是Dominator Tree上x的所有祖先以及x自身。

半必经点

- **半必经点（semi）**
- 在搜索树T上点y的祖先中，通过非树枝边可以到达y的深度最小的祖先x，称为y的半必经点。半必经点也是唯一的，因此可以记x=semi(y)。

半必经点定理

- 对于G中的一点y，考虑所有$x \in pre(y)$，定义一个临时变量$temp = +\infty$。
- 若$dfn[x] < dfn[y]$，则(x,y)为树枝边或前向边，此时$temp = \min(temp, dfn[x])$。
- 若$dfn[x] > dfn[y]$，则(x,y)为横叉边或后向边，此时$\forall x$在T中的祖先z，满足$dfn[z] > dfn[y]$时，有$temp = \min(temp, dfn\left[semi[z]\right])$。
- $semi[y] = id[temp]$。即在上述所有可能情况中取dfn值最小的一种，就是y的半必经点。

半必经点一定是必经点吗？

必经点定理

- 对于G中的一点x，考虑搜索树T中$semi(x)$到x的路径上除端点之外的点构成的集合$path$。
- 设$y = id[\min\{dfn[semi(z)] \mid z \in path\}]$，即$path$中半必经节点的时间戳最小的节点。
- $idom(x) = \begin{cases} semi(x) & semi(x) = semi(y) \\ idom(y) & semi(x) \neq semi(y) \end{cases}$

Lengauer-Tarjan算法

- Lengauer和Tarjan于1979年提出，经典Tarjan算法的进一步应用。
- 时间复杂度O(NlogN)或O(Nα(N))，几乎接近于线性。
- 基本思想：
- 通过DFS构建搜索树，并计算出每个节点的时间戳。
- 根据半必经点定理，按照时间戳从大到小的顺序计算每个节点的半必经节点。
- 根据必经点定理，按照时间戳从小到大的顺序，把半必经节点≠必经节点的节点进行修正。

使用的数据结构：并查集

- 在访问每个节点时，把该节点放入图的一个生成森林中。
- 根据半必经点定理，对于一条边(x,y)，若dfn[x]>dfn[y]，则计算semi[y]时需要考虑x的祖先中比y的时间戳大的节点。
- 由于算法按照时间戳从大到小的顺序计算，此时比y的时间戳小的节点还未加入生成森林，因此直接在生成森林中考虑x的所有祖先即可。
- 使用并查集维护生成森林，把dfn[semi[z]]作为z到其父节点的边的权值，在路径压缩的过程中对每个节点维护一个变量，很容易计算z到其祖先上所有边权的最小值。

算法实现

suc[u] – successors of u
pre[u] – predecessors of u
fa[u] – DFS tree father
dfn[u] – DFS order number
id[u] – vertex number of dfn
dom[u] – vertexes that u dominates
semi[u] – semidominator
idom[u] – immediate dominator
anc[u] – ancestor in disjoint set
best[u] – dfn of the vertex with the smallest semi in ancestor chain

```
procedure dfs(x is a vertex)
{
   dfn[x]←t←t+1, id[t]←x
   for each y in suc[x] do
      if dfn[y]==0 then
         dfs(y), fa[dfn[y]]←dfn[x]
}
function get(x is a vertex):a vertex
{
   if x=anc[x] then return x
   y←get(anc[x])
   if semi[best[x]]>semi[best[anc[x]]] then
      best[x]←best[anc[x]]
   anc[x]←y, return y
}
```

算法实现

```
procedure tarjan
{
  for y←t step -1 until 1 do
  {
    x←fa[y]
    for each z in pre[id[y]] do
    {
      z←dfn[z]
      if z=0 then continue
      get(z)
      semi[y]←min(semi[y],semi[best[z]])
    }
    push y into the back of dom[semi[y]]
    anc[y]←x
    for each z in dom[x] do
    {
      get(z)
      if semi[best[z]]<x then idom[z]←best[z]
        else idom[z]←x
    }
    empty dom[x]
  }
  for i←2 step 1 until t+1 do
  {
    if idom[i]≠semi[i] then idom[i]←idom[idom[i]]
    push i into the back of dom[idom[i]]
  }
  idom[1]←0
}
```

例题

- 首先是可以用来求有向图的割点（SPOJ BIA）。
- Problem：Entrapment（SPOJ EN）
- 冬令营讲义第124页
- 题目大意：有向图，求起点S到终点T的所有路径中最接近源的必经点。
- Lengauer-Tarjan算法求Dominator Tree，得到S到T的必经点，找最近的一个。
- 也可以用网络流，拆点构图，增广两次，第二次增广能够到达的最远点就是答案。

