第 1 次 算法分析

1.4.1

证明：

(1)当 $N=3$ 时，$f(3)=\binom{3}{3}=1=\frac{3(3-1)(3-2)}{6}$

(2)设当 $N=k$ 时，$f(k)=\binom{k}{3}=\frac{k(k-1)(k-2)}{6}$，则当 $N=k+1$ 时，

$$f(k+1)=\binom{k+1}{3}=\binom{k}{3}+\binom{k}{2}=\frac{k(k-1)(k-2)}{6}+\frac{k(k-1)}{2}$$

$$=\frac{k(k-1)(k+1)}{6}=\frac{(k+1)(k+1-1)(k+1-2)}{6}$$

(3)由(1)和(2)可得 $f(N)=\binom{N}{3}=\frac{N(N-1)(N-2)}{6}$

1.4.2

```
public class ThreeSum
{
    public static int count(int[] a)
    {
        int N = a.length;
        int cnt = 0;
        for(int i=0;i<N;i++)
            for(int j=i+1;j<N;j++)
                for(int k=j+1;k<N;k++)
                {
                    min=min(a[i],a[j],a[k]);
                    mid=mid(a[i],a[j],a[k]);
                    max=max(a[i],a[j],a[k]);
                    if(mid>0 && min+mid==-max)
                        cnt++;
                    else if(mid<0 && max+mid==-min)
                        cnt++;
                    else if(mid==0 && max=-min)
                        cnt++;
                }
}
```

备注：学生使用 float 或乘以 0.1 都属于强转。

1.4.4

```
public class TwoSum
{
    public static int count(int[] a)
    {
        int N = a.length;
        int cnt = 0;
        for(int i=0;i<N;i++)
            for(int j=i+1;j<N;j++)
                if(a[i]+a[j]==0)
                    cnt++;
        return cnt;
    }
}
```

语句块	运行时间	频率	总时间
D	t_0	x (取决于输入)	$t_0 x$
C	t_1	$N^2/2 - N/2$	$t_1\,(N^2/2 - N/2)$
B	t_2	N	$t_2 N$
A	t_3	1	t_3
		总时间	$\frac{t_1}{2}N^2 + (t_2 - \frac{t_1}{2})N + t_3 + t_0 x$
		近似	$\sim\frac{t_1}{2}N^2$ (假设 x 很小)
		增长的数量级	N^2

1.4.5

a. $N+1 \sim N$

b. $1 + 1/N \sim 1$

c. $(1 + 1/N)(1 + 2/N) \sim 1$

d. $2N^3 - 15N^2 + N \sim 2N^3$

e. $\lg(2N)/\lg(N) \sim 1$

f. $\lg(N^2+1)/\lg(N) \sim 2$

g. $N^{100}/2^N \sim 1/2^N$

1.4.6

a. $N(1 + 1/2 + 1/4 + 1/8 + \ldots + 1/N)$

$= N(2 - 1/N) = 2N - 1 \sim 2N$

b. $(1 + 2 + 4 + 8 + \ldots + N) = 2N - 1 \sim 2N$

c. $N\lg N$

or

$(N + N/2 + N/4 + \ldots + 1) \sim 2N$, 因为是公比为 1/2 的等比级数，共有 $\log N + 1$ 项，带入公式可得；

$(1 + 2 + 4 + 8 + \ldots + N) \sim 2N$, 原因类似，同上；

linearithmic (the outer loop loops $\lg N$ times).

1.4.7

如图 1.4.4 所示，设加法的成本为 t_0，比较的成本为 t_1。

则在 E 语句块中的成本为 t_0x，x 取决于输入。

在 D 语句块中，每次循环共有两次比较，三次加法，

所以总的成本为$(2t_1 + 3t_0)(N^3/6 - N^2/2 + N/3)$。

在 C 语句块中共有 2 次加法，1 次比较，时间成本为$(t_1 + 2t_0)(N^2/2 - N/2)$。

在 B 语句块中共有 2 次加法，1 次比较$(t_1 + 2t_0)N$。

类似的在 A 语句块总共成本为 t_2(求 a.length 的成本)。

所以，总的成本为四者累加为:

$(2t_1 + 3t_0)(N^3/6 - N^2/2 + N/3) + (t_1 + 2t_0)(N^2/2 - N/2) + (t_1 + 2t_0)N + t_0x + t_2$

