搜索问题中的meet in the middle技巧

南京外国语学校 乔明达

摘 要

meet in the middle是一种常用的搜索优化技巧。部分搜索问题可以使用meet in the middle技巧大幅优化时间复杂度，从而在时限内搜索出结果。

本文分三部分。第一部分将给出一个抽象的有向图模型，以此为基础介绍meet in the middle技巧的思想，并运用该技巧给出针对有向图模型设计的算法。第二部分将简要论证第一部分给出算法的正确性、时间复杂度及其使用条件。第三部分将研究几个使用该技巧的例题，以这些例题为基础，概括得出一个方程模型，再分析该技巧的几个变种。

1 第一部分：有向图模型

1.1 问题的实质

一部分搜索问题的实质是：求有向图G中点A到点B的长度为L的不同路径的数目。这里“长度为L的路径”定义为一个有向边的序列$(E_1, E_2, \ldots, E_L)$，满足有向边E_1的始点为点A，有向边E_L的终点为点B，并且对于$i = 1, 2, \ldots, L-1$，E_i的终点等于E_{i+1}的始点。两条路径$(E_1, E_2, \ldots, E_L)$和$(E'_1, E'_2, \ldots, E'_L)$不同，当且仅当存在$1 \le i \le L$使得$E_i \neq E'_i$。

假设图G中点的最大出度为常数D，描述图G中一个点需要的空间为M，并且计算一个点的某一个后继需要的时间为T。

例如：每个点的标号是一个长度为N的数列$P_1, P_2, \ldots, P_N$，将$P_1, P_2, \ldots, P_{N-1}$或者$P_2, P_3, \ldots, P_N$翻转，可以得到这个点的后继。这里“翻转$P_1, P_2, \ldots, P_{N-1}$”表示：交换$P_1$和$P_{N-1}$，交换$P_2$和$P_{N-2}$，……，交换$P_{\lfloor \frac{N}{2} \rfloor}$和$P_{\lceil \frac{N}{2} \rceil}$。那么：每个点恰有两个后继；描述一个点需要用$O(N)$的空间来存储这个数列；计算某一个

后继需要$O(N)$的时间翻转数列。因此在这个例子中$D = 2$，$M = O(N)$且$T = O(N)$。

根据以上假设，我们分析求解以上问题的朴素DFS算法。这个算法限制DFS的深度不超过L，当搜索深度达到L的时候，将当前点与点B比较，如果是同一个点就将答案增加1。容易证明，在搜索过程中经过的点数不超过$1 + D + D^2 + \cdots + D^L = \frac{D^{L+1}-1}{D-1} = O(D^L)$，并且搜索$L$步之后到达的点数不超过$D^L = O(D^L)$，注意这里多次经过同一个点需要算多次。

由于一个点占据$O(M)$空间，所以比较一个点是否是点B需要$O(M)$时间。另一方面，在搜索中计算每个后继需要$O(T)$时间。因此，朴素DFS算法的时间复杂度为$O(D^L(M + T))$。当L较大的时候，朴素算法的运行时间将会很长，多数情况下不能在时间限制内求出答案，我们需要考虑优化搜索的方法。

在这类问题中，图G中的点非常多，甚至图G可能是无穷图，因此搜索过程中的重复子问题较少，动态规划不是一个有效的解决方法。由于最坏情况下我们不得不遍历整棵搜索树，所以用来优化搜索最优解问题的技巧也不适用于此类求路径数目的问题。我们需要从整体的搜索方法上进行变革。

1.2 meet in the middle的思想

meet in the middle的字面意义是“在中间相遇”。假设有两个人甲和乙，分别从A地和B地出发，向对方的出发点走去。当他们都走了$\frac{L}{2}$ 米的时候恰好“在中间相遇”，那么可以肯定，我们找到了一条从A到B长度为L米的路径。

下面我们考虑如何把这个思想应用到有向图模型。

1.3 针对有向图模型的算法流程

为了便于描述，我们构造一张新图G'。G'中的点与G中相同，但G中一条从X到Y的有向边在G'中对应一条Y到X的有向边。此外，令$L_1 = \lfloor \frac{L}{2} \rfloor$，$L_2 = \lceil \frac{L}{2} \rceil$。通过讨论$L$的奇偶性不难证明$L_1 + L_2 = L$。

