中国计算机学会
China Computer Federation

数据结构与实际应用

清华大学 计算机科学与技术系 王逸松

wangyisong1996@gmail.com

迷思

- 我们学过很多数据结构
  - 向量、链表、栈、队列
  - 堆、线段树、平衡树、可持久化数据结构
- 这些数据结构在实际场景中有何应用呢？

本节课的内容

- 介绍一个实际问题
- 对它进行抽象和简化，使用信息学竞赛中的方法解决
- 再将其与实际场景中的解法相比较
- 希望给大家一个接触和思考实际问题的机会

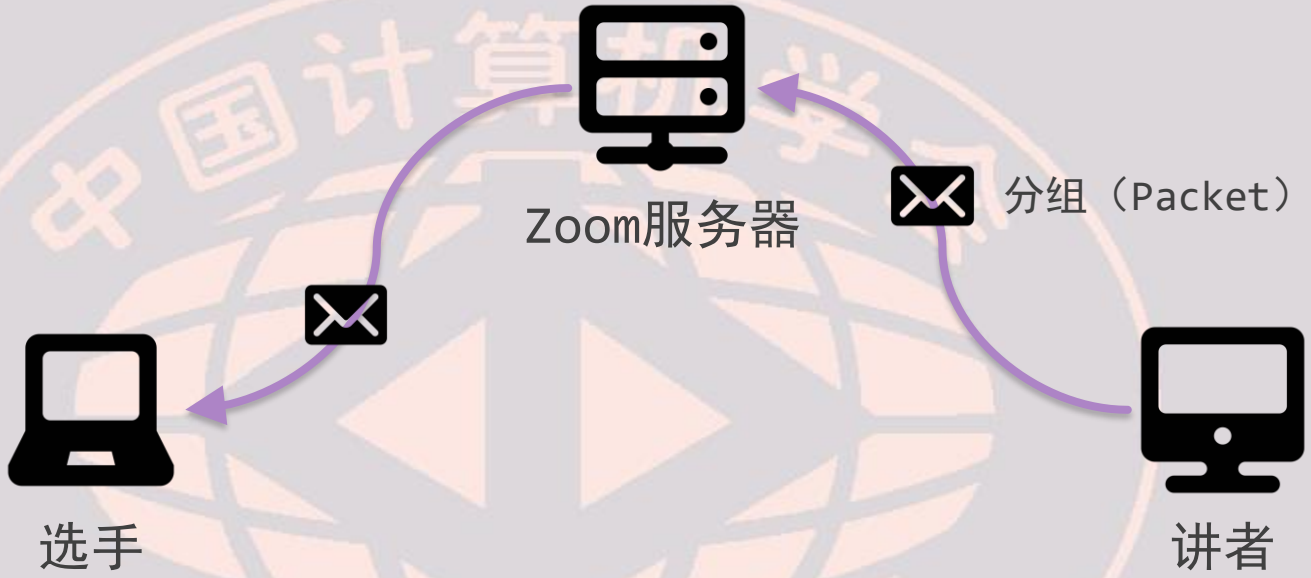
迷思

- 本节课在线上举行，讲者的声音和画面如何传输给选手？
- 通过网络（互联网）和 Zoom（会议软件）

- 这些数据（声音和画面）在网络中大致发生了什么？
- 从讲者传输到 Zoom 服务器；从 Zoom 服务器分发给各选手

- 假设能用字符串来描述这些数据，如何通过网络传输它？
- 网络传输有大小限制：拆成若干段（若干**分组**），逐个发送

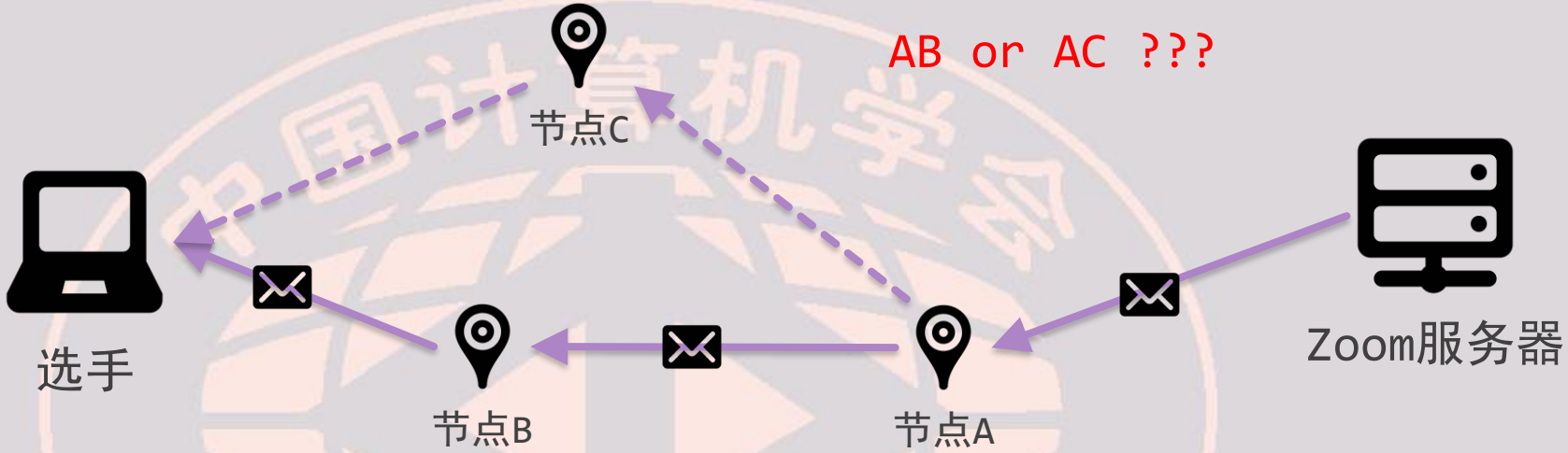
迷思



- 新的问题：网络如何知道一个分组的目的地？
- 在分组上直接标记“目的地址”
- （“这个分组发给Zoom，那个分组发给Baidu”）

图片来源：https://www.iconfinder.com/iconsets/wpzoom-developer-icon-set

迷思



- 另一个问题：网络如何确定一个分组的发送路径？
- 发送之前，在分组上直接标记发送路径？
- 不可行：网络规模巨大且持续变化，导致主机无法算出可行路径
- 网络地址的理论数量达到了 2^{32} (4.3×10^9)