令 $t_0 = 1$，$t_1 = 1$，渐近时间复杂度为 $5N^3/6$。

1.4.8

```
public class ThreeSum
{
    public static int count(int[] a)
    {
        int N = a.length();
        int cnt = 0;
        int tali = a[0];
        int subLength=1;
        arrays.sort(a);
        for(int i=1;i<N;++i)
        {
            if(a[i]==tail)
            {
                subLength++;
            }
            else
            {
```

次、合并查找。

```
                if(subLength>1)
                    cnt = cnt + subLength*(subLength-1)/2;
                subLength=1;
                tail = a[i];
            }
        }
    }
}
```

1.4.15

用两个指针分别指向数组的起始位置和末尾位置，求二者之和，小于 0，前指针++，大于 0 后指针--，等于 0，输出，二指针同时移动。指针指向同一位置时结束。

```
public class TwoSumFaster()
{
    public static int count(int[] a)
    {
        arrays.sort(a);
        int head=0;
        int tail=a.length()-1;
        cnt=0;
        while(head<tail)
        {
            int sum=a[head]+a[tail];
            if(sum>0)
                --tail;
            else if(sum<0)
                ++head;
            else
            {
                ++head;
                --tail;
                ++cnt;
            }
        }
        return cnt;
    }
}
```

利用类似上述方法可得到 N^2 级的 ThreeSum 算法。

注：本题目中假设数值都不相等。

1.5.1

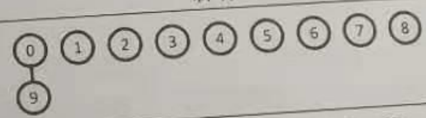
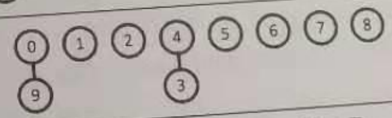
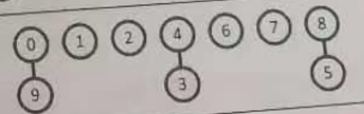
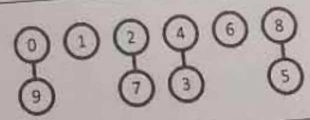
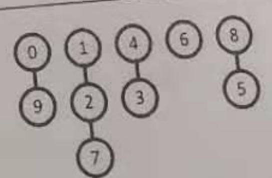
数对(p-q)	id 数组内容	数组访问次数
9-0	0 1 2 3 4 5 6 7 8 0	15
3-4	0 1 2 4 4 5 6 7 8 0	15
5-8	0 1 2 4 4 8 6 7 8 0	15
7-2	0 1 2 4 4 8 6 2 8 0	15
2-1	0 1 1 4 4 8 6 1 8 0	16
5-7	0 1 1 4 4 1 6 1 1 0	16
0-3	4 1 1 4 4 1 6 1 1 4	16
4-2	1 1 1 1 1 1 6 1 1 1	18

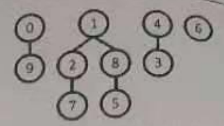
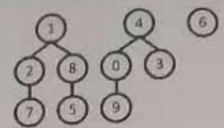
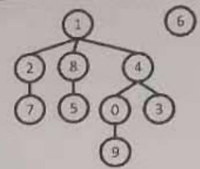
备注：

(1) 红色表示 id 数组改变的位置。

(2) 数组访问次数的计算方法如下。处理每一个数对(p-q)时，调用 connected(p, q)需要 2 次读操作，如果 p 和 q 当前没有连接(本题中每一对输入均是未连接的)，需要调用 union(p, q)；union(p, q)执行时，获取 p 和 q 的 id 需要 2 次读操作，for 循环需要 10 次读操作并有 x 次写操作(x 对应于数组中改变的位置个数)。因此，处理一个数对(p-q)的数组访问次数为：2+2+10+x。

1.5.2

数对(p-q)	id 数组内容	数组访问次数	森林
9-0	0 1 2 3 4 5 6 7 8 0	5	
3-4	0 1 2 4 4 5 6 7 8 0	5	
5-8	0 1 2 4 4 8 6 7 8 0	5	
7-2	0 1 2 4 4 8 6 2 8 0	5	
2-1	0 1 1 4 4 8 6 2 8 0	5	

