倍增

关注Akie秋绘谢谢喵~

目录

- 倍增思想
- ST表
- 树上倍增

倍增思想

- 先来看一个小学奥数题：
- 你可以选择一些正整数，需要满足 0~100 中的任何整数都能被你选择的一些数的和表示。
- 最少要选多少个数？应该选择哪些数？

倍增思想

- 7个。
- 一种选法：1、2、4、8、16、32、64。
- 这种选法可以表示 0~127 中的所有数。
- 如果要表示 [0,n) 中的所有数呢？需要选择多少个数？
- $\lfloor \log_2 n \rfloor + 1$。
- 二进制拆位思想。之前学的什么东西也用到了？
- 如果我们把这种思想用到处理区间问题上会如何呢？

ST表

- 又称 Sparse Table，稀疏表。
- 先看一个经典问题：给定 n 个数，m 个询问，每次询问区间 [l,r] 的 max。
- https://www.luogu.com.cn/problem/P3865
- N<=1e6
- 暴力？TLE！
- 树状数组 or 线段树？Over killed! 而且太慢啦！
- 可以把倍增的思想搬到这道题上吗？

ST表

- 如果我处理出了所有长度为 1, 2, 4, 8, 16, 32, … 的区间，我是不是可以用它们表示所有区间?
- 如何预处理呢?
- 长度为 2 的区间可以由两个长度为 1 的区间拼接而成。同理，长度为 2^(n+1) 的区间可以由长度为 2^n 的区间拼成。
- 所以预处理长这样:

```
for(int i=1; i<=n; i++)
{
    mx[i][0]=a[i];
}
for(int j=1; j<S; j++)
    for(int i=1; i<=n; i++)
    {
        mx[i][j]=max(mx[i][j-1],mx[min(i+(1<<j-1),n)][j-1]);
    }
```

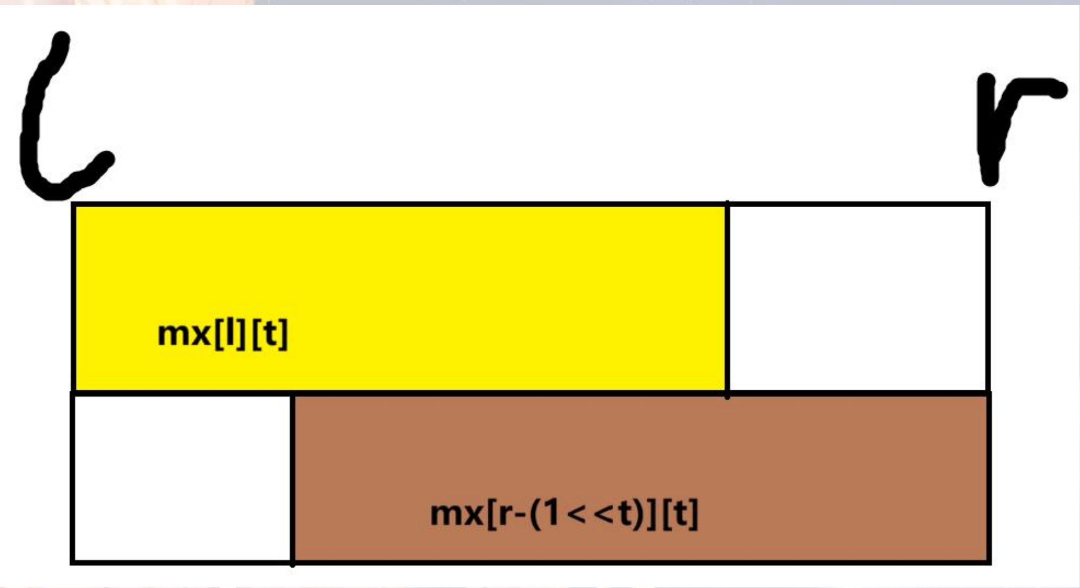
ST表

- 怎么查询呢?
- 同样是二进制拆位思想。和快速幂相反，ST表是从高位到低位枚举。
- 时间复杂度 O(logn)

```
int query(int l,int r)
{
    int now=l,res=0;
    for(int i=S-1; i>=0; i--)
    {
        if(now+(1<<i)<=r)
        {
            res=max(res,mx[now][i]);
            now+=(1<<i);
        }
    }
    res=max(res,mx[now][0]);
    return res;
}
```

