专题选讲

清华大学交叉信息研究院 杨骏昭

最大流 最小割 2-SAT

问题引入

- 给定有向带权图，两个点s和t，求s-t最小割是否唯一。（CF gym 100200A）
- 对于每条边，判断是否可能在最小割中?
- 是否一定在最小割中?

网络流的定义

- $f(u,v)$表示u到v的流量，满足$f(v,u)=-f(u,v)$
- $c(u,v)$表示u到v的最大流量限制，满足$f(u,v)<=c(u,v)$
- 对于除去源点s和汇点t以外的u，满足流量平衡限制：$\sum f(u, x) = 0$
- （假设$f(s,t)=0$）

几个性质

- 证明:

  1. \sum f(s, x) = \sum f(x, t)

  记这个值为网络流的流量：value(f)

  2. 若某集合S包含源点s且不包含汇点t，那么

$$\sum_{x \in S, y \notin S} f(x, y) = value(f)$$

s-t割的定义

- S和T是V的一个划分，且S包含s，T包含t。
- 定义 $c(S,T) = \sum_{x \in S \wedge y \in T} c(x,y)$
- 证明：value(f) <= c(S, T)

另一种割的定义

- 割是边集的一个子集，满足从原图中删去割中的边则s到t不连通。割的权值为割边权值之和。
- 之前定义的s-t割属于这种割。
- 证明：这种定义的割的最小权值不会小于之前定义的s-t割的最小权值。
- 接下来我们只考虑s-t割。

最大流-最小割定理

- s-t的最大流的值与s-t最小割的值相同。
- （线性规划中对偶定理的一个特殊情况）

残余网络

- $r(u, v) = c(u, v) - f(u, v)$
- 只考虑$r(u, v)>0$的边
- 增广路径：残余网络上的每条边的r都大于0
- ford-fulkerson算法：沿增广路径增广直到不存在增广路径

Proof

- 考虑残余网络上s能到达的点集S，设T = V/S
- 那么一定有c(S,T) = value(f)
- 此时一定满足最大流、最小割

最小割的条件

- 对于任意最大流f，s-t割(S,T)为最小割当且仅当对于任意割边(u,v)都有r(u,v)=0。
- 原图最小割 <=> 残余图的最小割
- 最小割的充要条件：s在S中，t不在S中
  对于所有r(u,v)>0，u在S中则v也要在S中
- 最小割解集 => 2-SAT解集

最小割的2-SAT结构

- 三种标记：未标记、必定在S中，必定在T中
  记为0, S, T
- 点限制：
  x在S中：将x能到达的点都标记为S
  x在T中：将能到达x的点都标记为T
- 边限制转化为点限制：
  (u,v)在s-t最小割中：u在S中，v在T中

最小割的2-SAT结构

- O标记的点同时记为S，或同时记为T都是合法的
- 判断当前限制下是否存在最小割：
  不存在某个点被同时标记为S或T
- 判断当前限制下最小割是否唯一：
  不存在O标记的点

例题应用

- 判断(u,v)是否可能在s-t最小割中?
- 判断(u,v)是否一定在s-t最小割中?

例题应用

- 判断(u,v)是否可能在s-t最小割中?
- 将(u,v)权值减少eps，检查最小割是否变小
- 若r(u,v)>0，则最小割一定不会变小
  若r(u,v)=0，则等价于能否退流
  即是否存在u到v的增广路径
- 2-SAT做法：
  u设为S，v设为T且不合法 => 存在u到v的增广路径

例题应用

- 判断(u,v)是否一定在s-t最小割中?
- 将(u,v)权值增加eps，检查最小割是否变大
- 最小割变大 => 最大流增加
  当且仅当存在S到u且v到T的增广路径
- 2-SAT做法：
  若u或v为0标记，则可能不在，否则一定在

动态凸包

二维凸包

- 凸包：上凸壳 下凸壳（只考虑上凸壳）
- 维护点集S的上凸壳
- 可能需要支持的操作：
  修改类：加入、删除一个点
  询问类：整体/区间 询问 凸包上的边/某个方向最远点

- 考虑支持求出凸包

- 修改操作的性质：
离线/在线
只有加/只有删/又有加又有删
加的点坐标单调/不单调
可持久化（回到历史版本）/不可持久化

- 离线只有删 <=> 离线只有加
在线 => 离线

几种凸包算法

- Graham 要求加点坐标单调
- 将凸壳上的点顺序存于数组中
- 支持在线加点，单次复杂度O(delta凸壳点数），总复杂度O(n)