有向图的桥边

- 给定一张Flow Graph (G,r)，若去掉图中的一条边e后，r不能到达图中的全部节点，则称e是(G,r)的桥。
- 给定一张有向图G和图中的两个点S、T，若图中的一条边e处于从S到T的所有路径上，则称e是从S到T的必经边。
- 有向无环图上的必经边？
- 有向图(可能有环)上的桥边？
- 注意在环上的边可能也是桥，因此不能简单地缩点转化为DAG。

有向图的桥边

- 有向无环图上的必经边？
- 求S到每个点的路径条数CntS，每个点到T的路径条数CntT；
- 若边(u,v)满足CntS[u]*CntT[v]=CntS[T]，则(u,v)是从S到T的必经边；
- 可以对几个大质数取模做上述计算。
- 有向图(可能有环)上的桥边？
- 在一条边u→v的中间加一个点，变成u→w→v，然后用Lengauer-Tarjan算法求有向图的割点，若一个新加的点是割点，则对应的边是割边。

PKU ACM Team's Excursion

- It is a traditional activity for PKU ACM Team to go hiking during the summer training camp. The coach has found a national park and acquired the map of this park. In this map, there are n intersections (numbered from 0 to n-1) connected by m one-way roads, and the map of the park is a Directed acyclic graph.
- The excursion will begin at a certain starting intersection s and end at a certain destination intersection t. You can find a route from intersection s to t. A road is called a "bridge" if and only if you cannot find a route from starting intersection to destination once it disappears. The other ordinary roads are not bridges so they are not dangerous at all. According to the experience of the tourists, the danger level of a "bridge" is equal to its length.

PKU ACM Team's Excursion

- The national park provides an "express" service. The hikers can order at most two buses beforehand to help them to escape some of the most dangerous routes. Each bus can start from anywhere (any intersection or any point on any road) in the national park. But the bus is powered by electricity so it can only travel no more than q meters continuously and the passenger should get off before the electricity runs out. When the team is on a bus, the danger level along the route can be ignored.
- The problem is reduced to find a route from the start intersection to the end with the least sum of danger levels. On the way we can take two special buses to make our route safe.
- $1 \le n \le 100\,000$, $1 \le m \le 200\,000$, $0 \le s,t < n$, $s \ne t$, $1 \le q \le 1\,000\,000\,000$

PKU ACM Team's Excursion

- 题目要求有向无环图中的一条危险程度最小的路径，开始节点和终止节点已经被指定。对于一条边，当它成为“桥”的时候，这条边被标记为危险，危险程度等于边的长度。可以选择搭乘两次车，每次连续行驶不超过q米，在车上经过的桥不计入路径的危险程度。
- 对于这道题目，我们首先用前面提到的计算路劲条数+取模的方法求出S到T的必经边。
- 由于S到T的任何一条路径都包含所有必经边，根据贪心，可以任取一条S到T的最短路。
- 把最短路上的点和边取出，就转化为了一个线性问题，用两段区间尽可能覆盖上面的桥。
- 正向、反向各扫描一遍，求出[S,x]和[x,T]上搭一次车的最小危险度，枚举切点x即可。

Disjoint Spanning Tree

Shannon Switching Game, Minimum Spanning Trees, Edge-disjoint Spanning Tree

Shannon Switching Game

- Problem：Game of Connect
- 冬令营讲义第125页
- 给定一张无向图和两个特定点A和B，图中的每条边可被染色或者删除。
- 有两个玩家，代号分别为Short Player和Cut Player，轮流操作。
- Short每次可以选择一条边染色，Cut每次可以删除一条尚未染色的边。
- 如果最终A和B不再连通，Cut获胜；如果最终从A到B有一条被染色的路径，Short获胜。

Game Time！

- 通过玩这个Demo的经验，你觉得怎样做才能获胜？

局面与必胜策略

- Shannon Switching Game只有三种局面：
- Short有必胜策略、Cut有必胜策略、先手有必胜策略。
- Short Player有必胜策略，当且仅当图中存在一个连接A和B的子图，子图中存在两棵边不相交的生成树。
- 此时如果Cut删除了一条边使得其中一棵生成树断开分成了两棵子树，Short只需要在另一棵生成树中找到一条连接这两棵子树的边染色。
- 除此之外，其它必胜情况可以考虑图中是否有两棵生成树仅有一条公共边。

