The Gracefulness of Graph $B(m, n)$

GUO WENFU, Translation, comments, and code by Pascal Welke

Abstract. In this paper, we shall prove that the graphs $B(m, n) = C_m \cup P_n$ are **graceful** when $m \equiv 1$ or $2 \pmod 4$, where $C_m = A_1A_2 \dots A_mA_1$ and $P_n = A_1B_1B_2 \dots B_n$ $(m \geq 3, n > 0)$.

Keywords: cycle, path, tadpole, graceful graph

I have obtained the original Chinese version [1] of this article with the help of collegues and translated it with the help of Google Gemini as of December 2025. I have checked the proof sketches in the translation carefully and have implemented all constructions in Python. All functions are named after the respective parts of the original paper and construct a sequence of vertices that are injectively labeled with numbers from 1 to $m + n$ and a sequence of edges. As the vertex labeling is injective, I used the labels as node ids. The code is provided next to the descriptions of the original paper and is available on github.

1 INTRODUCTION

Let the **cycle** be $C_m = A_1A_2 \dots A_mA_1$ and the **path** be $P_n = A_1B_1B_2 \dots B_n$. We define the graph $B(m, n) = C_m \cup P_n$ $(m \geq 3, n > 0)$.

It has been proven in [2] that $B(m, n)$ is a 1-graceful graph. It was also proven that $B(m, n)$ is a graceful graph when $m \equiv 0$ or $3 \pmod 4$. This paper gives an affirmative answer to the question: "Is $B(m, n)$ a graceful graph when $m \equiv 1$ or $2 \pmod 4$ and $n > 0$?".

Definition 1 (Graceful Labeling). *For a simple graph $G(V, E)$ we assign non-negative integers $\varphi(v) \in \{1, 2, \dots, |E|\}$ to every vertex $v \in V$ which we call* vertex labels. *We call the value $L(u, v) = |\varphi(u) - \varphi(v)|$ the* edge label *of the edge uv. A simple graph G is called a* graceful graph *if it is labeled such that:*

(1) *Different vertices have different vertex labels.*
(2) *Different edges have different edge labels.*

The set of edge labels for a graceful graph is $\{1, 2, \dots, |E|\}$.

The entrypoint to the python code is the following function that constructs a graceful labeling for any tadpole with $m \equiv 1$ or $2 \pmod 4$.

```
def graceful_tadpole(m, n):
    '''
    Implements all the constructions presented in the paper
    Guo Wenfu. The Gracefulness of Graph B(m,n).
    Journal of Inner Mongolia Normal University (Natural Science Edition),
    1994(2): 25-29.
    '''
    assert(m >= 3)
    assert(m % 4 == 2 or m % 4 == 1)

    if m % 4 == 2:
        return thm1(m, n)
    if m % 4 == 1:
        return thm2(m, n)
```

Author's address: Guo Wenfu, Translation, comments, and code by Pascal Welke, Lancaster University Leipzig, Germany.

2 CASE $m \equiv 2 \pmod 4$

THEOREM 1. *For all $m \equiv 2 \pmod 4$ and $n > 0$, $B(m, n)$ is a **graceful graph**.*

PROOF. Let $m = 4l + 2$, which implies $l = \frac{m-2}{4}$. A graceful labeling for $n = 1$ and arbitrary $l > 0$ is given in Figure 1 with comments and python code in Section 2.1.Similarly, a graceful labeling for $n = 2$ and arbitrary $l > 0$ is given in Figure 2 with comments and python code in Section 2.2. For the case $n \geq 3$ we will construct graceful labelings in Section 2.3. □

```python
def thm1(m, n):
    assert(m >= 3)
    assert(m % 4 == 2)
    assert(n > 0)

    if n == 1:
        return fig1(m, n)
    if n == 2:
        return fig2(m, n)
    else:
        if n % 6 == 3:
            return case_3_1(m, n, fig1)
        if n % 6 == 4:
            return case_3_2(m, n, fig1)
        if n % 6 == 5:
            return case_3_3(m, n, fig1)
        if n % 6 == 0:
            return case_3_4(m, n, fig1)
        if n % 6 == 1:
            return case_3_5(m, n, fig1)
        if n % 6 == 2:
            return case_3_6(m, n, fig1)
```

2.1 Case 1: $n = 1$

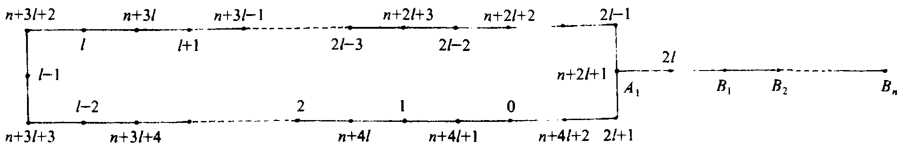

Fig. 1. Labeling for $B(m, 1)$ when $m \equiv 2 \pmod 4$

```python
def fig1(m, n):
    assert(m % 4 == 2)
    # careful. n >= 3 requires additional tail nodes
    # added in another step
    assert(n == 1 or n >= 3)

    l = m // 4

    vertices = list()
```

```
    # point A1 will be at index 0, followed by m-1 cycle nodes, clockwise.
    vertices.append(n + 2*l + 1)
    vertices.append(2*l + 1)

    for i in range(l+1):
        vertices.append(n + 4*l + 2 - i)
        vertices.append(i)

    for i in range(l+1,2*l):
        vertices.append(n + 4*l + 1 - i) # after l, we shift this by one.
        vertices.append(i)

    edges = list(zip(vertices, vertices[1:] + [vertices[0]]))

    # Then 1 path node
    vertices.append(2*l)

    # and the edge connecting it to the cycle
    edges.append((vertices[0], vertices[m])) # edge connecting path and cycle

    return vertices, edges
```