ST表

- 可以 O(1) 实现查询吗?
- 其实我只需要两段就可以覆盖整个区间。
- 所以查询的可以变成:




```
int query(int l,int r)
{
    if(l==r)
        return mx[l][0];
    int len=r-l,t=lg[len];
    return max(mx[l][t],mx[r-(1<<t)+1][t]);
}
```

ST表

• 最终代码:

```cpp
vector<int> lg;
int query(int l,int r)
{
    if(l==r)
        return mx[l][0];
    int len=r-l,t=lg[len];
    return max(mx[l][t],mx[r-(1<<t)+1][t]);
}
void O_o()
{
    int n=read(),Q=read();
    vector<int> a(n+1);
    for(int i=1; i<=n; i++)
        a[i]=read();
    mx.assign(n+1,vector<int>(S,inf));
    lg.assign(n+1,0);
    lg[0]=-1;
    for(int i=1; i<=n; i++)
    {
        mx[i][0]=a[i];
        lg[i]=lg[i>>1]+1;
    }
    for(int j=1; j<S; j++)
        for(int i=1; i<=n; i++)
        {
            mx[i][j]=max(mx[i][j-1],mx[min(i+(1<<j-1),n)][j-1]);
        }
    while(Q--)
    {
        int l=read(),r=read();
        cout<<query(l,r)<<"\n";
    }
}
```

ST表

- 尝试用 st 表做一下区间加，区间 gcd，区间按位与、或。

附录：ST 表求区间 GCD 的时间复杂度分析

在算法运行的时候，可能要经过 $\Theta(\log n)$ 次迭代。每一次迭代都可能会使用 GCD 函数进行递归，令值域为 w，GCD 函数的时间复杂度最高是 $\Omega(\log w)$ 的，所以总时间复杂度看似有 $O(n \log n \log w)$。

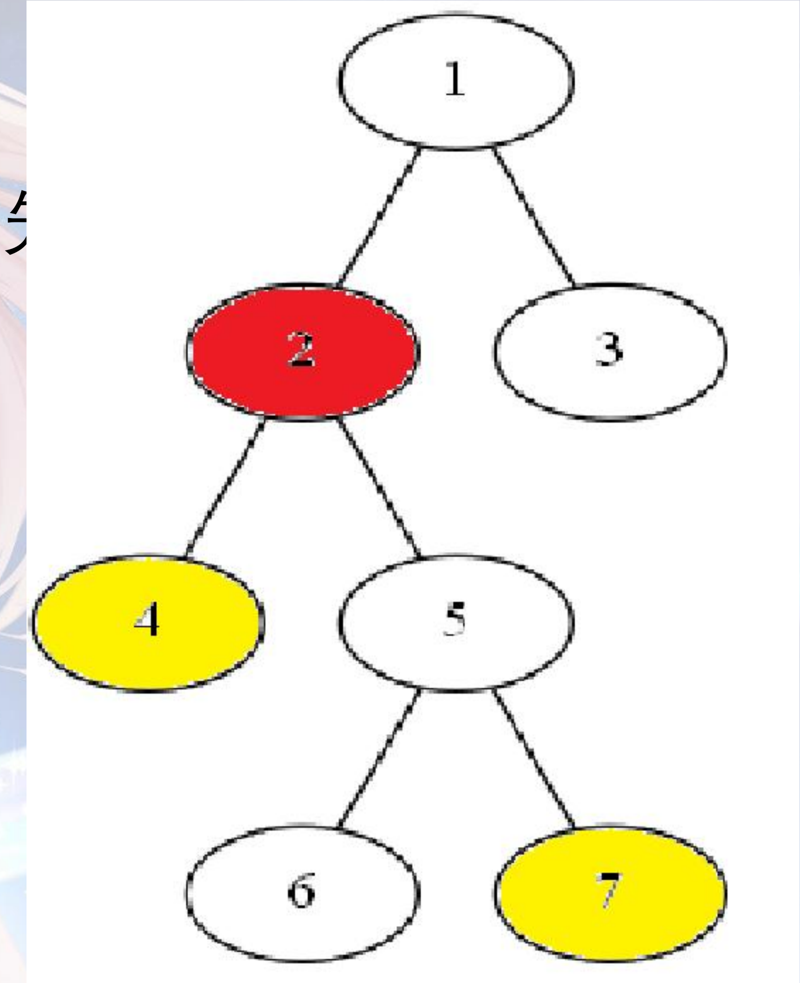
但是，在 GCD 的过程中，每一次递归（除最后一次递归之外）都会使数列中的某个数至少减半，而数列中的数最多减半的次数为 $\log_2(w^n) = \Theta(n \log w)$，所以，GCD 的递归部分最多只会运行 $O(n \log w)$ 次。再加上循环部分（以及最后一层递归）的 $\Theta(n \log n)$，最终时间复杂度则是 $O(n(\log w + \log x))$，由于可以构造数据使得时间复杂度为 $\Omega(n(\log w + \log x))$，所以最终的时间复杂度即为 $\Theta(n(\log w + \log x))$。

