浅析倍增思想在信息学竞赛中的应用

安徽省芜湖市第一中学 朱晨光

目 录

摘要

倍增思想是解决信息学问题的一种独特而巧妙的思想。本文就倍增思想在信息学竞赛中两个方面的应用进行了分析。全文可以分为五个部分：

第一部分引言，简要阐述了倍增思想的重要作用以及运用方法；

第二部分介绍倍增思想的第一个应用——在变化规则相同的情况下加速状态转移；

第三部分介绍倍增思想的第二个应用——加速对区间进行的操作；

第四部分探讨了一个有趣的问题，即为什么倍增思想每次只将考虑范围扩大一倍而不是两倍、三倍等；

第五部分总结全文，再次指出倍增思想的重要性以及应该怎样灵活运用倍增思想。

关键字

倍增思想　变化规则　状态转移　区间操作

正文

引言

倍增思想是一种十分巧妙的思想，在当今的信息学竞赛中应用得十分广泛。尽管倍增思想可以应用在许多不同的场合，但总的来说，它的本质是：每次根据已经得到的信息，将考虑的范围扩大一倍，从而加速操作。大家所熟悉的归并排序实际上就是倍增思想的一个经典应用。

在解决信息学问题方面，倍增思想主要有这两个方面的应用——

一、在变化规则相同的情况下加速状态转移；

二、加速区间操作。

下文将就这两个方面进行详细的讨论。

倍增思想应用之一　在变化规则相同的情况下加速状态转移[1]

首先，让我们来看一个简单的例子——已知实数 a，计算 a^{17}。

分析：

很显然，一种最简单的方法就是令 b=a，然后重复 16 次进行操作 b=b*a. 这样，为了得到 a^{17}，共进行了 16 次乘法。

[1] 这里是指由一种状态变化到另一种状态，并不只限于动态规划中的“状态转移”。

现在考虑另外一种方法，令 $a_0=a,a_1=a^2,a_2=a^4,a_3=a^8,a_4=a^{16}$,可以看出，$a_i=a_{i-1}^2,(1<=i<=4)$。于是，得到 a_0,a_1,a_2,a_3,a_4 共需要 4 次乘法。而 $a^{17}= a*a^{16}=a_0*a_4$, 也就是说，再进行一次乘法就可以得到 a^{17}.这样，总共进行 5 次乘法就算出了 a^{17}.

如果将这种方法推而广之，就可以解决这样一个一般性的例题：

例1、 已知 a，计算 a^n:

分析：

1、 将 n 表示成为二进制形式并提取出其中的非零位，即 $n=2^{b1}+2^{b2}+\cdots\cdots+2^{bw}$，不妨设 b1<b2<……<bw.

2、 由于已知 a，所以也就知道了 a^{2^0}，重复 bw 次将这个数平方并记录下来，就可以得到(bw+1)个数：a^{2^0}，a^{2^1}，a^{2^2}，……，$a^{2^{bw}}$；

3、 根据幂运算的法则，可以推出 $a^n=a^{2^{b1}+2^{b2}+\ldots\ldots+2^{bw}}=a^{2^{b1}}*a^{2^{b2}}*\cdots\cdots*a^{2^{bw}}$，而这些数都已经被求出，所以最多再进行(bw+1)次操作就可以得到 a^n。

由于 n 的二进制表示最多有 $\lfloor\log_2 n\rfloor+1$ 个非零位，所以 bw 最大为 $\lfloor\log_2 n\rfloor$。也就是说，最多进行 $O(\log_2 n)$次乘法就可以算出 a^n，这比进行 O(n)次乘法效率高得多。

当然，由于得到 n 的二进制表示的过程本身就是按照从低位到高位的顺序，所以并不需要记录 a^{2^0}，a^{2^1}，a^{2^2}，……，$a^{2^{bw}}$，只需要每次即算即用就可以了。伪代码如下（见下页）：

那么，这个算法是如何减少乘法次数的呢？显然，$a^n=a^{2^{b1}+2^{b2}+\ldots\ldots+2^{bw}}=a^{2^{b1}}*a^{2^{b2}}*\cdots\cdots*a^{2^{bw}}$ 使得求 a^n 转化为求不超过 $\lfloor\log_2 n\rfloor+1$ 个 a 的幂的积。而序列 a^{2^0}，a^{2^1}，a^{2^2}，……，$a^{2^{bw}}$ 中除了第一个数以外，每一个数都是前一个数的平方。即在从 a^{2^i} 得到 $a^{2^{i+1}}$ 的过程中，按照原始的方法需要进行 2^i 次乘法操作，而现在只需要利用已知结果 a^{2^i} 进行一次乘法操作（$a^{2^i}*a^{2^i}=a^{2^{i+1}}$）即可。大大减少了操作次数，从而降低了时间复杂度。

```
long double power(long double a,long n)
 {
  long double b,result;
  result=1;
  b=a;
  while (n)
   {
    if (n%2)
     result*=b;
    b*=b;
    n/=2;
   }
  return result;
 }
```

算法 1.1

而在实际情况中，a 可能是一个实数，也可能是一个矩阵或是一个抽象的状态。变化规则也可能是其他操作（如矩阵乘法、动态规划的状态转移等）。但是只要符合以下两个条件，就可以应用倍增思想并采用类似于上面的方法加速计算：

1、每次的变化规则必须相同；

2、变化规则必须满足结合律。

具体到上面的例子，每次的变化规则都是乘法，而乘法是满足结合律的。

下面将通过另一个例子更加深入地探讨倍增思想在加速状态转移方面的应用，同时得到更精确的定义。

例 2、骰子的运动[2]

给定一个六个面的骰子，每个面上都有一定的权值（1 到 50 之间的整数）。骰子运动的范围是一个宽度为 4，向左右无限延伸的带子（如图 1.1）。带子从左到右的横坐标值为……，-3，-2，-1，0，1，2，3，……，从前到后的纵坐标值依次为 1，2，3，4（这里的坐标对应的都是格子，而不是点）。这样，带子被分成了无限多个格子。每个格子恰好能与骰子的一个面完全重合。骰子每次可以向前后左右中的一个方向移动一格（但不能移出带子），花费是移动后朝上的面所附带的权值。