2.2 Case 2: $n = 2$

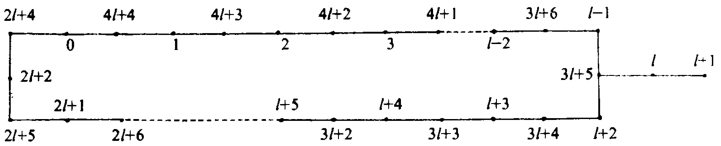

Fig. 2. Labeling for $B(m, 2)$ when $m \equiv 2 \pmod 4$

```
def fig2(m, n):
    assert(m % 4 == 2)
    assert(n == 2)

    l = m // 4
    vertices = list()

    # point A1 will be at index 0, followed by m-1 cycle nodes, counterclockwise.
    vertices.append(3*l+5)

    # we create increasing edge weights [2l+6, ..., m+n] by alternatingly
    # increasing and decreasing the vertex label.
    # the last node constructed by this loop is the vertex with label 0
    startlength_counterclockwise = 2*l + 6 #3*l + 5 - (l - 1)
    decrease = True
    for w in range(startlength_counterclockwise, m+n+1):
        current = vertices[-1]
        if decrease:
```

```
            vertices.append(current - w)
            decrease = False
        else:
            vertices.append(current + w)
            decrease = True

    vertices.append(2*l + 4)

    # we create increasing edge weights [2, ..., 2l+2] by alternatingly
    # increasing and decreasing the vertex label.
    # the last node constructed by this loop is the vertex with label l+2
    decrease = True
    for w in range(2, 2*l+3):
        current = vertices[-1]
        if decrease:
            vertices.append(current - w)
            decrease = False
        else:
            vertices.append(current + w)
            decrease = True

    # create cycle edges
    edges = list(zip(vertices, vertices[1:] + [vertices[0]]))

    # Then 2 path nodes
    vertices.append(l)
    vertices.append(l+1)

    # create tail edges
    edges.append((vertices[0], vertices[m])) # edge connecting path and cycle
    edges.append((vertices[m], vertices[m+1])) # path edge

    return vertices, edges
```

2.3 Case 3: $n \geq 3$

For the case $n \geq 3$ we will construct graceful labelings as follows:

- First, $C_m \cup \{B_1\}$ is labeled according to Figure 1.
- The vertex labels for $C_m \cup \{B_1\}$ are distinct and their set is $\{0, 1, \ldots, 2l+1\} \cup \{n+2l+1, n+2l+2, \ldots, n+3l\} \cup \{n+3l+2, \ldots, n+4l+1, n+4l+2\}$.
- The edge labels are distinct and their set is $\{n, n+1, \ldots, n+4l+1, n+4l+2\}$.
- The remaining path is $B_1B_2 \ldots B_n$, and the available edge label set is $\{1, 2, \ldots, n-1\}$.
- The labeling for the path $B_1B_2 \ldots B_n$ is then given across six cases based on $n \pmod 6$.

Labeling for Path $B_1B_2 \ldots B_n$ (for $n \geq 3$)

Case (3-1) Let $n = 6s+3$, $a = \varphi(A_1) = n+2l+1$, and $b = \varphi(B_1) = 2l$. The general assignments for $0 \le i \le s-1$ are:

$$\begin{aligned}
&\varphi(B_{6i+1}) = b + 3i && 0 \le i \le s-1\\
&\varphi(B_{6i+2}) = a - 2 - 3i && 0 \le i \le s-1\\
&\varphi(B_{6i+3}) = b + 2 + 3i && 0 \le i \le s-1\\
&\varphi(B_{6i+4}) = a - 1 - 3i && 0 \le i \le s-1\\
&\varphi(B_{6i+5}) = b + 4 + 3i && 0 \le i \le s-1\\
&\varphi(B_{6i+6}) = a - 3 - 3i && 0 \le i \le s-1\\
&\varphi(B_{6s+1}) = 2l + 3s\\
&\varphi(B_{6s+2}) = 2l + 2 + 3s\\
&\varphi(B_{6s+3}) = 2l + 3 + 3s
\end{aligned} \tag{1}$$

It is easy to see that the labels are all distinct, and they consist of all integers from $2l$ to $2l+n$ with $2l+1$ removed, and the label of B_1 is still $2l$. Comparing with the set of labels on $C_m \cup A_1 \cup B_1$, we can conclude that the labels on $C_m \cup P_m$ are also all distinct, and do not exceed $|E| = n+m = n+4l+2$.

Calculating the number of edge labels, we have:

$$\begin{aligned}
&L(B_{6i+1}B_{6i+2}) = n - 1 - 6i\\
&L(B_{6i+2}B_{6i+3}) = n - 3 - 6i\\
&L(B_{6i+3}B_{6i+4}) = n - 2 - 6i\\
&L(B_{6i+4}B_{6i+5}) = n - 4 - 6i\\
&L(B_{6i+5}B_{6i+6}) = n - 6 - 6i\\
&L(B_{6i+6}B_{6i+7}) = n - 5 - 6i
\end{aligned} \tag{2}$$

Here $0 \le i \le s-1$. In the above six formulas, letting $i = 0, 1, \ldots, s-1$ respectively, we see that the edge labels are all distinct and form the $n-3$ edge labels from 3 to $n-1$.

Moreover, since $L(B_{6s+1}B_{6s+2}) = 2$ and $L(B_{6s+2}B_{6s+3}) = 1$, it follows that the set of edge labels of $B_1, B_2, \ldots, B_n$ is $\{1, 2, \ldots, n-1\}$.

Considering the case on $C_m \cup P_n$ as a whole, it is easy to see that the labeling method we give is graceful.

The proof for the other cases below is similar; we only present the key steps and omit the rest.

```
def case_3_1(m, n, fig):
    assert(n % 6 == 3)

    head_vertices, head_edges = fig(m, n)

    tail_vertices = list()
    l = m // 4
    s = (n - 3) // 6
    a = n+2*l+1
    b = 2*l
    for i in range(s):
        tail_vertices.append(b+3*i)
        tail_vertices.append(a-2-3*i)
        tail_vertices.append(b+2+3*i)
```

```
        tail_vertices.append(a-1-3*i)
        tail_vertices.append(b+4+3*i)
        tail_vertices.append(a-3-3*i)
    tail_vertices.append(b+3*s)
    tail_vertices.append(2*l+2+3*s)
    tail_vertices.append(2*l+3+3*s)

    # construct tail edges
    tail_edges = [(u,v) for u,v in zip(tail_vertices, tail_vertices[1:])]

    # we are constructing edge labels for B_1, ... B_n, hence we drop B_1,
    # as it is already part of the output of fig1
    return head_vertices + tail_vertices[1:], head_edges + tail_edges
```

Case (3-2) When $n = 6s + 4$, the labels for B_1 to B_{6s+1} still use formula (1). Additionally, we set:

