Jak se neztratit v labyrintu

ParaDiSe seminář

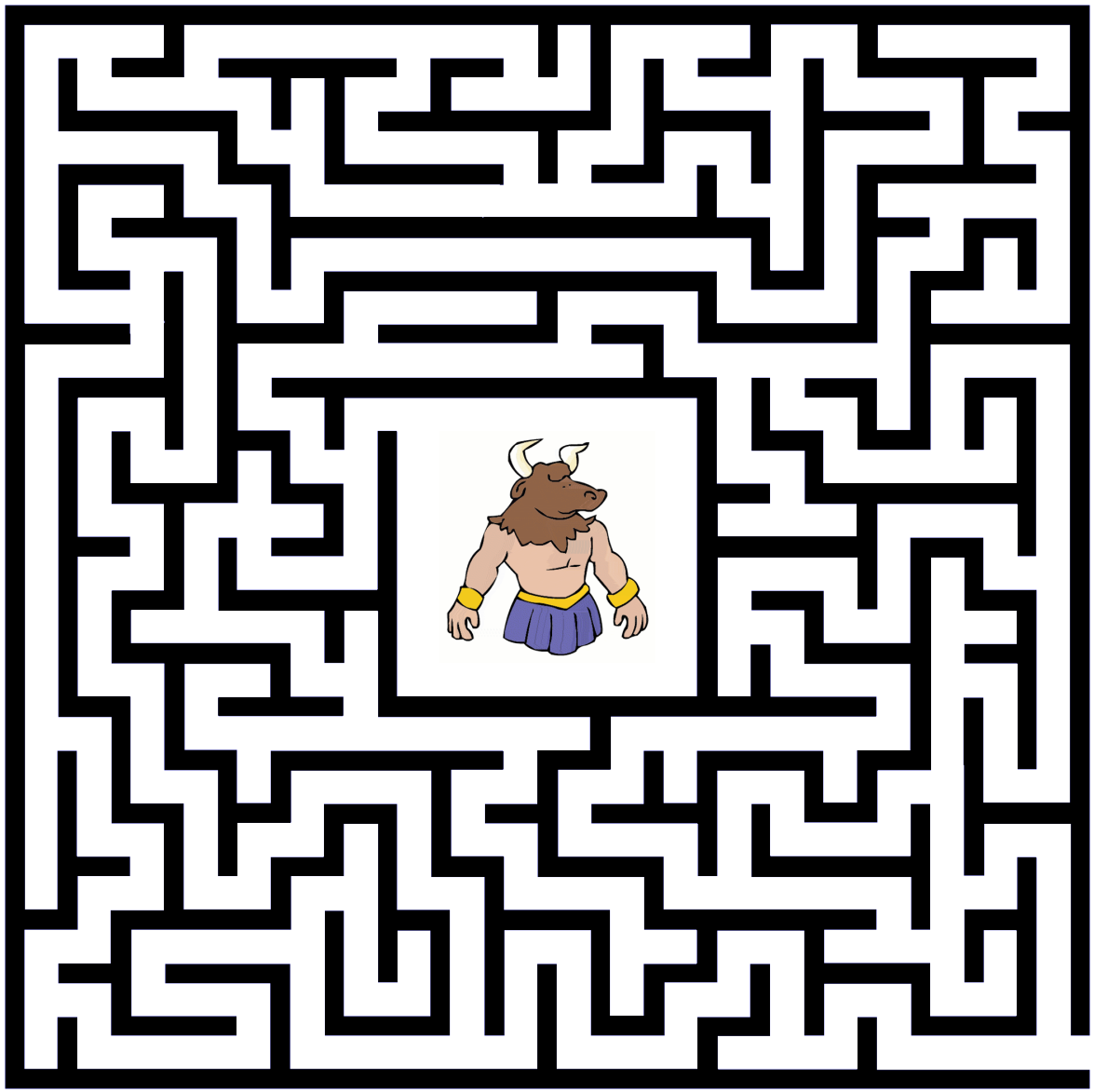
Nikola Beneš

19. listopadu 2018

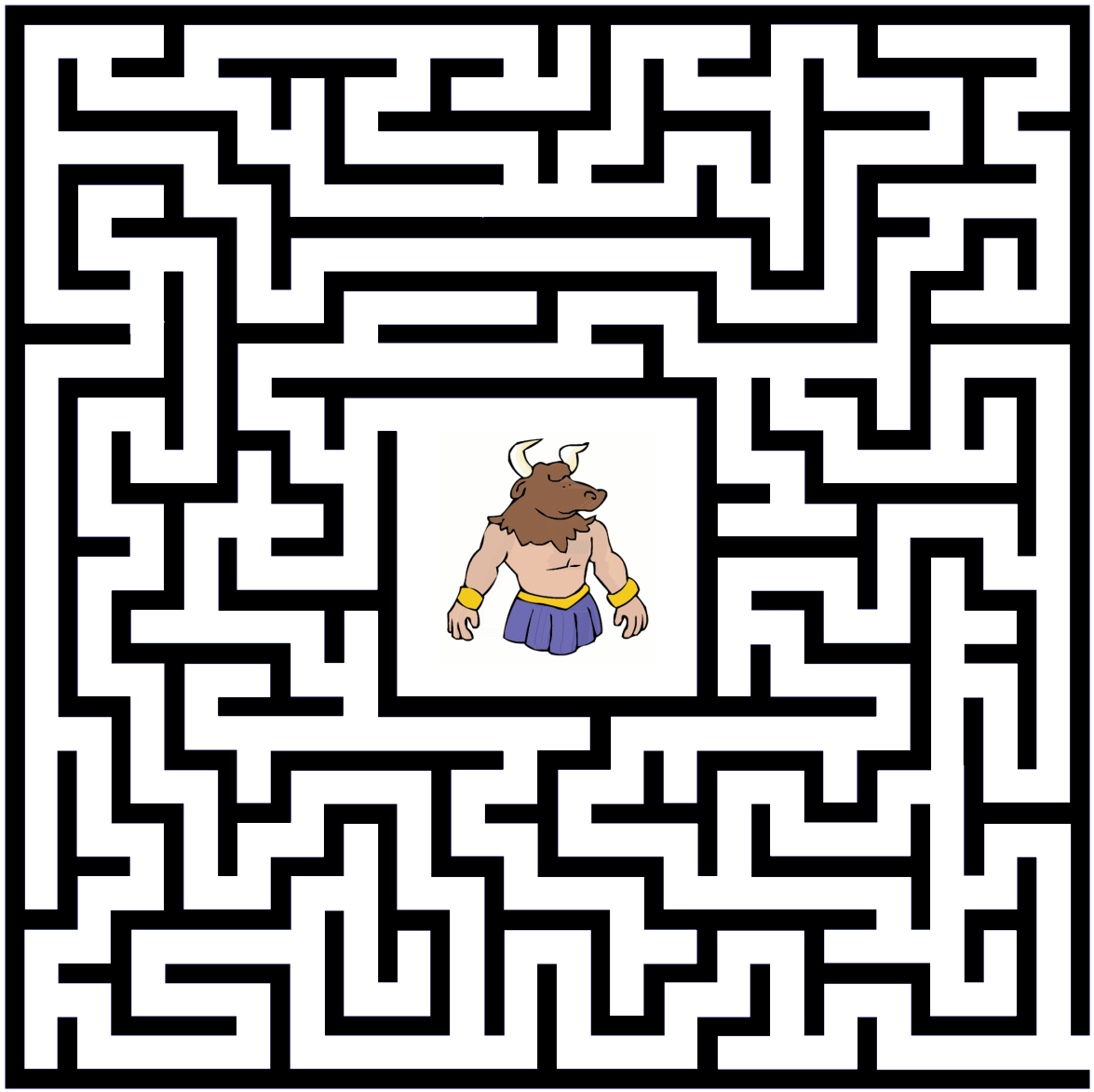
Příběh

Příběh

Příběh



Příběh



Jak projít bludiště s klubkem niti?

Varianta 1: Jen nit

- cestou tam se nit nikdy nenavíjí zpět
  - nit se navíjí zpět až při cestě ven z bludiště

Jak projít bludiště s klubkem niti?

Varianta 1: Jen nit

- cestou tam se nit nikdy nenavíjí zpět
  - nit se navíjí zpět až při cestě ven z bludiště

Varianta 2: Nit a kus křídy

- nit se navíjí zpět i cestou tam
- křída pro značení chodeb, kterými nit už dříve vedla, ale už nevede

Jak projít bludiště s klubkem niti?