算法的第一部分，在图G中从点A出发进行DFS，限制搜索深度不超过L_1。记录下搜索L_1步之后到达的所有点以及到达每个点的次数，记走L_1步到达点P的次数为$Count(P)$。如果搜索L_1步没有到达过点P，那么规定$Count(P) = 0$。

算法的第二部分，在图G'中从点B出发进行DFS，限制搜索深度不超过L_2。假设搜索L_2步之后到达点Q，我们就宣称找到了$Count(Q)$条长度为$L_1 + L_2 = L$的路径，答案累加$Count(Q)$。注意到：搜索L_2步之后我们可能多次到达同一个点，那么每一次到达都要累加$Count(Q)$。

从以上算法流程可以看出，meet in the middle是一种“双向搜索”的技巧。

2 第二部分：算法的分析

2.1 算法的正确性

令$L_1 = \lfloor \frac{L}{2} \rfloor$，$L_2 = \lceil \frac{L}{2} \rceil$。对于图$G$中的某一点$P$，设从点$A$到点$P$长度为$L_1$的不同路径有$f(P)$条，从点$P$到点$B$长度为$L_2$的不同路径有$g(P)$ 条。

引理1. *点A到点B长度为L的不同路径有$\sum_P [f(P)g(P)]$条。*

Proof. 考虑一条长度为L的路径$(E_1, \ldots, E_L)$，其中E_1的始点为A且E_L的终点为B。设E_{L_1}的终点为P，我们称这样一条路径是“途径”点P的。

根据定义，有$f(P)$种不同的路径$(E_1, \ldots, E_{L_1})$，有$g(P)$种不同的路径$(E_{L_1+1}, \ldots, E_L)$，并且这两类路径的任意一种组合，都能通过拼接得到一条满足：从A出发“途径”点P到达点B且长度恰为L的路径。

根据乘法原理，“途径”点P的路径共有$f(P)g(P)$条。由于从A到B长度为L的路径必然“途径”了某个点，所以不同的路径数目就是$\sum_P [f(P)g(P)]$。

□

引理2. *使用第一部分中给出的算法，得出的结果是$\sum_P [f(P)g(P)]$。*

Proof. 根据两次DFS，对于任意点P，算法第一部分中$Count(P) = f(P)$；算法第二部分中，搜索L_2步到达点P的次数为$g(P)$。

因此，对于$g(P) = 0$的点，算法并没有增加答案，相当于给答案加上了$f(P)g(P) = 0$；对于$g(P) > 0$的点，算法$g(P)$次给答案加上$f(P)$，即总共加上了$f(P)g(P)$。对所有点P进行求和，算法得出的结果就是$\sum_P [f(P)g(P)]$。

□

定理1. *第一部分中给出的算法可以正确地求出图G中从点A出发到达点B且长度为L的不同路径的总数。*

Proof. 结合引理1和引理2可知，算法给出的答案和实际上的路径数目都等于$\sum_P [f(P)g(P)]$。 □

2.2 算法的时间复杂度

下面我们来分析算法的时间复杂度。前文中我们已经假设图G中点的最大出度为D，现在我们假设图G'中点的出度均不超过D。另一方面，和前文中的描述相同，我们仍然假设在图G和G'中，描述一个点需要的空间为M，并且计算出某个点的指定后继的时间为T。

算法的时间复杂度主要取决于两部分的效率：一方面是DFS花费的时间，另一方面是程序计算和查询$Count(P)$的时间。以下分析中，由于D为常数，所以指数上的取整符号可以忽略，也就是说我们规定$O(D^{\frac{L}{2}}) = O(D^{\lfloor\frac{L}{2}\rfloor}) = O(D^{\lceil\frac{L}{2}\rceil})$。

根据以上假设，算法第一部分中DFS的代价为$O(D^{\lfloor\frac{L}{2}\rfloor}T)$，第二部分中DFS的代价为$O(D^{\lceil\frac{L}{2}\rceil}T)$。因此整个算法DFS花费的时间为：

$$O(D^{\lfloor\frac{L}{2}\rfloor}T) + O(D^{\lceil\frac{L}{2}\rceil}T) = O(D^{\frac{L}{2}}T)$$