[2] CEPC 2003 Problem D Dice Contest

给定当前骰子位置的坐标与各个面的朝向，求将这个骰子移动到某个新位置所需的最小花费。（所给横坐标的绝对值小于等于 10^9）。

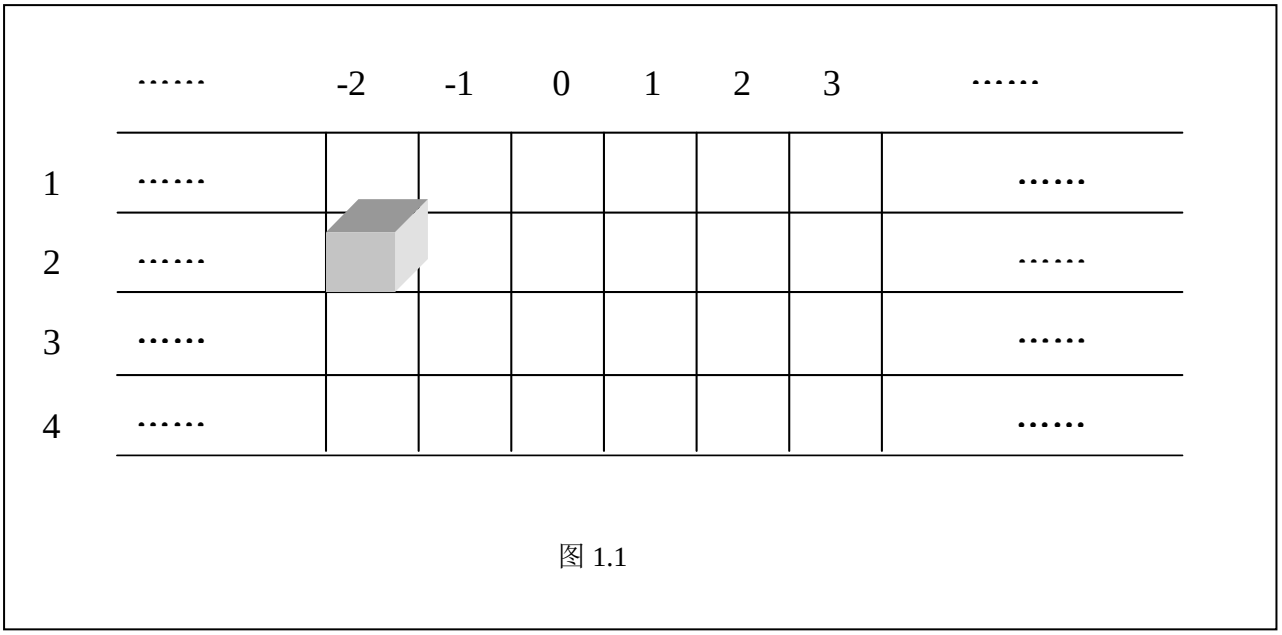


图 1.1

分析：

如果不考虑横坐标巨大的差值，本题完全可以用动态规划求解。方法是将每一格按照骰子的朝向拆分成 24 种状态，然后按列进行动态规划（本质上是一个分层图）得到最小花费。具体方法是从第一列 24*4=96 个状态推到第二列 96 个状态，再推到第三列，第四列……，一直推到终点所在的列，每次都用 Dijkstra 算法算出从一列中某个状态转移到相邻的一列中某个状态的最小花费。时间复杂度是与横坐标差值 n 是同阶的。但是，由于 n 最大可达 10^9，所以这个算法无论从时间还是空间上都难以满足要求。那么，是否可以采用倍增思想呢？答案是肯定的。

下面，我们来验证这里是否存在一种变化规则符合运用倍增思想的要求——相同并符合结合律。

设骰子从第 1 列某个状态 A 运动到第 2 列某个状态 B 需要花费代价 c，则骰子从第 2 列的状态 A（与前一个 A 的行号以及朝向均相同）运动到第 3 列的状态 B 同样需要花费代价 c！这就是一种“相同的变化规则”！

更一般地，如果骰子从第 i 列某个状态 A 运动到第(i+k) [3]列某个状态 B 需要花费代价 c，则骰子从第 j 列某个状态 A 运动到第(j+k)列某个状态 B 也需要花费代价 c，这就是相同的变化规则。

又由于前面所给出的算法是动态规划，而动态规划一个很重要的特点是具有最优子策略。所以如图 1.2 中按照 A->B->C 的顺序计算转移费用再加起来（对于一条路径来说，最小费用是它在这三个部分中的最小费用之和）与按照 B->C->A 的顺序计算转移费用再加起来完全一样，因为它们相互独立，而且合并方式是加法，所以符合结合律。

[3] 这里 k 可以是正数或负数，表示向右或向左运动

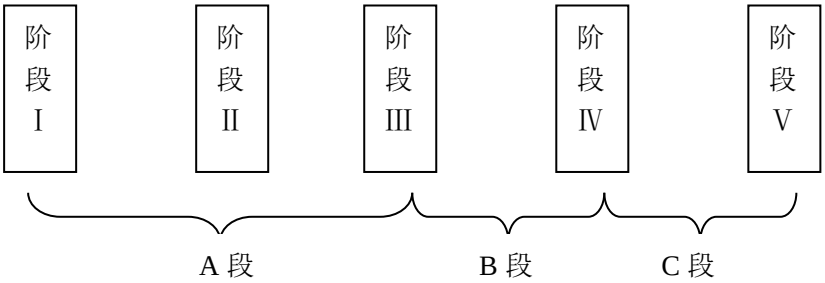


图 1.2

至此，我们证明了变化规则既满足相同性又满足结合律，可以运用倍增思想解决：

（以下默认终点在起点的右侧，如果是在左侧，处理方法类似；如果在同一列，用 Dijkstra 算法就可以得到答案）

1、用 Dijkstra 算法得到一列中某个状态 A 转移到右边相邻一列中某个状态 B 所需最小花费。这可以推出一个 96*96 的矩阵，不妨称为 a^1;

2、 根据已经得到的变化规则 a^1 可以计算出一列中某个状态 A 转移到右边与其距离为 2 的一列中某个状态 B 所需花费——a^2。

3、 依次类推得到 a^4，a^8，a^{16}，a^{32}，……, a^w(w<=n 且 2w>n);

4、 与例 1 相类似，设 $n=2^{b1}+2^{b2}+\cdots\cdots+2^{bw}$，然后从初始状态（即起点所在的那一列）开始，不断对其施行变化规则 $a^{2^{b1}}$、$a^{2^{b2}}$、……、$a^{2^{bw}}$，最终得到从起点到终点所在那一列每个状态的最小花费。

本题中，Dijkstra 算法所耗费的时间可以看成是常数（根据题目中骰子各面权值的取值范围可以算出必须考虑的点数为常数），每次倍增的时间为 96^3，而倍增的次数为 $\log_2 n$，所以上述算法的时间复杂度为 $O(96^3 \log_2 N)$。

从上题中，我们进一步加深了对于倍增思想的理解，并且纠正了一个以前错误的认识：倍增思想所作用的对象实际上是变化规则，而并非具体的状态！倍增思想的作用是算出一个状态到另一个状态的变化量。设初始状态为 A，目标状态为 B，A 到 B 的变化量为 c，则最终结果是 A*c（这里的乘号表示一种变化的手段）。也就是说，倍增思想计算出的量与具体的状态是无关的，而仅与状态之间的关系有关。将这个过程写成伪代码就是（见下页）：

```
Doubling Algorithm(A,n)
 {
  //a,b,c 均为变化规则，a 的初值由其他算法得到。如例 2 中 a 由 Dijkstra 算法得到
  c=b=a;
  while (n)
   {
    if (n%2)
      c=c•b;
    b=b•b;
    n/=2;
   }
  B=A*c;
  return B;
 }
```