Edge Disjoint Spanning Tree

- 从Shannong Switching Game的必胜策略出发，我们讨论以下几个问题：
- Minimum Spanning Trees 问题：求带边权无向图G中k棵边不相交的生成树，并且生成树的边权之和尽量小。
- （A set of k disjoint spanning trees such-that the total value of the edges is minimum）
- 求无向图G中边不相交的生成树的最多个数
- （A maximum number of spanning trees in G which are pairwise edge-disjoint）

Minimum Spanning Trees

- 问题概述：给定无向图G=(V,E)和一个正整数k，G中的每条边都有一个权值。求无向图G的k棵生成树，满足生成树两两之间没有公共边，并且生成树上所有边的权值之和最小。
- 边不相交生成树：edge-disjoint spanning trees
- 边不相交生成森林：edge-disjoint spanning forest
- 解决问题主要在于求出这k棵生成树的边集，基本思想是按边权排序后，贪心地尝试把每条边加入到边集中，并检测加入之后边集还能否分成k个边不相交的生成森林。
- 理论依据来自于拟阵(matroid)。

Var

- $i_+ = (i \ mod \ k) + 1$
- F是E的一个子集，包含G中的若干条边。
- 若F能分成k个集合，每个集合是G的生成森林的边集，则称F是独立的$(independent)$。
- 若F是独立的，则记$F_1, F_2, \dots, F_k$为F分成的k个边集构成的k个生成森林。
- 若$e = (u, v) \in E$ 并且u, v在F_i中的同一棵树上，则记$F_i(e)$为F_i中从u到v的路径。
- 在接下来的讨论中，我们不再区分生成树、森林与其边集之间的区别。

增广序列(Augmenting Sequence)

- 满足以下条件的序列$e_0, e_1, \ldots, e_t$被称为边集F的增广序列(Augmenting Sequence)：
- $\forall i \in [0, t]$，$e_i \in E$
- $\forall i \in [1, k]$，$e_0 \notin F_i$
- $\forall i \in [0, t-1]$，$e_{i+1} \in F_{i+}(e_i)$
- e_t的两个端点不在F_{t+}的同一棵树中

增广(Augmentation)

- 给定一个增广序列$e_0, e_1, \ldots, e_t$，以下两步过程称为对F的一次增广(Augmentation)：
- $\forall i \in [0, t-1]$，令$F_{i+} = F_{i+} \cup \{e_i\} - \{e_{i+1}\}$；
- 令$F_{t+} = F_{t+} \cup \{e_t\}$；
- 对F进行一次增广后，F变为$F \cup \{e_0\}$。

- 引理1：通过增广序列对独立的边集F进行增广后，F仍然独立。

增广的实际意义

- 增广的意义在于，尝试把一条新的边e_0加入到边集F中，同时维护F分成的k个生成森林。
- 把e_i加入到F_{i+}中，如果e_i在F_{i+}中不形成环，说明找到了一个增广序列；
- 否则令e_{i+1}为e_i在F_{i+}中形成的环上的某一条边；
- 从F_{i+}中删除e_{i+1}，并对e_{i+1}重复以上步骤。
- 因此如果能找到e_0为开头的增广序列，说明可以把e_0加入到F中，否则不行。

寻找增广序列

- 把F中的所有边置为未标记状态，把e_0加入队列，进行一次宽度优先搜索。
- 重复以下步骤直到队列为空，或者找到增广序列：
- 取出队头元素e，设i满足$e \in F_i$，若e不在任何F_i中$(e = e_0$时$)$，则令$i = 0$；
- 如果e的两个端点不在F_{i+}的同一棵树中（不形成环），说明找到了一个增广序列，结束；
- 否则把$F_{i+}(e)$中所有未标记的边依次标记为e并加入队尾；

BFS的结果

- 第一种情况：
- 如果宽搜中途结束，那么在BFS的过程中找到了e_0开头的长度最短的增广序列。
- 由于每条边在加入队尾时的标记都是当时的队头，因此通过边的标记很容易复原该序列。
- 第二种情况：
- 引理2：如果宽搜由于队列为空而结束，那么$F \cup \{e_0\}$是非独立的。
- 换言之，这说明从e_0出发找不到增广序列，因此e_0不能被加入到F中。

第二种情况的进一步思考

- 若宽搜由于队列为空而结束，$\forall i \in [1,k]$，设S_i为F_i中所有被标记的边的端点构成的集合，则有以下三条性质成立：
- 性质一：$S_1 = S_2 = \cdots = S_k = S$；
- 性质二：S包含e_0的两个端点；
- 性质三：$\forall i \in [1,k]$，F_i中所有被标记的边构成一棵生成S的子树（记为T_i）。

