# 搜索和倍增

许哲诚

天津大学

2023 年 6 月 29 日

# 目录

# 目录

# 搜索

## 简介

搜索，也就是对状态空间进行枚举，通过穷尽所有的可能来找到最优解，或者统计合法解的个数。有很多优化方式，如减小状态空间，更改搜索顺序，剪枝等。是一些高级算法的基础。在比赛中，纯粹的搜索往往也是得到部分分，打表的手段，但可以通过纯粹的搜索拿到满分的题目非常少。

# 搜索

## 分类

- DFS：深度优先搜索。
- BFS：广度优先搜索。
- 启发式/剪枝/双向搜索：各种优化方法。
- 记忆化：算是和动态规划的一种结合。
- 迭代加深搜索：DFS 的一种升级。
- 精确覆盖问题、重复覆盖问题的解决方法，不好写，建议弄模板。(不讲)
- Alpha-Beta 剪枝：局势的一种剪枝，算法题不常见。(不讲)

# DFS

## DFS 简介

在搜索算法中，该词常常指利用递归函数方便地实现暴力枚举的算法，与图论中的 DFS 算法有一定相似之处，但并不完全相同。
DFS 全称是 Depth First Search，中文名是深度优先搜索，是一种用于遍历或搜索树或图的算法。所谓深度优先，就是说每次都尝试向更深的节点走。
该算法讲解时常常与 BFS 并列，但两者除了都能遍历图的连通块以外，用途完全不同，很少有能混用两种算法的情况。

## DFS 常见运用

全排列，搜索，图上运用。

# DFS

## DFS

DFS 最显著的特征在于其递归调用自身。DFS 会对其访问过的点打上访问标记，在遍历图时跳过已打过标记的点，以确保每个点仅访问一次。符合以上两条规则的函数，便是广义上的 DFS。

```
DFS(v) // v 可以是图中的一个顶点，也可以是抽象的概念，如 dp 状态等。
  在 v 上打访问标记
  for u in v 的相邻节点
    if u 没有打过访问标记 then
      DFS(u)
    end
  end
end
```

# DFS

```
void dfs(int x)
{
    if(x==t){
        for(int i=1;i<tot;i++)
            printf("%d->",step[i]);
        printf("%d\n",t);
        return;
    }
    for(int i=1;i<=n;i++)
        if(a[x][i]&&!f[i]){
            f[i]=1;
            step[++tot]=i;
            dfs(i);
            f[i]=0;
            tot--;
        }
}
```

# BFS

## BFS

BFS 全称是 Breadth First Search，中文名是宽度优先搜索，也叫广度优先搜索。是图上最基础、最重要的搜索算法之一。
所谓宽度优先。就是每次都尝试访问同一层的节点。如果同一层都访问完了，再访问下一层。
这样做的结果是，BFS 算法找到的路径是从起点开始的最短合法路径。换言之，这条路径所包含的边数最小。
在 BFS 结束时，每个节点都是通过从起点到该点的最短路径访问的。
算法过程可以看做是图上火苗传播的过程：最开始只有起点着火了，在每一时刻，有火的节点都向它相邻的所有节点传播火苗。

## BFS 常见运用

最短路，搜索，图上运用。

# BFS

## BFS

BFS 最显著的特征在于最短路径。BFS 会对其访问的图按深度层序遍历，以确保每个点都是最短的路径访问的。

```
bfs(s) {
  q = new queue()
  q.push(s), visited[s] = true
  while (!q.empty()) {
    u = q.pop()
    for each edge(u, v) {
      if (!visited[v]) {
        q.push(v)
        visited[v] = true
      }
    }
  }
}
```

# 各种优化

## 优化简介

启发式搜索（英文：heuristic search）是一种在普通搜索算法的基础上引入了启发式函数的搜索算法。
启发式函数的作用是基于已有的信息对搜索的每一个分支选择都做估价，进而选择分支。简单来说，启发式搜索就是对取和不取都做分析，从中选取更优解或删去无效解。
双向同时搜索的基本思路是从状态图上的起点和终点同时开始进行广搜或深搜。
如果发现搜索的两端相遇了，那么可以认为是获得了可行解。
它适用于输入数据较小，但还没小到能直接使用暴力搜索的情况。
暴力搜索的复杂度往往是指数级的，而改用 meet in the middle 算法后复杂度的指数可以减半，即让复杂度从 $O(a^b)$ 降到 $O(a^{b/2})$ 。

