树

清华大学 茹逸中

目录

1. 有根树
    1. 有根树的定义与性质
    2. 有根树的遍历与动态规划
    3. 有根树适用的数据结构
2. 无根树
    1. 无根树的定义
    2. 倍增法
    3. 分治法
3. 环套树

1.1 有根树的定义与性质

定义

- 有唯一的根
- 除了根以外的节点有唯一的父亲节点

节点的深度：该节点与根节点的距离

树的高度：深度最大的节点的深度 +1

性质

- 无环图
- 连通图
- 分层图
- 边数 = 点数 - 1

有根树举例

- 一个公司中，除了老板以外的每个人都有一个唯一的上司，那么这个公司的所有员工组成一棵有根树
- 用任意一种顺序（深度优先、广度优先、其它顺序）搜索一个图，若不搜索重复节点，则得到一棵搜索树
- 对事物进行多层分类，可以得到一棵有根树
- 平衡树
- .....

1.2 有根树的遍历与动态规划

- 深度优先搜索

```
Visit(i):
    For each child j of i
        visit(j)
```

- 广度优先搜索

```
i = q.front()
    for each child j for i
        q.push(j)
```

1.2 有根树的遍历与动态规划

- 两种遍历方法均为先访问父节点后访问子节点。在一些简单的动态规划问题中，只有父子节点有直接的依赖关系。因此这两种遍历方法的效果往往是一样的。唯一需要注意的是直接用递归方法进行深度优先搜索可能会有爆栈的问题。
- 在一般的树型 DP 问题中，状态转移往往是从孩子节点转移到父亲节点（也有从父亲节点转移到孩子节点的），因此一般只需考虑父亲节点和孩子节点的关系。

1.2 例题：树的重心

- 给定一棵树G，对于一个点x，使得删除x后得到的若干棵树中节点数的最大值最小的x称为树的重心。

- $\text{size}[u] = \sum_{v \in \text{child}(u)} \text{size}[v]$
- $maxsize[u] = max(max_{v \in \text{child}(u)}(size[u]), n - size[u])$

1.2 例题：没有上司的舞会

- Ural大学有N个职员，编号为1~N。他们有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。每个职员有一个快乐指数。现在有个周年庆宴会，要求与会职员的快乐指数最大。但是，没有职员愿和直接上司一起与会。
- f[u][0]表示在u不参会的条件下，以u为根的子树中所有人能达到的最大快乐指数，f[u][1]表示在u参会的条件下，以u为根的子树中所有人能达到的最大快乐指数。
- $f[u][0] = \sum_{v \in child(u)} \max(f[v][0], f[v][1])$
- $f[u][1] = \sum_{v \in child(u)} f[v][0]$

1.2 例题：站岗

- 给定一棵有根树，在一个点上放置岗哨可以覆盖这个点距离<=1的点，在每个点上放哨的代价互不相同，求覆盖所有点的最小代价。
- f[u][0]表示以u为根的子树除u以外全部被覆盖的最小代价
- f[u][1]表示以u为根的子树全部被覆盖但节点u上没有放哨的最小代价
- f[u][2]表示以u为根的子树全部被覆盖且节点u上放哨的最小代价
- $f[u][0] = \sum_{v \in \text{child}(u)} f[v][1]$
- $f[u][1] = min_{v_0 \in child(u)}(f[v_0][2] + \sum_{v \in \text{child}(u), v \neq v0} \min(f[v][1], f[v][2]))$
- $f[u][2] = \sum_{v \in \text{child}(u), v \neq v0} \min(f[v][0], f[v][1], f[v][2]) + w[u]$

1.2 有根树的遍历与动态规划

- 在一些涉及到子树的题目中，深度优先搜索就可以发挥其独特的优势了。
- 注意到在一棵有根树中，任意一棵子树中的所有节点在深搜序列中总是连续的一段，而我们在处理连续的一段时往往拥有更好的方法。