三条性质的证明

- 设e_i为F_i中任意一条被标记的边。
- 由于未找到增广序列，所以e_i的两个端点在F_{i+}的同一棵树中，并且$F_{i+}(e_i)$上的边均被标记。
- $\Rightarrow S_i \subseteq S_{i+} \Rightarrow S_1 \subseteq S_2 \subseteq \cdots \subseteq S_k \subseteq S_1 \Rightarrow S_1 = S_2 = \cdots = S_k = S \Rightarrow S = S_1$包含$e_0$的两个端点。
- 性质三等价表述：若u是e_0的端点之一，则$\forall v \in S, \forall i \in [1,k]$，$F_i$中已标记的边连接$u, v$。
- 反证法，设e是满足"u和其端点v在F_{i+}中不被已标记边连接"的最早被标记的边，e被标记为e'，则u和e'的端点w在F_i中被已标记边连接。这些边均扩展了F_{i+}中的一条路径，因此u和w在F_{i+}中也被已标记边连接。而$v, w \in F_{i+}(e')$，故u和v在F_{i+}中被已标记边连接，矛盾！

Clump

- 若一个点集C被F的每个生成森林F_i中的一棵子树生成，则称该点集是一个丛(Clump)。
- 引理3：增广不破坏已有的丛（若C是F的一个丛，则对F增广后， C仍是一个丛）。
- 引理4：两个端点在同一个丛中的边一定不在增广序列中。
- 初始状态下， 边集F为空，每个点各自是一个丛。
- 一次增广如果成功，则扩大了F的大小；如果失败，则可以合并两个丛。
- 因此增广最多进行$O(kV)$次。

并查集维护

- 第0个并查集维护Clump，第1~k个并查集维护k个生成森林。
- $find_0(x)$返回点x所在的丛的代表节点，$union_0(x,y)$合并x和y所在的丛。
- $find_i(x)$返回F_i中点x所在子树的树根，$union_i(x,y)$连接F_i中x和y所在的子树。
- 对每条边$e=(u,v)$关联三条信息：$value(e)$表示其边权；
- $forest(e)\in[0,k]$表示其所在的生成森林的编号，不属于任何生成森林时为0；
- $label(e)$为标记法宽搜时e的标记，未标记时为null。

算法整体实现

- 初始化$k+1$个并查集、所有边的$forest$值置为0；
- 把所有边按照$value$从小到大排序；
- 按顺序依次考虑每条边$e_0=(u,v)$，检测是否有$find_0(u)=find_0(v)$；
- 若成立，则跳过这条边；否则从e_0出发进行宽搜；
- 如果找不到e_0出发的增广序列，说明u和v在同一个丛中，此时执行$union_0(u,v)$；
- 如果取出$e=(x,y)$时发现$find_i(x)\neq find_i(y)$而结束搜索，则执行$union_i(x,y)$，并重复执行$swap(index(e),i)$和赋值$e=label(e)$直到$e=\text{null}$。

宽搜具体实现(1)

- 队列清空、所有边的$label$值置为null，把$e_0 = (u, v)$加入队列；
- 在每个生成森林F_i中找到u所在的树T_i，令u为T_i的根并计算每个点x的父节点$p_i(x)$；
- 引理5：①F_i中已标记的边构成T_i的一棵包含u的子树；
- ②当$e = (x, y)$被从队头取出时，x或y在$F_{index(e)+}$已标记的边构成的子树中。
- 证明：每条被标记的边要么与u相连，要么与之前已经被标记的边相连，对搜索过程中边的标记进行归纳，引理显然正确。

宽搜具体实现(2)

- 因此我们每次取出队头元素$e=(x,y)$，令$i=index(e)+$；
- 若$find_i(x)\neq find_i(y)$，说明x,y在F_i的不同子树中，找到了增广序列，结束搜索；
- 否则令z是x,y中满足$z\neq u$并且$label(e'=(z,p_i(z)))=\text{null}$的点；
- 把边e'入队并令$label(e')=e,\ z=p_i(z)$，直到$z=u$或者$label((z,p_i(z)))\neq\text{null}$；
- 重复以上步骤，直到搜索结束或者队列为空；
- 一次宽搜的复杂度为$O(kV)$。

其它问题

- 至此我们在$O(k^2V^2+ElogE)$的时间内解决了Minimum Spanning Trees问题，当$k=2$时即可应用于Shannon Switching Game。
- 接下来对算法做一些修改来求无向图中边不相交生成树的最多个数：
- 尝试从e_0出发进行增广时，$\forall i=[1,k]$，在$F_1,F_2,\ldots,F_i$中BFS寻找增广序列；
- 如果失败，把F_i中已标记的边放回队列，不清空标记并继续在$F_1,F_2,\ldots,F_{i+1}$中寻找；
- 如果$i=k$时仍然失败，则e_0成为新的生成森林F_{k+1}中的边，并令$k\leftarrow k+1$。
- 对所有边增广结束后，找到最大的k满足F_k是一棵树，k即为所求的答案。