IPv4地址的理论数量为2^{32}，IPv6地址的理论数量为2^{128}，两者的实际有效地址共有10^9级别

图片来源：https://www.iconfinder.com/iconsets/wpzoom-developer-icon-set

解决方案：路由器

Linksys MR8300 Mesh Wi-Fi Router
家用路由器

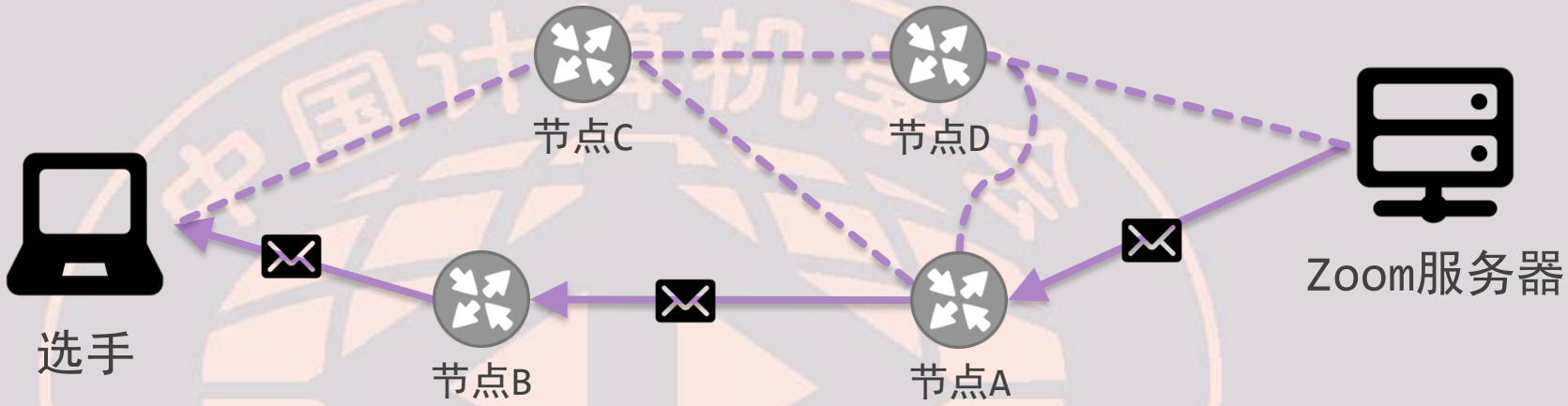
1 Gbps x 5

Cisco CRS 16-Slot Single Shelf System
互联网核心路由器

400 Gbps x 16

图片来源：https://www.apple.com/shop/product/HNKN2LL/A/linksys-mr8300-mesh-wi-fi-router
https://www.cisco.com/c/en/us/products/collateral/routers/carrier-routing-system/CRS-3_16-Slot_DS.html

解决方案：路由器

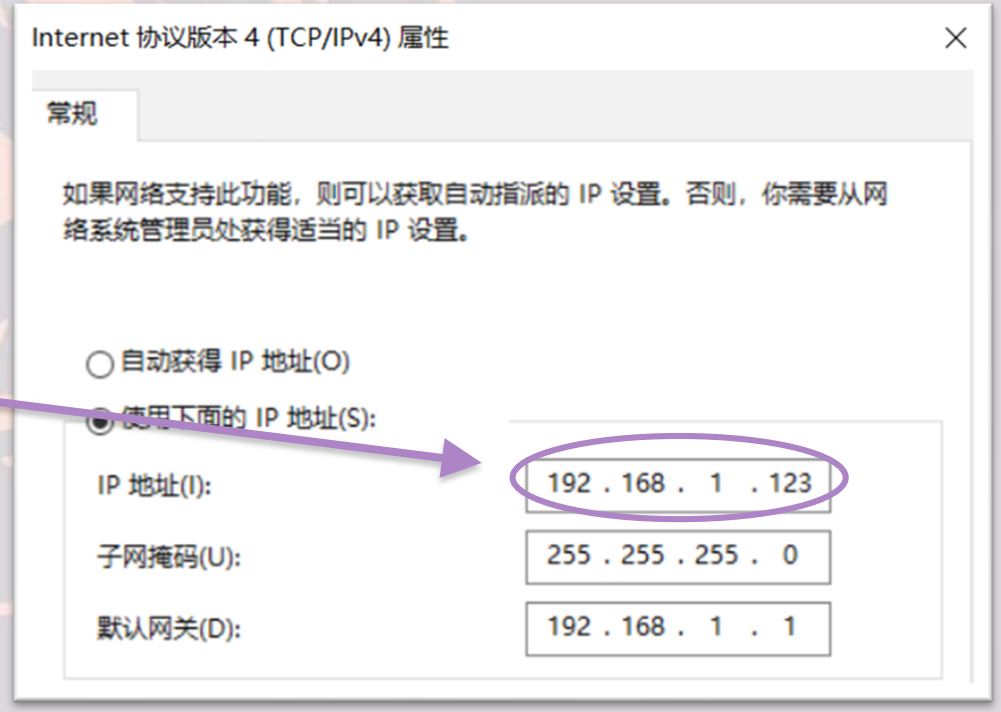


- 每个节点都有一个路由器：
  - 连接多个子网（多个节点）
  - 接收到分组时，选择正确的**下一跳**转发
- 路由器中的**路由表**存储了转发信息：
  - 给定分组的目的地址，查询其下一跳的地址

图片来源：https://www.iconfinder.com/iconsets/wpzoom-developer-icon-set
https://www.iconfinder.com/icons/575175/cisco_networking_router_stencil_visio_icon

路由表如何工作？

- “给定分组的目的地址，查询其下一跳的地址”
- 网络地址如何描述？
- IPv4地址：
  - xxx.xxx.xxx.xxx
  - 每个数在0~255之间
  - 可用一个32位整数表示



路由表如何工作？

- 路由表由一系列表项构成，每个表项包括：
  - 地址 addr（本课程仅讨论IPv4地址，即 32 位二进制整数）
  - 前缀长度 len（对于IPv4地址，len 为 0 ~ 32 之间的整数）
  - 下一跳地址 nexthop（IPv4地址）
- 每个表项中，地址从第 len 个二进制位开始均为 0
- 不存在相同地址和前缀长度的两个表项

编号	地址	前缀长度	下一跳
1	$(1000)_2$	1	0
2	$(1010)_2$	3	123
3	$(1100)_2$	2	20
4	$(0000)_2$	2	5

在此表中，为方便阅读，省略了地址中的高28位，并将下一跳视作有符号整数

路由表如何工作？

- 路由表对分组的目的地址进行**最长前缀匹配**：
  - 匹配(目的地址,addr) := 前缀(目的地址,len) == 前缀(addr,len)
  - 若有多个匹配，选择前缀最长的一个；若无匹配，返回 -1（未找到）
  - 查询结果为匹配表项的下一跳(nexthop)
- 可在路由表中添加前缀长度为0的表项，称作“默认路由”

编号	地址	前缀长度	下一跳
1	$(1000)_2$	1	0
2	$(1010)_2$	3	123
3	$(1100)_2$	2	20
4	$(0000)_2$	2	5

查询地址	查询结果
$(0100)_2$	-1
$(1001)_2$	0
$(1010)_2$	123
$(1111)_2$	20

在此表中，为方便阅读，省略了地址中的高28位，并将下一跳视作有符号整数

中国计算机学会
China Computer Federation

我们也来实现一个路由器吧

- 给定路由表（IPv4地址、前缀长度、下一跳）
- **在线**给定一些目的地址，要求给出查询结果
- 表项数 $N \approx 8 \times 10^5$，查询次数 $Q = 2 \times 10^6$
- 30s，1024MB
- https://duck.ac/problem/router32

