Micromouse

Micromouse é um evento onde pequenos robôs competem para resolver labirintos. Começando em uma posição inicial, os robôs devem encontrar o objetivo, voltar à posição inicial e retornar ao objetivo. Quanto mais rápido melhor!

Os robôs não possuem conhecimento da estrutura do labirinto: as paredes só são encontradas quando o robô está suficientemente perto para senti-las ou quando ocorre colisão.

Para este problema você irá construir um programa interativo, ou seja, que irá se comunicar com um árbitro.

O objetivo do seu programa é controlar um ratinho, virtual, que inicia em um ponto qualquer de um labirinto. O ratinho não conhece o labirinto e tão pouco a posição em que começa. Para alcançar as metas, o seu ratinho deve encontrar o ponto de saída do labirinto, depois ele deverá retornar para o ponto de partida e por fim deverá ir mais uma vez para a saída do labirinto.

Para conseguir se locomover pelo labirinto, o seu ratinho poderá executar comandos. Cada comando executado infere em um resultado gerado pelo árbitro do jogo. A comunicação entre o seu programa e o árbitro acontece por meio da entrada e saída padrão.

Note que a cada comando dado é obrigatório a leitura da resposta deste comando, mesmo que você deseje ignorar seu resultado. Todo comando dado deverá conter um `fflush(stdout)`, quando escrito na linguagem C, e outro comando similar a para outra linguagem utilizada.

Para cada comando executado, é somado uma (01) unidade de tempo gasto no domínio do jogo. O seu objetivo é executar as operações com a menor quantidade de tempo gasta.

Ações possíveis

Uma ação é realizada imprimindo um caractere junto com quebra de linha na saída padrão. As possíveis ações são:

1. `l` → rotacionar 90º para a esquerda (sentido anti-horário).
2. `r` → rotacionar 90º para a direita (sentido horário).
3. `w` → caminhar para frente. (anda uma casa)
4. `j` → corridinha para frente. (anda duas casas)
5. `R` → correr para frente. (anda três casas)
6. `s` → correr ao máximo para frente. (anda quatro casas)
7. `c` → ativar sensor de paredes próximas.
8. `d` → ativar sensor de proximidade do objetivo.

Após realizar qualquer ação, o robô deve ler da entrada padrão o resultado obtido. Esse resultado é sempre um número inteiro, mas seu significado muda de acordo com a ação previamente executada.

IMPORTANTE: O jogador sempre deve realizar a leitura da entrada padrão após qualquer movimento. Caso essa leitura não seja feita, a correção do exercício não será executada corretamente e a pontuação não será atribuída ao jogador.

Retornos das Ações

(`w`) caminhar para frente

Quando o robô caminha para frente (`w`), o valor de retorno pode ser `0` (o robô bateu em uma parede), `1` (o robô conseguiu se mover) ou 2 (o robô encontrou o objetivo).

(`j`) corridinha para frente

Quando o robô faz corridinha para frente (`j`), o valor de retorno pode ser `0` (bateu em uma parede sem ser mover), `1` (se moveu uma vez e bateu em uma parede) ou `2` (se moveu duas vezes).

A corridinha (`j`) penaliza o robô caso ele colida com uma parede antes do movimento finalizar. Essa penalidade é equivalente à 3 movimentos bem-sucedidos, logo, 3 unidades de tempo.

(R) correr para frente

Quando o robô corre (`R`), o valor de retorno pode ser `0` (bateu em uma parede sem ser mover), `1` (se moveu uma vez e bateu em uma parede), `2` (se moveu duas vezes e bateu em uma parede) ou `3` (se moveu três vezes).

A corrida (`R`) penaliza o robô caso ele colida com uma parede antes do movimento finalizar. Essa penalidade é equivalente à 4 movimentos bem-sucedidos, logo, 4 unidades de tempo.

(s) correr ao máximo para frente

Quando o robô corre em alta velocidade (`s`), o valor de retorno pode ser `0` (bateu em uma parede sem ser mover), `1` (se moveu uma vez e bateu em uma parede), `2` (se moveu duas vezes e bateu em uma parede), `3` (se moveu três vezes e bateu em uma parede) ou `4` (se moveu quatro vezes).

A corrida em alta velocidade (`s`) penaliza o robô caso ele colida com uma parede antes do movimento finalizar. Essa penalidade é equivalente à 5 movimentos bem-sucedidos, logo, 5 unidades de tempo.