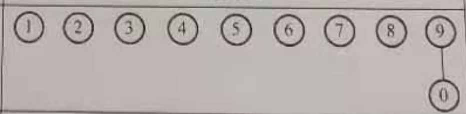
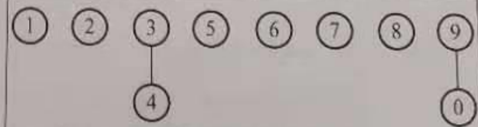
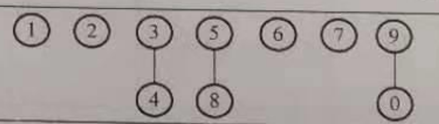
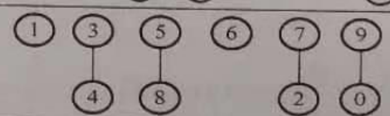
5-7	0 1 1 4 4 8 6 2 1 0	11	
0-3	4 1 1 4 4 8 6 2 1 0	7	
4-2	4 1 1 4 1 8 6 2 1 0	7	

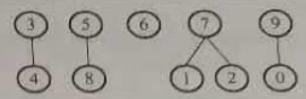
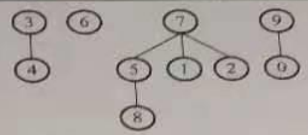
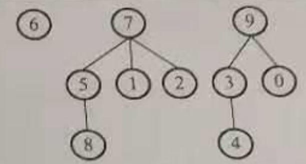
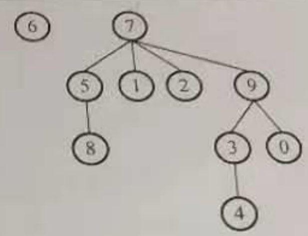
备注：

(1) 红色表示 id 数组改变的位置。

(2) 数组访问次数的计算方法如下。处理每一个数对(p-q)时，首先需要调用 connected(p, q)，执行时需要调用一次 root(p)和一次 root(q)，如果 p 和 q 当前没有连接(本题中每一对输入均是未连接的)，需要调用 union(p, q)；union(p, q)执行时，也需要调用一次 root(p)和一次 root(q)，同时执行一次写操作。执行一次 root(*node*)需要访问数组的次数为 *node* 的树深。因此，处理一个数对(p-q)的数组访问次数为：$2(depth_p + depth_q) + 1$，其中 $depth_{node}$ 表示 *node* 的树深。

1.5.3

数对(p-q)	id 数组内容	数组访问次数	森林
9-0	9 1 2 3 4 5 6 7 8 9	8	
3-4	9 1 2 3 3 5 6 7 8 9	8	
5-8	9 1 2 3 3 5 6 7 5 9	8	
7-2	9 1 7 3 3 5 6 7 5 9	8	

2-1	9773356759	10	
5-7	9773376759	8	
0-3	9779376759	10	
4-2	9779376757	14	

备注:

(1) 红色表示 id 数组改变的位置。

(2) 数组访问次数的计算方法如下。处理每一个数对(p-q)时，首先需要调用 connected(p, q)，执行时需要调用一次 root(p)和一次 root(q)，如果 p 和 q 当前没有连接(本题中每一对输入均是未连接的)，需要调用 union(p, q)；union(p, q)执行时，也需要调用一次 root(p)和一次 root(q)，同时执行 2 次 sz[]数组的读操作、1 次 id[]数组的写操作和 1 次 sz[]数组的写操作。执行一次 root(*node*)需要访问数组的次数为 *node* 的树深。因此，处理一个数对(p-q)的数组访问次数为：$2(depth_p + depth_q) + 4$，其中 $depth_{node}$ 表示 *node* 的树深。

(3) 注意：(2)给出的是计算所有数组访问次数的方法。根据题意，有些同学也可能会理解成只计算 id[]数组的访问次数，它等于(2)中的结果减去 3。

(4) 注意：这里给出森林供参考，题目并没有要求。

1.5.8

Answer. The value of id[p] changes to id[q] in the for loop. Thus, any object r > p with id[r] equal to id[p] will not be updated to equal id[q].

答：在 for 循环中，id[p]将会被赋值为 id[q]。这样，对于任意的满足 r > p 且 id[r]=id[p]的 id[r]将不会被更新为 id[q]。

1.5.10

Answer. Yes. However, it would be increase the tree height, so the performance guarantee would invalid.