而查询部分的时间复杂度很好分析，考虑最劣情况，即每次询问都询问最劣的一对数，时间复杂度为 $\Theta(\log w)$。因此，ST 表维护「区间 GCD」的时间复杂度为预处理 $\Theta(n(\log n + \log w))$，单次查询 $\Theta(\log w)$。

线段树的相应操作是预处理 $\Theta(n \log x)$，查询 $\Theta(n(\log n + \log x))$。

树上倍增（求LCA）

- 啥是LCA?
- Lowest Common Ancestor，最近公共祖
- 大家可以想想如何暴力求LCA?



树上倍增（求LCA）

- 不妨假设dep[u]>=dep[v]
- 先把 u 跳到和 v 同一深度的位置。
- 如果 u==v，那么LCA就是 u。
- 否则，两个点一起往上跳，直到 fa[u]==fa[v]。
- fa[u] 就是LCA。
- 单次时间复杂度O(n)。
- 我们是否可以通过倍增加速这个过程?

树上倍增（求LCA）

- 我们可以参考 st 表的思路，预处理一个 fa[u][i] 数组，代表 u 的 2^i 级祖先。
- 显然 fa[u][0] 就是 u 的父亲。
- fa[u][i]=fa[fa[u][i-1]][i-1]

```
void dfs(int u)
{
    dep[u]=dep[fa[u][0]]+1;
    for(auto v:e[u])
    {
        if(v==fa[u][0])
            continue;
        fa[v][0]=u;
        dfs(v);
    }
}
```

```
dfs(rt);
for(int j=1; j<S; j++)
{
    for(int i=1; i<=n; i++)
    {
        fa[i][j]=fa[fa[i][j-1]][j-1];
    }
}
```

树上倍增（求LCA）

- 显然，把 u 跳到和 v 同一个深度的过程可以用刚刚得到的数组优化：

```
if(dep[u]<dep[v])
    swap(u,v);
for(int i=S-1; i>=0; i--)
{
    if(dep[fa[u][i]]>=dep[v])
        u=fa[u][i];
}
if(u==v) return u;
```

树上倍增（求LCA）

- 同样的，u,v 一起往上跳也可以用倍增优化：

```
for(int i=S-1; i>=0; i--)
{
    if(fa[u][i]!=fa[v][i])
    {
        u=fa[u][i];
        v=fa[v][i];
    }
}
return fa[u][0];
```

树上倍增

- [洛谷P3865]

```cpp
void dfs(int u)
{
    dep[u]=dep[fa[u][0]]+1;
    for(auto v:e[u])
    {
        if(v==fa[u][0])
            continue;
        fa[v][0]=u;
        dfs(v);
    }
}
int lca(int u,int v)
{
    if(dep[u]<dep[v])
        swap(u,v);
    for(int i=S-1; i>=0; i--)
    {
        if(dep[fa[u][i]]>=dep[v])
            u=fa[u][i];
    }
    if(u==v) return u;
    for(int i=S-1; i>=0; i--)
    {
        if(fa[u][i]!=fa[v][i])
        {
            u=fa[u][i];
            v=fa[v][i];
        }
    }
    return fa[u][0];
}
```