在第一部分中，我们需要统计搜索$\lfloor\frac{L}{2}\rfloor$步之后到达每个点$P$的次数$Count(P)$；在第二部分中，我们需要对搜索$\lceil\frac{L}{2}\rceil$步之后到达的每个点$Q$查询$Count(Q)$。这要求我们寻找一个数据结构，能够快速地插入数据和查询某一数据出现的次数。可以发现，散列表是一个理想的选择。由散列表的性质可知，如果我们使用简单一致散列（simple uniform hashing），并使用链接法解决碰撞，那么平均情况下一次插入和一次查询操作仅需要$O(M)$时间。这是因为在本问题中，比较两个点是否相同需要$O(M)$时间。在第一部分中，我们需要对散列表进行$O(D^{\lfloor\frac{L}{2}\rfloor})$次插入操作；第二部分中，我们需要对散列表进行$O(D^{\lceil\frac{L}{2}\rceil})$次查询操作。因此计算和查询$Count(P)$的时间为：

$$\left[O(D^{\lfloor\frac{L}{2}\rfloor}) + O(D^{\lceil\frac{L}{2}\rceil})\right] \times O(M) = O(D^{\frac{L}{2}}M)$$

结合以上两点，算法的时间复杂度为

$$O(D^{\frac{L}{2}}(M+T))$$

2.3 算法的使用条件

一个问题要使用第一部分中给出的算法，首先必须符合有向图模型。其次，我们在分析该算法时间复杂度的过程中，实际上又加入了若干限制条件。

在之前的分析中，我们假设图G'中点的出度不超过图G中的最大出度。另一方面，图G'中点的出度实际上等于该点在图G中的入度。因此，一般需要保证图G中点的入度和出度是相同数量级的。

另一方面，我们假设能够在$O(T)$时间内访问G'中某个点的指定后继。这一条件等价于要求在$O(T)$时间内访问G中某个点的指定前驱。

在空间消耗方面，为了保证散列表期望$O(M)$的操作时间复杂度，我们需要耗费$O(D^{\frac{L}{2}}M)$的空间。

3 第三部分：应用

3.1 例题1：方程的解数[25]

3.1.1 题目大意

给出整数M，$k_1, k_2, \ldots, k_N$和$p_1, p_2, \ldots, p_N$，求方程$\sum_{i=1}^{N} k_i {x_i}^{p_i} = 0$满足$1 \leq x_i \leq M$ 的整数解的数目。

3.1.2 数据规模

$N \leq 6, M \leq 150$

3.1.3 分析

我们可以将本题转化为有向图模型，从而利用第一部分给出的算法求解。

我们用二元组$(Depth, Sum)$表示图G中的点，因此描述一个点只需要$O(1)$的空间。$Depth$表示之前的项数，Sum表示前$Depth$项的和。出发点为$(0, 0)$，终点为$(N, 0)$。

[25]题目来源：NOI2001第二试

对于满足$Depth < N$的点$(Depth, Sum)$而言，令$k' = k_{Depth+1}, p' = p_{Depth+1}$，则它恰好有$M$个后继，分别为$(Depth+1, Sum+k' \times 1^{p'}), (Depth+1, Sum+k' \times 2^{p'}), \ldots, (Depth+1, Sum+k' \times M^{p'})$。

由于减法是加法的逆运算，因此每个满足$Depth > 0$的点$(Depth, Sum)$也恰好有M个前驱。如果令$k'' = k_{Depth}, p'' = p_{Depth}$，那么这$M$个前驱分别为$(Depth-1, Sum-k'' \times 1^{p''}), (Depth-1, Sum-k'' \times 2^{p''}), \ldots, (Depth-1, Sum-k'' \times M^{p''})$。

为了计算一个点的后继或前驱，我们需要计算某个数的p_i次方，使用快速幂计算则时间复杂度为$O(\log MaxP)$，其中$MaxP$是$p_1, p_2, \ldots, p_N$中的最大值。

经过以上分析，我们可以直接使用第一部分给出的算法，时间复杂度为$O(M^{\frac{N}{2}} \log MaxP)$。

我们回过头来研究这个例题中使用上述算法的实际意义。下面我们研究$N = 6$的情况：

$$k_1 x_1^{p_1} + k_2 x_2^{p_2} + k_3 x_3^{p_3} + k_4 x_4^{p_4} + k_5 x_5^{p_5} + k_6 x_6^{p_6} = 0$$