- 魔改版Graham 要求加点坐标单调
- 将凸壳的点存在树上
- 二分在线加点，单次O(log 当前凸壳点数)，总复杂度O(n log n)
- 支持可持久化

- 平衡树版 Graham(?) 不要求加点坐标单调
- 将凸壳上的点按照某一维坐标为关键字，排在平衡树上
- 在线加点，暴力删除周围点
- 如果使用std::set(finger search)，则单次复杂度O(delta凸壳点数+log(凸壳点数))，总复杂度O(n log n)，不可持久化
- 如果手写平衡树，O(1)删除子树（修改指针），则单次复杂度O(log(凸壳点数))，可持久化

- 完全动态凸包
- 在线 不要求单调 可加可删 可持久化
- 两类实现方法：
  1. 类线段树实现
  2. 可持久化平衡树 cls论文《可持久化数据结构研究》

可持久化平衡树

- 只是可以回到历史状态的平衡树而已
- 可持久化的本质：记录所有时刻的历史版本，“垃圾不回收”
- 维护的所有版本原则上不可修改，修改应视为新的版本
- 适用于指针维护的数据结构，可持久化代价为一个点所连接的指针数
- 理论上非均摊复杂度的平衡树都可以可持久化，但是非旋treap比较好写

可持久化平衡树版完全动态凸包

- 对于两个x轴上不相交的上凸壳，合并后为前缀+一条边+后缀
- 为快速合并凸壳且保持结构完整，我们可以用可持久化平衡树维护按照x坐标排序的凸壳点集
- 外层使用平衡树等结构动态维护整个点集，树中每个点维护一个可持久化平衡树表示子树内点集凸壳即可

完全动态凸包

- 线段树结构，叶子节点维护一个单独的点，非叶子结点维护子树（区间）内点集凸壳信息
- 要求左子树的点集的坐标严格小于右子树的点集的坐标（相当于将所有点按坐标排序后建线段树）
- 假设我们已经求出了左右子树的凸壳，那么合并后的凸壳一定是左子树的凸壳的一段前缀+一条边+右子树凸壳的一段后缀
- 核心思想：每个节点只维护这条跨左右子树的边

- 记号：
  节点x维护的边记作bridge(x) = (p(x), q(x))
  叶子结点维护的点记作a(x)
  对于叶子结点，我们令bridge(x) = (a(x), a(x))

- 注意我们的结构并没有直接地表示出凸壳
  对于节点x，考虑它子树点集的凸壳，将凸壳上的点称为关键点，更小的凸壳上一定会出现这些关键点

- 设这些关键点对应的叶子节点集合为S，设T为S形成的虚树

- T有|S|个叶子结点，|S|-1个非叶子结点，而这|S|-1个非叶子结点维护的边按照中序排列即为x的子树的凸壳

- 如何判断哪些节点在虚树上？（哪些边是凸壳的边？）

- 考虑暴力过程：先求出左右子树的凸壳，再删除被bridge(x)包含的边（包含指坐标区间包含）

- 等价于：从x开始dfs，每次只加入没有被祖先的bridge包含的边

- 判断bridge(u)是否在x节点子树的凸壳上的条件：
  对于所有fa(u)到x的路径上的v，bridge(u)不被bridge(v)包含

- 如何计算bridge：（合并左右子树的凸壳）
- 考虑bridge(x)连接了左子树的p(x)和右子树的q(x)
- 所有p(x)左方的边向右延长的射线在右子树凸壳的上方
- 同理q(x)右方的边向左延长的射线在左子树凸壳的上方
- 考虑二分出p(x)和q(x)

