Dokumentacja

Diaminy

Paweł Maczuga

Zadania

	Nazwa	Wynik
ezy	ezy	100
gen	gen	40
han	han	40

Program składa się z dwóch części:

1) Zmiana danych wejściowych na graf
2) Znalezienie rozwiązania na grafie

Zmiana danych wejściowych na graf

Każdy **wierzchołek** ma:

- **id**

Id wierzchołków dla danych reprezentujących poniższą planszę:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

- **stan**

```
struct State { int movesLeft; int jewelsLeft; };
```

Aktualizowany podczas odwiedzin danego wierzchołka (gdy jest znacznie lepszy niż poprzedni).

- **listę krawędzi**

Każda **krawędź** ma:

- **kierunek** - od 0 do 7 - kierunek ruchu po tej krawędzi wynikający z treści zadania
- **sąsiada -** wierzchołek, do którego krawędź prowadzi
- **diamenty -** listę diamentów na tej krawędzi

Dane wejściowe zamieniane są na zdefiniowany powyżej graf, gdzie każda pozycja na planszy to jeden wierzchołek, a każda krawędź to możliwy ruch z tego wierzchołka (uwzględniając wszystkie diamenty po drodze).
Krawędź w danym kierunku nie jest dodawana, gdy:

- trafimy na minę
- zaraz obok nas jest ściana

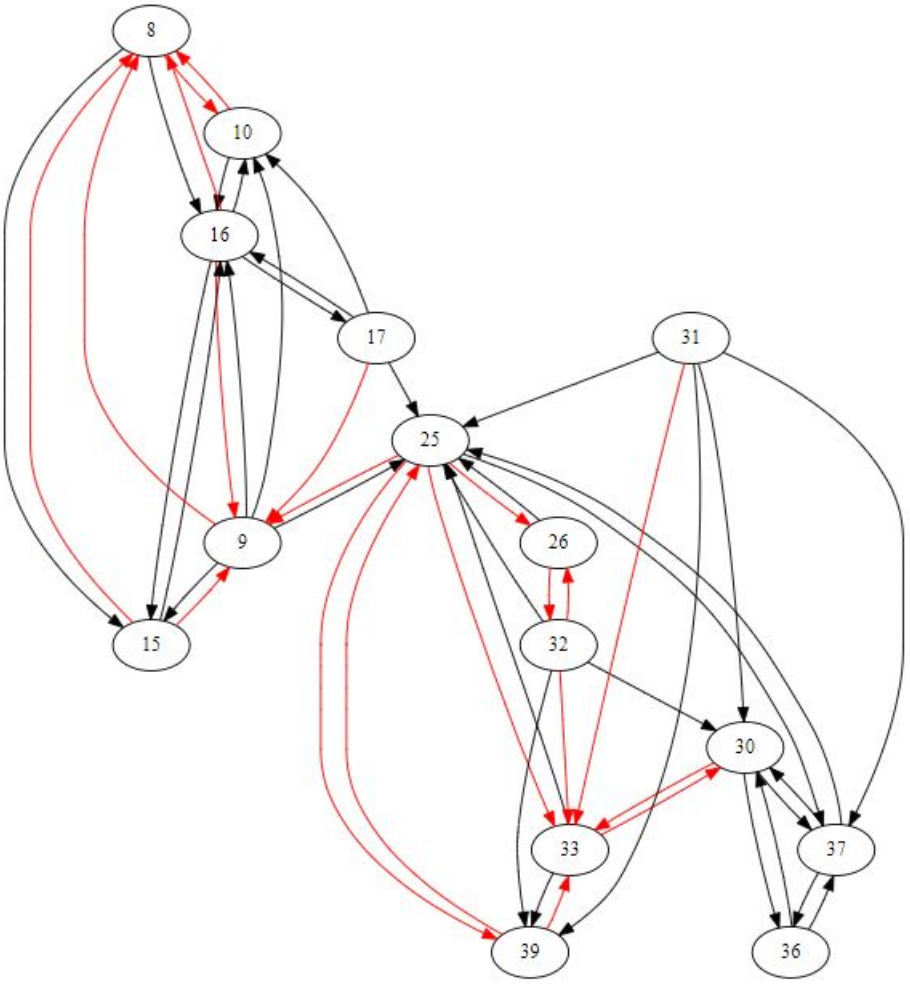
Uwzględniane są przypadki gdy nie ma muru wokół labiryntu.