$\varphi(B_{6s+2}) = 2l + 3 + 3s$

$\varphi(B_{6s+3}) = 2l + 2 + 3s$

$\varphi(B_{6s+4}) = 2l + 4 + 3s$

The edge labels calculated by formula (2) still apply, but the set of edge labels is now $\{4, 5, \ldots, n-1\}$. Furthermore, because

$$L(B_{6s+1}B_{6s+2}) = 3, \quad L(B_{6s+2}B_{6s+3}) = 1, \quad L(B_{6s+3}B_{6s+4}) = 2,$$

Therefore, the set of edge labels for the path $B_1B_2 \ldots B_n$ is $\{1, 2, \ldots, n-1\}$.

```
def case_3_2(m, n, fig):
    assert(n % 6 == 4)

    head_vertices, head_edges = fig(m, n)

    tail_vertices = list()
    l = m // 4
    s = (n - 3) // 6
    a = n+2*l+1
    b = 2*l
    for i in range(s):
        tail_vertices.append(b+3*i)
        tail_vertices.append(a-2-3*i)
        tail_vertices.append(b+2+3*i)
        tail_vertices.append(a-1-3*i)
        tail_vertices.append(b+4+3*i)
        tail_vertices.append(a-3-3*i)
    tail_vertices.append(b+3*s)
    tail_vertices.append(2*l+3+3*s)
    tail_vertices.append(2*l+2+3*s)
    tail_vertices.append(2*l+4+3*s)

    # construct tail edges
    tail_edges = [(u,v) for u,v in zip(tail_vertices, tail_vertices[1:])]

    # we are constructing edge labels for B_1, ... B_n, hence we drop B_1,
```

```
    # as it is already part of the output of fig1
    return head_vertices + tail_vertices[1:], head_edges + tail_edges
```

Case (3-3) When $n = 6s + 5$, the labels for B_1 to B_{6s+1} still use formula (1). Additionally, we set:

$$\varphi(B_{6s+2}) = 2l + 4 + 3s$$
$$\varphi(B_{6s+3}) = 2l + 3 + 3s$$
$$\varphi(B_{6s+4}) = 2l + 5 + 3s$$
$$\varphi(B_{6s+5}) = 2l + 2 + 3s$$

The edge labels calculated by formula (2) still apply, but the set of edge labels is now $\{5, 6, \ldots, n-1\}$. Furthermore, due to

$$L(B_{6s+1}B_{6s+2}) = 4$$
$$L(B_{6s+2}B_{6s+3}) = 1$$
$$L(B_{6s+3}B_{6s+4}) = 2$$
$$L(B_{6s+4}B_{6s+5}) = 3$$

the set of edge labels for the path $B_1B_2 \ldots B_n$ is $\{1, 2, \ldots, n-1\}$.

```
def case_3_3(m, n, fig):
    assert(n % 6 == 5)

    head_vertices, head_edges = fig(m, n)

    tail_vertices = list()
    l = m // 4
    s = (n - 3) // 6
    a = n+2*l+1
    b = 2*l
    for i in range(s):
        tail_vertices.append(b+3*i)
        tail_vertices.append(a-2-3*i)
        tail_vertices.append(b+2+3*i)
        tail_vertices.append(a-1-3*i)
        tail_vertices.append(b+4+3*i)
        tail_vertices.append(a-3-3*i)
    tail_vertices.append(b+3*s)
    tail_vertices.append(2*l+4+3*s)
    tail_vertices.append(2*l+3+3*s)
    tail_vertices.append(2*l+5+3*s)
    tail_vertices.append(2*l+2+3*s)

    # construct tail edges
    tail_edges = [(u,v) for u,v in zip(tail_vertices, tail_vertices[1:])]

    # we are constructing edge labels for B_1, ... B_n, hence we drop B_1,
    # as it is already part of the output of fig1
    return head_vertices + tail_vertices[1:], head_edges + tail_edges
```

Case (3-4) When $n = 6s + 6$, the labels for B_1 to B_{6s+1} still use formula (1). Additionally, set:

$$\varphi(B_{6s+2}) = 2l + 5 + 3s$$
$$\varphi(B_{6s+3}) = 2l + 2 + 3s$$
$$\varphi(B_{6s+4}) = 2l + 6 + 3s$$
$$\varphi(B_{6s+5}) = 2l + 4 + 3s$$
$$\varphi(B_{6s+6}) = 2l + 3 + 3s$$

The edge labels calculated by formula (2) still apply, but the set of edge labels is now $\{6, 7, \ldots, n-1\}$. Furthermore, because

$$L(B_{6s+1}B_{6s+2}) = 5$$
$$L(B_{6s+2}B_{6s+3}) = 3$$
$$L(B_{6s+3}B_{6s+4}) = 4$$
$$L(B_{6s+4}B_{6s+5}) = 2$$
$$L(B_{6s+5}B_{6s+6}) = 1$$

Therefore, the set of edge labels for the path $B_1B_2 \ldots B_n$ is $\{1, 2, \ldots, n-1\}$.

```
def case_3_4(m, n, fig):
    assert(n % 6 == 0)

    head_vertices, head_edges = fig(m, n)

    tail_vertices = list()
    l = m // 4
    s = (n - 3) // 6
    a = n+2*l+1
    b = 2*l
    for i in range(s):
        tail_vertices.append(b+3*i)
        tail_vertices.append(a-2-3*i)
        tail_vertices.append(b+2+3*i)
        tail_vertices.append(a-1-3*i)
        tail_vertices.append(b+4+3*i)
        tail_vertices.append(a-3-3*i)
    tail_vertices.append(b+3*s)
    tail_vertices.append(2*l+5+3*s)
    tail_vertices.append(2*l+2+3*s)
    tail_vertices.append(2*l+6+3*s)
    tail_vertices.append(2*l+4+3*s)
    tail_vertices.append(2*l+3+3*s)

    # construct tail edges
    tail_edges = [(u,v) for u,v in zip(tail_vertices, tail_vertices[1:])]

    # we are constructing edge labels for B_1, ... B_n, hence we drop B_1,
    # as it is already part of the output of fig1
    return head_vertices + tail_vertices[1:], head_edges + tail_edges
```

Case (3-5) When $n = 6s + 7$, the labels for B_1 to B_{6s+1} still use formula (1). Additionally, set:

$\varphi(B_{6s+2}) = 2l + 6 + 3s$
$\varphi(B_{6s+3}) = 2l + 2 + 3s$
$\varphi(B_{6s+4}) = 2l + 7 + 3s$
$\varphi(B_{6s+5}) = 2l + 4 + 3s$
$\varphi(B_{6s+6}) = 2l + 5 + 3s$
$\varphi(B_{6s+7}) = 2l + 3 + 3s$