Varianta 1: Jen nit

- cestou tam se nit nikdy nenavíjí zpět
  - nit se navíjí zpět až při cestě ven z bludiště

Varianta 2: Nit a kus křídy

- nit se navíjí zpět i cestou tam
- křída pro značení chodeb, kterými nit už dříve vedla, ale už nevede

Varianta 3: Nit a mapa

- když nemůžeme modifikovat bludiště
- značení chodeb, kterými už nechci jít
- značení křižovatek, kam už nechci jít

Algoritmus

Wienerův algoritmus, Trémauxův algoritmus, Tarryho algoritmus

- konec 19. století
- Trémaux, Tarry: stačí jen křída a dva druhy značek
- v podstatě zjednodušené verze DFS

Depth-First Search (DFS)

- Robert E. Tarjan, 70. léta 20. století
- procházení do hloubky
- implementace procházení bludiště s nití a mapou
- obecnější (libovolný *orientovaný* i *neorientovaný* graf)

Algoritmus

Wienerův algoritmus, Trémauxův algoritmus, Tarryho algoritmus

- konec 19. století
- Trémaux, Tarry: stačí jen křída a dva druhy značek
- v podstatě zjednodušené verze DFS

Depth-First Search (DFS)

- Robert E. Tarjan, 70. léta 20. století
- procházení do hloubky
- implementace procházení bludiště s nití a mapou
- obecnější (libovolný *orientovaný* i *neorientovaný* graf)

- nit: *zásobník*
- mapa: příznaky vrcholů a hran grafu

Procházení do hloubky

```
1 procedure dfs(G, v)
2     mark v as visited
3     while there exists an unvisited outgoing edge of v do
4         (v, w) ← choose one of the unvisited outgoing edges of v
5         mark (v, w) as visited
6         if w not visited then
7             dfs(G, w)
```

- kde je zásobník?
- jak poznáme, které vrcholy jsou na zásobníku?
  (tj. kterými křižovatkami vede nit?)
- a proč to chceme umět poznat?

Procházení do hloubky

```
1 procedure dfs(G, v)
2     mark v as visited
3     while there exists an unvisited outgoing edge of v do
4         (v, w) ← choose one of the unvisited outgoing edges of v
5         mark (v, w) as visited
6         if w not visited then
7             dfs(G, w)
```

- kde je zásobník?
- jak poznáme, které vrcholy jsou na zásobníku?
  (tj. kterými křižovatkami vede nit?)
- a proč to chceme umět poznat?
  - v orientovaných grafech nám to dává zajímavé informace
  - a co v neorientovaných grafech?

Procházení do hloubky

```
1 procedure dfs(G, v)
2     mark v as visited
3     while there exists an unvisited outgoing edge of v do
4         (v, w) ← choose one of the unvisited outgoing edges of v
5         mark (v, w) as visited
6         if w not visited then
7             dfs(G, w)
```

- kde je zásobník?
- jak poznáme, které vrcholy jsou na zásobníku?
  (tj. kterými křižovatkami vede nit?)
- a proč to chceme umět poznat?
  - v orientovaných grafech nám to dává zajímavé informace
  - a co v neorientovaných grafech?

- odbočka: *volba* vs. *náhodná volba*

Procházení do hloubky

```
1 procedure dfs(G, v)
2     mark v as gray
3     while there exists an unvisited outgoing edge of v do
4         (v, w) ← choose one of the unvisited outgoing edges of v
5         mark (v, w) as visited
6         if w is white then
7             dfs(G, w)
8     mark v as black
```

- na začátku je každý vrchol *bílý*
- vrcholy, kterou jsou na zásobníku, jsou *šedé*
  (tj. křižovatky, kterými vede nit)
- vrcholy, které už nejsou na zásobníku, jsou *černé*
  (tj. křižovatky, kterými už nevede nit)
- jak efektivně ověřovat podmínku příkazu **while**?

Procházení do hloubky

```
1 procedure dfs(G, v)
2     mark v as gray
3     for w in successors of v do
4         mark (v, w) as visited
5         if w is white then
6             dfs(G, w)
7     mark v as black
```

- co je všechno na zásobníku rekurze?