答：是，但这会增加树的高度，因此无法保证同样的性能。

第 2 章 排序

2.2.2 top-down

lo	m	hi	0	1	2	3	4	5	6	7	8	9	10	11
			E	A	S	Y	Q	U	E	S	T	I	O	N
0	0	1	A	E										
0	1	2	A	E	S									
3	3	4				Q	Y							
3	4	5				Q	U	Y						
0	2	5	A	E	Q	S	U	Y						
6	6	7							E	S				
6	7	8							E	S	T			
9	9	10										I	O	
9	10	11										I	N	O
6	8	11							E	I	N	O	S	T
0	5	11	A	E	E	I	N	O	Q	S	S	T	U	Y
			A	E	E	I	N	O	Q	S	S	T	U	Y

2.2.3 bottom-up

lo	m	hi	0	1	2	3	4	5	6	7	8	9	10	11
			E	A	S	Y	Q	U	E	S	T	I	O	N
0	0	1	A	E										
2	2	3			S	Y								
4	4	5					Q	U						
6	6	7							E	S				
8	8	9									I	T		
10	10	11											N	O
0	1	3	A	E	S	Y								
4	5	7					E	Q	S	U				
8	9	11									I	N	O	T
0	3	7	A	E	E	Q	S	S	U	Y				
0	7	11	A	E	E	I	N	O	Q	S	S	T	U	Y
			A	E	E	I	N	O	Q	S	S	T	U	Y

2.2.4

是的。如果输入的两个子数组都是有序的，那么原地归并排序将产生正确的输出。

如果一个子数组不是有序的，那么不能产生正确的输出，因为这个子数组中的元素在归并输出的中维持原有的次序。

比如：数组 A：2,1,3,4

数组 B：1,2,3,4

原地归并的结果是：1,2,2,1,3,3,4，明显是错的。

2.2.5

自顶向下的归并排序：

2, 3, 2, 5, 2, 3, 2, 5, 10, 2, 3, 2, 5, 2, 3, 2, 5, 10, 20, 2, 3, 2, 5, 2, 3, 2, 5, 10, 2, 3, 2, 5, 2, 2, 4, 9, 19, 39.

自底向上的归并排序：

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 8, 8, 8, 8, 7, 16, 16, 32, 39.

2.3.1

此为更新后的答案，原答案欠缺细致。

	i	j	0	1	2	3	4	5	6	7	8	9	10	11
初始值	0	12	E	A	S	Y	Q	U	E	S	T	I	O	N
扫描左、右部分	2	6	E	A	S	Y	Q	U	E	S	T	I	O	N
交换	2	6	E	A	E	Y	Q	U	S	S	T	I	O	N
扫描左、右部分	3	2	E	A	E	Y	Q	U	S	S	T	I	O	N
最后一次交换	3	2	E	A	E	Y	Q	U	S	S	T	I	O	N
结果		2	E	A	E	Y	Q	U	S	S	T	I	O	N

划分元素V

2.3.2

lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11
			E	A	S	Y	Q	U	E	S	T	I	O	N
			Y	A	T	N	S	S	E	Q	O	I	U	E
0	11	11	E	A	T	N	S	S	E	Q	O	I	U	Y
0	2	10	E	A	E	N	S	S	T	Q	O	I	U	Y
0	1	1	A	E	E	N	S	S	T	Q	O	I	U	Y
0		0	A	E	E	N	S	S	T	Q	O	I	U	Y
3	4	10	A	E	E	I	N	S	T	Q	O	S	U	Y
3		3	A	E	E	I	N	S	T	Q	O	S	U	Y
5	8	10	A	E	E	I	N	O	S	Q	S	T	U	Y
5	5	7	A	E	E	I	N	O	S	Q	S	T	U	Y
6	7	7	A	E	E	I	N	O	Q	S	S	T	U	Y
6		6	A	E	E	I	N	O	Q	S	S	T	U	Y
9	9	10	A	E	E	I	N	O	Q	S	S	T	U	Y
10		10	A	E	E	I	N	O	Q	S	S	T	U	Y
			A	E	E	I	N	O	Q	S	S	T	U	Y