算法 1.2

（A 为原状态，B 为末状态。“状态*变化规则[4]”的结果表示对于某个状态施行变化规则后得到的状态，“变化规则•变化规则”的结果表示与依次施行两个变化规则等价的变化规则。）

可以看出，算法 1.2 与算法 1.1 十分相似。其实，算法 1.1 就是算法 1.2 的一个实例。

需要再次强调的是，这里的变化规则必须是相同的（这比较容易检验），并且满足结合律。一个不满足结合律的运算——除法就不能用倍增思想求解（除非转化成乘法）：$a_1/a_2/a_3/a_4\cdots\cdots/a_n \neq a_1/(a_2/a_3/a_4\cdots\cdots/a_n)$。这也提醒我们，在运用倍增思想解题之前，必须检验是否可行。如果不满足条件，就要构造出合法的变化规则或者思考其他的方法。

以上便是倍增思想的第一个应用——在变化规则相同时加速状态转移。当然，这个应用还有一些其他的实现方法。比如在计算 a^n 时,不一定要计算 $a^1,a^2,a^4,a^8\cdots\cdots$，可以按照如下的规则进行递归：

$$a^n=\begin{cases} a & n=1, \\ a^{n/2}*a^{n/2} & n \text{ 为偶数}, \\ a^{(n-1)/2}*a^{(n-1)/2}*a & n \text{ 为奇数}. \end{cases}$$

很显然，这种算法的时间复杂度也是 $O(\log_2 N)$，这也说明倍增思想在实际操作中是灵活多变的，要根据实际情况选择相对简便的形式加以利用。

[4] 这里的规则是灵活多变的，可以指“能否达到”，也可以指“转移后新增的花费”等，要根据具体的题目制定特定的规则。

倍增思想应用之二　加速区间操作

在区间操作方面，有许多表现优秀的算法与数据结构，线段树便是其中的代表。这里之所以提出倍增思想，是因为它也可以胜任在区间上的一些操作，并且在某些特定的情况下可以做得更好。下面先介绍在这种情况下使用倍增思想的模式，再通过几个例题详细说明。

在区间操作中运用倍增思想的一般模式：

预处理：

对于区间中每一个点 A，记录$[A,A+2^0-1],[A,A+2^1-1],[A,A+2^2-1],\cdots\cdots[A,A+2^{\lfloor \log_2 N \rfloor}-1]$[5],这$(\lfloor \log_2 N \rfloor+1)$个区间的性质。在记录的过程中，按照区间长度依次求解（也可以按照点的顺序求解，下文中求树上最近公共祖先的例题中将会有所介绍）。设要求$[A,A+2^{2i}-1]$的性质，则可以通过已知结果$[A,A+2^i-1]$与$[A+2^i, (A+2^i)+ 2^i-1]$得到有关性质。

取用：

在取用区间[A,B]时，有两种适用范围不同的方法：

1、如果区间的重叠对于所需结果无影响（如两段区间重叠对于求最大值没有影响），则令 $k=\lfloor \log_2(B-A+1) \rfloor$. 根据区间$[A,A+2^k-1]$与$[B-2^k+1,B]$就可以得到所需结果（如图 1.3），时间复杂度为 O(1)；

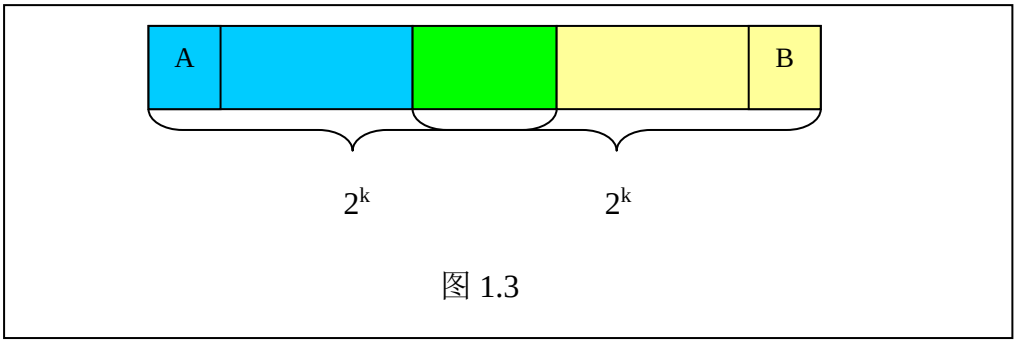


图 1.3

2、如果区间的重叠对于所需结果有影响（如两段区间重叠对于计数问题有影响），则将[A,B]划分成为$[A,A+2^{\lfloor \log_2(B-A+1) \rfloor}-1]$，$[A+2^{\lfloor \log_2(B-A+1) \rfloor}, A+2^{\lfloor \log_2(B-A+1) \rfloor}+2^{\lfloor \log_2(B-(A+2^{\lfloor \log_2(B-A+1) \rfloor})+1) \rfloor}-1]$，……[x,B]；（x 为某个数）每次用对数运算直接得到划分点（如图 1.4）。可以证明每次至少将区间减半，所以最多分成了$\lfloor \log_2(B-A+1) \rfloor+1$ 个区间。因此这样处理的时间复杂度为 $O(\log_2 N)$.

[5] 如果某些区间[A，B]中的 B 已经大于 N，则取[A，N]代替。

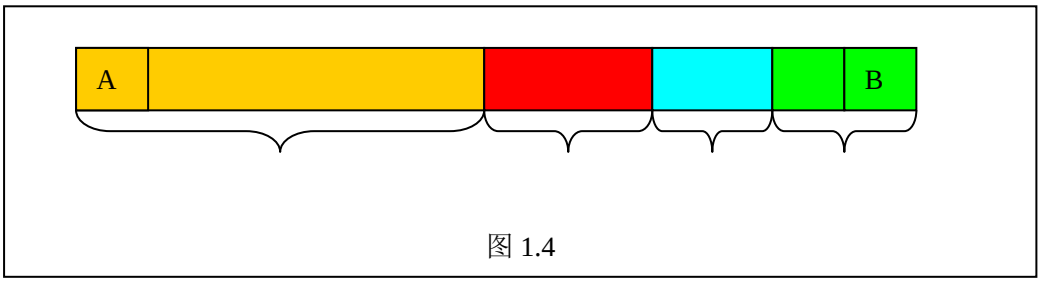


图 1.4

综上所述，预处理时间复杂度为 $O(N\log_2 N)$，处理一个区间的时间复杂度为 $O(1)$或 $O(\log_2 N)$（视处理方法而定）。空间复杂度为 $O(N\log_2 N)$

例 3　一般 RMQ 问题[6]

给定一个长度为 n 的数组 A，每次询问？？ i,j(i<=j)，要求得到 A[i…j]中最小数的下标。

分析：

构造数组 B，使得 B[i,j]表示 A[i…i+2^j-1]中最小数的下标。根据一般模式，可以在 $O(N\log_2 N)$的时间内得到这个数组。

然后对于每对 i,j，可以利用一般模式中第一种取用区间的方法，令 $k=\lfloor \log_2(j-i+1) \rfloor$，则比较 A[B[i,k]]与 A[B[j-$2^k$+1,k]]的大小就可以得到结果了。时间复杂度为 O(1).