# 各种优化

## 优化运用

排除等效冗余：在搜索过程中，若能判断从搜索树当前节点上沿某几条不同分支到达的子树是相同的，那么只需对其中一条分支执行搜索。

可行性剪枝：可行性剪枝也叫上下界剪枝，其是指在搜索过程中，及时对当前状态进行检查，若发现分支已无法到达递归边界，就执行回溯。

最优性剪枝：在最优化问题的搜索过程中，若当前花费的代价已超过当前搜索到的最优解，那么无论采取多么优秀的策略到达递归边界，都不可能更新答案，此时可以停止对当前分支的搜索进行回溯。

# 各种优化

## 来个简单例子

### 题目描述

将整数 $n$ 分成 $k$ 份，且每份不能为空，任意两个方案不相同（不考虑顺序）。

例如：$n = 7$，$k = 3$，下面三种分法被认为是相同的。

$1, 1, 5$;
$1, 5, 1$;
$5, 1, 1$.

问有多少种不同的分法。

### 输入格式

$n, k$ $(6 < n \leq 200,\ 2 \leq k \leq 6)$

### 输出格式

1 个整数，即不同的分法。

# 各种优化

```
void dfs(int last,int sum,int cur)
{
    if(cur==k)
    {
        if(sum==n) cnt++;
        return;
    }
    for(int i=last;sum+i*(k-cur)<=n;i++)
        dfs(i,sum+i,cur+1);
    //剪枝，只用枚举到sum+i*(k-cur)<=n为止
}
```

# 各种优化

**优化运用**

太常见了，暴力的时候非常需要，但这些大多数并不能优化复杂度。

# 记忆化

## 记忆化简介

用空间换时间的一种方式，但其实算是动态规划的一种。
相当常见，如果动态规划不好写循环的转移，那么就可以考虑使用这种记忆化的操作。但其实这还能算搜索吗？(我也不好定义)

# 迭代加深搜索

## 迭代加深搜索简介

迭代加深是一种每次限制搜索深度的深度优先搜索。
迭代加深搜索的本质还是深度优先搜索，只不过在搜索的同时带上了一个深度，当达到设定的深度时就返回，一般用于找最优解。如果一次搜索没有找到合法的解，就让设定的深度加一，重新从根开始。
既然是为了找最优解，为什么不用 BFS 呢？我们知道 BFS 的基础是一个队列，队列的空间复杂度很大，当状态比较多或者单个状态比较大时，使用队列的 BFS 就显出了劣势。事实上，迭代加深就类似于用 DFS 方式实现的 BFS，它的空间复杂度相对较小。
当搜索树的分支比较多时，每增加一层的搜索复杂度会出现指数级爆炸式增长，这时前面重复进行的部分所带来的复杂度几乎可以忽略，这也就是为什么迭代加深是可以近似看成 BFS 的。

# 目录

# 八皇后 Checker Challenge

## 问题简述

在一个 $n \times n$ 的棋盘上放置皇后，求不冲突的方案数。

# 八皇后 Checker Challenge

**分析**

就是把所有可能的皇后的摆放枚举，然后判断是否可行。

# 八皇后

```
1  void queen(int i){
2      if(i>n){print();return;}
3      for(int j=1;j<=n;j++) {
4          if((!b[j])&&(!c[i+j])&&(!d[i-j+n])){//如果没有皇
               后占领，执行以下程序
5              a[i]=j;b[j]=1;//宣布占领纵列，标记i排是第j个
6              c[i+j]=1;d[i-j+n]=1;//宣布占领两条对角线
7              queen(i+1);//进一步搜索，下一个皇后
8              b[j]=0;c[i+j]=0;d[i-j+n]=0;//（回到上一步）清
                   除标记
9          }
10     }
11 }
```