1.2 例题： Apple Tree(POJ3321)

- 给定一棵有根树，有两种操作：改变一个点的权值和询问一棵子树的权值和。
- 得到 DFS 序列后，用树状数组维护。

1.3 有根树适用的数据结构

- 线段树
  - 首先进行 DFS ，得到 DFS 序列。
  - 对 DFS 序列建立一棵线段树
  - 所有子树都可以表示为 DFS 序列中的一段
- 动态树

2.1 无根树的定义

定义

- 无向连通图
- 无环 或 点数比边数少 1

2.2 倍增法

倍增法是无根树中非常重要的一种基本思想，只需要离线预处理后即可在线处理询问。

倍增法一般用来处理和链有关的问题（无法处理和子树有关的问题），一般要求问题满足区间可加性（可以不满足区间可减性）。

满足区间可加性是指，知道 [a,b) 和 [b,c) 的结果后，可以算出 [a,c) 的结果。例如区间和、区间最大值、树中节点的祖先等

满足区间可减性是指，知道 [a,c) 和 [b,c) 的结果后，可以算出 [a,b) 的结果。例如区间和。注意：区间最大值不满足区间可减性。

除此之外，朴素的倍增法还要求问题满足翻转不变性（顺序无关），即 [a,b) 的结果等于 [b,a) 的结果。这个约束可以通过计算两个方向的答案解决。

倍增法的预处理复杂度是 O(n log n) 的，一次询问的复杂度是 O(log n) 的。

2.2 倍增法

- 倍增法处理的一个基本问题是寻找lca（最近公共祖先）问题。

  要点：预处理时倍增、询问时“倍减”。

预处理：

1. 预处理出每个点的深度
2. 预处理出每个点的1-祖先、2-祖先、4-祖先、8-祖先……其中的i-祖先为它的父亲，u的2^i-祖先（$i>0$）为u的2^{i-1}-祖先的2^{i-1}-祖先。

询问：

3. 用“倍减”法找到较深节点与较浅节点深度相同的祖先
4. 两个深度相同的节点同时“倍减”，找到深度最小的不同的祖先
5. 其中一个祖先的父亲即为lca

2.2 倍增法

- 在解决了寻找lca问题后，所有的“链形”问题都可以解决了。只需要在预处理中附加问题答案的预处理，然后在寻找lca时同时附带问题答案的计算即可。

以求点权和为例：

$$s[u][0] = w[u]$$

$$s[u][i] = s[u][i-1] + s\big[f[u][i-1]\big][i-1]$$

$$u = f[u][i], ans \mathrel{+}= s[u][i]$$

2.2 例题：次小生成树 (BZOJ1977)

- 求一张图的严格次小生成树，所谓严格次小是指其边权和不能等于最小生成树的边权和。
- 先求最小生成树
- 要求次小生成树，即要替换一条边
- 对于不在最小生成树上的每一条边，加入后形成一个环，切掉这个环上原来就存在的最大的边
- 若原来的最大边和新边权值一样，则切掉原来的次大边
- 转化为找一条链上的最大和次大问题，可以倍增解决

2.3 分治法

分治法是无根树中另一种重要的方法，它用来处理两类问题

1. 满足某种条件的链的数量 / 满足某种条件的链中某个属性的最值
2. 给定 m 条链，求这些链的某种属性值（离线）

一般来说，分治法比倍增法解决问题的能力更强，但它的编程复杂度和思维复杂度也更大。分治法的基本思想是将树分治，每次只处理经过一个特定点的所有链，对于不经过这个点的链，就可以将这个点删去后处理。这样处理的最大好处是可以处理第一类问题。

复杂度：每一个点最多出现在 O(log n) 棵子树中。如果处理经过一个特定点的所有链的复杂度是 O(t) 的 (t 为该子树大小），则总复杂度是 O(n log n)，如果是 O(t log t) 的，则总复杂度是 O(n log^2 n)