这便是巧妙地运用了倍增思想降低了时间复杂度，整个算法也十分简洁明了。

例4　求树上最近公共祖先问题[7]

给定一棵树，节点数为 n. 每次询问两点 i,j 在树上最近的公共祖先 k，询问次数为 m.

分析：

如果每次直接去寻找两点的公共祖先，时间复杂度为 O(MN)，不能满足要求。那么，如何用倍增思想来解决这道题目呢？

对于每个节点 A，记录 A，A 的第 2^0 层祖先（即父亲），A 的第 2^1 层祖先，A 的第 2^2 层祖先，……，A 的第 $2^{\lfloor \log_2 levelA \rfloor}$ 层祖先。如果按照先序遍历的顺序来记录，就可以利用以前的结果（即利用祖先的计算结果）。根据一般模式，可以得出预处理的时间复杂度是 $O(N\log_2 N)$.

然后，对于每对节点 A，B，所要做的就是求 A 的所有祖先与 B 的所有祖先

[6] 经典问题

[7] USACO 2004 February Contest Green Problem 3 Distance Queries

中层次最小的相同节点。做法是这样的：

二分地找到一个 B 的祖先 X，如果 A 的与 X 同层的祖先即为 X[8]，则可以把范围缩小到[B,B 的父亲，……，X]，否则缩小到[X 的父亲，……，根]。

1、二分地找到 B 的祖先 X

首先，令上界为根，下界为 B。因为已经记录了下界 B 的第 2^0 层祖先（即父亲），B 的第 2^1 层祖先，B 的第 2^2 层祖先，……，B 的第 $2^{\lfloor \log_2 levelB \rfloor}$ 层祖先，所以可以令 X 为 B 的第 $2^{\lfloor \log_2 levelB \rfloor}$ 层祖先。当 A 的与 X 同层的祖先 Y 即为 X 时，将上界变为 X，否则将下界变为 X 的父亲。因此，这个步骤的时间复杂度为 $O(\log_2 N)$。

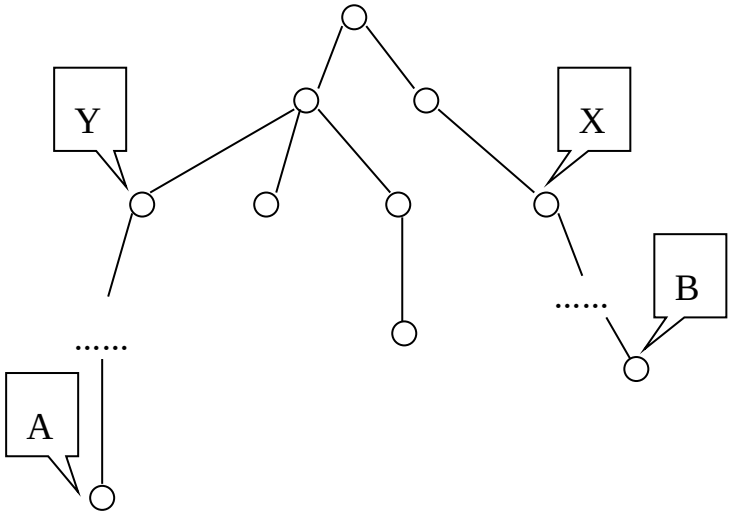


图 1.5

2、找到 A 的与 X 同层的祖先

上文提到了如何在区间重叠对所需结果有影响的情况下取用区间[A,B],这里只不过是把 B 换成了 A 的与 X 同层的祖先。所以可以在 $O(\log_2 N)$的时间内找到这个祖先。

步骤 2 中可以加上这样一个优化，即如果本次找到的祖先并非 X，则根据步骤 1，下一次的 X 一定是这一次的 X 的祖先。那么，下一次步骤 2 就可以不从 A 开始找，而是从本次的 X 的父亲开始找。但是这个优化并不能降低最坏情况下的时间复杂度。

综上所述，步骤 1 与步骤 2 各需要 $O(\log_2 N)$时间，所以这个算法回答每次询问的时间复杂度是 $O(\log_2^2 N)$. 预处理的时间复杂度为 $O(N\log_2 N)$。总的时间复

[8] 如果 X 的层次比 A 还要大，认为 A 的与 X 同层的“祖先”不等于 X。

杂度为 O(Nlog₂N+M$\log_2^2 N$)，空间复杂度为 O(Nlog₂N).

参考文献[1]中给出了一种转化为±1RMQ 求解的方法，可以使得预处理的时间复杂度降为 O(N)，每次回答询问的时间复杂度降为 O(1)，但是操作比较繁琐，这里不再介绍。

以上两个例题说明了倍增思想是如何解决区间操作问题的。虽然在这两题中倍增思想都不是最好的解决办法，但是它的编程复杂度很低，思考起来也不复杂。而在构造后缀数组[9]当中，倍增思想更是发挥了巨大的作用。

在例 4 以及构造后缀数组当中，本身并没有什么明显的“区间”，而都是人为构造出来的。这也体现了倍增思想作为一种思想，并不拘泥于一种死板的模式，而是靠选手通过这种思想来得到巧妙而高效的算法。

一个有趣的探讨

倍增思想的本质是每次将考虑的范围扩大一倍，那么是否可以每次扩大两倍、三倍甚至更多呢？下面就来探讨这个问题。

设每次将考虑的范围扩大(k-1)倍(k>=2)的倍增思想为 k 增思想，则本文所介绍的倍增思想为 2 增思想。

对于 k 增思想，最多通过$\lfloor \log_k N \rfloor$次扩大就可以得到所有需要的值，这里为了讨论方便，简化为 logₖN. 但是，为了每次扩大(k-1)倍，必须操作(k-1)次（这里不再考虑使用倍增思想）。所以时间复杂度为 O((k-1)logₖN)。

如 k=3,N=30，则所需要的值（即 3 的幂）为 3 与 27 (30=3+27)。需要 3 次扩大可以得到 1,3,9,27。但是每次扩大需要进行 3-1=2 次操作。如从 3 得到 9，需要进行两次加法 3+3=6，6+3=9（尽管这里也可以通过一次乘法得到 9，但是如果是矩阵乘法或是更加复杂的操作就只能通过(k-1)次操作得到所需要的值）。

本文中 k=2，因此复杂度为(2-1)log₂N= log₂N.