Krawędzie dla każdego wierzchołka są sortowane od w kolejności największej ilości diamentów.

Na przykład dla poniższych danych wejściowych:

```
7 7
10
#######
#++.###
#OO #*#
#***O+#
##O ++#
#O # *#
#######
```

Jest tworzony następujący graf:



Gdzie czerwone krawędzie zawierają diamenty.
Użyłem generatora ze strony: http://www.webgraphviz.com/

Znalezienie rozwiązania na grafie

Algorytm znajdowania rozwiązania jest rekurencyjny. Pseudokod:

```
solve(G, position, prevoisPosition, jewelsLeft, movesLeft):
      // znalezione rozwiązanie
      if jewelsLeft == 0:
            return ""

      // koniec ruchów
      if movesLeft == 0:
            return "BRAK"

      if movesLeft * maxJewelsInRow < jewelsLeft:
            return "BRAK"

      if stan wierzchołka gorszy niż poprzednio:
            return "BRAK"
```

```
if stan wierzchołka znacznie lepszy niż poprzednio:
      zaktualizuj stan

for edge in position.edges:
      // nie ma sensu wracać tam skąd przyszliśmy jeśli nie
      // zebraliśmy po drodze diamentów
      if edge.sąsiad == previousPosition and edge nie ma diamentów:
            continue

      dodaj diamenty z krawędzi edge do listy zebranych

      solution = solve(G, edge.sąsiad, position,
                        jewelsLeft-zebrane, movesLeft-1)

      if solution != "BRAK":
            return edge.kierunek + solution

      // brak rozwiązania
      usuń diamenty z krawędzi edge z listy zebranych

return "BRAK"
```

Ważniejsze objaśnienia:

maxJewelsInRow

Jeśli niemożliwe jest dotarcie do pozostałych diamentów w obecnej ilości ruchów - zwracamy BRAK. Bardzo prosty i niezbyt efektywny warunek (`movesLeft * maxJewelsInRow < jewelsLeft`), ale spowodował pewien wzrost wydajności.

Stan wierzchołka

Stan wierzchołka to:

- liczba ruchów
- liczba pozostałych diamentów

Jeśli doszliśmy do danego wierzchołka sprawdzamy jego stan i porównujemy z obecnym. Następnie:

- gdy **obie** wartości są gorsze - zwracamy BRAK
- gdy **obie** wartości są lepsze - aktualizujemy stan

Dodanie stanu spowodowało dość znaczny wzrost wydajności.

Brak powrotów

Dość znaczny wzrost wydajności.

Lista zebranych diamentów

Jest to:

```
bool foundJewels[allJewelsCount];
```

Gdzie każdy indeks to **id** diamentu - znaczenie, oraz odznaczanie zebrania diamentu w czasie stałym.

Niedziałające pomysły

- Próba zapamiętywania więcej niż jednego wierzchołka, tak by nie wracać do poprzednio odwiedzonych, gdy nie zostały zebrane diamenty.
  Problemy z implementacją, dłuższy czas działania
- Przed uruchomieniem algorytmu próba znalezienia podgrafów, które można bezpiecznie usunąć, np. takich oddzielonych mostem od punktu startu i nie zwierających diamentów.
  Problem w grafie skierowanym, praktycznie nie występujące są na tyle duże podgrafy, aby przyniosło to jakikolwiek zysk