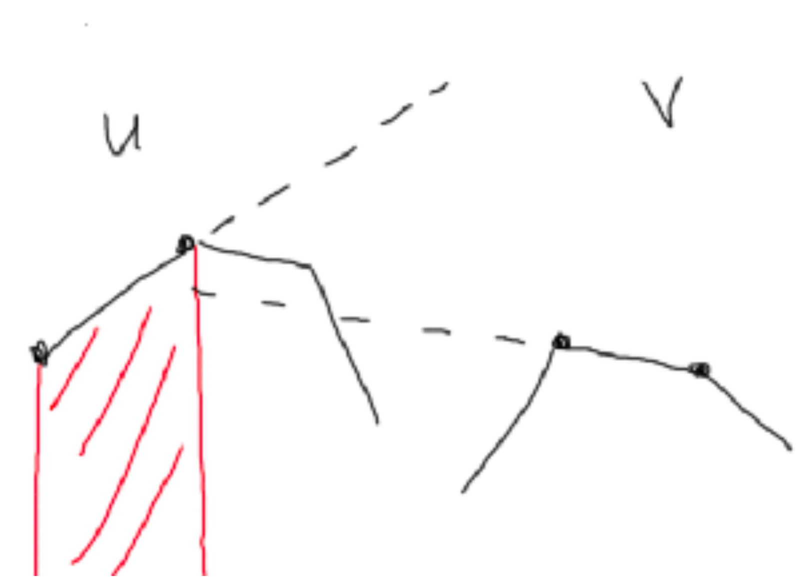
- 单独二分一边时需要对另一边求某个方向的最远点，再次二分的话需要两个log

- 考虑一起在树上二分，假设我们肯定p(x)和u的子树内，q(x)在v子树内

- 若bridge(v)中的某一点在bridge(u)射线上方，那么p(x)一定在u的左子树内

- 同理，若bridge(u)中的某一点在bridge(v)射线上方，那么q(x)一定在v的右子树内

- 如果出现这两种情况，我们就可以缩小范围



- 如果两个情况都不满足，我们可以考虑bridge(u)和bridge(v)射线的交点
- 假设交点的坐标不在u子树的坐标范围内，那么q(x)一定在v的左子树内。同理如果不在v子树的坐标范围内，那么p(x)一定在u的右子树内。
- u和v子树的坐标范围不交，所以以上两种情况至少会发生一种。
- 计算bridge(x)的复杂度O(x子树的深度)

- 我们数据结构里节点需要维护的信息有：
bridge(x), x子树的坐标范围

- 其中bridge的计算复杂度依赖于深度，坐标范围可以O(1)直接合并

- 对于离线（预先知道用到的点坐标），可以使用线段树

- 对于在线，可以用非均摊的平衡树（如treap）

- 单点修改（包括删点、加点）只会影响到到根的路径上的节点，复杂度严格O(log^2 n)

- 区间询问凸包需要合并O(log n)个结点，复杂度严格O(log^2 n)

- 对于用这个数据结构表示的凸包，可以直接遍历求出O(|S|)，也可以用隐式树状结构作询问（如某方向最远点）

- 空间复杂度O(n)

凸壳结构

- 注意到每次合并，新建的节点的左儿子是左子树的一段前缀，右儿子是右子树的一段后缀，我们也可以用cls的可持久化平衡树方法来实现，从而显式地表达出凸壳结构

- 前缀、后缀刚好就是可持久化treap的split操作。时间复杂度不变（常数就不知道变不变了），空间复杂度O(n log n)

- 好处是表示出了凸壳上的点集本身，有更多的应用；合并复杂度严格O(log n)，不与深度挂钩

思考

- 如何维护一段凸壳上的点信息？（询问凸包面积？）
- 注意可持久化treap是定义在我们的树结构上的（~~而且这违背了我们不写可持久化平衡树的初心）~~
- 我们已经用树结构维护了点集，有着天然的树结构。可持久化平衡树是必要的吗？

凸壳的可持久化树结构

- 我们称叶子结点维护一个点，非叶子结点维护一条bridge的树状结构称为“外部树结构”。

- 凸壳上的点在树中是子树里叶子的一个子集，其在外部树结构中形成的虚树称为“隐式凸壳结构”。

- 合并左右子树时，新的凸壳为左凸壳的前缀+bridge+右凸壳的后缀，而隐式凸壳结构的前缀后缀可以由O(d)个它的子树表示。（设d为深度）

- 我们可以使用可持久化树结构维护每个点的隐式凸壳结构。

可持久化树?

- 可持久化树实际上是一个DAG。对于一个点，它维护的信息实际上是它在DAG上不去重的dfs搜索树中所有点的信息和。（一个点可能会重复贡献信息）

- 相当于是一个树的压缩结构。

凸壳的可持久化树结构

- 一个点的凸壳集合可以表示为O(d)个点的已有可持久化树结构维护的集合的并，这O(d)个点在merge过程中即可求出。（d为该点子树深度）

- 除了“外部树结构”外，我们称我们维护凸壳的可持久化树结构为“内部树结构”。内部树结构中的每个可持久化节点具有O(d)条出边。

- 注意我们的修改操作并不会修改“内部树结构”，只会抛弃一些历史版本再加入一些新的版本。对于维护凸壳信息，只需要信息可加性即可。

- 该结构应该被认为是定义在外部树结构节点上的一种可加信息，它具有天然的可持久化性，可以支持历史版本回归。

完全动态凸包总结与应用

- 我们本质上用树状结构（“外部树结构”）按照坐标顺序维护了点集，并维护了在这个树状结构上定义的可加信息bridge，以此维护出了隐式凸壳结构。

- 我们也可以显式地使用可持久化树维护凸壳结构（“内部树结构”），虽然空间复杂度会变成O(n log n)，但是可以进行与凸壳点集相关的询问。（如凸包面积）

- 操作时间复杂度严格O(log^2 n)。

- 由于是可加信息（类似区间和，区间矩阵的积），应用十分广泛，可以有很多推广。比如说我们可以区间增加/修改y坐标，区间翻转y坐标（这里假设我们按照x坐标排序），也可以再加上可持久化。~~（传说中的可持久化平衡树套可持久化平衡树）~~

