Théseus hledá Minotaura

- Z Algoritmizace znáte problém Thésea (najít Minotaura).
- Théseus dostal od Ariadny niť – a kyblík s barvou (a malířský váleček).
- Théseův algoritmus:
  1. Najdi Minotaura,
  2. zabij Minotaura.
- Nás zajímá jen první část.

Théseův algoritmus

hledání Minotaura

- Je-li tu Minotaurus, GOTO 2 (tj. `zabij_minotaura()`).
- Jinak: Vidíme dosud nenavštívenou chodbu? $\Rightarrow$ Chodbou projdeme a označíme ji jako navštívenou.
- Jinak: Lze-li namotat niť, namotáme niť.
- Nelze-li namotat niť (a nevidíme dosud nenavštívenou chodbu)? $\Rightarrow$ Konec.
- Analýza algoritmu byla na Algoritmizaci (konečnost a parciální správnost).

Théseův algoritmus

jiná varianta

- Je-li tu Minotaurus, GOTO 2 (tj. `zabij_minotaura()`).
- Jinak: Vidíme dosud nenavštívenou chodbu a do současné místnosti vede niť jen za námi? $\Rightarrow$ Chodbou projdeme a označíme ji jako navštívenou.
- Jinak: Lze-li namotat niť, namotáme niť.
- Nelze-li namotat niť (a nevidíme dosud nenavštívenou chodbu)? $\Rightarrow$ Konec.
- Korektnost algoritmu plyne stejně (jako u minulého).

Théseův algoritmus

vyplavování Minotaura

- Utěsni Labyrint,
- do vchodu zaveď hadici z hydrantu,
- pusť vodu,
- počkej, až bude Labyrint pod vodou ($\Leftarrow$ `zabij_minotaura()`),
- konec.

Prohledávání grafu

do hloubky, s návratem a do šířky.

- První algoritmus je prohledání do hloubky,
- druhý prohledání s návratem,
- třetí prohledání do šířky (a. k. a. algoritmus vlny),
- všechny najdou Minotaura (je-li v Labyrintu).
- Jak je naprogramovat?

Krok stranou

aneb kam král nemůže

- Na šachovnici jsou některá pole zakázaná (třeba obsazena figurkami vlastní barvy),
- na jednom poli stojí král,
- chceme najít všechna pole, kam král může.
- Jak to naprogramovat?

Kam kráľ nemůže

jak to naprogramovat?

- Vytvoříme zásobník, označíme všechny vrcholy jako nenavštívené, `push`(startovní pole)
- `while` zásobník neprázdný:
  - `ted=pop()`
  - `for v in sousedi(ted):`
    - Není-li v navštívený, označ ho jako navštívený a `push(v)`

Implementace

bude předvedena přímo na místě

Využije toho, co už umíme.
`sachovnice=[[0]*8]*8`, vyplníme načtením ze souboru, `push` a `pop` bylo.
Na místě napsat načítání, zpracování, výpis (čísel na šachovnici).

Jiný algoritmus

pro tentýž problém

- Vytvoříme frontu, označíme všechny vrcholy jako nenavštívené, `enqueue`(startovní pole)
- `while` fronta neprázdná:
  - `ted=dequeue()`
  - `for v in sousedi(ted):`
    - Není-li v navštívený, nastav mu vzdálenost `ted`+1, `enqueue(v)`

Souvislosti

a jejich stíny

- Jak souvisí Théseus a Král?
- Je to totéž, jen na speciálním grafu (vrcholy jsou pole šachovnice, hrany možnosti přechodu).
- Od obecného případu nás dělí jen reprezentace grafu. Možnosti?
- Matice sousednosti, incidence (matice jsou dvourozměrné seznamy), seznam sousedů (je seznam seznamů).
- Hledání sousedů provedeme podle matice sousednosti, označovat budeme vrcholy.
- Hledání se zásobníkem je prohledání do hloubky, to s frontou zase do šířky (a hledá nejkratší cestu v neohodnoceném grafu).