Procházení do hloubky

```
1 procedure dfs(G, v)
2     mark v as gray
3     for w in successors of v do
4         mark (v, w) as visited
5         if w is white then
6             dfs(G, w)
7     mark v as black
```

- co je všechno na zásobníku rekurze?
  - lokální proměnné, návratové adresy
- jaké (podstatné) lokální proměnné se tady vyskytují?

Procházení do hloubky

```
1 procedure dfs(G, v)
2     mark v as gray
3     for w in successors of v do
4         mark (v, w) as visited
5         if w is white then
6             dfs(G, w)
7     mark v as black
```

- co je všechno na zásobníku rekurze?
  - lokální proměnné, návratové adresy
- jaké (podstatné) lokální proměnné se tady vyskytují?
  - aktuální vrchol
  - *iterátor* do seznamu následníků aktuálního vrcholu

Procházení do hloubky iterativně

```
1  procedure dfs(G, u)
2      S ← empty stack
3      mark u as gray
4      push(S, (u, 1))
5      while S is not empty do
6          (v, k) ← pop(S)
7          if k ≤ number of v's successors then
8              push(S, (v, k + 1))
9              w ← kth successor of v
10             mark (v, w) as visited
11             if w is white then
12                 mark w as gray
13                 push(S, (w, 1))
14         else
15             mark v as black
```

Procházení do hloubky iterativně

Co když nemám předem danou iterovatelnou strukturu?

- místo iterátorů kandidáti

Procházení do hloubky iterativně

Co když nemám předem danou iterovatelnou strukturu?

- místo iterátorů kandidáti
- generování všech následníků při první návštěvě vrcholu
  a uložení všech těchto následníků na zásobníku spolu s iterátorem

Procházení do hloubky iterativně

Co když nemám předem danou iterovatelnou strukturu?

- místo iterátorů kandidáti
- generování všech následníků při první návštěvě vrcholu
  a uložení všech těchto následníků na zásobníku spolu s iterátorem
- totéž jako minulý bod, ale bez ukládání iterátoru – jak?

Procházení do hloubky iterativně

Co když nemám předem danou iterovatelnou strukturu?

- místo iterátorů kandidáti
- generování všech následníků při první návštěvě vrcholu
  a uložení všech těchto následníků na zásobníku spolu s iterátorem
- totéž jako minulý bod, ale bez ukládání iterátoru – jak?
  - dva druhy objektů na zásobníku – šedé vrcholy a neoznačené hrany

Procházení do hloubky iterativně

Co když nemám předem danou iterovatelnou strukturu?

- místo iterátorů kandidáti
- generování všech následníků při první návštěvě vrcholu
  a uložení všech těchto následníků na zásobníku spolu s iterátorem
- totéž jako minulý bod, ale bez ukládání iterátoru – jak?
  - dva druhy objektů na zásobníku – šedé vrcholy a neoznačené hrany

Poslední varianta může svádět k jisté nepravdivé generalizaci, zvláště když

- ignorujeme obarvování vrcholů na černo,
- nebudeme nijak zpracovávat hrany,
- místo hran si budeme ukládat jen jejich cílové vrcholy.

Procházení do hloubky iterativně

```
1  procedure dfs(G, v)
2      S ← empty stack
3      mark u as visited
4      push(S, u)
5      while S is not empty do
6          v ← pop(S)
7          if v not visited then
8              mark v as visited
9              for w in successors of v do
10                 push(S, w)
```

- „vždyť to vypadá jako BFS, jen místo fronty je zásobník“

Procházení do hloubky iterativně

```
1  procedure dfs(G, v)
2      S ← empty stack
3      mark u as visited
4      push(S, u)
5      while S is not empty do
6          v ← pop(S)
7          if v not visited then
8              mark v as visited
9              for w in successors of v do
10                 push(S, w)
```

- „vždyť to vypadá jako BFS, jen místo fronty je zásobník“
  - **NE**

Procházení do hloubky iterativně

```
1  procedure dfs(G, v)
2      S ← empty stack
3      mark u as visited
4      push(S, u)
5      while S is not empty do
6          v ← pop(S)
7          if v not visited then
8              mark v as visited
9              for w in successors of v do
10                 push(S, w)
```