知识扩充篇

图灵机与计算复杂性理论

图灵机、递归定理、可归约性、时间复杂性与空间复杂性

生命科学中的一个悖论

- （1）生物都是机器；

  这是现代生物学的宗旨之一，生物体可以看做是以机械方式运作的。
- （2）生物都能自再生；

  显然，自再生（繁殖）是生物物种的本质特征。
- （~~3~~）机器都不能自再生；

  设计生产线比设计它制造的东西复杂，因为生产线的设计包含其制造物品的设计。

  因此没有机器能够制造自己，故机器不可能自再生。

引入

- 机器可以制造自己吗？如何制造？
- 请大家思考这样一个问题：
- 编写一段C++程序，该程序打印出自身。

  （程序的代码和输出是完全一样的）
- 可以在纸上尝试一下……

抽象化

- 在前几天的课程中，乔明达大神给大家讲解了正则语言、DFA与NFA。
- 而程序设计语言中的语法大多是上下文无关文法，可以用附带一个栈的自动机——下推自动机（PDA）来识别。
- 我们称一个机器的实现方法叙述为它的"描述"。
- 由于丘奇-图灵论题告诉我们，算法的定义等价于图灵机描述，因此我们来考虑：
- 制造一个图灵机，该图灵机打印出自己的描述。

初识图灵机

- 图灵机(TM)是一种通用的计算机模型，它与有穷自动机类似，但是具有无限大容量的存储和任意访问内部数据的能力。
- 图灵机具有一个无限长的纸带，开始运行时带上只有输入串，其余位置为空；
- 图灵机具有一个双向读写头，可以纸带上左右移动，读取或者修改纸带上的信息；
- 图灵机预置接受和拒绝两种状态，一旦进入则立即停机，没有进入则持续运行永不停止；
- 与有穷自动机类似，分为确定型（DTM）和非确定型（NTM）、单带和多带等。

图灵机的形式化定义

- $TM = (Q, \Sigma, \Gamma, \delta, q_0, q_{ac}, q_{re})$
- Q是有穷状态集；
- Σ是输入字符集；
- Γ是纸带字符集（必须包含空格符）；
- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$是转移函数；
- $q_0, q_{ac}, q_{re} \in Q$分别是初始状态、接收状态、拒绝状态，其中$q_{ac} \neq q_{re}$；

格局与语言

- 设当前纸带上的内容为$s_0, s_1, s_2, \ldots, s_m$，当前状态为$q$，读写头处于$s_i$位置。
- 把字符串$C = s_0, s_1, \ldots, s_{i-1}, q, s_i, s_{i+1}, \ldots, s_m$称为它的格局。
- 如果格局C_1经过一次转移可以到达C_2，则称C_1产生C_2。
- 对于$TM\ M$和输入w，如果存在格局序列$C_0, C_2, \ldots, C_k$满足：①C_0是M在输入w上的初始格局；②C_k是接受格局；③每个C_i产生C_{i+1}；则称$TM\ M$接受输入w。
- M接受的字符串的集合称为M识别的语言$L(M)$。

图灵机的运行结果

- 停机接受；
- 停机拒绝；
- 不停机。
- 对于所有输入都停机的机器称为判定器。
- 如果一个语言能被某一图灵机识别，则称该语言是图灵可识别的。
- 如果一个语言能被某一图灵机判定，则称该语言是图灵可判定的。

图灵机的三种描述

- 形式化描述

  就像上面那样使用数学符号严格定义。
- 实现描述

  使用文字叙述，精确到指出每步的情况和读写头移动的方式。
- 高水平描述（通常使用）

  使用文字叙述，就像叙述算法那样指出图灵机执行的工作和停机条件。
- 机器的描述可以看做对一个对象进行编码，用<M>代表TM M的描述。

可计算函数

- 函数$f\colon \Sigma^* \to \Sigma^*$是一个可计算函数，如果有某个图灵机$M$，使得对于任意输入$w$，$M$停机，且此时只有$f(w)$出现在带上。
- 引理：存在可计算函数$q\colon \Sigma^* \to \Sigma^*$，对任意串$w$，$q(w)$是$TM\ P_w$的描述。其中$P_w$打印出$w$，然后停机。
- 证明：右面的$TM\ D$实现了$q(w)$，因此$q(w)$是可计算函数。