2.3 分治法

点分法：

1. 求出树的重心 g
2. 处理经过 g 的所有链
3. 删去 g ，得到若干子树，递归处理所有子树

2.3 例题 链计数 POJ1741

给定一棵带正边权的无根树，求链长不超过 k 的链的数量。

- 先点分，考虑所有经过 u 的链
- 求出当前子树中所有点与 u 的距离
- 排序后求出和小于 k 的点对数
- 减掉属于 u 的同一个孩子的点对数
- 复杂度 O(n log^2 n)

2.3 例题 链的条件最值

给定一棵无根树，每个点上带两个权值，分别是 w 和 h 。要求一条 w 值和最大的链，其 h 值和不超过给定的值 hmax 。

- 点分
- 考虑经过点 u 的所有链
- 起点和终点来自 u 的不同孩子的子树
- 统计出每个点到 u 的 w 和与 h 和
- 问题转化为给定 k 个集合，每个集合中是若干个数对 <w,h> ，要求从任意两个集合中各选出一个数对 <w1,h1>,<w2,h2> 满足 h1+h2<hmax ，求 w1+w2 的最大值

2.3 例题 链的条件最值

- 依次处理每个集合，维护之前集合总的单调队列（用平衡树维护）
- 单调队列： h 增大时 w 应增大
- 以 h 为关键字维护平衡树
- 插入时若其前一个的 w 比它更大，则不插入
- 插入后若其后一个点的 w 比它更小，则删除后一个点，并继续检查

2.3 例题 LCA

现在我们考虑第二类问题，仍然以寻找 LCA 作为例子。

与倍增不同的是，点分只能通过离线的方法来寻找 m 个点对的 LCA 。

点分后，扫描每个询问，属于 u 不同孩子的节点的 LCA 是 u ，将属于 u 相同孩子 v 的询问交给 v 子树递归处理。

点分层数 O(log n) ，每个询问处理一次复杂度 O(1) ，最多被传递 O(log n) 层，总复杂度 O((n+m)log n)

2.3 例题 cost

C 国共有 n 个城市。有 n-1 条双向道路，每条道路连接两个城市，任意两个城市之间能互相到达。小 R 来到 C 国旅行，他共规划了 m 条旅行的路线，第 i 条旅行路线的起点是 S_i ，终点是 T_i 。在旅行过程中，小 R 每行走一单位长度的路需要吃一单位的食物。 C 国的食物只能在各个城市中买到，而且不同城市的食物价格可能不同。

然而，小 R 不希望在旅行中为了购买较低价的粮食而绕远路，因此他总会选择最近的路走。现在，请你计算小 R 规划的每条旅行路线的最小花费是多少。

2.3 例题 cost

- 先考虑暴力做法
  - 由于小 R 不会绕远路，所以每次行程的路线是固定的
  - 每次购买刚好能坚持到下一个食物价格比当前食物价格低的城市或者终点城市的食物
- 这个问题涉及到状态问题，不满足区间加！不能倍增。
- 试试点分？
- 点分后将每一条旅行路线分为两段，分别是 S--u 和 u--T 。

2.3 例题 cost

- S--u:
- 从上到下计算每个点到 u 所需要的费用。
- 首先需要向上找到第一个比当前点小的点 v ， S--u = S--v + v -- u 。由于 v 的深度比 u 小，因此 v--u 已经算过了。
- u--T:
- 这个更复杂了。
- 首先任何一个点到 u 都会自带状态，即当前最便宜的粮食价格。首先要从 u 往下找到第一个比这个价格更便宜的点 v ，然后走到那个点。然后再计算 v 到 T 所需要的费用。
- 由于要计算 v 到 T 所需要的费用，这时的起点和终点都不固定。我们考虑可以用 u 到 T 的费用减掉 u 到 v 的费用来计算。（为什么满足区间减？）