# 全排列

## 问题简述

按照字典序输出自然数 1 到 $n$ 所有不重复的排列，即 n 的全排列，要求所产生的任一数字序列中不允许出现重复的数字。

# 全排列

**分析**

就是最常见的搜索实现。

# 全排列

```
12 void dfs(int x){//X表示当前搜索到那个数
13     if(x>n){//如果N位都搜索完了，就输出方案并返回
14         for(int i=1;i<=n;i++) printf("% 5d",ans[i]);//输出
               方案
15         return;
16     }
17     for(int i=1;i<=n;i++)//从小到大枚举
18         if(!use[i]){//判断这个数是否用过
19             ans[x]=i;//保存到方案中
20             use[i]=1;//标记这个数被使用了
21             dfs(x+1);//进行下一步搜索
22             use[i]=0;//撤销标记
23         }
24 }
```

# 回家

**问题简述**

给你一张图，求两个点之间的的最短时间。

# 回家

**分析**

考虑到状态太多，我们就使用 Bfs 更好。

# 世界冰球锦标赛

## 问题简述

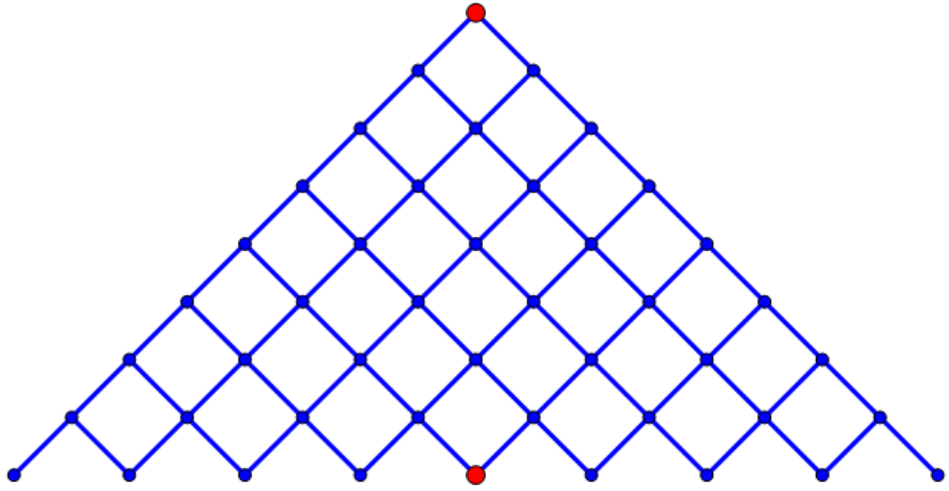
给出 Bobek 的预算和每场比赛的票价，试求：如果总票价不超过预算，他有多少种观赛方案。如果存在以其中一种方案观看某场比赛而另一种方案不观看，则认为这两种方案不同。

# 世界冰球锦标赛

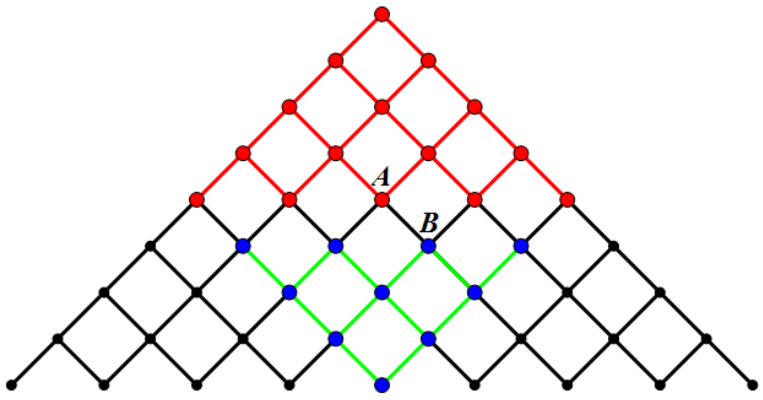
**分析**

$2^{40}$ 还是太大了，纯纯的暴力过不了。我们可以考虑折半搜索。这样就把复杂度控制在我们可以接受的一个范围了。

# 世界冰球锦标赛







# 砝码称重

## 问题简述

现有 $n$ 个砝码，重量分别为 $a_i$，在去掉 $m$ 个砝码后，问最多能称量出多少不同的重量（不包括 0）。
请注意，砝码只能放在其中一边。

# 砝码称重

**分析**

是一个先搜索，在动态规划的例题，本身难度不大。

# 骑士精神

在一个 5×5 的棋盘上有 12 个白色的骑士和 12 个黑色的骑士，且有一个空位。在任何时候一个骑士都能按照骑士的走法（它可以走到和它横坐标相差为 1，纵坐标相差为 2 或者横坐标相差为 2，纵坐标相差为 1 的格子）移动到空位上。