编号	地址	前缀长度	下一跳
1	$(1000)_2$	1	0
2	$(1010)_2$	3	123
3	$(1100)_2$	2	20
4	$(0000)_2$	2	5

查询地址	查询结果
$(0100)_2$	-1
$(1001)_2$	0
$(1010)_2$	123
$(1111)_2$	20

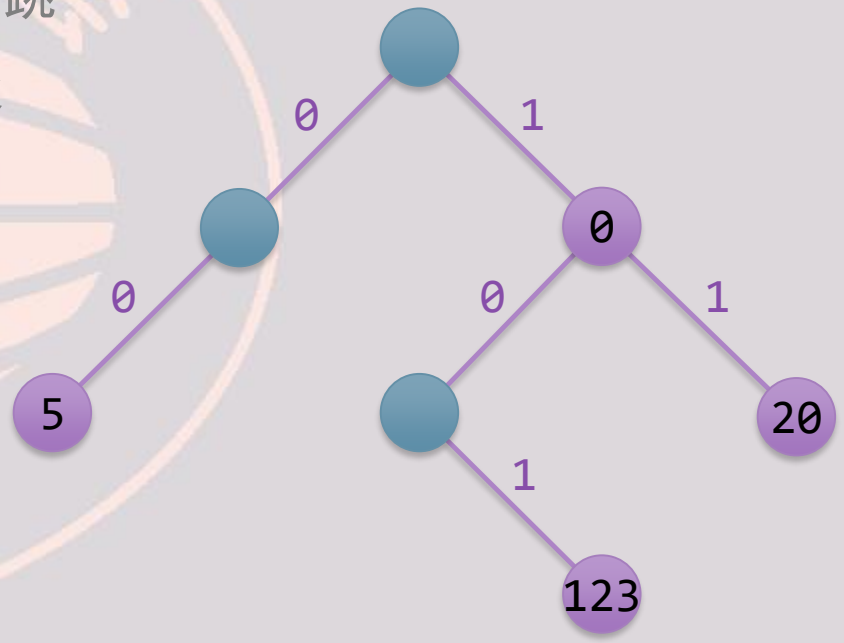
在此表中，为方便阅读，省略了地址中的高28位，并将下一跳视作有符号整数

简单的算法

- 直接存下每个地址对应的下一跳信息
  - IPv4地址有 $2^{32} \approx 4.3 \times 10^9$ 种
  - `uint32_t nexthop[1ull<<32]`
- 空间开销：16GB（16384MB）
- 时间开销：很长的初始化时间；每次查询仅需访问一次内存
- 实际性能：无法实现

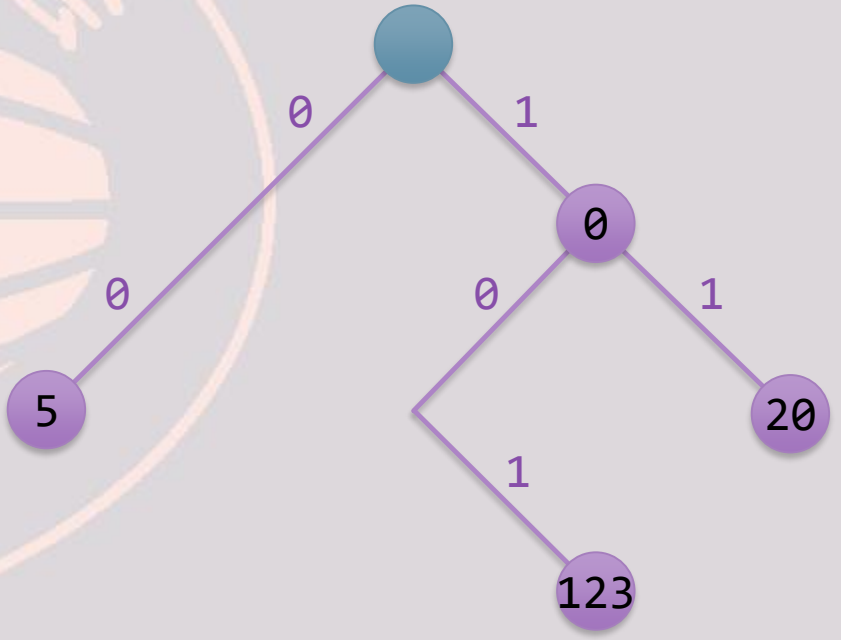
基于 Trie 树的算法

- 将IPv4地址看作32位01串，用Trie树存储
  - 在Trie树的每个结点上保存对应的下一跳
  - 查询时沿着Trie树找到最长的匹配前缀
- 时间开销：每次查询32次内存访问
- 空间开销：不超过32N个结点
- 实际性能：266 ms，52 MB
- https://duck.ac/submission/13218



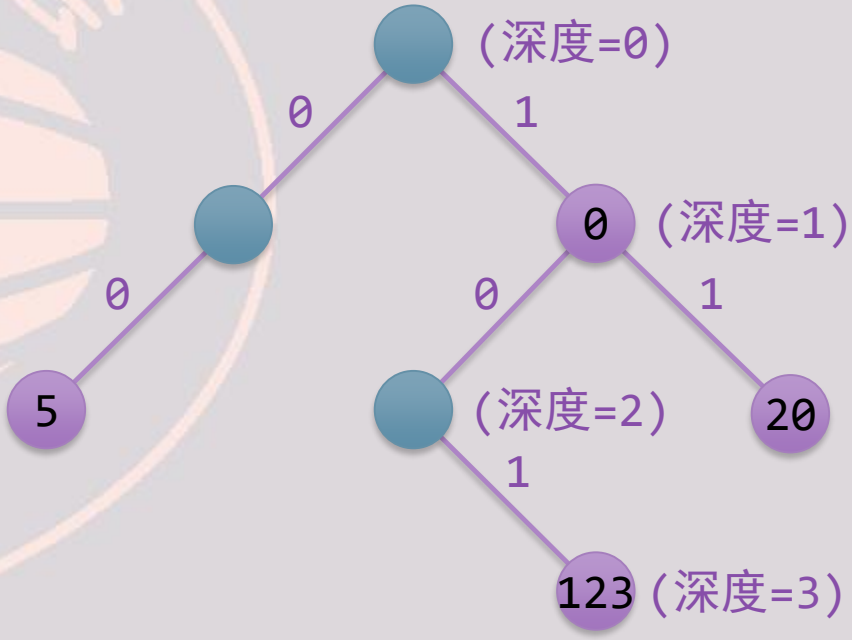
路径压缩的 Trie (Patricia)

- 在Trie树的基础上，删去“中间结点”
  - 中间结点：只有一个子结点的结点
  - 在其他结点记录该结点的前缀和深度
- 时间开销：最多32次内存访问/查询
- 空间开销：不超过2N-1个结点
- 实际性能：381 ms, 55 MB
- https://duck.ac/submission/13219