为了将 k>=3 与 k=2 进行比较。这里采用商值比较法：

$$\frac{(k-1)\log_k N}{\log_2 N}=\frac{(k-1)\dfrac{\log_2 N}{\log_2 k}}{\dfrac{\log_2 N}{\log_2 2}}=\frac{k-1}{\log_2 k}$$. 令 f(k)=(k-1)-log₂k. 当 k=2 时，f(k)=0；

当 k>=3 时，$f'(k)=1-\frac{1}{k}\log_2 e=1-\frac{\ln e}{k\ln 2}=1-\frac{1}{k\ln 2}>=1-\frac{1}{3\ln 2}\approx 0.5191$，也就是说当 k>=3 时 f(k)为增函数，即恒有 f(k)>f(2)=0，即(k-1)-log₂k>0，(k-1)> log₂k.

当然，这个结论也可以根据图象得到（见下页图 1.6）。图中 y_1=k-1 与 y_2=log₂k 的图象共有两个交点(1,0)与(2,1). 而当 k>2 时，y_1 总大于 y_2，所以

[9] 详见参考文献[2]

$\frac{(k-1)\log_k N}{\log_2 N} = \frac{k-1}{\log_2 k} > 1$ (k>=3) ，即(k-1)$\log_k$N>$\log_2$N.

这就从数学角度解释了倍增思想的高效性。

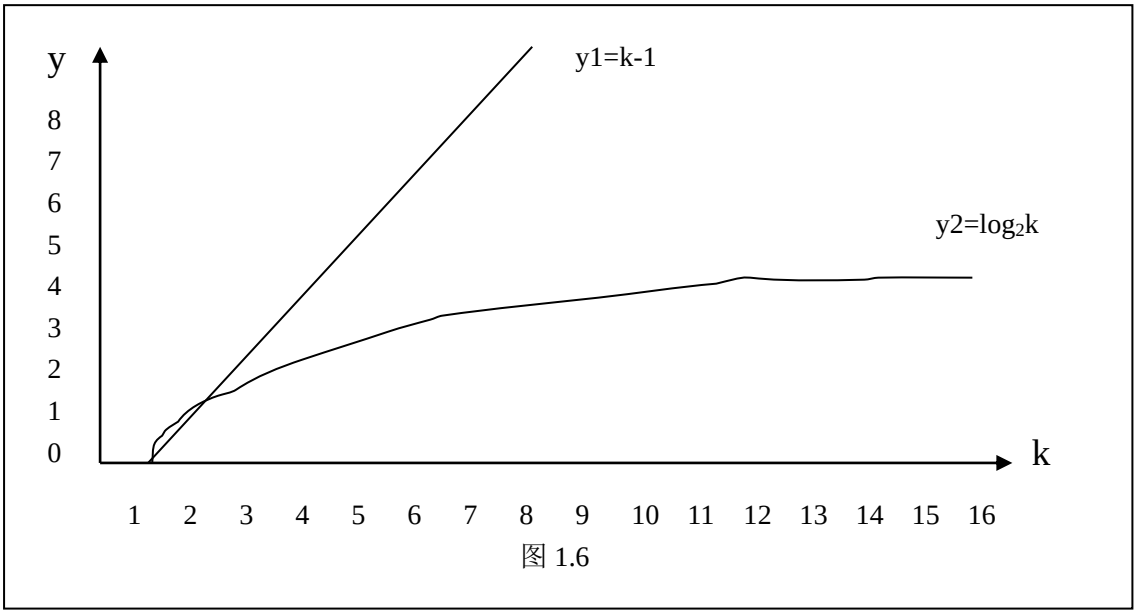


图 1.6

总结

本文简要介绍了倍增思想的重要作用以及它在两个方面的应用——在变化规则相同的情况下加速状态转移以及加速区间操作。倍增思想高效的根源在于它恰当地利用了以前的计算结果。从前文的理论分析可以得知，倍增思想将这种利用发挥到了极致。正因为如此，根据倍增思想设计出的算法中没有冗余的运算，使得它能够高效、简洁地解决许多信息学问题。灵活掌握倍增思想，可以使我们在思考问题时能独辟蹊径，快速地找到时空复杂度与编程复杂度都相对较低的算法，从而在紧张的信息学比赛中占得先机。

倍增思想本身作为一种思想，应用是十分广泛的。尽管本文给出了倍增思想的两类应用以及其一般模式，但应该认识到，倍增思想的作用决不仅在于此，其千变万化的实战运用也决非一两个模式可以涵盖。本文作为一篇浅析倍增思想的论文，只是起到抛砖引玉的作用。希望读者能够从中体会到倍增思想博大精深的内涵，并在实际运用中逐渐积累经验，将倍增思想自然地融入到自己的思考过程当中，指导自己解决各类问题，从而在面对规模巨大的题目时能够做到举重若轻，挥洒自如。当然，这也需要长期的磨练与不断的总结和体会。毕竟，

纸上得来终觉浅，绝知此事要躬行。

感谢

感谢江涛老师阅读我的论文并提出宝贵的修改意见和建议。
感谢刘汝佳教练与许智磊同学，他们与作者就论文写作进行了深入的讨论。

参考文献

[1] 刘汝佳 黄亮，2004，《算法艺术与信息学竞赛》。 北京：清华大学出版社。
[2] 许智磊 IOI2004 国家集训队论文 《后缀数组》
[3] CEPC 2003 Problem D Dice Contest
[4] USACO 2004 February Contest Green Problems

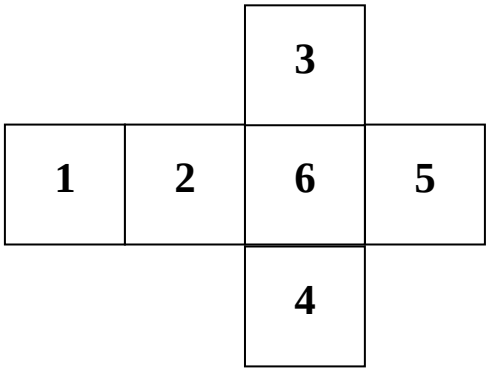
附录

一、文中例子的原题
1、例 2（此处给出许智磊同学翻译并改编后的题目）

Farewell, My Friend!