给定一个初始的棋盘，怎样才能经过移动变成如下目标棋盘：

为了体现出骑士精神，他们必须以最少的步数完成任务。

# 骑士精神

**分析**

如果单纯使用 DFS 和 BFS 对于这道题，可能空间和时间上都是不允许的。我们考虑使用迭代加深算法。设置最大深度来操作。

# 目录

# 倍增简介

## 倍增

倍增法（英语：binary lifting），顾名思义就是翻倍。它能够使线性的处理转化为对数级的处理，大大地优化时间复杂度。

# 快速幂

## 简述

求 $a^x$ 的值。

## 分析

自己思考一下。

# 快速幂

**分析**

快速幂啊，想想怎么做。观点决定成败。

$a^b \times a^c = a^{b+c}$

有了这个我们再来考虑一下优化。

# 快速幂

```
25 int ksm(int a,int b,int p) {
26     int x = 1;
27     a %= p;
28     for(;b;b >>= 1,a = a * 1ll * a % p) {
29         if(b & 1) x = x * 1ll * a % p;
30     }return x;
31 }
32 int Ksm(int a,int b,int p) {
33     int B = (int)sqrt(b);
34     a %= p;
35     int c1 = b / B,c2 = b % B,x = 1,y = 1;
36     for(int i = 1;i <= B;i++) x = x * 1ll * a % p;
37     for(int i = 1;i <= c1;i++) y = y * 1ll * x % p;
38     for(int i = 1;i <= c2;i++) y = y * 1ll * a % p;
39     return y;
40 }
```