The edge labels calculated by formula (2) still apply, but the set of edge labels is now $\{7, 8, \ldots, n-1\}$. Furthermore, because

$L(B_{6s+1}B_{6s+2}) = 6$
$L(B_{6s+2}B_{6s+3}) = 4$
$L(B_{6s+3}B_{6s+4}) = 5$
$L(B_{6s+4}B_{6s+5}) = 3$
$L(B_{6s+5}B_{6s+6}) = 1$
$L(B_{6s+6}B_{6s+7}) = 2$

the set of edge labels for the path $B_1B_2 \ldots B_n$ is $\{1, 2, \ldots, n-1\}$.

```
def case_3_5(m, n, fig):
    assert(n % 6 == 1)

    head_vertices, head_edges = fig(m, n)

    tail_vertices = list()
    l = m // 4
    s = (n - 3) // 6
    a = n+2*l+1
    b = 2*l
    for i in range(s):
        tail_vertices.append(b+3*i)
        tail_vertices.append(a-2-3*i)
        tail_vertices.append(b+2+3*i)
        tail_vertices.append(a-1-3*i)
        tail_vertices.append(b+4+3*i)
        tail_vertices.append(a-3-3*i)
    tail_vertices.append(b+3*s)
    tail_vertices.append(2*l+6+3*s)
    tail_vertices.append(2*l+2+3*s)
    tail_vertices.append(2*l+7+3*s)
    tail_vertices.append(2*l+4+3*s)
    tail_vertices.append(2*l+5+3*s)
    tail_vertices.append(2*l+3+3*s)

    # construct tail edges
    tail_edges = [(u,v) for u,v in zip(tail_vertices, tail_vertices[1:])]

    # we are constructing edge labels for B_1, ... B_n, hence we drop B_1,
```

```
    # as it is already part of the output of fig1
    return head_vertices + tail_vertices[1:], head_edges + tail_edges
```

Case (3-6) When $n = 6s + 8$, the labels for B_1 to B_{6s+1} still use formula (1). Additionally, set:

$\varphi(B_{6s+2}) = 2l + 7 + 3s$

$\varphi(B_{6s+3}) = 2l + 2 + 3s$

$\varphi(B_{6s+4}) = 2l + 8 + 3s$

$\varphi(B_{6s+5}) = 2l + 4 + 3s$

$\varphi(B_{6s+6}) = 2l + 5 + 3s$

$\varphi(B_{6s+7}) = 2l + 3 + 3s$

$\varphi(B_{6s+8}) = 2l + 6 + 3s$

The edge labels calculated by formula (2) still apply, but the set of edge labels is now $\{8, 9, \cdots, n-1\}$. Furthermore, because

$L(B_{6s+1}B_{6s+2}) = 7$

$L(B_{6s+2}B_{6s+3}) = 5$

$L(B_{6s+3}B_{6s+4}) = 6$

$L(B_{6s+4}B_{6s+5}) = 4$

$L(B_{6s+5}B_{6s+6}) = 1$

$L(B_{6s+6}B_{6s+7}) = 2$

$L(B_{6s+7}B_{6s+8}) = 3$

Therefore, the set of edge labels for the path $B_1B_2 \cdots B_n$ is $\{1, 2, \cdots, n-1\}$.

```
def case_3_1(m, n, fig):
    assert(n % 6 == 3)

    head_vertices, head_edges = fig(m, n)

    tail_vertices = list()
    l = m // 4
    s = (n - 3) // 6
    a = n+2*l+1
    b = 2*l
    for i in range(s):
        tail_vertices.append(b+3*i)
        tail_vertices.append(a-2-3*i)
        tail_vertices.append(b+2+3*i)
        tail_vertices.append(a-1-3*i)
        tail_vertices.append(b+4+3*i)
        tail_vertices.append(a-3-3*i)
    tail_vertices.append(b+3*s)
    tail_vertices.append(2*l+2+3*s)
    tail_vertices.append(2*l+3+3*s)

    # construct tail edges
    tail_edges = [(u,v) for u,v in zip(tail_vertices, tail_vertices[1:])]

```

```
    # we are constructing edge labels for B_1, ... B_n, hence we drop B_1,
    # as it is already part of the output of fig1
    return head_vertices + tail_vertices[1:], head_edges + tail_edges
```

In all cases, the set of edge labels for the path $B_1B_2 \dots B_n$ is $\{1, 2, \dots, n-1\}$. By combining this with the labels of $C_m \cup A_1B_1$, the entire graph $C_m \cup P_n$ has a graceful labeling.

3 CASE $m \equiv 1 \pmod 4$

Theorem 2. *For all $m \equiv 1 \pmod 4$ and $n > 0$, $B(m, n)$ is a **graceful graph**.*

Proof. Let $m = 4l + 1$. If n is odd, we proceed as described in Section 3.1. If n is even, we distinguish three cases in Section 3.2. □

```
def thm2(m, n):
    assert(m >= 3)
    assert(m % 4 == 1)
    assert(n > 0)

    k = n // 2
    l = m // 4

    if n % 2 == 1:
        return fig3(m, n)
    if n % 2 == 0:
        if k == 1:
            return fig4(m, n)
        elif k >= l:
            return fig5(m, n)
        elif k < l:
            if n % 6 == 4:
                return case_3_2(m, n, fig6)
            if n % 6 == 0:
                return case_3_4(m, n, fig6)
            if n % 6 == 2:
                return case_3_6(m, n, fig6)
        else:
            raise ValueError(f'Uncaught case: k={k} l={l}, n={n} m={m}')
```

3.1 Case n is odd

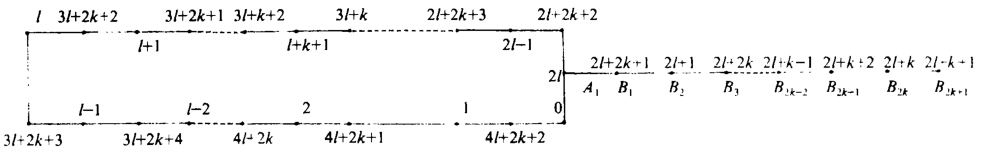

Fig. 3. Labeling for $B(m, n)$ when $m \equiv 1 \pmod 4$ and $n = 2k + 1$ (odd, $0 \le k \le l - 2$)

Let $n = 2k + 1$, where k is a non-negative integer.

- When $0 \le k \le l - 2$, a graceful labeling is given in Figure 3.