2.3 例题 cost

- 还有一个重要问题没有解决，如何求出一个点向上的第一个比它价格小的点？
- 倍增！
- 倍增预处理出每一个点向上 $2^0, 2^1, 2^2, \dots$ 个点的最小值。询问时只需要倍增找到最长的最小值高于该点价格的区间即可。

2.3 例题 动态点分

给定一棵带点权的无根树，有 m 次修改点权的操作，要求每次修改后的最长链。

- 每次修改权值最多影响到 O(log n) 棵点分树
- 对于每棵点分树，对于根的每个孩子，维护根到每个节点的链长。每次修改一个点的权值，即修改一个子树中的所有节点的链长。前面提到过，可以用 DFS 序列 + 线段树来维护。
- 维护得根的每个孩子中所有点到根的最长链，用堆维护，选取最大值和次大值得到该点分树的最长链。
- 对于每棵点分树，再用堆维护得总的最长链。
- 复杂度 O((n+m)log^2n)

3 环套树

为了增加难度，有些出题人会把树的题目改装成环套树的题目。

环套树是 n 个点 n 条边的无向图，包括一个环和若干棵外向树。

环套树的题目一般按照如下思路解决：

1. 找到环
2. 环上每个点为根都对于一棵外向树，处理每棵外向树
3. 处理环

如何找到环？

- 直接深搜，找到的第一条回边与原树边构成唯一的环

3 例题 noi2012 迷失游乐园

放假了，小 Z 觉得呆在家里特别无聊，于是决定一个人去游乐园玩。进入游乐园后，小 Z 看了看游乐园的地图，发现可以将游乐园抽象成有 n 个景点、 m 条道路的无向连通图，且该图中至多有一个环（即 m 只可能等于 n 或者 n-1 ）。小 Z 现在所在的大门也正好是一个景点。小 Z 不知道什么好玩，于是他决定，从当前位置出发，每次随机去一个和当前景点有道路相连的景点，并且同一个景点不去两次（包括起始景点）。贪玩的小 Z 会一直游玩，直到当前景点的相邻景点都已经访问过为止。小 Z 所有经过的景点按顺序构成一条非重复路径，他想知道这条路径的期望长度是多少？小 Z 把游乐园的抽象地图画下来带回了家，可是忘了标哪个点是大门，他只好假设每个景点都可能是大门（即每个景点作为起始点的概率是一样的）。同时，他每次在选择下一个景点时会等概率地随机选择一个还没去过的相邻景点。

环长很小，只有 20

3 例题 noi2012 迷失游乐园

- 一棵树怎么做？
- 随便定一个根节点，先考虑只往下走的情况。
- 从下往上DP
- $down[u] = \frac{\sum_{v \in child(u)} down[v]}{|child(u)|} + 1$
- 然后考虑从一个点往上走的情况
- 这是需要从上往下DP，同时需要考虑往上走之后不能回来的条件
- 对于每个点，计算 $up[u] = \frac{up[v] + (|child(v)| down[v] - down(u) - 1)}{|child(v)|} + 1, v = f[u]$
- 对于每个点，计算 $e[u] = \frac{|child(u)| down[u] + up[u]}{|child(u)| + 1}$

3 例题 noi2012 迷失游乐园

- 环套树呢？
- 求 down 时不需要管环，但是 up 不一样，由于向上走到根时有可能会进入环，所以求 up 时需要先处理环。
- 由于环长很小，所以可以随便搞。考虑到一个点不能经过两次，所以在进入环时，除了第一次有两个方向可以选择，其余只能往一个方向走，或者走出环。走出环的期望就是 down[v] ，其中 v 是走出去的那个点。只需要枚举起点，枚举两个方向，然后枚举终点即可算出概率和期望。

总结

- 有根树
  1. 动态规划
  2. DFS 序列
- 无根树
  1. 倍增
  2. 分治
     - 复杂的点分算法
     - 动态点分
- 树形 DP 加强版——环套树形 DP