朴素的算法是枚举方程左边的6个未知数，判断左边的结果是否等于右边的常数，这样时间复杂度为$O(M^6 \log MaxP)$，非常不理想。

仔细想想，上述方程看起来两边非常不平衡，6个未知数均在左边，导致我们不得不枚举6个未知数。将上述方程移项，可以得到：

$$k_1 x_1^{p_1} + k_2 x_2^{p_2} + k_3 x_3^{p_3} = -k_4 x_4^{p_4} - k_5 x_5^{p_5} - k_6 x_6^{p_6}$$

这样方程两边就“平衡”了。我们假设先枚举左侧的三个数，把枚举出的所有结果存在一张表里，再枚举右侧的三个数，每当计算得到和表中某个数相同的结果时，就得到了方程的一组解。在比较计算结果时，我们并不需要比较每一对数是否相同，我们可以利用数据结构（例如散列表）加快统计的效率，从而得到时间复杂度为$O(M^{\frac{N}{2}} \log MaxP)$的算法。实际上，meet in the middle技巧的关键就在于通过优化合并过程减少运算量。

3.2 例题2：ABCDEF[26]

3.2.1 题目大意

给出一个整数的集合S，求有多少组(a,b,c,d,e,f)满足$a,b,c,d,e,f \in S, d \neq 0$且$\frac{ab+c}{d} - e = f$。

3.2.2 数据规模

$|S| \leq 100$

3.2.3 分析

题目中的方程可以通过移项变为$ab + c = d(e + f)$，于是转化为与例题1本质相同的题目，可以得出时间复杂度为$O(|S|^3)$的算法。

3.3 例题3：EllysBulls[27]

3.3.1 题目大意

有一个N位十进制数A，可能存在前导零。现在给出M个条件，第i个条件包含一个N位数X_i和$Same(A, X_i)$的值。$Same(A, B)$表示N位十进制数A和B数字相同的数位数目，例如$Same(\text{"0312"}, \text{"0321"}) = 2$。如果有唯一的$A$满足要求则输出这个解，否则输出表示无解或多解的信息。

3.3.2 数据规模

$N \leq 9, M \leq 50$

3.3.3 分析

为了便于叙述，定义行向量

$$Sum = \begin{pmatrix} Same(A, X_1) & Same(A, X_2) & \cdots & Same(A, X_M) \end{pmatrix}$$

[26]**题目来源：** SPOJ

[27]**题目来源：** TopCoder Single Round Match 572 - Division I, Level Two

记X_i的第j位为$X_{i,j}$，A的第j位为a_j。定义函数$S(a,b)$，其中$0 \le a,b \le 9$，当$a=b$时$S(a,b)=1$，$a \ne b$ 时$S(a,b)=0$。

此外再定义行向量$Cost(i,j) = \begin{pmatrix} S(X_{1,i},j) & S(X_{2,i},j) & \cdots & S(X_{M,i},j) \end{pmatrix}$。这表示，如果$A$的第$i$位是$j$，那么这一位能与$M$个条件中的哪些数匹配。

根据题目的条件，我们可以列出以下方程：

$$Cost(1,a_1) + Cost(2,a_2) + \cdots + Cost(N,a_N) = Sum$$

我们仍然和前两题一样进行移项，令$Mid = \lfloor \frac{N}{2} \rfloor$，得到以下方程：

$$Cost(1,a_1)+\cdots+Cost(Mid,a_{Mid}) = Sum-Cost(Mid+1,a_{Mid+1})-\cdots-Cost(N,a_N)$$

因为a_i有10种可能的情况，所以搜索方程的左边有$O(10^{Mid})$种不同结果，右边有$O(10^{N-Mid})$种不同结果。由于方程两边都是$1 \times M$的行向量，所以每一步计算和在散列表中操作都需要$O(M)$的时间。因此，我们得到了一个时间复杂度为$O(10^{\frac{N}{2}}M)$的算法，可以通过本题的测试数据。