- When $k > l - 2$, the point labeled $l + k + 1$ in Figure 3 will be on the path $A_1B_1B_2 \dots B_{2k+1}$.

The construction for this case labels the path $C_0C_2 \dots C_{m+n-1}$ with the node labels

$$\varphi(C_i) = \begin{cases} \frac{i}{2} & \text{if } i \text{ is even} \\ n + m - \frac{i-1}{2} & \text{if } i \text{ is odd} \end{cases}$$

for $i < 2l + 2k$ and

$$\varphi(C_i) = \begin{cases} \frac{i}{2} - 1 & \text{if } i \text{ is even} \\ n + m - \frac{i-1}{2} - 1 & \text{if } i \text{ is odd} \end{cases}$$

for $i \geq 2l + 2k$, skipping the edge label $2l$. This yields all edge labels $\{m, \dots, 1\}$ in decreasing order, except $2l$. We note that $\varphi(C_{m-1}) = 2l$ and can hence connect C_0 and C_{m-1} to obtain a graceful labeling for the tadpole $B(m, n)$.

Both variants are implemented in the `fig3` function.

```
def fig3(m, n):
    assert(m % 4 == 1)
    assert(n % 2 == 1)

    l = m // 4

    vertices = [0]
    decrease = False
    offset = 0
    for w in range(n+m, 1, -1):
        current = vertices[-1]
        if w == 2*l:
            offset = -1
        if decrease:
            vertices.append(current - (w + offset))
            decrease = False
        else:
            vertices.append(current + w + offset)
            decrease = True

    # construct a long path
    edges = [(u,v) for u,v in zip(vertices, vertices[1:])]
    # add the edge connecting 0 with 2l that closes the head cycle
    edges.append((vertices[0], vertices[m-1]))

    return vertices, edges
```

3.2 Case n is even

Let $n = 2k$, where k is a natural number.

Case (2-1) If $k = 1$ ($n = 2$), a graceful labeling is given in Figure 4.

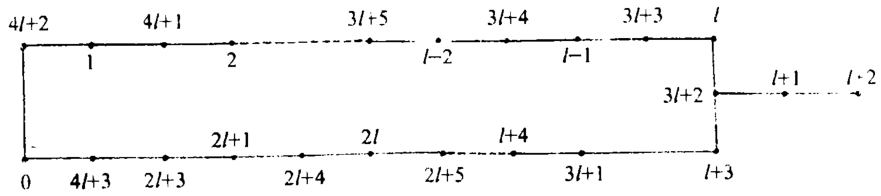

Fig. 4. Labeling for $B(m, 2)$ when $m \equiv 1 \pmod 4$ and $n = 2$

```
def fig4(m, n):
    assert(m % 4 == 1)
    assert(n == 2)

    l = m // 4

    vertices = [l+2, l+1]

    decrease = False
    for w in range(2*l+1, 4*l+4, 1):
        current = vertices[-1]
        if decrease:
            vertices.append(current - w)
            decrease = False
        else:
            vertices.append(current + w)
            decrease = True

    if 2*l+3 != 3*l+2:
        vertices.append(2*l+3)

    decrease = True
    for w in range(2, 2*l-1, 1):
        current = vertices[-1]
        if decrease:
            vertices.append(current - w)
            decrease = False
        else:
            vertices.append(current + w)
            decrease = True

    # construct a long path
    edges = [(u,v) for u,v in zip(vertices, vertices[1:])]
    # add the edge connecting 3l+2 with l+3 that closes the head cycle
    edges.append((vertices[-1], vertices[2]))

    return vertices, edges
```

Case (2-2) If $k \geq l$ $(k \neq 1)$, graceful labeling is given in Figure 5.

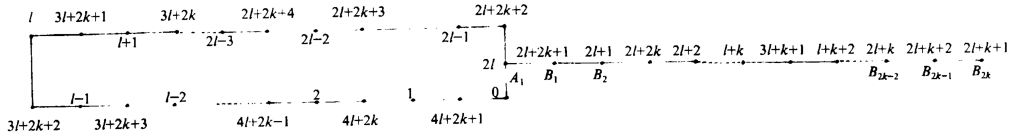

Fig. 5. Labeling for $B(m, n)$ when $m \equiv 1 \pmod 4$ and $n = 2k$ (even, $k \geq l$)

```python
def fig5(m, n):
    assert(m % 4 == 1)
    assert(n % 2 == 0)

    k = n // 2
    l = m // 4

    assert (k >= l)
    assert (k != 1)

    vertices = [2*l + k + 1]
    decrease = False
    offset = 0
    for w in range(1, 4*l+2*k+1, 1):
        current = vertices[-1]
        # skip edge length 2l
        if w == 2*l:
            offset = 1
        if decrease:
            vertices.append(current - (w + offset))
            decrease = False
        else:
            vertices.append(current + w + offset)
            decrease = True

    # construct a long path
    edges = [(u,v) for u,v in zip(vertices, vertices[1:])]
    # add the edge connecting 0 with 2l that closes the head cycle
    edges.append((vertices[-1], vertices[n]))

    return vertices, edges
```

Case (2-3) If $1 < k < l$:

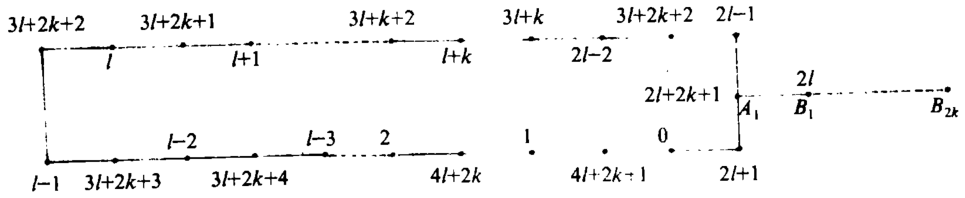

Fig. 6. Labeling for $C_m \cup A_1B_1$ when $m \equiv 1 \pmod 4$ and $n = 2k$ (even, $1 < k < l$)