谢谢！

树

清华大学 茹逸中

目录

1. 有根树
    1. 有根树的定义与性质
    2. 有根树的遍历与动态规划
    3. 有根树适用的数据结构
2. 无根树
    1. 无根树的定义
    2. 倍增法
    3. 分治法
3. 环套树

1.1 有根树的定义与性质

定义

- 有唯一的根
- 除了根以外的节点有唯一的父亲节点

节点的深度：该节点与根节点的距离

树的高度：深度最大的节点的深度 +1

性质

- 无环图
- 连通图
- 分层图
- 边数 = 点数 - 1

有根树举例

- 一个公司中，除了老板以外的每个人都有一个唯一的上司，那么这个公司的所有员工组成一棵有根树
- 用任意一种顺序（深度优先、广度优先、其它顺序）搜索一个图，若不搜索重复节点，则得到一棵搜索树
- 对事物进行多层分类，可以得到一棵有根树
- 平衡树
- .....

1.2 有根树的遍历与动态规划

- 深度优先搜索

```
Visit(i):
    For each child j of i
        visit(j)
```

- 广度优先搜索

```
i = q.front()
    for each child j for i
        q.push(j)
```

1.2 有根树的遍历与动态规划

- 两种遍历方法均为先访问父节点后访问子节点。在一些简单的动态规划问题中，只有父子节点有直接的依赖关系。因此这两种遍历方法的效果往往是一样的。唯一需要注意的是直接用递归方法进行深度优先搜索可能会有爆栈的问题。
- 在一般的树型 DP 问题中，状态转移往往是从孩子节点转移到父亲节点（也有从父亲节点转移到孩子节点的），因此一般只需考虑父亲节点和孩子节点的关系。

1.2 例题：树的重心

- 给定一棵树G，对于一个点x，使得删除x后得到的若干棵树中节点数的最大值最小的x称为树的重心。

- $\text{size}[u] = \sum_{v \in \text{child}(u)} \text{size}[v]$
- $maxsize[u] = max(max_{v \in \text{child}(u)}(size[u]), n - size[u])$

1.2 例题：没有上司的舞会

- Ural大学有N个职员，编号为1~N。他们有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。每个职员有一个快乐指数。现在有个周年庆宴会，要求与会职员的快乐指数最大，但是没有职员愿意和直接上司一起与会。
- f[u][0]表示在u不参会的条件下，以u为根的子树中所有人能达到的最大快乐指数，f[u][1]表示在u参会的条件下，以u为根的子树中所有人能达到的最大快乐指数。
- $f[u][0] = \sum_{v \in child(u)} \max(f[v][0], f[v][1])$
- $f[u][1] = \sum_{v \in child(u)} f[v][0]$

1.2 例题：站岗

- 给定一棵有根树，在一个点上放置岗哨可以覆盖这个点距离<=1的点，在每个点上放哨的代价互不相同，求覆盖所有点的最小代价。
- f[u][0]表示以u为根的子树除u以外全部被覆盖的最小代价
- f[u][1]表示以u为根的子树全部被覆盖但节点u上没有放哨的最小代价
- f[u][2]表示以u为根的子树全部被覆盖且节点u上放哨的最小代价
- $f[u][0] = \sum_{v \in child(u)} f[v][1]$
- $f[u][1] = min_{v_0 \in child(u)}(f[v_0][2] + \sum_{v \in child(u), v \neq v0} min(f[v][1], f[v][2]))$
- $f[u][2] = \sum_{v \in child(u), v \neq v0} min(f[v][0], f[v][1], f[v][2]) + w[u]$

1.2 有根树的遍历与动态规划

- 在一些涉及到子树的题目中，深度优先搜索就可以发挥其独特的优势了。
- 注意到在一棵有根树中，任意一棵子树中的所有节点在深搜序列中总是连续的一段，而我们在处理连续的一段时往往拥有更好的方法。