(c) ativar sensor de paredes próximas

Quando o robô ativa o sensor de paredes próximas (`c`), o valor de retorno é um número inteiro de 4 bits representando para quais direções o robô pode se mover. Ou então -1 quando estiver quebrado

Por exemplo:

```
int retorno = 13; // 0b1101
int frente = (retorno >> 0) & 1; // 1 (parede)
int direita = (retorno >> 1) & 1; // 0 (livre)
int tras = (retorno >> 2) & 1; // 1 (parede)
int esquerda = (retorno >> 3) & 1; // 1 (parede)
```

(d) ativar sensor de proximidade do objetivo

Quando o robô ativa o sensor de distância (`d`), o valor de retorno é um número inteiro representando a distância de **Manhattan** da posição atual até o objetivo. Ou então -1 quando o sensor estiver quebrado.

O uso de qualquer um dos dois sensores penaliza o robô em o equivalente à 10 unidades de tempo.

(l) rotacionar 90º para a esquerda e (r) rotacionar 90º para a direita

Quando robô rotaciona (`l` ou `r`), o valor de retorno é insignificante. **Contudo, é importante que o robô leia o valor de retorno.** Se o robô não ler o valor de retorno, a solução não será corrigida corretamente.

Pontuação

A pontuação final é calculada pela quantidade de ações tomadas para finalizar o caminho *inicial* → *objetivo* → *inicial* → *objetivo* somadas com as penalidades aplicadas por colisões ou uso de sensores.

Lembrando que na última ida ao objetivo, depois de conhecer o mapa e ter voltado para o inicio, toda unidade de tempo das ações e penalidades são multiplicadas por 2. Então garanta que essa sua última volta será a mais rápida possível.

Não será possível utilizar os sensores na última volta, será retornado -1 que é o código de sensor quebrado.

Ganha o robô que tiver a menor quantidade de tempo acumulado. Em caso de empate, o desempate será a quantidade de tempo gasto de processamento da solução.

Exemplo

Aqui está um exemplo de um labirinto e o ínicio de uma possível solução.

Lembrando que o mapa não será conhecido pelo robô, ele que deve descobrir. Isso é apenas um exemplo visual para facilitar o entendimento.

No caso real, apenas o árbitro conhece o mapa e não irá ser impresso para o jogador.

A baixo tem uma representação do estado do mapa e abaixo de cada tem o comando executado por seu programa e o que o árbitro responderia.

```
Estado inicial:
+---+---+---+
|         < |
+   +---+---+
|           |
+---+---+   +
| x         |
+---+---+---+
Comando: j MOJ: 2
+---+---+---+
| <         |
+   +---+---+
|           |
+---+---+   +
| x         |
+---+---+---+
Comando: j MOJ: 0
+---+---+---+
| <         |
+   +---+---+
|           |
+---+---+   +
| x         |
+---+---+---+
Comando: l MOJ: 1
+---+---+---+
| v         |
+   +---+---+
|           |
+---+---+   +
| x         |
+---+---+---+
Comando: w MOJ: 1
+---+---+---+
|           |
+   +---+---+
| v         |
+---+---+   +
| x         |
+---+---+---+
Comando: w MOJ: 0
+---+---+---+
|           |
+   +---+---+
| v         |
+---+---+   +
| x         |
+---+---+---+
Comando: l MOJ: 1
+---+---+---+
|           |
+   +---+---+
| >         |
+---+---+   +
| x         |
+---+---+---+
Comando: s MOJ: 2
+---+---+---+
|           |
+   +---+---+
|         > |
+---+---+   +
| x         |
+---+---+---+
```

```
Comando: r MOJ: 1
+---+---+---+
|           |
+   +---+---+
|         v |
+---+---+   +
| x         |
+---+---+---+
Comando: w MOJ: 1
+---+---+---+
|           |
+   +---+---+
|           |
+---+---+   +
| x       v |
+---+---+---+
Comando: r MOJ: 1
+---+---+---+
|           |
+   +---+---+
|           |
+---+---+   +
| x       < |
+---+---+---+
Comando: w MOJ: 1
+---+---+---+
|           |
+   +---+---+
|           |
+---+---+   +
| x   <     |
+---+---+---+
Comando: w MOJ: 2
+---+---+---+
|           |
+   +---+---+
|           |
+---+---+   +
| <         |
+---+---+---+
```

Author: Bruno Ribas, Guilherme Puida, Thalisson Alves, Cauã Corrêa e Bruno Ribeiro.