满分：100
时限：5s
程序：fmf.cpp/fmf.c/fmf.pas fmf.exe
输入：fmf.in
输出：fmf.out

骰子的空间和我们人类所处的空间很不一样，当然，骰子和人长得也不一样。骰子是用如下的六个连在一起的面折叠后粘贴起来形成的，六个面的编号如图所



示：

粘的方式是把 6 放在桌面上，2,3,4,5 都向上折，1 折到顶部（这需要一些空间想象力），然后再粘起来。骰子的每个面上不光有编号，还有分值，每个骰子

六个面上的分值是给定的，分别是 f(1)到 f(6)。

骰子的空间是一个摆放在你的桌面上的带子，这个带子向左延伸至无限长，向右延伸至无限长。带子从前向后（“前”指的是向着你所坐的桌子边缘的方向）被均匀地分成四行，分别用 1 到 4 来编号，每行的宽度恰好是一个骰子的边长。同时，这条带子从左到右被划分成无限多列，这些列用连续的整数来编号，从左到右编号依次增大，每列的宽度恰好也是一个骰子的边长。于是，带子就被分成了无限多个小方格，每个方格恰好全等于骰子的一个面。骰子总是摆放在某个方格上，且某一个面与方格表面重合。用(x,y)可以代表一个骰子的位置，其中 x 代表所处方格的列编号，y 代表所处方格的行编号，x 是整数，y 是 1,2,3,4 中的某一个。

一个骰子可以向它的前后左右四个相邻的方格移动，移动的方法是以它的底面上的一条边为轴向外滚动 90 度，如果滚动的轴是底面的左边那么它就移动到了左边的相邻格，滚动轴是前边那么它就移动到了前边的相邻格，等等。当然，如果骰子处在第 1 行那它没有前边的相邻格，处在第 4 行则没有后边的相邻格，移动的时候不能超出这两个边界。从一个旧方格移动到相邻的一个新方格的花费是移动以后这个骰子顶部那个面上的分值。

你的朋友，它是一个骰子，目前正处于(x1,y1)，它的 6 号面贴着带子，1 号面处于顶部，2 号面对着前面。它想做一次远行，移动到(x2,y2)上，这是一个冒险性的举动，除了说“再见”之外，你唯一能为它做的就是计算一下它这次远足的最小花费。

输入第一行是六个数，依次给出 f(1)到 f(6)，均在 1 到 50 范围内。第二行四个数 x1,y1,x2,y2，给出了出发地(x1,y1)和目的地(x2,y2)。x1,x2 的绝对值不超过 10^9。输出一行，单独的一个整数 M，代表从(x1,y1)移动到(x2,y2)的最小花费。

样例：

输入	输出
1 2 8 3 1 4 -1 1 0 2	7

2、例 3　一般 RMQ 问题（经典问题）

给定 N 与 N 个整数 A[1…N],M 与 M 对下标(x,y) (x<=y),对于每对下标(x,y)求出 A[x…y]中最小数的下标。

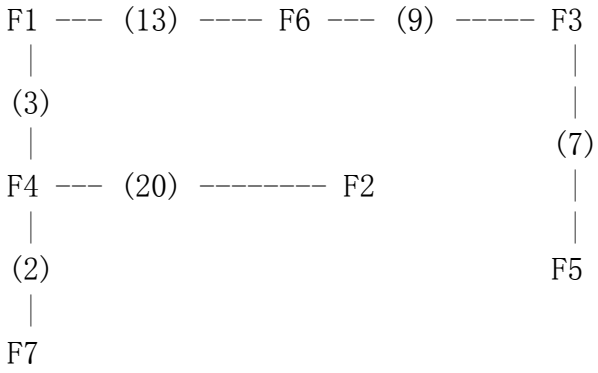
3、例 4 原题

USACO 2004 February Contest Green Problem 3 Distance Queries

Distance Queries [Brian Dean, 2004]

Farmer John's pastoral neighborhood has N farms (2 <= N <= 40,000), usually numbered/labeled 1..N. A series of M (1 <= M < 40,000) vertical and horizontal roads each of varying lengths (1 <= length <= 1000) connect the farms. A map of these farms might look something like the illustration below in which farms are labeled F1..F7 for clarity

and lengths between connected farms are shown as (n):



Being an ASCII diagram, it is not precisely to scale, of course.

Each farm can connect directly to at most four other farms via roads that lead exactly north, south, east, and/or west. Moreover, farms are only located at the endpoints of roads, and some farm can be found at every endpoint of every road. No two roads cross, and precisely one path (sequence of roads) links every pair of farms.

FJ lost his paper copy of the farm map and he wants to reconstruct it from backup information on his computer. This data contains lines like the following, one for every road:

> There is a road of length 10 running north from Farm #23 to Farm #17
> There is a road of length 7 running east from Farm #1 to Farm #17
> ...

Farmer John's cows refused to run in his marathon since he chose a path much too long for their leisurely lifestyle. He therefore wants to find a path of a more reasonable length. The input to this problem consists of the description of the farm and is followed by a line containing a single integer K, followed by K "distance queries". Each distance query is a line of input containing two integers, giving the numbers of two farms between which FJ is interested in computing distance (measured in the length of the roads along the path between the two farms). Please answer FJ's distance queries as quickly as possible!

PROBLEM NAME: dquery
INPUT FORMAT:
* Line 1: Two space-separated integers: N and M
* Lines 2..M+1: Each line contains four space-separated entities, F1, F2, L, and D that describe a road. F1 and F2 are numbers of two farms connected by a road, L is its length, and D is a character that is either 'N', 'E', 'S', or 'W' giving the direction of the road from F1 to F2.
* Line 2+M: A single integer, K. 1 <= K <= 10,000
* Lines 3+M..2+M+K: Each line corresponds to a distance query and contains the indices of two farms.
SAMPLE INPUT (file dquery.in):
7 6
1 6 13 E
6 3 9 E
3 5 7 S
4 1 3 N
2 4 20 W
4 7 2 S
3
1 6
1 4
2 6

INPUT DETAILS:
This is the farm layout drawn above.
OUTPUT FORMAT:
* Lines 1..K: For each distance query, output on a single line an integer giving the appropriate distance.
SAMPLE OUTPUT (file dquery.out):
13
3
36
OUTPUT DETAILS:
Farms 2 and 6 are 20+3+13=36 apart.