- First, $C_m \cup A_1B_1$ is labeled according to Figure 6.
- The vertex label set for $C_m \cup A_1B_1$ is $\{0, 1, \ldots, 2l + 1\} \cup \{2l + 2k + 1, \ldots, 3l + k\} \cup \{3l + k + 2, \ldots, 4l + 2k + 1\}$.
- The edge label set is $\{2k, 2k + 1, \ldots, 4l + 2k + 1\}$.
- The remaining path is $B_1B_2 \ldots B_n$, and its required edge label set is $\{1, 2, \ldots, 2k - 1\}$.
- Since n is even, the construction methods from Theorem 1 for the even n cases—**(3-2)**, **(3-4)**, and **(3-6)**—can be used to give a graceful labeling for the remaining path.

```
def fig6(m, n):
    assert(m % 4 == 1)
    assert(n % 2 == 0)

    k = n // 2
    l = m // 4

    assert (k < l)
    assert (k > 1)

    vertices = [2*l+1, 0]
    decrease = False
    offset = 0
    for w in range(4*l+2*k+1, 2*k+2, -1):
        current = vertices[-1]
        # # skip edge length 2*l+1
        if w == 2*l+1:
            offset = -1
        if decrease:
            vertices.append(current - (w + offset))
            decrease = False
        else:
            vertices.append(current + w + offset)
            decrease = True

    # add cycle edges
    edges = list(zip(vertices, vertices[1:] + [vertices[0]]))

    # add tail
    vertices.append(2*l)
    edges.append((vertices[-1], vertices[-2]))
```

```
32
33      return vertices, edges
```

4 REFERENCES

(1) Guo Wenfu. The Gracefulness of Graph $B(m, n)$. Journal of Inner Mongolia Normal University (Natural Science Edition), 1994(2): 25-29.
(2) Chen Zhizeng. On the $B(m, n, p)$ and $B(m, n)$ 3-optimality. Journal of Inner Mongolia Normal University (Natural Science Edition), 1991(3): 11-19.

A HELPER FUNCTIONS

To check if a constructed graph labeling is indeed graceful, i have written a test function. The python file contains a loop to check if the constructed labelings for tadpoles with $m \in \{3, \ldots, 1000\}$ and $n \in \{1, \ldots, 1000\}$ are indeed graceful. They are. :)

```
def is_graceful(vertices, edges, verbose=False):
    '''
    Check if a graph, given by a list of vertex labels
    and a list of edges is graceful.
    '''

    # vertex labels in range 0...m (inclusive)?
    proper_range = min(vertices) >= 0 and max(vertices) <= len(edges)

    # vertex labels injective?
    injective_vertex_labels = len(set(vertices)) == len(vertices)

    # edge labels injective?
    labels = set()
    for e in edges:
        labels.add(abs(e[0] - e[1]))
    injective_edge_labels = len(labels) == len(edges)

    if verbose:
        print(f'Proper vertex range? {proper_range}')
        print(f'Injective vertex labels? {injective_vertex_labels}')
        print(f'Injective edge labels? {injective_edge_labels}')
    return proper_range and injective_vertex_labels and injective_edge_labels
```

If you want to draw one of these tadpoles, there is the following function that may be of some use:

```
def draw_graph(e):
    import networkx as nx
    import matplotlib.pyplot as plt

    g = nx.from_edgelist(e)
    pos1 = nx.circular_layout(g)
    nx.draw(g, pos1, with_labels=True, node_color='skyblue')
    nx.draw_networkx_edge_labels(g, pos1, edge_labels=edge_labels(e))
    plt.show()
```

B ORIGINAL PAPER

关于图 $B(m, n)$的优美性

郭 文 富

（乌兰察布师专数学系）

摘 要 证明了：当$m \equiv 1$或2(mod4)时，$B(m, n) \overset{\triangle}{=} C_m \cup P_n$是优美图，其中$C_m = A_1A_2\cdots A_mA_1$，$P_n = A_1B_1B_2\cdots B_n$ $(m \geqslant 3,\ n > 0)$.

关键词 圈，路，优美图

设圈 $C_m = A_1A_2\cdots A_mA_1$，路 $P_n = A_1B_1B_2\cdots B_n$. 定义 $B(m, n) \overset{\triangle}{=} C_m \cup P_n (m \geqslant 3,\ n > 0)$. [1] 中已证明 $B(m, n)$ 是 1 − 优美图. 还证明了，当 $m \equiv 0$ 或 3(mod4) 时. $B(m, n)$ 是优美图. 然后，提出“当 $m \equiv 1$ 或 2(mod4)，$n > 0$ 时，$B(m, n)$ 是否优美图”. 本文对此给出肯定的回答.

对于简单图 $G(V, E)$. $\forall v \in V$，赋予一个非负整数 $\varphi(v)$，则称图 G 是标定的，$\varphi(v)$ 称为顶点 v 的标号. $|\varphi(u) - \varphi(v)|$ 称为 uv 的标数，记为 $L(uv)$，又设整数集 $I = \{0, 1, 2, \cdots, |E|\}$.

定义 若简单图 G 是标定的，顶点的标号取自 I，不同顶点的标号不同，不同边的标数不同，则称图 G 是优美图.

根据上述定义，实际上，边标数集合为 $\{1, 2, \cdots, |E|\}$. 我们用 $N_1(N_2)$ 表示全体奇（偶）数的集合. 我们仍采用 [1] 中的述语.

定理1 当$m \equiv 2(\mathrm{mod}4)$，$n > 0$时，$B(m, n)$是优美图.

证 令$m \equiv 4l + 2$，则$l = \dfrac{m-2}{4}$.

1. $n = 1$时，可按图1给出优美标号.

2. $n = 2$时，可按图2给出优美标号.

3. $n \geqslant 3$ 时，首先，在 $C_m \cup A_1B_1$ 上按图 1 标号，易知标号互不相同，其数集为 $\{0, 1, \cdots, 2l + 1\} \cup \{n + 2l + 1, n + 2l + 2, \cdots, n + 3l\} \cup \{n + 3l + 2, \cdots, n + 4l + 1, n + 4l + 2\}$，且边标数互不相同，其数集为 $\{n, n + 1, n + 2, \cdots, n + 4l + 1, n + 4l + 2\}$. 还剩余路 $B_1B_2\cdots B_n$，其边标数集应为 $\{1, 2, \cdots, n - 1\}$.

其次，在路$B_1B_2\cdots B_n$上分6种情形给出标号.