1.2 例题： Apple Tree(POJ3321)

- 给定一棵有根树，有两种操作：改变一个点的权值和询问一棵子树的权值和。
- 得到 DFS 序列后，用树状数组维护。

1.3 有根树适用的数据结构

- 线段树
  - 首先进行 DFS ，得到 DFS 序列。
  - 对 DFS 序列建立一棵线段树
  - 所有子树都可以表示为 DFS 序列中的一段
- 动态树

2.1 无根树的定义

定义

- 无向连通图
- 无环 或 点数比边数少 1

2.2 倍增法

倍增法是无根树中非常重要的一种基本思想，只需要离线预处理后即可在线处理询问。

倍增法一般用来处理和链有关的问题（无法处理和子树有关的问题），一般要求问题满足区间可加性（可以不满足区间可减性）。

满足区间可加性是指，知道 [a,b) 和 [b,c) 的结果后，可以算出 [a,c) 的结果。例如区间和、区间最大值、树中节点的祖先等

满足区间可减性是指，知道 [a,c) 和 [b,c) 的结果后，可以算出 [a,b) 的结果。例如区间和。注意：区间最大值不满足区间可减性。

除此之外，朴素的倍增法还要求问题满足翻转不变性（顺序无关），即 [a,b) 的结果等于 [b,a) 的结果。这个约束可以通过计算两个方向的答案解决。

倍增法的预处理复杂度是 O(n log n) 的，一次询问的复杂度是 O(log n) 的。

2.2 倍增法

- 倍增法处理的一个基本问题是寻找lca（最近公共祖先）问题。

  要点：预处理时倍增、询问时“倍减”。

预处理：

1. 预处理出每个点的深度
2. 预处理出每个点的1-祖先、2-祖先、4-祖先、8-祖先……其中的1-祖先为它的父亲，i的2^i-祖先（i>0）为i的2^(i-1)-祖先的2^(i-1)-祖先。

询问：

3. 用“倍减”法找到较深节点与较浅节点深度相同的祖先
4. 两个深度相同的节点同时“倍减”，找到深度最小的不同的祖先
5. 其中一个祖先的父亲即为lca

2.2 倍增法

- 在解决了寻找lca问题后，所有的“链形”问题都可以解决了。只需要在预处理中附加问题答案的预处理，然后在寻找lca时同时附带问题答案的计算即可。

以求点权和为例：

$$s[u][0] = w[u]$$

$$s[u][i] = s[u][i-1] + s\big[f[u][i-1]\big][i-1]$$

$$u = f[u][i], ans \mathrel{+}= s[u][i]$$

2.2 例题：次小生成树 (BZOJ1977)

- 求一张图的严格次小生成树，所谓严格次小是指其边权和不能等于最小生成树的边权和。
- 先求最小生成树
- 要求次小生成树，即要替换一条边
- 对于不在最小生成树上的每一条边，加入后形成一个环，切掉这个环上原来就存在的最大的边
- 若原来的最大边和新边权值一样，则切掉原来的次大边
- 转化为找一条链上的最大和次大问题，可以倍增解决

2.3 分治法

分治法是无根树中另一种重要的方法，它用来处理两类问题

1. 满足某种条件的链的数量 / 满足某种条件的链中某个属性的最值
2. 给定 m 条链，求这些链的某种属性值（离线）

一般来说，分治法比倍增法解决问题的能力更强，但它的编程复杂度和思维复杂度也更大。分治法的基本思想是将树分治，每次只处理经过一个特定点的所有链，对于不经过这个点的链，就可以将这个点删去后处理。这样处理的最大好处是可以处理第一类问题。

复杂度：每一个点最多出现在 O(log n) 棵子树中。如果处理经过一个特定点的所有链的复杂度是 O(t) 的 (t 为该子树大小)，则总复杂度是 O(n log n)，如果是 O(t log t) 的，则总复杂度是 O(n log^2 n)