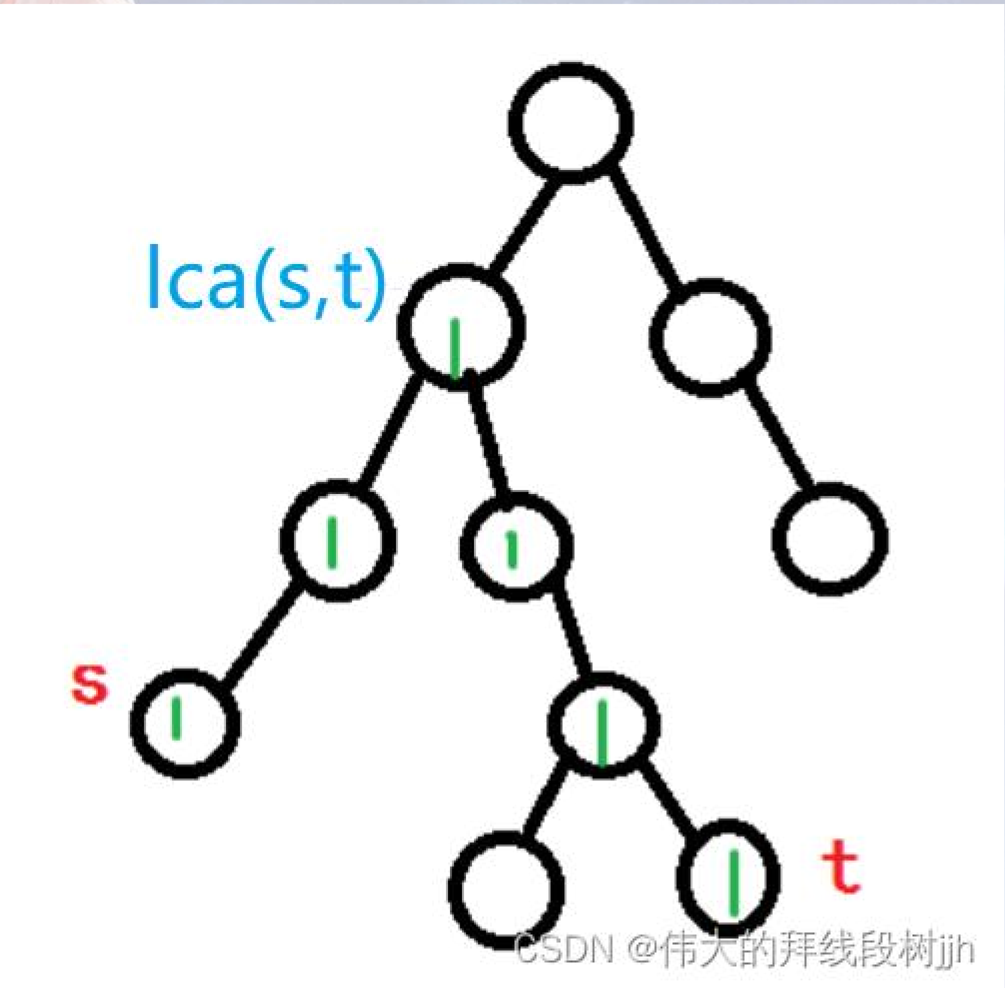
```cpp
void O_o()
{
    int n,Q,rt;
    cin>>n>>Q>>rt;
    e.assign(n+1,{});
    for(int i=1; i<n; i++)
    {
        int x,y;
        cin>>x>>y;
        e[x].push_back(y);
        e[y].push_back(x);
    }
    fa.assign(n+1,vector<int>(S,0));
    dep.assign(n+1,0);
    dfs(rt);
    for(int j=1; j<S; j++)
    {
        for(int i=1; i<=n; i++)
        {
            fa[i][j]=fa[fa[i][j-1]][j-1];
        }
    }
    while(Q--)
    {
        int x,y;
        cin>>x>>y;
        cout<<lca(x,y)<<"\n";
    }
}
```

树上倍增（树上差分）

- 给你一棵 n 个点的树，点有点权，初始全部为0。
- 有 k 次操作，每次操作给出一个二元组 (u,v)，表示将 u 到 v 的简单路径上的点权全部+1。
- 所有操作结束后，你需要输出树上最大点权。
- （洛谷翻译翻得依托，如果要看原题建议直接读英文）
- [洛谷P3128]()

树上倍增（树上差分）

- 暴力怎么写?
- 使用暴力跳 lca 的方法，把途径的点都 +1。
- 单次操作时间复杂度 O(n)
- 有办法优化吗?