3.4 小结：方程模型

总结以上三个例题，可以发现题目都可以转化为一种方程模型：求方程$\sum_{i=1}^{N} f(x_i) = S$的解数。

在例题1和2中，$f(x_i)$和S为整数，例题3中$f(x_i)$和S为向量。在这三题中，我们的做法都是基于将方程变形为：

$$\sum_{i=1}^{\lfloor \frac{N}{2} \rfloor} f(x_i) = S - \sum_{i=\lfloor \frac{N}{2} \rfloor+1}^{N} f(x_i)$$

我们先搜索方程的左边，将得到的结果记录在散列表中；再搜索方程的右边，并在散列表中查询之前存储的结果。可以看出，以上简要的解题过程与第一部分给出的算法非常相似。并且从例题1的分析过程可以看出，方程模型可以转化为第一部分中给出的有向图模型，因此方程模型是有向图模型的一种特殊情况。对于方程模型的题目，我们比较容易联想到meet in the middle技巧。

运用以上两个模型，我们可以快速地把具体问题模型化，并使用meet in the middle技巧求解。然而，有些题目不能转化为方程模型或有向图模型，从而不能直接使用第一部分中的算法。但是，这些题与上述两个模型具有相似之处，

我们仍然可以对第一部分的算法进行修改，从而使用meet in the middle技巧优化这些问题。下面我们来分析几个这样的例题。

3.5 例题4：Balanced Cow Subsets[28]

3.5.1 题目大意

给出正整数$a_1, a_2, \ldots, a_N$，先从N个数中选出若干个数（至少一个），再把这些数分为两部分，使得两部分的数之和相等，求第一步选数的方案数。

注意：可能有一种选数方案使得存在多种划分方式，但是只能算一种选数方案。

3.5.2 数据规模

$N \leq 20$

3.5.3 分析

我们将题目换一种等价的描述方式：对于数列$x_1, x_2, \ldots, x_N$，其中$x_i \in \{-1, 0, 1\}$。在满足$\sum_{i=1}^{N} x_i a_i = 0$的前提下，求存在多少个不同的非空集合$S = \{i | 1 \leq i \leq N, x_i \neq 0\}$。

这道题类似于之前给出的方程模型，但我们要求的并不是方程的解数，而是方程的解对应的集合的数目，因此算法需要做一些改动。

定义一个新的数列$y_1, y_2, \ldots, y_N$，规定当$x_i = 0$时$y_i = 0$，当$x_i \neq 0$时$y_i = 2^{i-1}$。于是集合S可以用$\sum_{i=1}^{N} y_i$表示。

设M是一个$[0, N]$范围内的整数，将$\sum_{i=1}^{N} x_i a_i = 0$变形为$x_1 a_1 + x_2 a_2 + \cdots + x_M a_M = -x_{M+1} a_{M+1} - x_{M+2} a_{M+2} - \cdots - x_N a_N$。记方程左边为*LeftSum*，右边为*RightSum*。

那么$\sum_{i=1}^{N} y_i$也可以分为两部分：$LeftSet = y_1 + y_2 + \cdots + y_M$和$RightSet = y_{M+1} + y_{M+2} + \cdots + y_N$。

在算法的第一部分中，我们依然像之前的题目一样先搜索方程左边的M个未知数，共有3^M种情况。我们不仅要记录方程左边的*LeftSum*，还要记录对应

[28]**题目来源**：USACO 2012 US Open, Gold Division

的*LeftSet*。

在算法的第二部分中，如果发现*RightSum*和某个*LeftSum*相等，那么相应的*LeftSet* + *RightSet*就是一个符合要求的集合。我们可以用一个0到$2^N - 1$的布尔数组来记录这些集合。

最后，扫描一遍整个布尔数组，就能统计得到答案。

不难发现，第一部分的时间复杂度为$O(3^M)$。第二部分中枚举了$O(3^{N-M})$种情况，但是对于每种情况需要访问对应的所有*LeftSet*。在最坏情况下，我们需要访问$O(2^M)$个*LeftSet*，因此第二部分的时间复杂度为$O(3^{N-M}2^M)$。最后扫描的时间复杂度为$O(2^N)$。