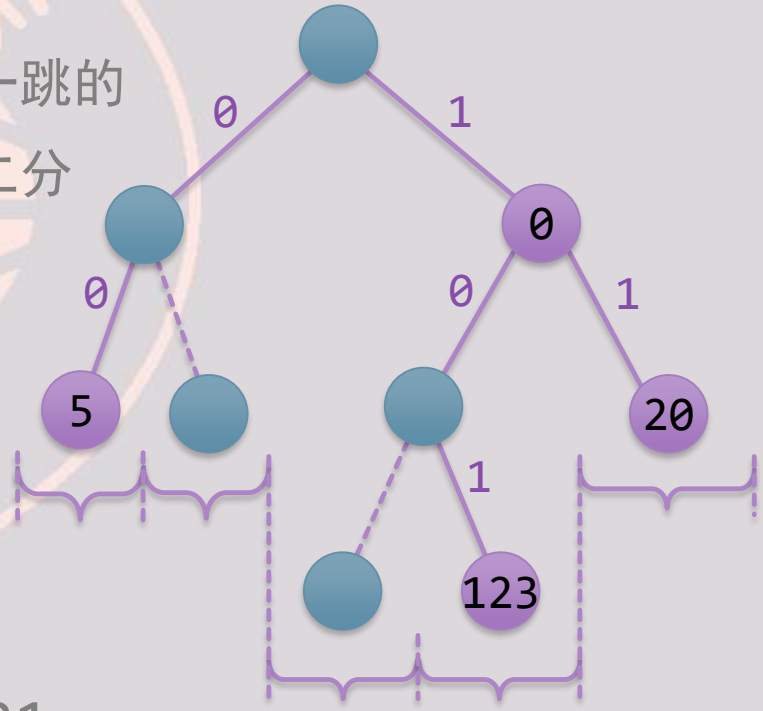
基于 Hash 和二分查找的算法

- 如何避免Trie树为找到最长匹配前缀的32次结点访问？
  - 二分查找
  - 在Hash表中保存所有结点
  - 查询时二分匹配长度（结点深度）
- 时间开销：6次Hash表访问/查询
- 空间开销：不超过32N个结点
- 实际性能：1603 ms, 86 MB
- https://duck.ac/submission/13220



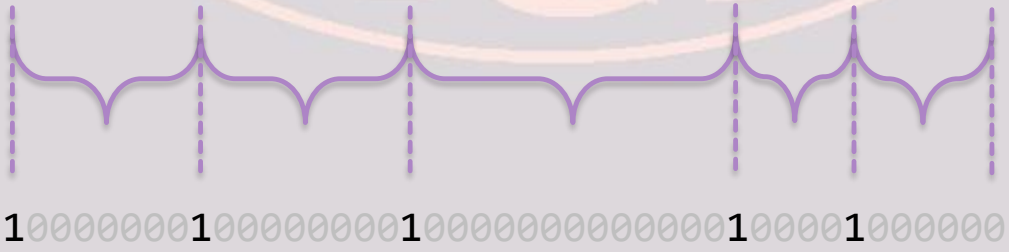
基于二分查找的算法

- 考虑在网络地址范围进行二分
  - “补全”所有结点
  - 取出所有叶结点表示的区间，合并相同下一跳的
  - 查询时用目的地址在这个“区间列表”中二分
- 时间开销：约log N次内存访问/查询
- 空间开销：不超过2N-1个区间
- 实际性能：245 ms, 17 MB
- https://duck.ac/submission/13221



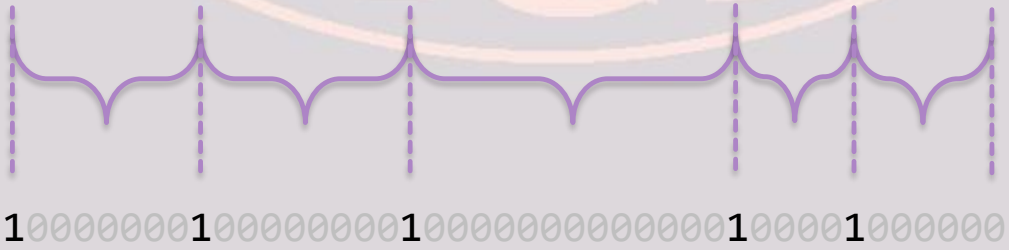
基于位图的算法

- 上一个算法的瓶颈：找到目的地址所在的区间
  - 使用二分查找， log N 次数组访问
- 如何快速查找？使用位图(Bitmap)结构
  - 每个地址对应一个位(bit)，区间左端点对应位为 1，其余位为 0
  - 目的地址所在区间编号 = sum(bit[0..addr])
- 如何计算上述前缀和？



基于位图的算法

- 如何在位图上快速计算前缀和？
  - 每64位一组，预处理以组为单位的前缀和，用32位整数保存
  - 查询时读取一次前缀和，再统计不满一组的64位中“1”的个数
  - 可用位运算和 __builtin_popcountll 快速进行64位的统计
- 空间开销：512MB（位图）+ 256MB（前缀和）
- 时间开销：前缀和访问、位图访问、下一跳数组访问各一次
- 实际性能：322 ms, 501 MB



算法的比较

算法	预处理时间	每次查询时间	空间开销
简单算法	很长	单次内存访问	16384 MB
Trie	37.2 ms	114 ns	52 MB
Patricia Trie	50.0 ms	165 ns	55 MB
Hash+二分深度	669 ms	467 ns	86 MB
二分查找	11.5 ms	117 ns	17 MB
位图前缀和	230 ms	46.3 ns	~768 MB

迷思

- 上述算法对于路由查找是否足够？

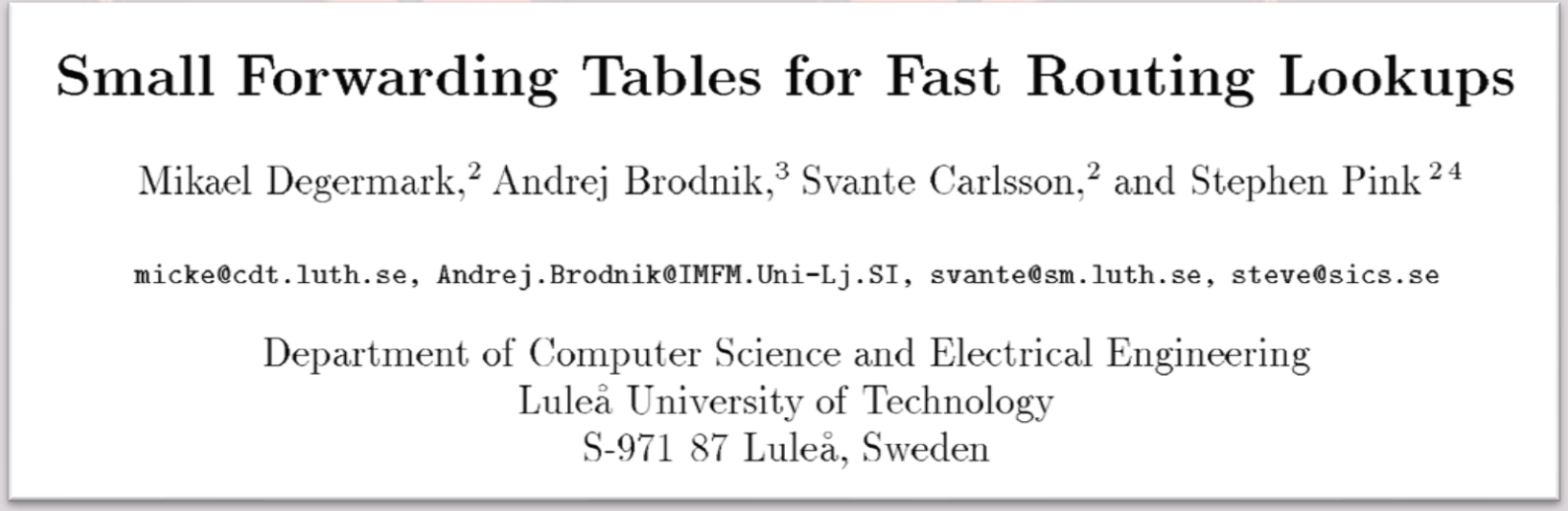
算法	每次查询时间	每秒查询次数	对应网络速率
Trie	114 ns	8.74 M	5.87 Gbps
二分查找	117 ns	8.58 M	5.76 Gbps
位图前缀和	46.3 ns	21.62 M	14.53 Gbps