2.3 分治法

点分法：

1. 求出树的重心 g
2. 处理经过 g 的所有链
3. 删去 g ，得到若干子树，递归处理所有子树

2.3 例题 链计数 POJ1741

给定一棵带正边权的无根树，求链长不超过 k 的链的数量。

- 先点分，考虑所有经过 u 的链
- 求出当前子树中所有点与 u 的距离
- 排序后求出和小于 k 的点对数
- 减掉属于 u 的同一个孩子的点对数
- 复杂度 O(n log^2 n)

2.3 例题 链的条件最值

给定一棵无根树，每个点上带两个权值，分别是 w 和 h 。要求一条 w 值和最大的链，其 h 值和不超过给定的值 hmax 。

- 点分
- 考虑经过点 u 的所有链
- 起点和终点来自 u 的不同孩子的子树
- 统计出每个点到 u 的 w 和与 h 和
- 问题转化为给定 k 个集合，每个集合中是若干个数对 <w,h> ，要求从任意两个集合中各选出一个数对 <w1,h1>,<w2,h2> 满足 h1+h2<hmax ，求 w1+w2 的最大值

2.3 例题 链的条件最值

- 依次处理每个集合，维护之前集合总的单调队列（用平衡树维护）
- 单调队列： h 增大时 w 应增大
- 以 h 为关键字维护平衡树
- 插入时若其前一个的 w 比它更大，则不插入
- 插入后若其后一个点的 w 比它更小，则删除后一个点，并继续检查

2.3 例题 LCA

现在我们考虑第二类问题，仍然以寻找 LCA 作为例子。

与倍增不同的是，点分只能通过离线的方法来寻找 m 个点对的 LCA 。

点分后，扫描每个询问，属于 u 不同孩子的节点的 LCA 是 u ，将属于 u 相同孩子 v 的询问交给 v 子树递归处理。

点分层数 O(log n) ，每个询问处理一次复杂度 O(1) ，最多被传递 O(log n) 层，总复杂度 O((n+m)log n)

2.3 例题 cost

C 国共有 n 个城市。有 n-1 条双向道路，每条道路连接两个城市，任意两个城市之间能互相到达。小 R 来到 C 国旅行，他共规划了 m 条旅行的路线，第 i 条旅行路线的起点是 S_i ，终点是 T_i 。在旅行过程中，小 R 每行走一单位长度的路需要吃一单位的食物。 C 国的食物只能在各个城市中买到，而且不同城市的食物价格可能不同。

然而，小 R 不希望在旅行中为了购买较低价的粮食而绕远路，因此他总会选择最近的路走。现在，请你计算小 R 规划的每条旅行路线的最小花费是多少。

2.3 例题 cost

- 先考虑暴力做法
  - 由于小 R 不会绕远路，所以每次行程的路线是固定的
  - 每次购买刚好能坚持到下一个食物价格比当前食物价格低的城市或者终点城市的食物
- 这个问题涉及到状态问题，不满足区间加！不能倍增。
- 试试点分？
- 点分后将每一条旅行路线分为两段，分别是 S--u 和 u--T 。

2.3 例题 cost

- S--u:
- 从上到下计算每个点到 u 所需要的费用。
- 首先需要向上找到第一个比当前点小的点 v ， S--u = S--v + v -- u 。由于 v 的深度比 u 小，因此 v--u 已经算过了。
- u--T:
- 这个更复杂了。
- 首先任何一个点到 u 都会自带状态，即当前最便宜的粮食价格。首先要从 u 往下找到第一个比这个价格更便宜的点 v ，然后走到那个点。然后再计算 v 到 T 所需要的费用。
- 由于要计算 v 到 T 所需要的费用，这时的起点和终点都不固定。我们考虑可以用 u 到 T 的费用减掉 u 到 v 的费用来计算。（为什么满足区间减？）