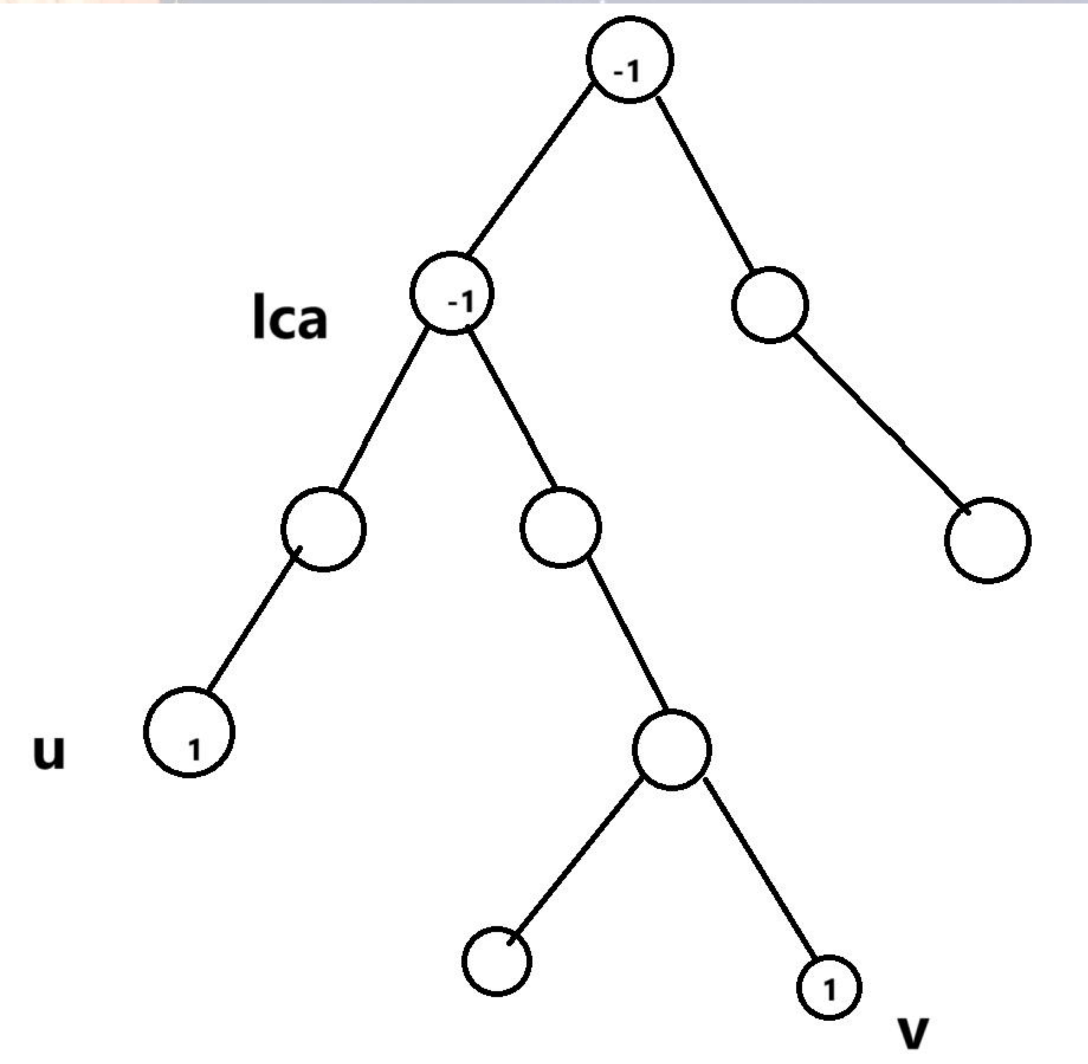


树上倍增（树上差分）

- 优化1：树链剖分+前缀和。可惜你们没学树链剖分。
- 优化2：树上差分。
- 大家先回忆一下普通的差分是怎么做的？
- 有奖问答：如果树的形态保证 i+1 的父亲是 i 怎么做？