- 1 Gbps = 10^9 bits/sec = 1.488 M packets/sec

- 家用路由器：1 Gbps x 5
- 互联网核心路由器：400 Gbps x 16

迷思

- 学术界如何解决此问题？
- 二十多年前，Luleå工业大学的Degermark提出了Luleå算法
- 我们在现代计算机上实现了它，性能依然优越
  - 11.08 ns/查询 (60.68 Gbps)，41 MB
  - https://duck.ac/submission/13223



Small Forwarding Tables for Fast Routing Lookups

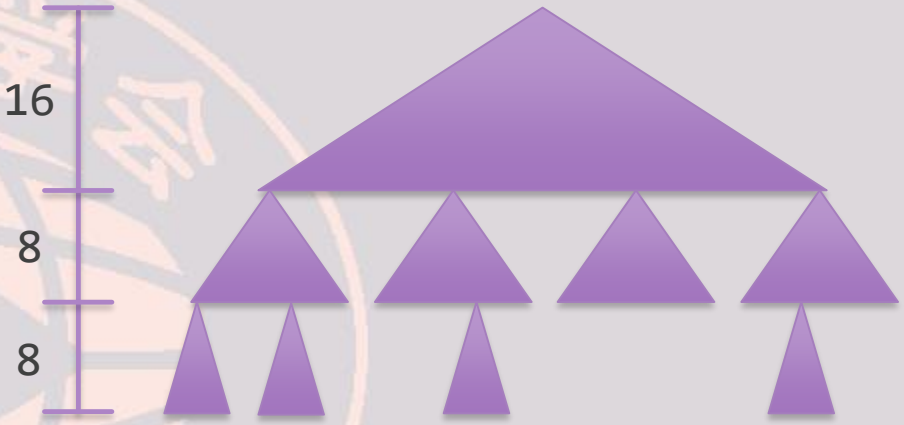
Mikael Degermark,[2] Andrej Brodnik,[3] Svante Carlsson,[2] and Stephen Pink[24]

micke@cdt.luth.se, Andrej.Brodnik@IMFM.Uni-Lj.SI, svante@sm.luth.se, steve@sics.se

Department of Computer Science and Electrical Engineering
Luleå University of Technology
S-971 87 Luleå, Sweden

Luleå 算法（简化版）

- 三层 Trie 树
  - 第一层：前 16 个 bit
  - 第二层：中间 8 个 bit
  - 第三层：最后 8 个 bit



- 预处理
  - 每一层保存“位图前缀和”结构
  - 前两层结构指向下一层，第三层结构保存下一跳
- 查询
  - 不超过 3 次“位图前缀和”查询，不超过 9 次内存访问

迷思

- 为什么 Luleå 算法快？
  - 因为空间开销小
  - （原版算法更复杂，甚至用时间换空间）

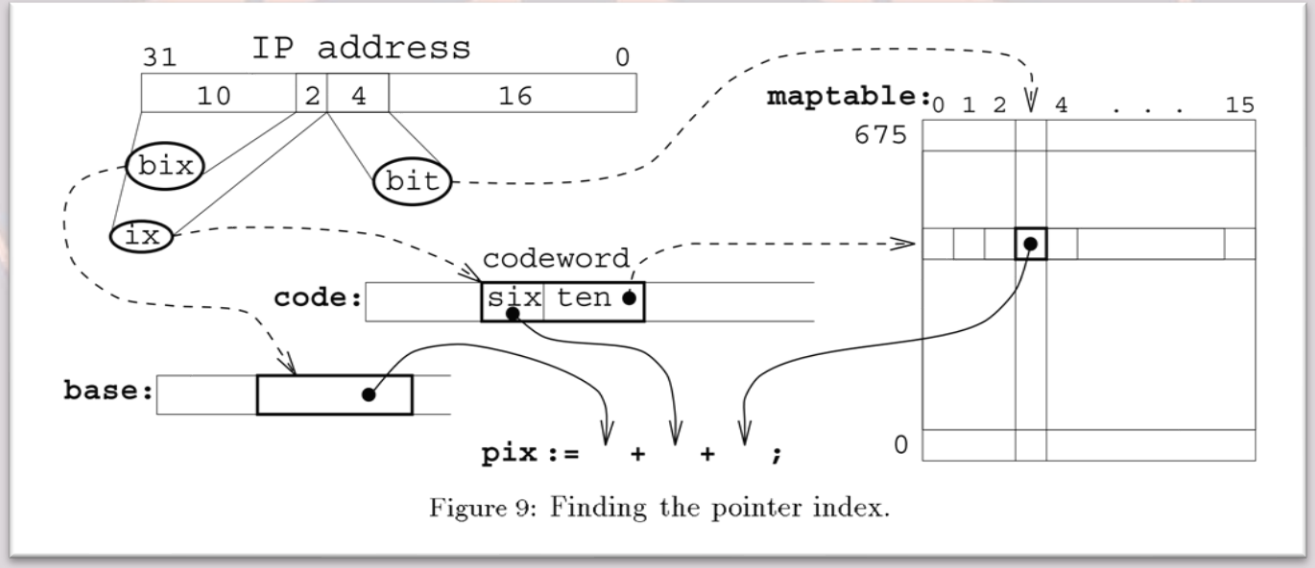


Figure 9: Finding the pointer index.