# RMQ 问题

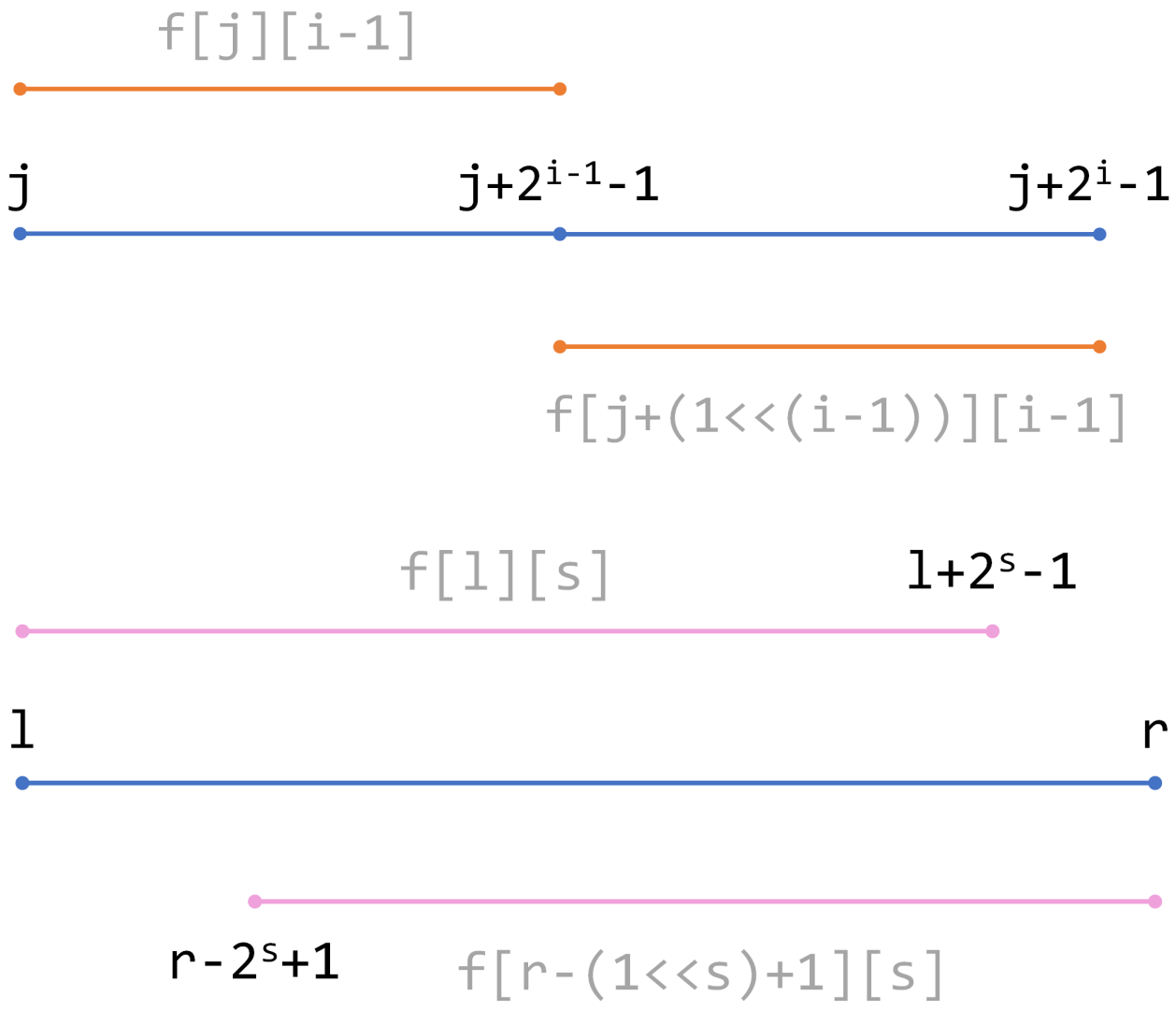
## 问题简述

给你一个长度为 $n$ 的序列，$Q$ 次询问，查找 $[l, r]$ 的最大值。

## 分析

相信很多同学知道线段树之类的 $O(n \log n) - O(\log n)$ 前者为初始化复杂度，后者为单次查询复杂度的做法。

# RMQ 问题



# RMQ 问题

## 分析

这里我们可以通过该题无修改的优秀性质，考虑一个 $O(n\log n)-O(1)$ 的做法。我们可以分析 max 函数其实是有一个优美的性质的可重复贡献，对于一个询问 $Q(l,r)$ ，我们可以形式化的写成 $\max Q(l,r)$ 那么等同于 $\max Q(l,l+2^p-1),Q(r-2^p+1,r)$ ，其中 $p=\log(r-l+1)$ 的，这样我们就做到了对于一个区间拆分成两个区间，而且两个区间的并集是等于原区间的。而 $\max Q(l,l+2^p-1)$ 就比较简单的。我们定义 $f(i,p)=\max Q(i,i+2^p-1)$ 是等同于 $\max f(i,p-1),f(i+2^{p-1},p-1)$ 那么 $f(i,p)$ 一共有 $n\log n$ 的状态，初始化的时间复杂度为 $O(n\log n)$ ，而每次查询为 $O(1)$ ，空间复杂度为 $O(n\log n)$ 。

# RMQ 问题

**分析**

这种做法可以拓展到，只要一个函数满足可重复贡献的性质，那么都可以使用这种方法的。例如区间 gcd 。而且区间 gcd 用 st 表维护，时间复杂度为 $O(n(\log n + \log w)) - O(\log w)$ 的。

# RMQ 问题

```
41 int query(int l,int r) { // 询问
42     int k = Log[r - l + 1];
43     return max(st[l][k],st[r-(1<<k)+1][k]);
44 }
45 // 初始化
46 for(int i = 2;i <= n;i++) Log[i] = Log[i >> 1] + 1;
47 for(int j = 1;j <= Log[n] + 1;j++) {
48     for(int i = 1;i + (1 << j) - 1 <= n;i++) {
49         st[i][j] = max(st[i][j-1],st[i+(1<<j-1)][j-1]);
50     }
51 }
```

# 树上倍增求 LCA

**问题简述**

求有根树上两个节点 $u, v$ 的最近公共祖先。

# LCA

**分析**

我们可以分析到，如果 $dep_u, dep_v$ 不相同，那么 $lca(u, v)$ 的深度一定小于 $\min\{dep_u, dep_v\}$ 。那么这里定义 $dep_u \geq dep_v$ ，那么我们可以先把 $u$ 提高为它的 $dep_u - dep_v$ 级祖先，使这两个节点深度相同，便于之后的操作。

# LCA

**分析**

如果 $u = v$，那么证明原本的 $u$ 是 $v$ 的子树中的某一个节点，他们的 $lca$ 为 $v$。
如果 $u \neq v$，那么 $u, v$ 属于 $lca(u, v)$ 的不同子树。那么我们可以考虑二分 $dis(lca(u, v), u)$ 的距离，这样我们就得到 $\mathrm{I}$ $O(\log^2 n)$ 的做法。