uoj 319 分身术

- 给定一个点集，每次询问删除其中k个点后形成的凸包面积，强制在线
- n<=100000, M=sum k <= 2000000

- 删除k个点后的凸壳相当于k+1个区间的凸壳的并
- 直接用动态凸包维护可以做到O(M log^2 n)或用论文实现O(M log n)
- 可以预处理线段树区间每个前缀后缀的凸壳信息，这样任意一个区间可以通过某个前缀+某个后缀合并而成。总复杂度O(n log^2 n + M log n) （看起来能过）

其他几个凸包算法

- 假设只需要询问某个方向最远点
- 性质：
  对于询问点集可以拆分后再取max
  若询问的斜率具有单调性可以使用two pointers优化
- 对于单个凸包：
  离线可以将询问按斜率排序，再用two pointers O(n)
  在线需要二分，单次O(log n)

- 对于区间询问某个方向最远点，可以用线段树拆分成log个询问
- 如果坐标单调且只有加点操作，可以只建线段树的一部分（等价于二进制分组）
- 如果又有加点又有删点（撤销），可以用随机增加乱数防止复杂度退化

- 完全动态凸包问题理论可以做到下界O(n log n)！
- https://arxiv.org/pdf/1902.11169.pdf

背包问题

定义

- 有n种物品，第i种质量为m[i]，价值为v[i]，数量有1个或无限个（前者称为0-1背包，后者称为无穷背包），背包容量为T，设最大质量为s。
- 特殊情况：
  v[i]=1
  恰好装满背包
  只求是否存在一种方案恰好装满背包
  求容量为1～T的所有答案
  T远大于s
  ……

判断能否恰好装满背包

- n个物品，第i种质量为m[i]，对于t=1～T询问是否存在方案使得质量和恰好为t。
- 0-1情况
- 无穷情况

判断能否恰好装满背包

- 多项式ln 再 exp
- 复杂度O(T log T)算出所有答案
- 概率算法，只能算方案数模大质数。

v[i]=1 硬币找零问题

- n个物品，第i种质量为m[i]且有无穷多个，对于t=1～T询问最少选择几个物品使得质量和恰好为t。
- 只询问一个质量的最优复杂度?
- O~(T^1.5)? 还能更优吗?

v[i]=1 硬币找零问题

- 对于质量大于B的物品，最多只会使用T/B次。用O(T/B)次FFT在O~(T^2/B)的时间内计算使用T/B次物品的情况，再此基础上用质量小于B的物品每次O(T)更新dp数组，即O(BT)。
- 设B = sqrt(T)，可达到O~(T^1.5)。

进一步优化

- 考虑如何优化质量小物品的更新效率?
- 能否一次增加多个质量小的物品?
- 0-1情况下，如何在大dp数组上增加小物品?
  (1-inf数组和任意数组的(+,min)卷积)

(+,min)卷积特殊情况优化

- 设A数组为任意数组，B数组要么为1要么为inf，两个数组长度都为L。他们的(+,min)卷积C数组为：

$$C[k] = \min_{i+j=k} A[i] + B[j]$$

- 将A数组按照值从小到大排序，我们相当于询问能从(i,j)转移到k的最小的A[i]是什么
- 将A分成根号块，每块与B数组卷积。询问时由每块的卷积数组快速判断，可以将A的候选数限制在根号内，然后暴力枚举即可。总复杂度O~(L^1.5)。

优化

- 考虑将质量在[K/2, K]之间的物品的转移加到我们的dp数组上，我们依次对初始质量位于[0,K/2)，[K/2,K)，[K, K*3/2)进行0-1物品转移，使用上述优化可以做到O~(T/K*K^1.5) = O~(T*K^0.5)。

最终算法

- 令B = T^(3/2)，处理质量大于B物品的情况。对于小于B的物品，分别取K = B, B/2, B/4, B/8 ...
- 最终复杂度O~(T^(4/3))。
- Open problem：能否做到更优？