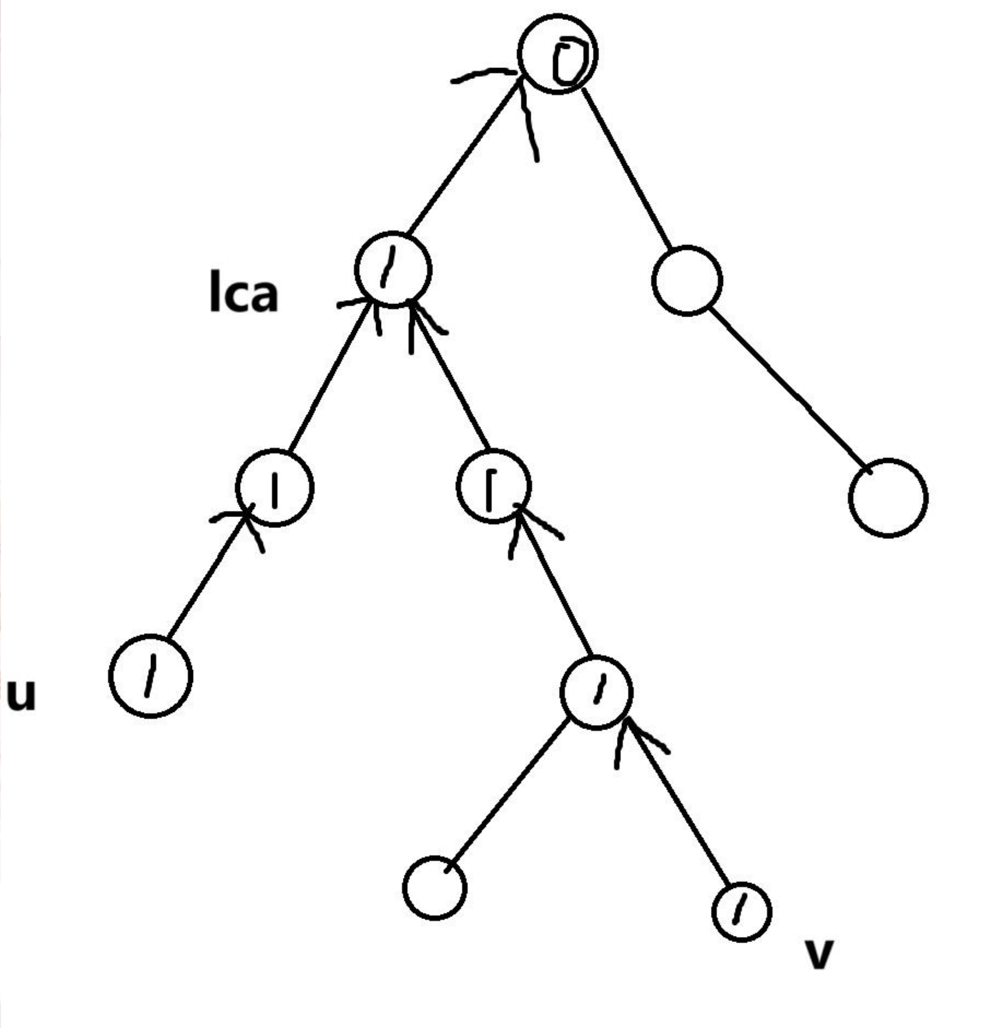
树上倍增（树上差分）

- 怎么样把差分搬到树上呢?
- 如果在链上，我只需要在头部+1，尾部-1，然后前缀和就行了。但在树上，如果只在路径的两端做文章，我们无法统一所有操作的传递方向。
- 我们不妨把 lca(u,v) 纳入考虑
- a[u]+=1, a[v]+=1
- a[lca(u,v)]-=1, a[fa[lca(u,v)]]-=1



树上倍增（树上差分）

- 所有操作结束之后，所有点的权值都向上传给它的父亲（相当于前缀和）
- 这就是树上差分的**点差分**
- 课后思考：边差分怎么做？



树上倍增（树上差分）

- 核心代码：

```
while(k--)
{
    int x,y;
    cin>>x>>y;
    sum[x]++; sum[y]++;
    int l=lca(x,y);
    sum[l]--; sum[fa[l][0]]--;
}
dfs2(1);
```

```
void dfs2(int u)
{
    for(auto v:e[u])
    {
        if(v==fa[u][0]) continue;
        dfs2(v);
        sum[u]+=sum[v];
    }
}
```