

Problem Menu

< Back

Statement

Submissions

>

Your score

not attempted

Spoilers

Show difficulty

Show tags

solved by 29 / 30

graph: shortest path

Top users by time

#	User	Time
1	levirs565	129 ms
2	irdacin	129 ms
3	AhmadRomy	131 ms
4	JUstRadr	133 ms
5	Arman10	147 ms

Top users by memory

#	User	Memory
1	zhoelzx	8152 KB
2	levirs565	9032 KB
3	vandykas	9196 KB
4	AhmadRomy	9416 KB
5	Arman10	10568 KB

Arkavidia 9.0 - Penyisihan CP > G

Submission #4744624

Arkavidia 9.0 - Penyisihan CP / G. Gerbang Sihir Arkavidia

Accepted · LilyFlower · Go · January 9, 2026 at 19:02:04

▶ Sample Test Data Results



▶ Test Data Results



```
1 package main
2
3 import (
4     "bufio"
5     "container/heap"
6     "fmt"
7     "os"
8 )
9
10 var (
11     in      = bufio.NewReader(os.Stdin)
12     out     = bufio.NewWriter(os.Stdout)
13     println = Println
14     scan    = Scan
15     print   = Print
16 )
17
18 type Edge struct {
19     to int64
20     c  int64
21     t  int64
22     dir int64
23 }
24
25 type Node struct {
26     Value int64
27     Edges []Edge
28 }
29
30 func NewNode(value int64) *Node {
31     return &Node{
32         Value: value,
33         Edges: make([]Edge, 0),
34     }
35 }
36
37 func travelTime(u, v int64, currentTime int64, edge Edge) int64 {
38     if edge.dir == 1 {
39         cycle := 2 * edge.t
40         phase := currentTime % cycle
41         if phase < edge.t {
42             return currentTime + edge.c
43         } else {
44             wait := cycle - phase
45             return currentTime + wait + edge.c
46         }
47     } else {
48         cycle := 2 * edge.t
49         phase := currentTime % cycle
50         if phase >= edge.t {
51             return currentTime + edge.c
52         } else {
53             wait := edge.t - phase
54             return currentTime + wait + edge.c
55         }
56     }
57 }
58
59 }
60 }
61 }
62
63 type PQItem struct {
64     node int64
65     time int64
66     index int
67 }
68
69 type PriorityQueue []*PQItem
70
71 func (pq PriorityQueue) Len() int { return len(pq) }
72 func (pq PriorityQueue) Less(i, j int) bool { return pq[i].time < pq[j].time }
73 func (pq PriorityQueue) Swap(i, j int) {
74     pq[i], pq[j] = pq[j], pq[i]
75     pq[i].index = i
76     pq[j].index = j
77 }
78
79 func (pq *PriorityQueue) Push(x any) {
80     n := len(*pq)
81     item := x.(*PQItem)
82     item.index = n
83     *pq = append(*pq, item)
84 }
85
86 func (pq *PriorityQueue) Pop() any {
87     old := *pq
88     n := len(old)
89     item := old[n-1]
90     item.index = -1
91     *pq = old[0 : n-1]
92     return item
93 }
94
95 func Solve() {
96     var n, k int64
97     scan(&n, &k)
98
99     nodes := make([]*Node, n+1)
100     for i := int64(1); i <= n; i++ {
101         nodes[i] = NewNode(i)
```

```

102     }
103
104     for i := int64(0); i < k; i++ {
105         var a, b, c, t int64
106         scan(&a, &b, &c, &t)
107
108         nodes[a].Edges = append(nodes[a].Edges, Edge{
109             to: b,
110             c: c,
111             t: t,
112             dir: 1,
113         })
114
115         nodes[b].Edges = append(nodes[b].Edges, Edge{
116             to: a,
117             c: c,
118             t: t,
119             dir: -1,
120         })
121     }
122
123     var x, y int64
124     scan(&x, &y)
125
126     dist := make([]int64, n+1)
127     for i := range dist {
128         dist[i] = -1
129     }
130     dist[x] = 0
131
132     pq := make(PriorityQueue, 0)
133     heap.Init(&pq)
134     heap.Push(&pq, &PQItem{node: x, time: 0})
135
136     for pq.Len() > 0 {
137         current := heap.Pop(&pq).(*PQItem)
138         u := current.node
139         currentTime := current.time
140
141         if currentTime > dist[u] && dist[u] != -1 {
142             continue
143         }
144
145         if u == y {
146             println(currentTime)
147             return
148         }
149
150         for _, edge := range nodes[u].Edges {
151             v := edge.to
152             nextTime := travelTime(u, v, currentTime, edge)
153
154             if dist[v] == -1 || nextTime < dist[v] {
155                 dist[v] = nextTime
156                 heap.Push(&pq, &PQItem{node: v, time: nextTime})
157             }
158         }
159     }
160
161     println(-1)
162 }
163
164 func Print(a ...any) {
165     fmt.Fprint(out, a...)
166 }
167
168 func Println(a ...any) {
169     fmt.Fprintln(out, a...)
170 }
171
172 func Scan(a ...any) {
173     fmt.Fscan(in, a...)
174 }
175
176 func main() {
177     defer out.Flush()
178     Solve()
179 }

```