综上，算法的时间复杂度为$O(3^M + 3^{N-M}2^M + 2^N) = O(3^M + 3^{N-M}2^M)$。如果我们取$M = \lfloor \frac{N}{2} \rfloor$，那么就可以得到$O(6^{\frac{N}{2}}) \approx O(2.45^N)$的时间复杂度，足以通过本题的测试数据。但是实际上，如果我们取$M = N \log_{\frac{9}{2}} 3$，可以得到一个更好的时间复杂度$O((3^{\log_{\frac{9}{2}} 3})^N) \approx O(2.23^N)$。

例题4表明，我们在使用meet in the middle技巧时，可以在散列表中存储一些其它的信息，以便于问题的求解。另一方面，使用技巧时如果将搜索的未知数不均匀地分为两部分，可能得到更优的时间复杂度。

3.6 例题5：AlphabetPaths[29]

3.6.1 题目大意

有一个$R \times C$的矩阵，矩阵的每个格子里要么是空的，要么包含一个$0 \sim 20$的整数。求有多少条包含21个格子的路径满足：路径上相邻两个格子必须有一条公共边，并且$0 \sim 20$每个数都在路径上出现一次。

3.6.2 数据规模

$R, C \leq 21$

[29]**题目来源：** TopCoder Single Round Match 523 - Division I, Level Three

3.6.3 分析

这道题看上去并不是之前提到的方程模型，而是类似于第一部分中的有向图模型。但是在这道题中，路径的起点和终点是不确定的，各有$O(RC)$种选择。如果我们枚举了起点和终点，就已经做了$O(R^2C^2)$次枚举，再乘上搜索需要的时间，根本无法通过。

我们刚刚的想法是枚举起点和终点，再从路径的两端向中间搜索。我们发现，实际上“中间点”的选择也是$O(RC)$种，我们为什么不枚举中间点，从中间向两端搜索呢？

于是我们枚举路径的中间点Mid，也就是第11个格子。从Mid出发搜索10步，可以得到一条包含11个格子的路径P，设$Num(P)$为不包括Mid的10个格子里数值的集合。设Mid中的数值为X。那么假如存在两条路径P_1和P_2满足$Num(P_1) \cup Num(P_2) \cup \{X\} = \{0, 1, \ldots, 20\}$，那么将$P_1$和$P_2$拼接起来一定是一条满足要求的路径。注意到这里因为$0 \sim 20$各出现了一次，所以我们可以肯定$P_1$和$P_2$只在$Mid$处相交。我们和上一题一样使用一个二进制数表示集合$Num(P)$，就可以快速地判断$Num(P_1) \cup Num(P_2) \cup \{X\}$是否等于$\{0, 1, \ldots, 20\}$了。

我们估计一下上述算法的运算量。首先枚举的Mid数目为$O(RC)$，其次由于搜索10步，每步可以向四个方向走，所以路径数为4^{10}。实际上这个估计可以更加精确一些，从第二步搜索开始，“往回走”显然是不合法的，因此第二步开始可能的方向只有三个，从而实际的路径数不会超过4×3^9。对于每一条路径，进行散列表的操作和比较都只需要常数时间。综上，我们得到了一个时间复杂度为$O(RC \times 4 \times 3^9)$的算法。尽管$RC \times 4 \times 3^9$可能达到$21^2 \times 4 \times 3^9 = 34720812$，但是考虑到路径不能超出矩阵的边界或者到达没有数字的格子，并且如果路径中某个数字出现了两次可以直接剪枝，实际的搜索量会远小于以上估计。在实际测试中，按照上述算法实现的程序通过了所有测试数据。

例题5表明，在使用meet in the middle技巧时，如果起点和终点未知，并且中间点的数目较少，我们不妨尝试将“在中间相遇”变为“从中间出发”。

3.7 例题6：FencingGarden[30]

3.7.1 题目大意

有N根木棍，长度分别为$L_1, L_2, \ldots, L_N$。你可以选择先将某一根木棍分成两段，长度不一定为整数。从得到的木棍中选出一部分，靠着一堵无限长的墙围一个矩形区域。求当矩形区域的面积达到最大值时，矩形与墙平行的边的长度。