收稿日期：1992−04−17，修改稿于 1993−04−20 收到

(3 − 1) $n = 6s + 3$时，记$a = \varphi(A_1) = n + 2l + 1$，$b = \varphi(B_1) = 2l$，置

$$\left.\begin{array}{l}\varphi(B_{6i+1}) = b + 3i, \quad 0 \leqslant i \leqslant s\\ \left.\begin{array}{l}\varphi(B_{6i+2}) = a - 2 - 3i,\\ \varphi(B_{5i+3}) = b + 2 + 3i,\\ \varphi(B_{6i+4}) = a - 1 - 3i,\\ \varphi(B_{6i+5}) = b + 4 + 3i,\\ \varphi(B_{6i+6}) = a - 3 - 3i,\end{array}\right\} \quad 0 \leqslant i \leqslant s - 1\\ \varphi(B_{6s+2}) = 2l + 2 + 3s,\\ \varphi(B_{6s+3}) = 2l + 3 + 3s.\end{array}\right\} \tag{1}$$

易知标号互不相同，而且是由$2l$至$2l + n$中间除去$2l + 1$的所有整数，且B_1的标号仍为$2l$. 与$C_m \cup A_1B_1$上的标号集比较，可知$C_m \cup P_n$上的标号互不相同，且不超过$|E| = n + m = n + 4l + 2$.

计算边标数，则

$$\left.\begin{array}{l}L(B_{6i+1}B_{6i+2}) = n - 1 - 6i,\\ L(B_{6i+2}B_{6i+3}) = n - 3 - 6i,\\ L(B_{6i+3}B_{6i+4}) = n - 2 - 6i,\\ L(B_{6i+4}B_{6i+5}) = n - 4 - 6i,\\ L(B_{6i+5}B_{6i+6}) = n - 6 - 6i,\\ L(B_{6i+6}B_{6(i+1)+1}) = n - 5 - 6i,\end{array}\right. \tag{2}$$

其中$0 \leqslant i \leqslant s - 1$. 以上6式中分别令$i = 0$，1，…，$s - 1$，可知边标数互不相同，且组成由3到$n - 1$的$n - 3$个边标数. 又因$L(B_{6s+1}B_{6s+2}) = 2$，$L(B_{6s+2}B_{6s+3}) = 1$，于是，路$B_1B_2\cdots B_n$的边标数集合为{1，2，…，$n - 1$}.

综合$C_m \cup P_n$上的情形，易知我们给出的标号法是优美的.

以下证法类似，我们只写出关键步骤，其余省略.

(3 − 2) $n = 6s + 4$时，对B_1至B_{6s+1}仍用(1)式标号，此外置

$$\varphi(B_{6s+2}) = 2l + 3 + 3s,\ \varphi(B_{6s+3}) = 2l + 2 + 3s,\ \varphi(B_{6s+4}) = 2l + 4 + 3s.$$

(2)式计算的边标数仍适用，但此时边标数集合为{4，5，…，$n - 1$}. 又因

$$L(B_{6s+1}B_{6s+2}) = 3,\ L(B_{6s+2}B_{6s+3}) = 1,\ L(B_{6s+3}B_{6s+4}) = 2,$$

所以路$B_1B_2\cdots B_n$的边标数集合为{1，2，…，$n - 1$}.

(3 − 3) $n = 6s + 5$时，对B_1至B_{6s+1}仍用(1)式标号，此外置

$$\varphi(B_{6s+2}) = 2l + 4 + 3s,\ \varphi(B_{6s+3}) = 2l + 3 + 3s,\ \varphi(B_{6s+4}) = 2l + 5 + 3s,$$
$$\varphi(B_{6s+5}) = 2l + 2 + 3s.$$

(2) 式计算的边标数仍适用，但此时边标数集合为 {5，6，…，$n - 1$}. 又因

$L(B_{6s+1}B_{6s+2})=4$，$L(B_{6s+2}B_{6s+3})=1$，$L(B_{6s+3}B_{6s+4})=2$，$L(B_{6s+4}B_{6s+5})=3$，

所以路$B_1B_2\cdots B_n$的边标数集合为$\{1，2，\cdots，n-1\}$.

(3－4) $n=6s+6$时，对B_1至B_{6s+1}仍用(1)式标号，此外置

$\varphi(B_{6s+2})=2l+5+3s$，$\varphi(B_{6s+3})=2l+2+3s$，$\varphi(B_{6s+4})=2l+6+3s$，

$\varphi(B_{6s+5})=2l+4+3s$，$\varphi(B_{6s+6})=2l+3+3s$.

(2)式计算的边标数仍适用，但此时边标数集合为$\{6，7，\cdots，n-1\}$．又因

$L(B_{6s+1}B_{6s+2})=5$，$L(B_{6s+2}B_{6s+3})=3$，$L(B_{6s+3}B_{6s+4})=4$，

$L(B_{6s+4}B_{6s+5})=2$，$L(B_{6s+5}B_{6s+6})=1$，

所以路$B_1B_2\cdots B_n$的边标数集合为$\{1，2，\cdots，n-1\}$.

(3－5) $n=6s+7$时，对B_1至B_{6s+1}仍用(1)式标号，此外置

$\varphi(B_{6s+2})=2l+6+3s$，$\varphi(B_{6s+3})=2l+2+3s$，$\varphi(B_{6s+4})=2l+7+3s$，$\varphi(B_{6s+5})$

$=2l+4+3s$，$\varphi(B_{6s+6})=2l+5+3s$，$\varphi(B_{6s+7})=2l+3+3s$.

(2)式计算的边标数仍适用，但此时边标数集合为$\{7，8，\cdots，n-1\}$．又因

$L(B_{6s+1}B_{6s+2})=6$，$L(B_{6s+2}B_{6s+3})=4$，$L(B_{6s+3}B_{6s+4})=5$，

$L(B_{6s+4}B_{6s+5})=3$，$L(B_{6s+5}B_{6s+6})=1$，$L(B_{6s+5}B_{6s+7})=2$，

所以路$B_1B_2\cdots B_n$的边标数集合为$\{1，2，\cdots，n-1\}$.

(3－6) $n=6s+8$时，对B_1至B_{6s+1}仍用(1)式标号，此外置

$\varphi(B_{6s+2})=2l+7+3s$，$\varphi(B_{6s+3})=2l+2+3s$，$\varphi(B_{6s+4})=2l+8+3s$，

$\varphi(B_{6s+5})=2l+4+3s$，$\varphi(B_{6s+6})=2l+5+3s$，$\varphi(B_{6s+7})=2l+3+3s$，

$\varphi(B_{6s+8})=2l+6+3s$.