16 bit 的数被拆成 10 bit 和 6 bit:
10 bit 表示 676 种不同情况
6 bit 表示 0~63 之间的整数

迷思

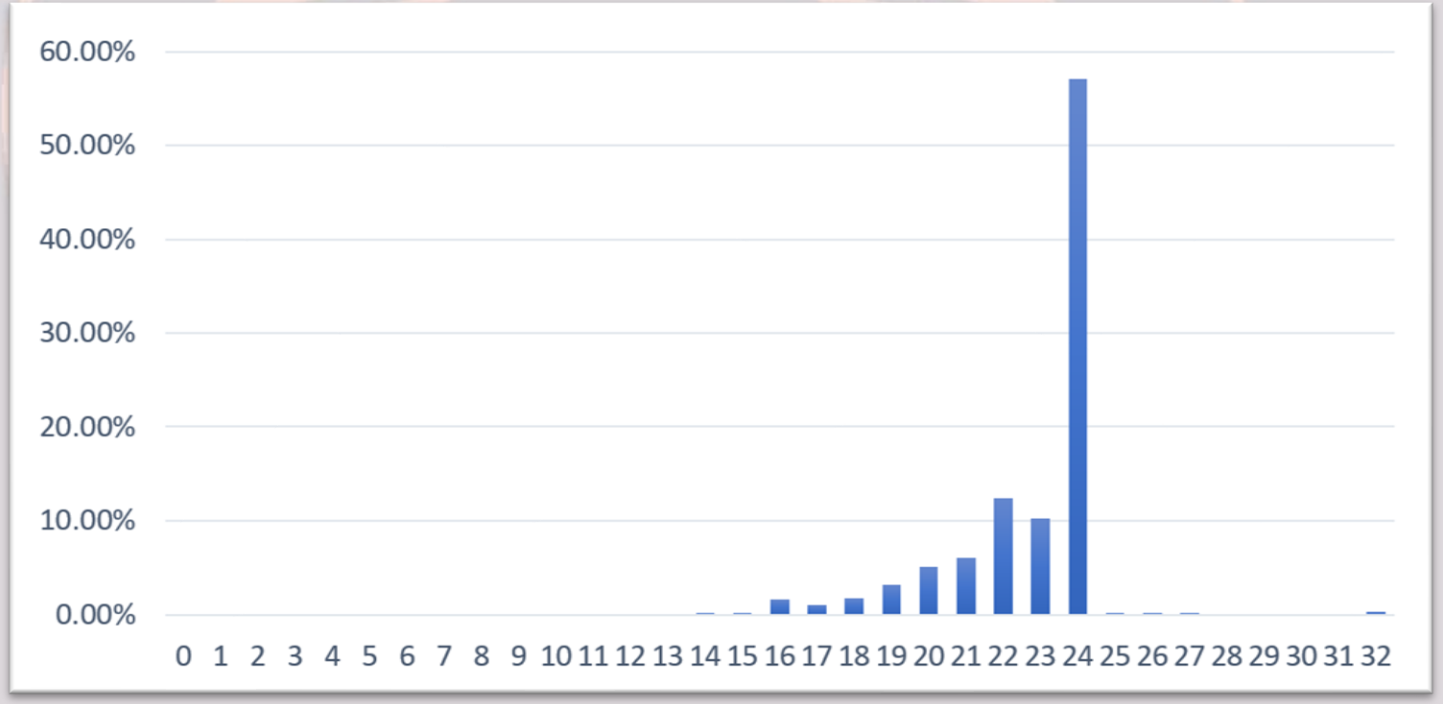
- 为什么 Luleå 算法快？
  - 空间开销小
  - 内存访问次数少
- 为什么空间小的算法更快？
  - 因为缓存命中率更高
- 为什么 Luleå 算法空间开销小？

数据缓存/算法需求	大小
一级缓存	32 KB
二级缓存	256 KB
三级缓存	6 MB
Trie	~ 50 MB
Luleå	< 40 MB

缓存大小在 i3-8100 CPU 上测得

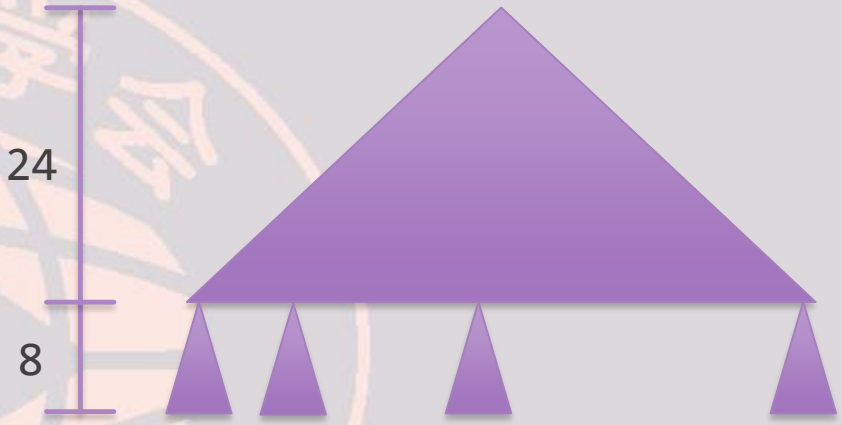
观察数据

- 前缀长度为24的表项：57.1%
- 前缀长度在16到24之间的表项：98.4%



24 + 8

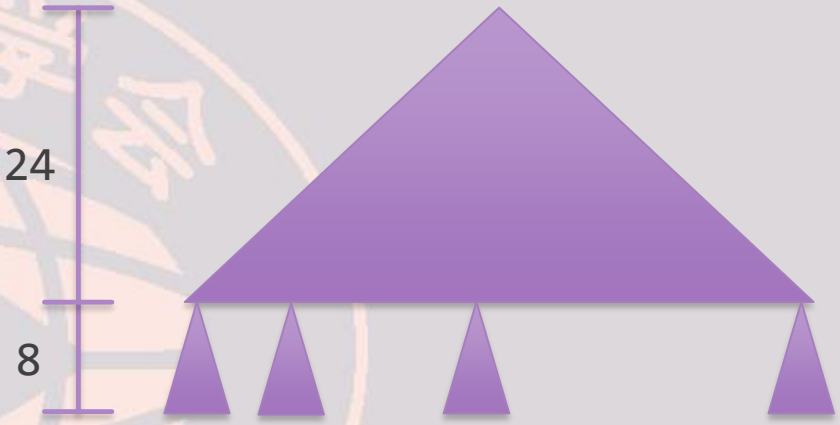
- 两层 Trie 树
  - 第一层：前 24 个 bit
  - 第二层：后 8 个 bit



- 预处理
  - 每一层保存“位图前缀和”结构
  - 第一层结构指向第二层，第二层结构保存下一跳
- 查询
  - 不超过 2 次“位图前缀和”查询，不超过 6 次内存访问

24 + 8

- 实际性能
  - 空间开销：78 MB
  - 预处理：20.52 ms
  - 每次查询：17.23 ns
  - 每秒查询次数：58.03 M
  - 对应网络速率：39.00 Gbps



- https://duck.ac/submission/13224

Poptrie 算法（6+6+6+6+6+2）

- Asai等人在SIGCOMM 2015会议上提出
- 每层高度为 6 的 Trie 树，使用位图前缀和结构
  - 2^6 = 64，能够在常见的 64 位计算机上快速运行
  - 该算法也可将前 16 或 18 位合并成一层

Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup

Hirochika Asai
The University of Tokyo
panda@hongo.wide.ad.jp

Yasuhiro Ohara
NTT Communications Corporation
yasuhiro.ohara@ntt.com

路由查找算法的总结和讨论

- 信息学竞赛中的算法：
  - Trie、Patricia Trie、Hash、二分查找、位图前缀和
  - 追求更好的时空复杂度
  - 追求更好的“常数”（可能是操作次数）
- 实际应用中的算法：
  - Luleå 算法、Poptrie 算法 等
  - 面向指令集和缓存进行优化
  - 从数据中挖掘特性来设计更优的结构（比如，16~24长度占了98%）
- 这些算法均基于 Trie 结构：
  - 1+1+…+1，32，16+8+8，6+6+6+6+6+2

下课休息时间☺

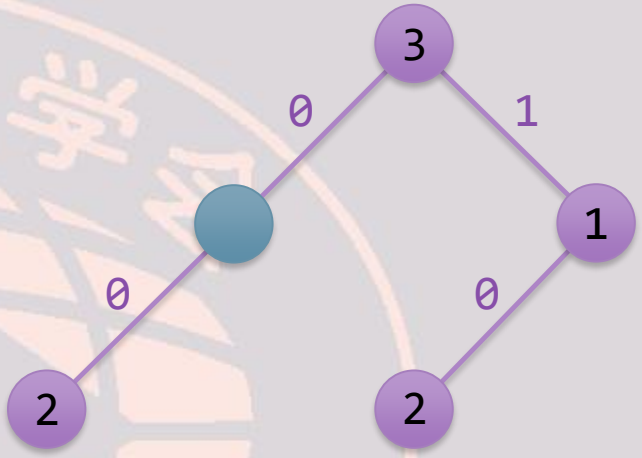
- 欢迎提问

迷思

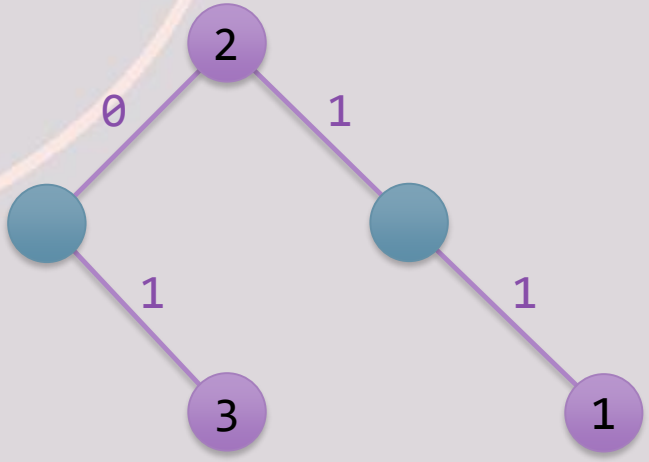
- 刚才讨论了软件路由查找算法，而实际上大多路由器基于硬件
  - 特殊硬件可以同时查询地址和所有表项
- 然而，硬件的造价也更昂贵，且空间限制也更苛刻
- 所以？
  - 使用软件路由查找算法（低成本，但不够快！）
  - 压缩路由表？（等价即可）

路由表压缩？

地址	前缀长度	下一跳
$(0000)_2$	0	3
$(0000)_2$	2	2
$(1000)_2$	1	1
$(1000)_2$	2	2



地址	前缀长度	下一跳
$(0000)_2$	0	2
$(0100)_2$	2	3
$(1100)_2$	2	1

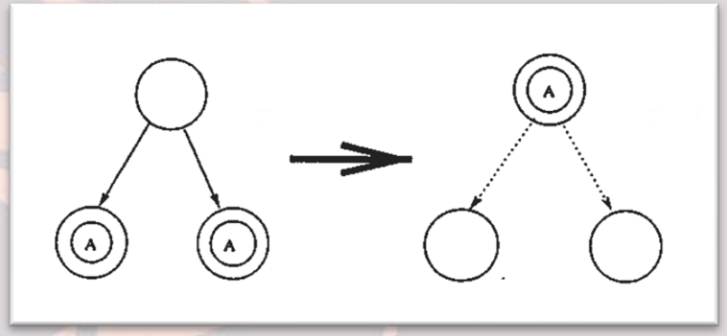


路由表压缩 —— 问题描述

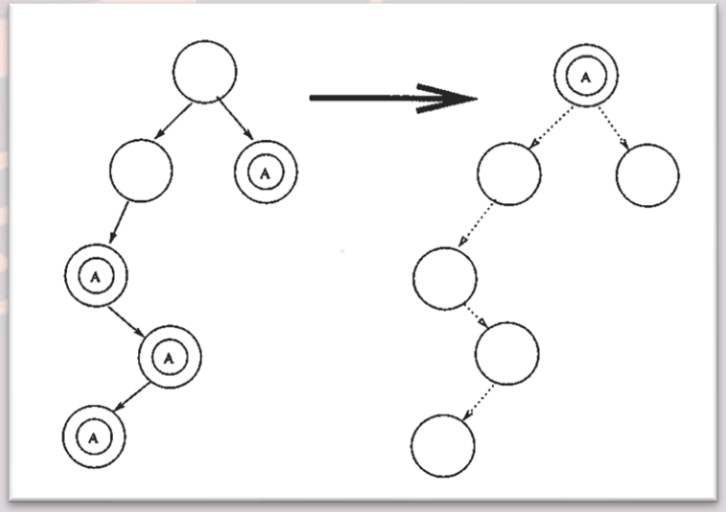
- 给定路由表（IPv4地址、前缀长度、下一跳）
- 输出一个与之等价且最优压缩的路由表
  - 等价：每个IPv4地址在两张表中对应的下一跳分别相同
  - 最优：不存在表项更少的等价压缩路由表
- 表项数 $N \approx 8 \times 10^5$
- 10s, 1024MB
- https://duck.ac/problem/routecomp