3.7.2 数据规模

$N \leq 40, L_i \leq 10^8$

3.7.3 分析

这道题与之前的所有题目都不同，并不是求方案数，而是求一个最优解。

这道题的一个难点在于切割木棍的方案非常多，枚举这个方案显然不现实。

由于我们把木棍分成了两段，但是围成的矩形有三条边，所以必然有一条边完全是由完整的木棍拼成的。这样一条边有两种可能：与墙平行或者与墙垂直。记这条边的长度为Len。

我们考虑这条边与墙平行的情况，记$S = \sum_{i=1}^{N} L_i$。那么矩形剩下的两条边长度必须相等。我们把仍未使用的木棍排列起来，它们的总长为$S - Len$。在$\frac{S-Len}{2}$处切下一刀，如果这里恰好是两根木棍的连接处，那么我们并没有改变木棍的数量，如果这里是某根木棍中间的某个点，那么我们就把这根木棍切成两段。无论如何，我们总能把剩下的木棍分成长度均为$\frac{S-Len}{2}$的两部分，从而得到一个面积为$Len\frac{S-Len}{2}$的矩形。

同理可得，如果一条长度为Len且与墙垂直的边完全由完整的木棍拼成，那么我们可以得到一个面积为$Len(S - 2Len)$的矩形。

由二次函数的性质可知，第一种情况下$Len = \frac{S}{2}$最优，第二种情况下$Len = \frac{S}{4}$最优。进而我们需要求出，由完整的木棍拼出的长度中，最接近$\frac{S}{2}$和$\frac{S}{4}$的长度分别是多少。

[30]**题目来源：** TopCoder Single Round Match 461 - Division I, Level Three

与之前的题目相同，我们仍然把N根木棍分为两部分，分别有M根和$N-M$根木棍。我们先搜索M根木棍，将得到的长度记录下来。之后我们搜索另外$N-M$根，假设得到了一个长度为T的部分，那么需要在之前搜索的长度中找到一个x，使得$x+T$尽量接近$\frac{S}{2}$或$\frac{S}{4}$，也就是要使x尽量接近$\frac{S}{2}-T$或$\frac{S}{4}-T$。一个较好的解决方式是：先将第一部分中得到的$O(2^M)$个长度排序，第二部分查询时使用两次二分查找。

将$O(2^M)$个数排序的时间复杂度为$O(2^M M)$，在其中进行$O(2^{N-M})$次二分查找的时间复杂度为$O(2^{N-M}M)$。如果我们取$M=\lfloor\frac{N}{2}\rfloor$，那么算法的时间复杂度为$O(2^{\frac{N}{2}}N)$。

例题6表明，运用meet in the middle技巧不仅可以解决计数类问题，还可以结合二分查找等方法，处理一些特殊的最优性问题。

4 总结

从之前的介绍可以看出，meet in the middle是一种十分有效的搜索优化技巧。它的核心思想是将问题分成两部分分别搜索，通过快速地合并两部分搜索结果达到优化算法的目的。虽然我们并不能因此得到一个多项式时间复杂度的算法，但是往往可以使时间复杂度的指数减小一半，在一些问题中这就能使得我们在时限内求出结果。

该技巧的优点在于算法容易实现，算法使用的搜索过程和朴素算法基本相同，仅需要使用一些较为简单的数据结构——例如散列表。该技巧的不足在于使用范围并不是很广，并且优化后的时间复杂度仍然是指数级别，能够有效解决的数据的规模并不能大幅增加。同时，算法必须消耗大量的空间。

有向图模型和方程模型概括了meet in the middle技巧能够解决的问题的普遍特征，使用这两个模型可以解决一类问题。另一方面，仍然有题目不能直接转化为上述模型，需要我们对meet in the middle技巧进行修改。由此看出，同一个技巧在不同题目中的使用方法也是多种多样的。我们需要根据情况灵活地使用meet in the middle 技巧，从而达到优化算法的目的。

5 感谢

感谢亲人和朋友对我的鼓励与支持!

感谢李曙、吴效时、史钋镭等老师对我的指导!
感谢贾志鹏、许昊然等众多学习信息竞赛的同学对我的帮助和启发!

参考文献

[1] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein, Introduction to Algorithms.

[2] vexorian, TopCoder Algorithm Problem Set Analysis.

[3] Bruce Merry, USACO 2012 US Open Gold Division, Balanced Cow Subsets Solution.

[4] Zhipeng Jia, TopCoder SRM 450∼499 Solution.