(2)式计算的边标数仍适用，但此时边标数集合为$\{8，9,\cdots，n-1\}$．又因

$L(B_{6s+1}B_{6s+2})=7$，$L(B_{6s+2}B_{6s+3})=5$，$L(B_{6s+3}B_{6s+4})=6$，$L(B_{6s+4}B_{6s+5})$

$=4$，$L(B_{6s+5}B_{6s+6})=1$，$L(B_{6s+6}B_{6s+7})=2$，$L(B_{6s+7}B_{6s+8})=3$，

所以路$B_1B_2\cdots B_n$的边标数集合为$\{1，2，\cdots，n-1\}$.

证毕

定理2 当$m\equiv 1(\mathrm{mod}4)$，$n>0$时，$B(m，n)$是优美图.

证 令$m\equiv 4l+1$.

1. $n\in N_1$．令$n=2k+1$，k为非负整数．当$0\leqslant k\leqslant l-2$时，可按图3给出优美标号．当$k>l-2$时，图3中标号为$l+k+1$的点将在路$A_1B_1B_2\cdots B_{2k+1}$上．

2. $n\in N_2$．令$n=2k$，k为自然数.

(2－1) $k=1$时，$n=2$，可按图4给出优美标号.

(2－2) $k\geqslant l(k\neq 1)$时，可按图5给出优美标号.

(2－3) $1<k<l$时，首先在$C_m\cup A_1B_1$上按图6标号．易知标号互不相同，其数集

为$\{0, 1, \cdots, 2l+1\}\bigcup\{2l+2k+1, \cdots, 3l+k\}\bigcup\{3l+k+2, \cdots, 4l+2k+1\}$，且边标数集为$\{2k, 2k+1, \cdots, 4l+2k+1\}$，还剩余路$B_1B_2\cdots B_n$，其边标数集应为$\{1, 2, \cdots, 2k+1\}$.

其次在路$B_1B_2\cdots B_n$上，由于n为偶数，故只用定理1的证明中(3－2)、(3－4)、(3－6)的作法就可给出优美标号.

证毕

参 考 文 献

[1] 陈志增. 关于$B(m, n, p)$和$B(m, n)$的$\frac{1}{2}$-优美性. 内蒙古师大学报(自然科学汉文版), 1991, 3: 11～19

The Gracefulness on Graph $B(m, n)$

Guo Wenfu

(Depl. of Math, Wulanchab teachers' college)

Abstract In this paper, we shall prove that "the graphs $B(m, n)\overset{\triangle}{=}C_m\bigcup P_n$ are graceful when m≡1 or 2 (mod 4), in which $C_m=A_1A_2\cdots A_mA_1$, $P_n=A_1B_1B_2\cdots B_n$ $(m>3, n>0)$

Key words cycle, path, graceful graph

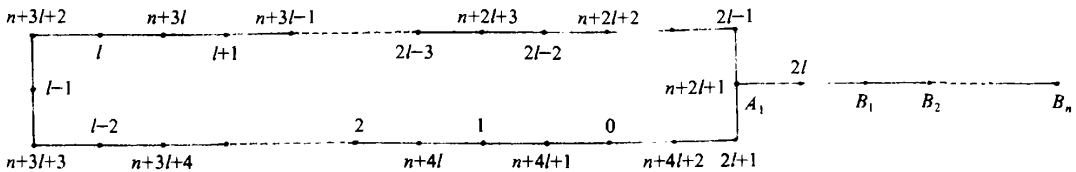

图 1

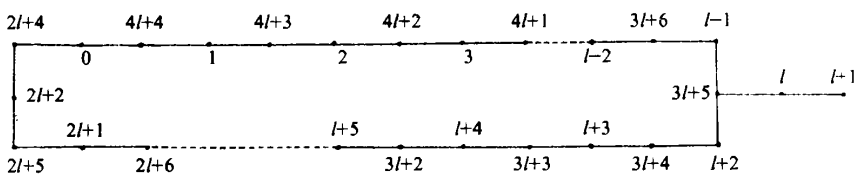

图 2

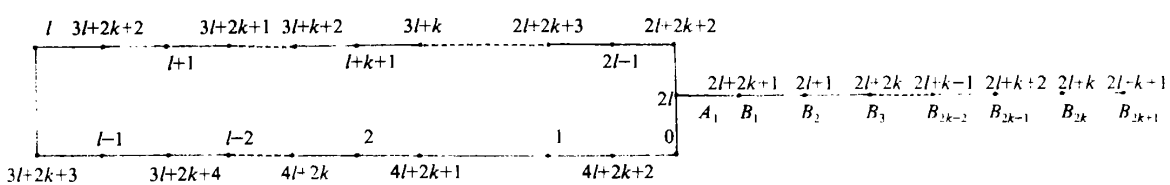

图 3

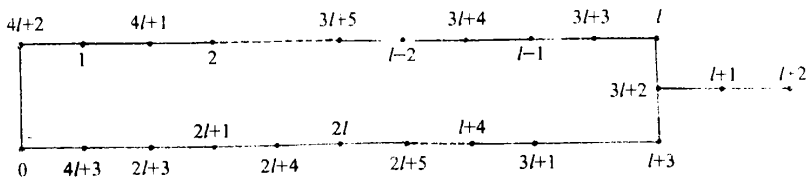

图 4

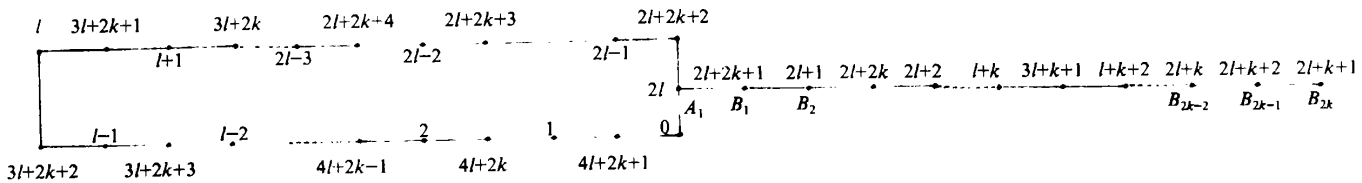

图 5

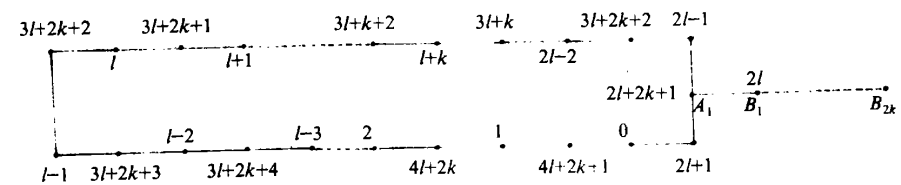

图 6