# LCA

## 分析

但其实我们可以再考虑一下倍增的性质，由于 2 进制表示是可以贪心的，那么如果两个节点 $2^p$ 级祖先相同，那么 $dis(lca(u,v),u) \leq 2^p$ ，反之 $dis(lca(u,v),u) > 2^p$ 。这样我们每次把问题范围减少为 $\frac{1}{2}$ ，那么由高位到低位贪心，就可以在 $O(\log n)$ 的时间范围内求出两个节点 $u, v$ 的 $lca(u,v)$

# 树上倍增求 LCA

```
52 int LCA(int x,int y) {
53     if(dep[x] > dep[y]) swap(x,y);
54     int k = dep[y] - dep[x];
55     for(int i = Log2[k];i >= 0;i--) {
56         if((k >> i) & 1) y = f[i][y];
57     }
58     if(x == y) return x;
59     for(int i = Log2[n];i >= 0;i--) {
60         if(f[i][x] != f[i][y]) x = f[i][x],y = f[i][y];
61     }
62     return f[0][x];
63 }
```

# 小总结

## 小总结

倍增一般优化的是静态的数据，而且 $g(2^k)$ 唯一对应。这样可以将 $g(x)$ 变为多个函数复合的形式，从而优化复杂度。

## 小总结

在树上倍增，序列上倍增都是非常常见的。而且二分查找也可以用倍增替换，常数更小。

# 目录

# MEET 紧急集合

## 问题简述

给出三个点 $x, y, z$ ，找到一个节点 $p$ 使 $dis(p, x) + dis(p, y) + dis(p, z)$ 最小。

# MEET 紧急集合

## 分析

我们考虑两个节点，那么比较简单，只要不走回头路就是最短的，换而言之就是 $path(x, y)$ 上的任意一个节点就可以。那么两个节点可以这样做，三个节点呢？我们可以类比，只要 $path(p, x), path(p, y), path(p, z)$ 这三条路径没有重叠，那么这样的 $p$ 点就是合法的。那么现在如何找这个 $p$ 点，对于 $x, y, z$ 中任意两个点来说，只要 $p$ 在简单路径上就可以了。那么我们可以分类讨论。

# MEET 紧急集合

## 分析

令 $A = lca(x, y)$ 如果 $lca(x, z) \neq A$ ，那么由于 $A$ 在 $path(x, y)$ 上，则 $lca(x, z)$ 必然是 $A$ 的祖先，或者子树。那么等同于 $lca(z, y) = lca(x, z)$ ，或者 $lca(z, y) = lca(y, x)$ 。那么容易得到 $x, y, z$ 两两求出 $lca$ 那么必然存在两个 $lca$ 是相同的，那么另一个 $lca$ 就是最优答案，那么剩下的就是树上距离公式了
$dis(x, y) = dis(root, x) + dis(root, y) - 2 \times dis(root, lca(x, y))$ 。考虑倍增求 $lca$ ，那么总的复杂度为 $O(n \log n + Q \log n)$ 。

# A and B and Lecture Rooms

## 问题简述

给你俩个节点 $A, B$ 和一棵无根树，找出树上满足 $dis(x, A) = dis(x, B)$ 的节点个数。

# A and B and Lecture Rooms

## 分析

我们比较好思考的就是 $path(A, B)$ 至多只有一个合法点，而且这个合法点一定是满足 $dis(x, A) = \frac{1}{2}dis(A, B)$ 的，所以当 $dis(A, B) \bmod 2 = 1$ 时是一定无解的。那么我们现在考虑 $x \in path(A, B)$ 的节点位置，比较好分析的是 $path(A, B)$ 只会在 $lca(A, B)$ 处发生一次转折，所以 $x$ 是 $A, B$ 中深度较深的祖先，具体来讲是它的 $\frac{dis(A,B)}{2}$ 级祖先 $P$ 。现在只需要讨论 $lca$ 和 $P$ 的关系就好了。