二、例 2——例 4 源程序

1、例 2 程序

```
#include <stdio.h>
#include <stdlib.h>
#define infile "fmf.in"
```

```
#define outfile "fmf.out"
#define shapenum 24
#define mid 3
#define distlen 5
#define maxnum 10000000
#define df 1e12

const long long shape[shapenum+1][4]={{0,0,0,0},
                        {0,6,2,4},
                        {0,6,4,5},
                        {0,6,5,3},
                        {0,6,3,2},
                        {0,5,1,3},
                        {0,5,3,6},
                        {0,5,6,4},
                        {0,5,4,1},
                        {0,4,1,5},
                        {0,4,5,6},
                        {0,4,6,2},
                        {0,4,2,1},
                        {0,3,6,5},
                        {0,3,5,1},
                        {0,3,1,2},
                        {0,3,2,6},
                        {0,2,1,4},
                        {0,2,4,6},
                        {0,2,6,3},
                        {0,2,3,1},
                        {0,1,5,4},
                        {0,1,4,2},
                        {0,1,2,3},
                        {0,1,3,5}};

const long long opposite[7]={0,6,5,4,3,2,1};
const long long deltax[5]={0,0,1,0,-1};
const long long deltay[5]={0,-1,0,1,0};
const long long gb[5][4]={{0,0,0,0},
                  {0,2,-1,3},
                  {0,3,2,-1},
                  {0,-2,1,3},
                  {0,-3,2,1}};

long long dy[7][7][7],f[7],done[distlen+1][5][shapenum+1],
    dist[distlen+1][5][shapenum+1],zhy[5][shapenum+1][5][shapenum+1],
```

```
    now[5][shapenum+1],x1,y1,x2,y2;
FILE *fin=fopen(infile,"r"),
    *fout=fopen(outfile,"w");

void change(long long sign,long long x,long long y,long long z,long long
*nx,long long *ny,long long *nz)
{
 long long a[4],b[4],i;
 (*nx)=x+deltax[sign];
 (*ny)=y+deltay[sign];
 for (i=1; i<=3; i++)
  a[i]=shape[z][i];
 for (i=1; i<=3; i++)
  {
   if (gb[sign][i]<0)
    b[i]=opposite[a[-gb[sign][i]]];
    else b[i]=a[gb[sign][i]];
  }
 (*nz)=dy[b[1]][b[2]][b[3]];
}

void calc_dist_zhy()
//calculate the initial regulation of changes---zhy(a[1])
{
 long long a,b,i,j,k,x,y,z,nx,ny,nz,xynum,min,qq,zeng;
 qq=3*4*24;
 for (a=1; a<=4; a++)
  for (b=1; b<=24; b++)
   {
    for (i=1; i<=distlen; i++)
     for (j=1; j<=4; j++)
      for (k=1; k<=shapenum; k++)
       {
        dist[i][j][k]=df;
        done[i][j][k]=0;
       }
    dist[mid][a][b]=0;
    xynum=0;
    while (xynum<qq)
     {
      min=df;
      for (i=1; i<=distlen; i++)
       for (j=1; j<=4; j++)
        for (k=1; k<=shapenum; k++)
```

```
      if ((done[i][j][k]==0)&&(dist[i][j][k]<min))
       {
        min=dist[i][j][k];
        x=i;
        y=j;
        z=k;
       }
   if (min==df)
    break;
   done[x][y][z]=1;
   if ((x>=mid-1)&&(x<=mid+1))
    xynum++;
   for (i=1; i<=4; i++)
    {
     change(i,x,y,z,&nx,&ny,&nz);
     if ((nx>=1)&&(nx<=distlen)&&(ny>=1)&&(ny<=4))
      if (done[nx][ny][nz]==0)
       {
        zeng=f[opposite[shape[nz][1]]];
        if (dist[x][y][z]+zeng<dist[nx][ny][nz])
         dist[nx][ny][nz]=dist[x][y][z]+zeng;
       }
    }
  }
 if (x1<x2)
  {
   for (i=1; i<=4; i++)
    for (j=1; j<=shapenum; j++)
     zhy[a][b][i][j]=dist[mid+1][i][j];
  }
 if (x1>x2)
  {
   for (i=1; i<=4; i++)
    for (j=1; j<=shapenum; j++)
     zhy[a][b][i][j]=dist[mid-1][i][j];
  }
 if (x1==x2)
  {
   if ((a==y1)&&(b==1))
    {
     min=df;
     for (i=1; i<=shapenum; i++)
      if (dist[mid][y2][i]<min)
       min=dist[mid][y2][i];
```

```
       fprintf(fout,"%Ld\n",min);
       //change!!!!!!!!!!!!!!!!!!!!
       fclose(fout);
       exit(0);
      }
    }
   }
}
void init()
{
 long long i;
 for (i=1; i<=6; i++)
  fscanf(fin,"%Ld",&f[i]);
 fscanf(fin,"%Ld%Ld%Ld%Ld",&x1,&y1,&x2,&y2);
 //change!!!!!!!
 fclose(fin);
 for (i=1; i<=shapenum; i++)
  dy[shape[i][1]][shape[i][2]][shape[i][3]]=i;
 calc_dist_zhy();
}

void zhuan1()  //to update now---the specific state
{
 long long i,j,k,h,temp[5][shapenum+1];
 for (i=1; i<=4; i++)
  for (j=1; j<=shapenum; j++)
   temp[i][j]=df;
 for (i=1; i<=4; i++)
  for (j=1; j<=shapenum; j++)
   if (now[i][j]!=df)
    for (k=1; k<=4; k++)
     for (h=1; h<=shapenum; h++)
      if ((now[i][j]+zhy[i][j][k][h]<temp[k][h]))
       temp[k][h]=now[i][j]+zhy[i][j][k][h];
 for (i=1; i<=4; i++)
  for (j=1; j<=shapenum; j++)
   now[i][j]=temp[i][j];
}

void zhuan2()  //to update zhy---the regulation of changing
{
 long long i,j,k,h,a,b,qq,temp[5][shapenum+1][5][shapenum+1];
 for (i=1; i<=4; i++)
  for (j=1; j<=shapenum; j++)
```

```
   for (k=1; k<=4; k++)
    for (h=1; h<=shapenum; h++)
     temp[i][j][k][h]=df;
 for (i=1; i<=4; i++)
  for (j=1; j<=shapenum; j++)
   for (k=1; k<=4; k++)
    for (h=1; h<=shapenum; h++)
     if (zhy[i][j][k][h]!=df)
      for (a=1; a<=4; a++)
       for (b=1; b<=shapenum; b++)
        if (zhy[i][j][k][h]+zhy[k][h][a][b]<temp[i][j][a][b])
         temp[i][j][a][b]=zhy[i][j][k][h]+zhy[k][h][a][b];
 for (i=1; i<=4; i++)
  for (j=1; j<=shapenum; j++)
   for (k=1; k<=4; k++)
    for (h=1; h<=shapenum; h++)
     zhy[i][j][k][h]=temp[i][j][k][h];
}

void work()
{
 long long x,i,j,temp;
 x=x2-x1;
 if (x<0)
  x=-x;
 for (i=1; i<=4; i++)
  for (j=1; j<=shapenum; j++)
   now[i][j]=df;
 now[y1][1]=0;
 while (x>0)
  {
   temp=x%2;
   if (temp==1)
    zhuan1();
   zhuan2();
   x/=2;
  }
}

void output()
{
 long long i,result;
 result=df;
 for (i=1; i<=shapenum; i++)
```

```
  if (now[y2][i]<result)
   result=now[y2][i];
 fprintf(fout,"%Ld\n",result);
 //change!!!!!!!!!!!!!!!!
 fclose(fout);
}

int main()
{
 init();
 work();
 output();
 return 0;
}
```