2.3 例题 cost

- 还有一个重要问题没有解决，如何求出一个点向上的第一个比它价格小的点？
- 倍增！
- 倍增预处理出每一个点向上2^0，2^1…个点的最小值。询问时只需要倍增找到最长的最小值大于等于该点价格的区间即可。

2.3 例题 动态点分

给定一棵带点权的无根树，有 m 次修改点权的操作，要求每次修改后的最长链。

- 每次修改权值最多影响到 O(log n) 棵点分树
- 对于每棵点分树，对于根的每个孩子，维护根到每个节点的链长。每次修改一个点的权值，即修改一个子树中的所有节点的链长。前面提到过，可以用 DFS 序列 + 线段树来维护。
- 维护得根的每个孩子中所有点到根的最长链，用堆维护，选取最大值和次大值得到该点分树的最长链。
- 对于每棵点分树，再用堆维护得总的最长链。
- 复杂度 O((n+m)log^2n)

3 环套树

为了增加难度，有些出题人会把树的题目改装成环套树的题目。

环套树是 n 个点 n 条边的无向图，包括一个环和若干棵外向树。

环套树的题目一般按照如下思路解决：

1. 找到环
2. 环上每个点为根都对于一棵外向树，处理每棵外向树
3. 处理环

如何找到环？

- 直接深搜，找到的第一条回边与原树边构成唯一的环

3 例题 noi2012 迷失游乐园

放假了，小 Z 觉得呆在家里特别无聊，于是决定一个人去游乐园玩。进入游乐园后，小 Z 看了看游乐园的地图，发现可以将游乐园抽象成有 n 个景点、 m 条道路的无向连通图，且该图中至多有一个环（即 m 只可能等于 n 或者 n-1 ）。小 Z 现在所在的大门也正好是一个景点。小 Z 不知道什么好玩，于是他决定，从当前位置出发，每次随机去一个和当前景点有道路相连的景点，并且同一个景点不去两次（包括起始景点）。贪玩的小 Z 会一直游玩，直到当前景点的相邻景点都已经访问过为止。小 Z 所有经过的景点按顺序构成一条非重复路径，他想知道这条路径的期望长度是多少？小 Z 把游乐园的抽象地图画下来带回了家，可是忘了标哪个点是大门，他只好假设每个景点都可能是大门（即每个景点作为起始点的概率是一样的）。同时，他每次在选择下一个景点时会等概率地随机选择一个还没去过的相邻景点。

环长很小，只有 20

3 例题 noi2012 迷失游乐园

- 一棵树怎么做？
- 随便定一个根节点，先考虑只往下走的情况。
- 从下往上DP
- $down[u] = \frac{\sum_{v \in child(u)} down[v]}{|child(u)|} + 1$
- 然后考虑从一个点往上走的情况
- 这是需要从上往下DP，同时需要考虑往上走之后不能回来的条件
- 对于每个点，计算 $up[u] = \frac{up[v] + (|child(v)| down[v] - down(u) - 1)}{|child(v)|} + 1, v = f[u]$
- 对于每个点，计算 $e[u] = \frac{|child(u)| down[u] + up[u]}{|child(u)| + 1}$

3 例题 noi2012 迷失游乐园

- 环套树呢？
- 求 down 时不需要管环，但是 up 不一样，由于向上走到根时有可能会进入环，所以求 up 时需要先处理环。
- 由于环长很小，所以可以随便搞。考虑到一个点不能经过两次，所以在进入环时，除了第一次有两个方向可以选择，其余只能往一个方向走，或者走出环。走出环的期望就是 down[v] ，其中 v 是走出去的那个点。只需要枚举起点，枚举两个方向，然后枚举终点即可算出概率和期望。

总结

- 有根树
  1. 动态规划
  2. DFS 序列
- 无根树
  1. 倍增
  2. 分治
     - 复杂的点分算法
     - 动态点分
- 树形 DP 加强版——环套树形 DP

谢谢！