- „vždyť to vypadá jako BFS, jen místo fronty je zásobník“
  - **NE**, správné BFS označuje vrcholy jako navštívené *ve chvíli, kdy jsou vloženy do fronty*

Procházení do hloubky iterativně

```
1  procedure dfs(G, v)
2      S ← empty stack
3      mark u as visited
4      push(S, u)
5      while S is not empty do
6          v ← pop(S)
7          if v not visited then
8              mark v as visited
9              for w in successors of v do
10                 push(S, w)
```

- „vždyť to vypadá jako BFS, jen místo fronty je zásobník“
  - **NE**, správné BFS označuje vrcholy jako navštívené *ve chvíli, kdy jsou vloženy do fronty*
- pro naše účely je tohle nepoužitelné, proč?

Odbočka: Prohledávání do šířky

```
1  procedure bfs(G, v)
2      Q ← empty queue
3      mark v as visited
4      enqueue(Q, v)
5      while Q is not empty do
6          v ← dequeue(Q)
7          for w in successors of v do
8              if w not visited then
9                  mark w as visited
10                 enqueue(Q, w)
```

- kdybych do tohoto algoritmu místo fronty dal zásobník, dostanu tzv. *PseudoDFS*
  - není to korektní DFS (ani pořadí prvního objevení vrcholů)
  - a už vůbec to není fyzicky realizovatelné robotem

Procházení do hloubky – fyzická realizace robotem

- potřebujeme zaručit, aby se robot nacházel přesně na místě aktuální vrcholu (tj. vrcholu zásobníku)
- zejména tedy je třeba se fyzicky vracet (*backtracking*)
  - a proto nutně potřebujeme variantu s obarvováním na černo

Procházení do hloubky – fyzická realizace robotem

- potřebujeme zaručit, aby se robot nacházel přesně na místě aktuální vrcholu (tj. vrcholu zásobníku)
- zejména tedy je třeba se fyzicky vracet (*backtracking*)
  - a proto nutně potřebujeme variantu s obarvováním na černo
- ve **chvíli obarvení vrcholu na černo** se podíváme na vrchol zásobníku – tam je křižovatka, kam se máme vrátit

Procházení do hloubky robotem

```
1  procedure dfs(G, u)
2      S ← empty stack; mark u as gray; push(S, (u, 1))
3      while S is not empty do
4          (v, k) ← pop(S)
5          if k ≤ number of directions from v then
6              push(S, (v, k + 1))
7              w ← kth direction from v
8              if move from v to w possible and w is white then
9                  mark w as gray
10                 move from v to w
11                 push(S, (w, 1))
12         else
13             (p, ?) ← top(S)
14             mark v as black
15             move from v to p
```

Zajímavé implementační otázky

- jak volit pořadí následníků?
  - nápad: jeď rovně, dokud to jde

Zajímavé implementační otázky

- jak volit pořadí následníků?
  - nápad: jeď rovně, dokud to jde
- mohlo by být lepší křižovatku tvaru + projet rovně, i když už je šedá?
  - jak se s tím vypořádat v algoritmu?
  - „smyčky na zásobníku“

Zajímavé implementační otázky

- jak volit pořadí následníků?
  - nápad: jeď rovně, dokud to jde
- mohlo by být lepší křižovatku tvaru + projet rovně, i když už je šedá?
  - jak se s tím vypořádat v algoritmu?
  - „smyčky na zásobníku“
- detekce zdí za běhu
  - jak to pomůže při volbě následníků?
  - detekce vzdálených zdí?

Zajímavé implementační otázky

- jak volit pořadí následníků?
  - nápad: jeď rovně, dokud to jde
- mohlo by být lepší křižovatku tvaru + projet rovně, i když už je šedá?
  - jak se s tím vypořádat v algoritmu?
  - „smyčky na zásobníku“
- detekce zdí za běhu
  - jak to pomůže při volbě následníků?
  - detekce vzdálených zdí?
- efektivnější backtracking
  - může se stát, že umím dojet na nejbližší nenavštívené místo nějak efektivněji?
  - co kdybych se po nalezení cíle chtěl vrátit na start?
  - hint: BFS
    (ale co když nemám odhalený celý graf?)