2.3.3

$\lg N$ 次

2.3.4

给出任意六个满足如下条件的数组：(1) 含有 10 个元素；(2) 升序排列。

2.3.5

思路：对输入的数组执行一次快速排序的划分操作，这样就完成了只含两个键值的数组的排序。

参考代码如下：

```
private static int twoKeySort(Comparable []a)
{
    int i = 0, j = a.length+1;
    comparable v = a[0];
    while(i<j)
    {
        while(i<j && less(a[++i], v));
        while(i<j && less(v, a[--j]));
        exch(a[i],a[j]);
}
```

备注：当数组中元素都相同时，每次划分将数组均分成两部分。

2.3.9

参考解答：

标准的快速排序在处理只有两种主键值的数组时，执行完第一次划分即已经将整个数组排定，后边的所有操作都是不必要的，整体的时间是线性对数级别的；在处理只有三种主键值的数组时，执行完第一次划分后，再分别对每个子数组至多执行一次划分即已经将整个数组排定，后边的所有操作都是不必要的，整体的时间是线性对数级别的。

利用三向划分的快速排序可以高效地处理含有大量重复元素的数组，可以将排序时间从线性对数级降低到线性级。

2.4.1

R R P O T Y I I U Q E U

2.4.2

Will need to update the maximum value from scratch after a *remove-the-maximum* operation.

2.4.3

方法	无序数组	有序数组	无序链表	有序链表
插入元素	1	N	1	N
删除元素	N	1	N	1

2.4.4

是。

2.4.5

将 EASYQUESTION 顺序插入一个面向最大元素的堆，数组变化过程：

插入的元素	插入该元素后的数组
E	E
A	E A
S	S A E
Y	Y S E A
Q	Y S E A Q
U	Y S U A Q E
E	Y S U A Q E E
S	Y S U S Q E E A
T	Y T U S Q E E A S
I	Y T U S Q E E A S I
O	Y T U S Q E E A S I O
N	Y T U S Q N E A S I O E

2.4.6

操作	堆的内容
插入 P	P

插入 R	R P
插入 I	R P I
插入 O	R P I O
删除最大元素	P O I
插入 R	R P I O
删除最大元素	P O I
删除最大元素	O I
插入 I	O I I
删除最大元素	I I
插入 T	T I I
删除最大元素	I I
插入 Y	Y I I
删除最大元素	I I
删除最大元素	I
删除最大元素	
插入 Q	Q
插入 U	U Q
插入 E	U Q E
删除最大元素	Q E
删除最大元素	E
删除最大元素	
插入 U	U
删除最大元素	
插入 E	E

2.4.7

k	可能出现位置	不可能出现位置
2	[2, 3]	[1, 1], [4, 31]
3	[2, 7]	[1, 1], [8, 31]
4	[2, 15]	[1, 1], [16, 31]

2.4.8

k	可能出现位置	不可能出现位置
2	[16, 31]	[1, 15]
3	[8, 31]	[1, 7]
4	[8, 31]	[1, 7]

2.4.9

A B C D E 五个元素可能构造出来的所有堆：

E D A B C

E D A C B

E D C B A

E D C A B

E D B A C
E D B C A
E C D A B
E C D B A
A A A B B 五个元素可能构造出来的所有堆：
B A B A A
B B A A A

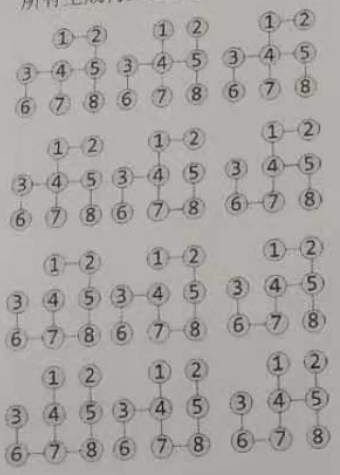
2.4.10
pq[k]的父结点是 pq[(k+1)/2 − 1]，子结点是 pq[2k+1]和 pq[2k+2]（如果有的话）。