$TM\ P_w =$ "对于任意输入:

1)抹去输入;

2)在带上写下w;

3)停机;"

$TM\ D =$ "对于输入w:

1)抹去输入;

2)输出$< P_w >$;

3)停机;"

构造自我复制机SELF

- 把SELF分成两部分A、B实现，A的任务是打印出B，B的任务是打印出A，但是它们的实现方式不同。
- B是对输入w，打印q(w)和w的图灵机；
- A是对任意输入，均打印<B>的图灵机；
- 首先运行A部分，结束时纸带上出现<B>，接着运行B部分，它打印q(<B>)=<A>，把<A>接在<B>的前面就得到了SELF。

$TM\ A = P_{<B>}$

$TM\ B$ = "对于输入 $< M >$, 其中M是TM:

1)计算$q(< M >)$;

2)将其结果与 $< M >$ 合成，组成一个完整的TM描述;

3)打印这个描述，然后停机;"

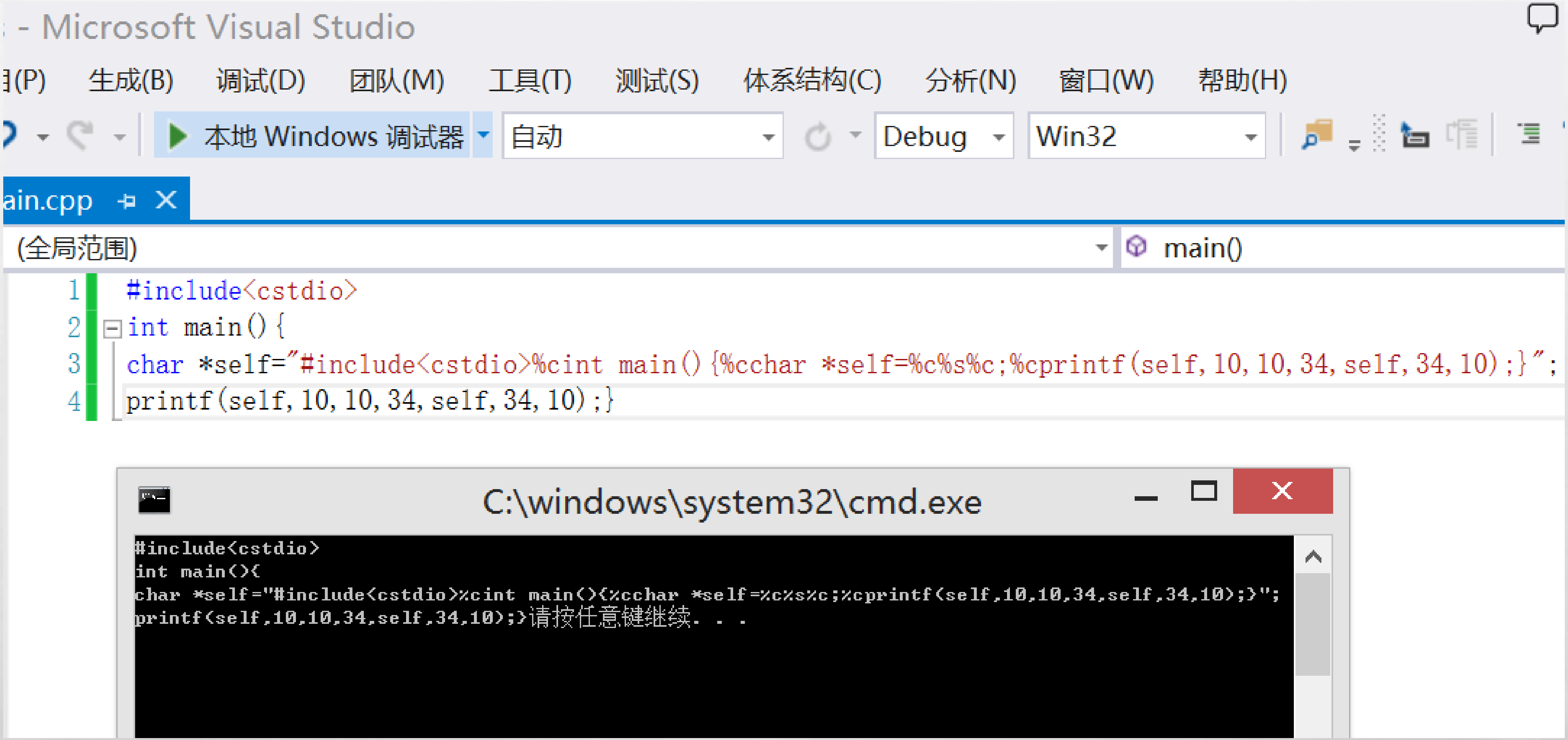
$TM\ SELF = AB$

递归定理

- 递归定理：设T是计算$t:\ \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$的一个图灵机，则存在计算$r:\ \Sigma^* \rightarrow \Sigma^*$的一个图灵机$R$，使得对于每一个$w$，有：$r(w) = t(< R >, w)$。
- 递归定理实际上是说，如果我们想制造一个得到自己的描述并用于计算的图灵机R，只需要制造一个定理中提到的图灵机T，如果我们给T一张说明书和一堆原材料，T就会看着这张说明书去在原材料上模拟说明书上的操作。这样的"模拟器"是不难制造的，当我们制造出T后，就有一个叫做"递归定理"的奇♂妙的神犇跑过来瞬间给我们制造了一个图灵机R，这个图灵机在w上运行的结果，和T看着$< R >$在w上运行的结果是一样的。这也就是说R跟一个得到自己的描述并用于计算的图灵机是等价的！

递归定理

- 递归定理的证明与SELF的构造类似，此处略去。
- 根据递归定理，以下描述在图灵机中是合法的：“得到自己的描述<M>”。
- 所以我们可以运用递归定理来实现$SELF$的描述：
- $TM\ SELF$ = “对于任意输入:
  1)利用递归定理得到自己的描述$< SELF >$;
  2)输出$< SELF >$; ”


- Microsoft Visual Studio
](P) 生成(B) 调试(D) 团队(M) 工具(T) 测试(S) 体系结构(C) 分析(N) 窗口(W) 帮助(H)
本地 Windows 调试器
自动
Debug
Win32
ain.cpp
(全局范围)
main()
#include<cstdio>
int main(){
