**Описание алгоритма ROU версии 2**

**Описание алгоритма на английском псевдокоде:**

**algorithm Find\_route\_ROU** is

**Input:** startNode – start node, endNode – end node, K – count of cycles, N – count of nodes, S1 ­­– first generate, S2 ­­– second generate.

**Output:** startNode – next start node.

1. **If**  **then**
2. **else**
3. **If**  **then**
4. **else**
5. **If**  **then**
6. **return *startNode***

**function Step\_cicles** is

**Input:** startNode – start node, endNode – end node, K – count of cycles, N – count of nodes, S1 ­­– first generate, S2 ­­– second generate.

**Output:** the function returns the best step (direction is also selected)

1. ,
2. **If** **then**
3. ,
4. **else**
5. **If**  **then**
6. ,
7. **else**
8. ,
9. **If then**
10. **for i to (K)**
11. **,**
12. **If then**
13. **,**
14. **If then**
15. **,**
16. **end for**
17. *,* ,
18. **If** **then**
19. ,
20. **else**
21. **If**  **then**
22. ,
23. **else**
24. ,
25. **If then**
26. **for** *i* **to** (K)
27. ,
28. **If**  **then**
29. ,
30. **If**  **then**
31. ,
32. **end for**
33. **If**  **then**
34. **return stepR**
35. **else**
36. **return stepL**

**Фрагмент кода алгоритма на языке С#:**

Алгоритм **ROU** версии 2:

private int Find\_route\_ROU(int start\_node, int end\_node, int K, int N, int s1=1, int s2=2)

{

if (start\_node > end\_node)

{

start\_node = start\_node - Step\_cicles(end\_node, start\_node,K, N, s1, s2);

}

else

{

start\_node = start\_node + Step\_cicles(start\_node, end\_node,K, N, s1, s2);

}

if (start\_node > N)

{

start\_node = start\_node - N;

}

else if (start\_node <= 0)

{

start\_node = start\_node + N;

}

return start\_node;

}

Функция **Step\_cicles**:

private int Step\_cicles(int start\_node, int end\_node, int K, int N, int s1 = 1, int s2 = 2)

{

int best\_way\_R = 0, step\_R = 0, best\_way\_L = 0, step\_L = 0;

int s = end\_node - start\_node;

int R1 = s / s2 + s % s2;

int R2 = s / s2 - s % s2 + s2 + 1;

if (s % s2 == 0)

{

best\_way\_R = R1;

step\_R = s2;

}

else

{

if (R1 < R2)

{

best\_way\_R = R1;

step\_R = s1;

}

else

{

best\_way\_R = R2;

step\_R = s2;

}

}

if (K > 0)

for (int i = 1; i < K+1; i++)

{

int Ri1 = (s + N\*i) / s2 + (s + N\*i) % s2;

int Ri2 = (s + N\*i) / s2 - (s + N\*i) % s2 + s2 + 1;

if (Ri1 < best\_way\_R)

{

best\_way\_R = Ri1;

step\_R = s2;

}

if (Ri2 < best\_way\_R)

{

best\_way\_R = Ri2;

step\_R = s2;

}

}

s = start\_node - end\_node + N;

int L1 = s / s2 + s % s2;

int L2 = s / s2 - s % s2 + s2 + 1;

if (s % s2 == 0)

{

best\_way\_L = L1;

step\_L = -s2;

}

else

{

if (L1 < L2)

{

best\_way\_L = L1;

step\_L = -s1;

}

else

{

best\_way\_L = L2;

step\_L = -s2;

}

}

if (K > 0)

for (int i = 1; i < K+1; i++)

{

int Ri1 = (s + N \* i)/ s2 + (s + N \* i) % s2;

int Ri2 = (s + N \* i) / s2 - (s + N \* i) % s2 + s2 + 1;

if (Ri1 < best\_way\_L)

{

best\_way\_L = Ri1;

step\_L = -s2;

}

if (Ri2 < best\_way\_L)

{

best\_way\_L = Ri2;

step\_L = -s2;

}

}

if (best\_way\_R < best\_way\_L)

{

return step\_R;

}

else

{

return step\_L;

}

}

private int Step(int start\_node, int end\_node, int N, int s1 = 1, int s2 = 2)

{

int best\_way\_R = 0, step\_R = 0, best\_way\_L = 0, step\_L = 0;

int s = end\_node - start\_node;

int R1 = s / s2 + s % s2;

int R2 = s / s2 - s % s2 + s2 + 1;

if (s % s2 == 0)

{

best\_way\_R = R1;

step\_R = s2;

}

else

{

if (R1 < R2)

{

best\_way\_R = R1;

step\_R = s1;

}

else

{

best\_way\_R = R2;

step\_R = s2;

}

}

s = start\_node - end\_node + N;

int L1 = s / s2 + s % s2;

int L2 = s / s2 - s % s2 + s2 + 1;

if (s % s2 == 0)

{

best\_way\_L = L1;

step\_L = -s2;

}

else

{

if (L1 < L2)

{

best\_way\_L = L1;

step\_L = -s1;

}

else

{

best\_way\_L = L2;

step\_L = -s2;

}

}

if (best\_way\_R < best\_way\_L)

{

return step\_R;

}

else

{

return step\_L;

}

}