# A and B and Lecture Rooms

**分析**

$P = lca$ 那么要使 $dis(x, A) = dis(x, B)$ ，那么可以选择的节点就是，所有的节点除去以 $lca$ 为根的，$A, B$ 所在的两个子树的所有节点。
$P \neq lca$ 那么要使 $dis(x, A) = dis(x, B)$ ，那么可以选择的节点就是，以祖先 $P$ 为根的子树中，除去深度较深的 $A, B$ 所处的子树节点。那么总的复杂度为 $O(n \log n)$ ，因为我们需要查找 $K$ 级祖先和查找 $lca$ 。

# [SCOI2015] 国旗计划

## 问题简述

给出 $n$ 条线段，要求覆盖完整个环。求出某一条必选之后，至少需要的线段数量，线段无包含关系。

# [SCOI2015] 国旗计划

## 分析

常见的破环成链，那么我们把链倍长一次，现在就是要覆盖长度为 $n$ 的序列就可以了。那么由于线段没有包含关系，那么线段相离或者有交集。那么我们按左端点排序，得到的区间右端点应该是递增的。这样我们就可以贪心，显然，如果我们要从一个区间转移到另一个区间，那么另一个区间的右端点一定是，所有左端点小于当前右端点的所有区间中的最大值。那么我们的每一次跳跃是固定的，这里我们就可以考虑用倍增来维护，这样单次询问的时间复杂度为 $O(\log n)$ 加上离散化的复杂度，总的复杂度为 $O(n \log n)$ 。

# [SCOI2016] 萌萌哒

## 问题简述

在某些区间必须一样的情况下，可以组成多少个大数。

# [SCOI2016] 萌萌哒

**分析**

这个区间必须一样我们可以考虑并查集来维护，但是我们合并并查集时就需要 $O(n)$ 的时间，这个必须优化，还是像 $St$ 表，我们只需要维护 $2^p$ 长度的所有区间就可以了，我们可以考虑一个区间拆除 2 个，然后不断递归下去，这样总复杂度看似为 $O(n^2)$ 的，但是考虑到只有 $O(n\log n)$ 状态，而每一次递归势能都会减少，那么最后的复杂度为 $O(n\log n)$ 了，那么最后就只需要考虑集合个数了。

# 疫情控制

## 问题简述

给你一个有根树，根节点为 1 ，要求同时调动多支军队，使军队出现任意叶子节点到 1 的路径上（1 号节点不能出现军队）。求时间的最小值。

# 疫情控制

## 分析

由于你可以同时调动，那么其实就使求最大调度时间最小，那么这个可以考虑二分答案，将问题转化为判断问题。那么现在就是考虑在 $time = T$ 的情况下是否可以使军队出现在任意叶子节点到 1 的路径上。

贪心，我们可以容易想到，由于可以同时调动，那么对于任意一支军队，都要使他尽量的往上，因为这样不会使答案更劣，但是这里我们并不能慢慢跳父亲，而且考虑倍增，考虑的方式可以类比倍增求 $lca$ 的过程，而且同时保存 $dis(fa_{2^p}, x)$ 的答案。

# 疫情控制

## 分析

我们做完上一步，现在发现，军队被划分成两种。可以到达 1 号节点。不可以到达 1 号节点。那么我们先对所有子树遍历，考虑这个子树是否可以不需要军队了。那么最后我们剩下的就是，一些还需要军队来出现的子树和一些搁置在 1 号节点的军队。对于每个子树我们记录它的根节点到 1 的距离为 $B$，和每一支可以到达 1 号节点还剩余的时间 $A$ 。那么现在问题转化为。是否可以在 $A$ 序列中选取 $|B|$ 个元素，使 $\forall i$ 满足 $A_i \geq B_i$ 。这个可以考虑贪心，将 $A_i$ 和 $B_i$ 都从小到大排序，维护两个指针 $P_a, P_b$ 。

# 疫情控制

**分析**

如果 $P_a$ 的子树需要一个军队，那么我们可以考虑不让 $P_a$ 跳到 1 号节点，直接使 $P_a$ 的子树变为已覆盖，因为每一个没有被覆盖的子树至少需要一个军队到覆盖，那么在 $B_{P_a}$ 还没有被访问时覆盖，这样一定可以使答案不更劣。如果 $P_a \geq P_b$ 直接覆盖，考虑下一个 $P_a$ 和 $P_b$ 。如果 $P_a < P_b$ ，考虑下一个 $P_a$ 。

# 目录

# Thank you! Any Question?