2、例 3 程序

```
#include <stdio.h>
#include <math.h>
#define infile "rmq.in"
#define outfile "rmq.out"
#define maxn 50000
#define maxlogn 17

long a[maxn+1],b[maxn+1][maxlogn+1],mi[maxlogn+1],logn,n,m,x,y;

FILE *fin=fopen(infile,"r"),
     *fout=fopen(outfile,"w");

void init()
{
 long i;
 fscanf(fin,"%ld",&n);
 for (i=1; i<=n; i++)
  fscanf(fin,"%ld",&a[i]);

 logn=0;
 mi[0]=1;
 while (mi[logn]*2<=n)
  {
   logn++;
   mi[logn]=mi[logn-1]*2;
  }
}
```

```
long min(long sa,long sb)
{
 if (a[sa]<a[sb])
  return (sa);
  else return (sb);
}

void calc_b()
{
 long i,j;
 for (i=1; i<=n; i++)
  b[i][0]=i;
 for (j=1; j<=logn; j++)
  for (i=1; i<=n; i++)
   if (i+mi[j-1]>n)
    b[i][j]=b[i][j-1];
    else b[i][j]=min(b[i][j-1],b[i+mi[j-1]][j-1]);
}

void deal_with_query()
{
 long i,k,result;
 fscanf(fin,"%ld",&m);
 for (i=1; i<=m; i++)
  {
   fscanf(fin,"%ld%ld",&x,&y);
   k=(long) (log2(y-x+1));
   result=min(b[x][k],b[y-mi[k]+1][k]);
   fprintf(fout,"%ld\n",result);
  }
 fclose(fin);
 fclose(fout);
}

void work()
{
 calc_b();
 deal_with_query();
}

int main()
{
 init();
 work();
```

```
 return 0;
}
```

3、例 4 程序

```
/*
PROG: dquery
LANG: C++
*/
#include <fstream.h>
#define infile "dquery.in"
#define outfile "dquery.out"
#define maxn 41000
#define maxm 41000
#define maxlogn 20

struct anedge{
            long x,y,len;
           }edge[maxm+1];

struct qqedge{
            long node,len;
           }*g[maxn+1];

struct jl{
         long node,dist;
        }df[maxn+1][maxlogn+1];

long degree[maxn+1],xl[maxn+1],level[maxn+1],
    father[maxn+1],done[maxn+1],top[maxn+1],
    n,m,result,asknum;             //top[i] stands for the size of df[i].

ifstream qin(infile);

long calc_father(long node)
{
 long j,k,temp;
 j=node;
 while (father[j]>0)
  j=father[j];
 k=node;
 while (k!=j)
  {
   temp=father[k];
   father[k]=j;
```

```
   k=temp;
  }
 return (j);
}

void combine(long a,long b)
{
 father[b]+=father[a];
 father[a]=b;
}

void calc_xl()
//calculate the pre-order of nodes in the tree and get the level of every
node
{
 long i,j,now,node,head,tail,qq;
 for (i=1; i<=n; i++)
  level[i]=father[i]=done[i]=0;
 head=0;
 tail=1;
 xl[1]=done[1]=level[1]=1;
 top[1]=0;
 while (head<tail)
  {
   head++;
   node=xl[head];
   for (j=1; j<=degree[node]; j++)
    {
     qq=g[node][j].node;
     if (done[qq]==0)
      {
       done[qq]=1;
       father[qq]=node;
       tail++;
       xl[tail]=qq;
       level[qq]=level[node]+1;

       top[qq]=1;
       df[qq][1].node=node;
       df[qq][1].dist=g[node][j].len;
       i=node;
       now=1;
       while (1)
        {
```

```
        if (now>top[i])
         break;
        top[qq]++;
        df[qq][top[qq]]=df[i][now];
        df[qq][top[qq]].dist+=df[qq][top[qq]-1].dist;
        i=df[i][now].node;
        now++;
       }
      }
    }
  }
}

void init()
{
 long i,j,x,y,fx,fy,len,nm;
 char c;
 qin>>n>>m;
 for (i=1; i<=n; i++)
  {
   father[i]=-1;
   degree[i]=0;
  }
 nm=0;
 for (i=1; i<=m; i++)
  {
   qin>>x>>y>>len>>c;
   fx=calc_father(x);
   fy=calc_father(y);
   if (fx!=fy)
    {
     nm++;
     edge[nm].x=x;
     edge[nm].y=y;
     edge[nm].len=len;
     degree[x]++;
     degree[y]++;
     combine(fx,fy);
    }
  }
 m=nm;
 for (i=1; i<=n; i++)
  {
   g[i]=new (struct qqedge [degree[i]+2]);
```

```
   g[i][0].node=0;
  }
 for (i=1; i<=m; i++)
  {
   x=edge[i].x;
   y=edge[i].y;
   len=edge[i].len;
   g[x][0].node++;
   g[x][g[x][0].node].node=y;
   g[x][g[x][0].node].len=len;
   g[y][0].node++;
   g[y][g[y][0].node].node=x;
   g[y][g[y][0].node].len=len;
  }
 calc_xl();
}

void suan(long dian,long ceng,long *node,long *len)
//calculate an ancestor of "dian" and the ancestor is at level "ceng"
{
 long now,i;
 if (ceng>level[dian])
  {
   (*node)=(*len)=-1;
   return;
  }
 if (ceng==level[dian])
  {
   (*node)=dian;
   (*len)=0;
   return;
  }
 now=dian;
 (*len)=0;
 while (level[now]>ceng)
  {
   i=1;
   while ((i<=top[now])&&(level[df[now][i].node]>=ceng))
    i++;
   i--;
   (*len)+=df[now][i].dist;
   now=df[now][i].node;
  }
 (*node)=now;
```

```
 return;
}

void calc_result(long x,long y)
//calculate the sum of the distance from x to z and from y to z(z is the
nearest common ancestor of node x and y)
{
 long l1,l2,n1,n2,start,stop,mid;
 if (x==y)
  {
   result=0;
   return;
  }
 start=1;
 stop=level[y];
 result=0;
 while (start<=stop)
  {
   mid=(start+stop)/2;
   suan(y,mid,&n1,&l1);
   suan(x,mid,&n2,&l2);
   if (n1==n2)
    start=mid;
    else {
         stop=mid;
         y=n1;
         result+=l1;
        }
   if (start==stop)
    {
     result+=l2;
     break;
    }
   if (start+1==stop)
    {
     suan(y,stop,&n1,&l1);
     suan(x,stop,&n2,&l2);
     if (n1==n2)
      {
       result+=l1+l2;
       return;
      }
     suan(y,start,&n1,&l1);
     suan(x,start,&n2,&l2);
```

```
    if (n1==n2)
     {
      result+=l1+l2;
      return;
     }
    return;
   }
  }
}

void work()
{
 long i,x,y;
 qin>>asknum;
 ofstream cout(outfile);
 for (i=1; i<=asknum; i++)
  {
   qin>>x>>y;
   calc_result(x,y);
   cout<<result<<endl;
  }
}

int main()
{
 init();
 work();
 return 0;
}
```