char *self="#include<cstdio>%cint main(){%cchar *self=%c%s%c;%cprintf(self,10,10,34,self,34,10);}";
printf(self,10,10,34,self,34,10);}
C:\windows\system32\cmd.exe
#include<cstdio>
int main(){
char *self="#include<cstdio>%cint main(){%cchar *self=%c%s%c;%cprintf(self,10,10,34,self,34,10);}";
printf(self,10,10,34,self,34,10);}请按任意键继续. . .

可判定性

- 对于可判定语言，我们需要用高水平描述给出它的一个判定器。
- 然而存在着不可判定的语言，即图灵机可能不会停机。
- $A_{TM} = \{< M, w > | M$是一个TM, w是一个串, M接受$w\}$ 是不可判定的。
- 假设$TM\ H$判定A_{TM}，构造右面的图灵机$TM\ D$。

$TM\ D$ = “对于输入w:

1)利用递归定理得到自己的描述 $< D >$

2)在输入 $< D, w >$ 上运行H;

3)若H拒绝,则接受;若H接受,则拒绝;”

如果$TM\ H$判定A_{TM}，若它接受$< D, w >$，则说明$TM\ D$接受输入w，而$TM\ D$实际上总返回与H相反的结果，即D拒绝w。这产生了矛盾，因此A_{TM}不可能被判定。

归约

- 若存在可计算函数$f:\Sigma^*\to\Sigma^*$，对于每一个w，有$w\in A\Leftrightarrow f(w)\in B$，则称语言$A$映射可归约到语言$B$，记为$A\le_m B$。
- 若$A\le_m B$且B可判定，则A可判定；
- 若$A\le_m B$且A不可判定，则B不可判定；
- 例：$\mathrm{HALT}_{\mathrm{TM}}=\{<M,w>|M$是一个TM，且对输入w停机$\}$是不可判定的。

假设TM R判定$\mathrm{HALT}_{\mathrm{TM}}$，构造TM S判定$\mathrm{A}_{\mathrm{TM}}$。

$TM\ S=$ “在输入$<M,w>$上，

M是TM，w是串：

1)在输入$<M,w>$上运行$TM\ R$。

2)如果R拒绝，则拒绝。

否则在w上模拟M直到停机。

3)若M接受，则接受；

若M拒绝，则拒绝。”

因此$\mathrm{A}_{\mathrm{TM}}\le_m \mathrm{HALT}_{\mathrm{TM}}$，故$\mathrm{HALT}_{\mathrm{TM}}$不可判定。