一个简单但并非最优的算法

- Binary Tree Collapse
- 由Turner等人于1998年提出

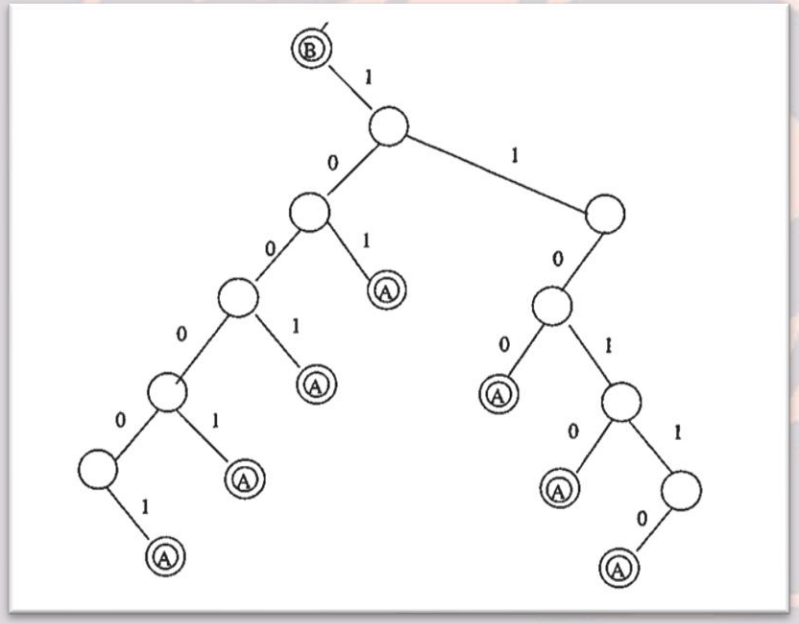


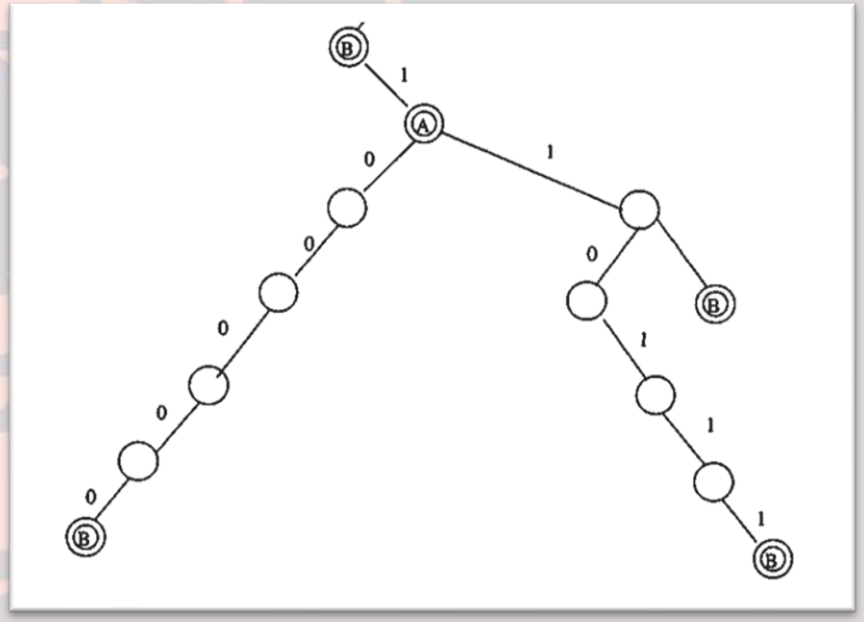
- 方法1：合并相同叶子结点
- 方法2：删除冗余子结点



一个简单但并非最优的算法

- 方法3：范围压缩





一个最优算法

- 首先假设所有表项均为叶子结点
  - 若存在内部结点，则可以执行“标记下传”
- 观察发现该问题有最优子结构
  - 一个子树的“最优解”只与它祖先给它下传的标记有关
  - 一个子树的表项数 = 它的左子树表项数 + 它的右子树表项数
- 树形动态规划（DP）
  - F[i][j]：子树 i 的祖先下传的标记为 j 时，该子树的最优表项个数

树形动态规划

- F[i][j]
  - 子树 i 的祖先下传的标记为 j 时，该子树的最优表项个数
  - = 0，如果结点 i 是叶子结点，且 j == nexthop[i]
  - = 1，如果结点 i 是叶子结点，且 j != nexthop[i]
  - = min{

```
F[l[i]][j] + F[r[i]][j],              // 该结点留空
min{1 + F[l[i]][k] + F[r[i]][k]}      // 该结点添加表项k
}
```

- 时间复杂度：O(N * M)，M 为不同下一跳的个数
- 在实际路由器中，M 通常不超过 64 ☺

学术界的算法：ORTC

- 全称 Optimal Routing Table Constructor
- 第一步：标记下传
- 第二步：对每个结点定义 $$\mathrm{set}(i)=\begin{cases}\{\mathrm{nexthop}_i\} & \text{叶子结点}\\ S\left(\mathrm{set}\left(l(i)\right),\mathrm{set}\left(r(i)\right)\right) & \text{非叶子结点}\end{cases}$$
$$S(A,B)=\begin{cases}A\cap B, & A\cap B\neq\emptyset\\ A\cup B, & A\cap B=\emptyset\end{cases}$$
- 第三步：从根结点往下，若该结点已被其祖先的下一跳满足，则跳过，否则从set(i)中任意选取一个下一跳

ORTC 为什么正确？

- 留作课后练习☺
- 提示：可从树形动态规划入手，尝试证明
  - 对每个子树，当下传的标记不同时，最优解表项数只有两种值，且差1
  - ORTC中的set(i)就是每个子树的最优决策集合
- ORTC论文：
  - https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-98-59.pdf

实际应用中的需求

- $N \approx 8 \times 10^5$ 的路由表，每秒最多有几万次更新操作
- 压缩后的路由表，如何快速更新？
  - 保持最优压缩率 — 更新可能需要 O(N)
  - 更新后不进行压缩/调整 — 压缩率不可控
  - 更新后进行局部次优压缩 — 如何控制时间、压缩率？
- 多次更新后，若路由表过大，则需要重压缩
  - 重压缩时间间隔与压缩率如何权衡？
  - 能否避免重压缩？

另一个问题

- 如何快速判定压缩算法正确？
- 如何快速找出两个路由表中不等价的区间？

- 给定路由表 A 和 B，判断它们是否等价
  - 每个IPv4地址在两张表中对应的下一跳分别相同

- 如果不等价，找出下一跳不相同的区间
- $N \approx 8 \times 10^5$

两遍扫描即可

- 第一遍：将路由表转换成“区间列表”
  - 每个区间对应一段连续相同的下一跳
  - 需先将路由表按照前缀地址和前缀长度排序
- 第二遍：检查两个区间列表是否相同
- 时间复杂度：O(N)
- 空间复杂度：O(N)

另一个问题 —— 增量更新

- 给定路由表 A 和 B，多次询问它们是否等价
  - 每个IPv4地址在两张表中对应的下一跳分别相同
- 如果不等价，找出下一跳不相同的区间
- 询问之间可以有更新操作
- $N \approx 8 \times 10^5$, $Q = 10^5$

使用数据结构维护

- 每个子树的路由信息是独立的
  - 除了受到祖先下传的标记影响
- 每个子树的信息由它的根结点、左右子树组成
  - 可以自下而上统计
- 在 Trie 树上增加额外的统计信息
  - 将标记下传影响到的，与不受标记下传影响的信息分开统计
  - 用多项式 Hash 来表示子树范围内的下一跳信息
- 每次更新时间复杂度：O(Trie 树深度)

路由压缩算法的总结和讨论

- 信息学竞赛中的问题
  - 静态路由压缩问题 — 存在最优算法（树形动态规划、ORTC）
  - 路由表等价性判断问题 — 存在简单的算法（两遍扫描、Hash）
  - 此类问题通常优化目标单一，容易彻底解决
- 实际应用中的问题
  - 动态路由压缩问题 — 不存在完美算法，存在多种次优算法
  - 此类问题需要权衡多种指标，不存在绝对的最优解

总结和讨论

- 在解决实际问题的过程中，锻炼自己的思维
- 多方向多角度地思考问题☺

信息学竞赛中的问题与算法
更关心“复杂度”
更关心“最优解”

实际应用中的问题与算法
面向具体软硬件和数据优化
权衡多种评价标准

谢谢

- 欢迎提问