2.4.11
无序数组。插入元素的操作是常量时间。

2.4.12
有序数组。找出最大元素的操作是常量时间。

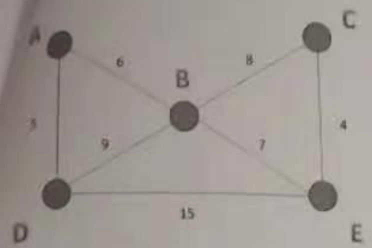
第 4 章 Graphs

4.3.2
所有生成树如图所示



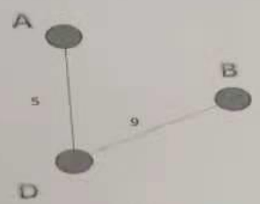
4.3.3
For the sake of contradiction, suppose there are two different MSTs of G, say T1 and T2. Let e = v-w be the min weight edge of G that is in one of T1 or T2, but not both. Let's suppose e is in T1. Adding e to T2 creates a cycle C. There is at least one edge, say f, in C that is not in T1 (otherwise T1 would be cyclic). By our choice of e, w(e) ≤ w(f). Since all of the edge weights are distinct, w(e) < w(f). Now, replacing f with e in T2 yields a new spanning tree with weight less than that of T2 (contradicting the minimality of T2).

4.3.13
举例说明问题即可。

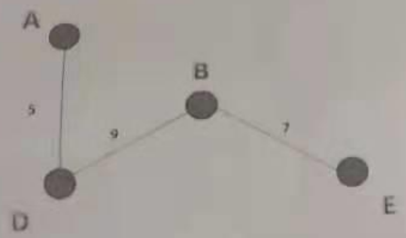


(1) A 为任意顶点，先找到 A-D;

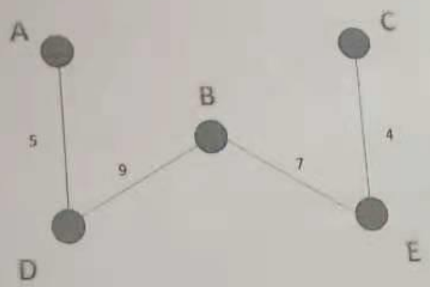
(2) 再以 D 为顶点找到 D-B;



(3) 再以 B 为顶点找到 B-E;

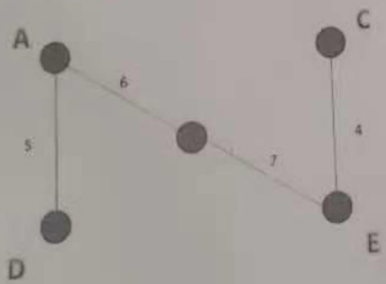


(4) 再以 E 为顶点找到 E-C;



结果：这样总权重为 5+9+7+4=25;

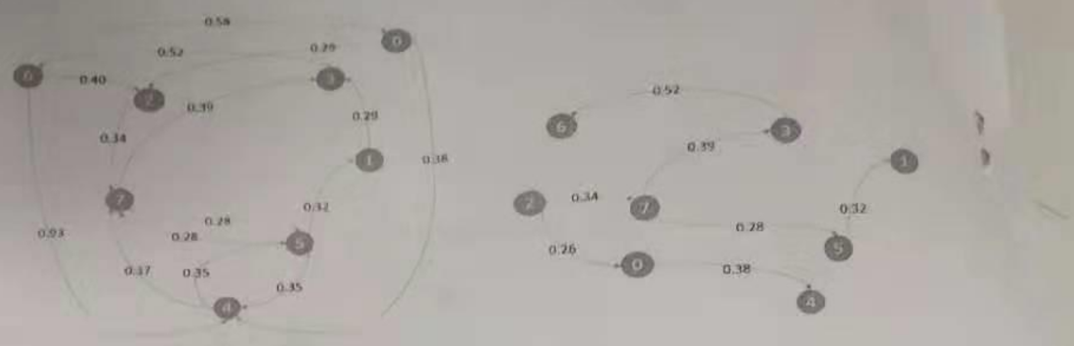
(5) 但是存在以下情况，总权重为 5+9+7+4=22<25;



4.4.1

假。与路径的长度有关。

4.4.5



最小生成树

4.4.9

	Providence	Westerly	New London	Norwich
Providence	-	53	54	48
Westerly	53	-	18	101
New London	54	18	-	12
Norwich	48	101	12	-

Norwich 到 Westerly 的最短路径为：

Norwich→ New London→ Westerly，路程为：12+18=30

其他路径不用绕道，直达就是最短路径。