P与NP

- 时间复杂性类$TIME\left(t(n)\right) = \{L \mid 存在O\left(t(n)\right)时间的单带DTM判定语言L\}$
- 时间复杂性类$NTIME\left(t(n)\right) = \{L \mid 存在O\left(t(n)\right)时间的单带NTM判定语言L\}$
- P是确定型单带图灵机在多项式时间内可判定的语言类：$P = \bigcup_k TIME\left(n^k\right)$
- NP是非确定型单带图灵机在多项式时间内可判定的语言类：$NP = \bigcup_k NTIME\left(n^k\right)$
- NP是具有多项式时间验证机的语言类。
- 验证：给定一个问题和一个解，检查解的正确性。（区分验证、判定、识别）

NP完全性

- 若存在多项式时间可计算函数$f: \Sigma^* \rightarrow \Sigma^*$，对于每一个$w$，有$w \in A \Leftrightarrow f(w) \in B$，则称语言$A$多项式时间映射可归约到语言$B$，记为$A \leq_p B$。
- 若语言$B \in P$且$A \leq_p B$，则$A \in P$。这意味着只要B在多项式时间有解，则A也有。
- 若一个问题是多项式时间可验证的，则称它是NP问题；
- 若一个问题满足所有的NP问题多项式时间可归约到它，则称它是NP-Hard问题；
- 同时满足上面两个条件的问题被称为NP完全问题（NPC问题）。
- 只要有一个NPC问题$\in P$，那么$P = NP$。

第一个NP完全问题 – SAT

- 若存在一种赋值使包含布尔变量和运算的表达式为真，则称该表达式是可满足的布尔公式。
- 库克-列文定理：SAT={φ|φ是可满足的布尔公式} 是NP完全的。
- 这个定理从图灵机的格局出发，把经历的格局写成一个矩阵，并建立若干个布尔变量a[i,j,k]，取值为真时表示矩阵的(i,j)位置是k。然后通过构造布尔公式及其可满足性来检查初始和接受格局是否合法、格局之间的转移是否合法。由于任何一个问题都可以用图灵机实现，所以问题的是否有解就归约为布尔公式是否可满足。
- 在更多的时候，从一个已知的NPC问题出发归约到另一问题来证明其NP完全性。

多项式时间归约 – 3SAT

- 布尔变量或它的非称为文字，V连接的若干文字称为子句，^连接的若干子句称为合取范式。每个子句都有3个文字的合取范式称为3cnf公式。
- 3SAT={φ|φ是可满足的3cnf公式} 是NP完全问题。
- 显然 3SAT∈NP，下面只要证明SAT$\le_p$3SAT。
- 利用分配律可以把库克-列文定理中证明SAT的布尔公式转化为等价的合取范式，然后对子句$(a_1 \vee a_2 \vee \ldots\ldots \vee a_l)$，用$(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \wedge \ldots\ldots \wedge (\neg z_{l-3} \vee a_{l-1} \vee a_l)$替换，就得到了等价的3cnf公式。

团问题 - Clique

- CLIQUE={<G,k>|G是包含k团的无向图} 是NP完全的。
- CLIQUE的验证机="对输入<<G,k>,c>：1) 检查c是否为G中k个结点的集合；2) 检查G是否包含连接c中结点的所有边；3) 若两项检查都通过，则接受，否则拒绝。"
- 3SAT$\leq_p$CLIQUE：设φ是k个子句的公式，该归约生成<G,k>。G中的节点分为k个三元组，三元组对应子句，节点对应文字并做相应标记。除了同一三元组内节点、相反标记的两个节点之外都有边相连。φ可满足当且仅当G有k团。

PSPACE

- 类似地，我们还可以定义PSPACE、NPSPACE类以及它们的完全性。
- P⊆PSPACE, NP⊆PSPACE.

- 大部分时间、空间复杂性类之间的关系还是悬而未决的，例如P是否等于NP。
- 更多问题还有待大家讨论和思考。

总结

- 今天上午我们解决了以下问题：
- 无向图割点、桥、点/边双连通分量的求法；
- 有向图必经点(割点)/必经边(桥)的求法——Dominator Tree与Lengauer-Tarjan算法；
- 边不相交生成树的相关问题，与Shannon Switching Game的必胜策略的联系；
- 图灵机的相关定义、描述与基本应用；
- 计算复杂性的初步认知；

谢谢大家

E-mail：lydrainbowcat@pku.edu.cn

CCF NOI2014 Winter Camp, 2014年2月10日

鸣谢：

CCF NOI 科学委员会

Robert E. Tarjan

复旦大学 刘炎明

北京大学 唐浩 杜宇飞

石家庄市第二中学 章玄润

最佳浏览环境：

Microsoft Office 2013 Pro
on Windows 8.1 Pro