Další algoritmus

ještě uvidíte několikrát

- Hranám přidáme délku,
- jako datovou strukturu použijeme prioritní frontu, v níž budou vrcholy setříděné podle dosud zjištěné vzdálenosti, odebíráme vrchol s nejmenší vzdáleností,...
- a vznikne Dijkstrův algoritmus hledající nejkratší cestu v nezáporně ohodnoceném grafu.
- Při hledání lze též využít rekurzi (ale jen při hledání do hloubky). Jak?

Prohledání s rekurzí

opět předvést na místě

- Jak se prohledává soused?
- Úplně stejně jako my.
- Nemáme na to již nějakého odborníka?
- Povšimněte si, že zásobník volání funkce bude simulovat zásobník z našeho algoritmu.

Další optimalizační úlohy

řešitelné rekurzí

- Maximální klika (úplný podgraf),
- maximální nezávislá množina (prázdný indukovaný podgraf),
- dominance (minimální počet figurek ohrožující celou šachovnici),
- nezávislost (maximální počet figurek navzájem se neohrožujících),
- všechno jsou to zamaskované grafové optimalizační problémy,
- lze je řešit vyzkoušením všech možností, které s výhodou vygenerujeme rekurzí (alias cvičení "vypište všechna n-ciferná čísla v zadané číselné soustavě").

Teorie her

- Kombinatorická hra je hrou dvou hráčů. Stav hry je určen pozicí nějakých předmětů. Všechny zúčastněné předměty jsou viditelné. Jde o tzv. hru s úplnou informací.
- Příklad: Nimm, Podivná hra, Dáma, Šachy, Halma, Mlýn, Otrávená čokoláda...
- Kombinatorickými hrami nejsou: Poker, Prší, Mariáš, Black Jack, závody formulí...
- Zaměříme se na hrací část, ne na vstup a výstup.
- U her předpokládáme, že hrají rozumně se chovající jedinci (s motivací vyhrát).
- Nejjednodušší příklad: Hra Al-Capone a Babinský na sebe práskají.

Shannonova věta

Theorem (Shannon)

Každá kombinatorická hra má pro některého z hráčů neprohrávající strategii.

Důkaz.

Náznak: Buďto platí, že si jeden z hráčů může vynutit zacyklení hry (a tak neprohrát), nebo budeme zkoumat predikáty:
Existuje náš tah, že pro každý tah protihráče existuje náš tah, že pro každý tah protihráče... protihráč prohraje.
Pro každý náš tah existuje tah protihráče, že pro každý náš tah... my prohrajeme.
Formule jsou konečné, počty tahů jsou také konečné, jsou to vzájemně negace a lze je algoritmicky rozhodnout. □

Graf hry

- Ke hře (případně její instanci) definujeme orientovaný graf:
- Vrcholy: Stavy hry,
- Hrany: Možnosti přechodů mezi jednotlivými stavy.
- Příklad pro Nimm, kdy odebíráme 1 nebo 2 sirky (na tabuli).
- Každému stavu můžeme přiřadit barvu říkající, zda se odtud vyhrává nebo prohrává.

Příklad grafů her

- Na hracím plánu tvaru orientovaného grafu vyrážíme z určeného vrcholu. Taháme jedním padesátníkem.
- Máme dojet do jednoho z cílových vrcholů. Kdo dojede, vyhraje.
- Graf hry máme přímo zadaný a jde jen o to, který vrchol vyhrává.
- Podivná hra: Graf hry si zakreslíme na šachovnici. Vrcholy jsou políčka, hrany vedou tudy, kudy může figurka.
- Stačí říct, ze kterého vrcholu se vyhrává, prohrává, nebo zda existuje cyklus, po kterém mají oba hráči zájem bloudit (resp. zda si někdo z hráčů může bloudění po této kružnici vynutit).

AND-OR stromy

- Máme-li graf konečné hry, můžeme z něj postavit strom odpovídající hře.
- V tomto stromě nás zajímá, zda existuje větev, po které pokud pojedeme, tak vyhrajeme.
- Tato větev se pozná tak, že ve všech synech jejího koncového vrcholu existuje vyhrávající cesta, tedy ...
- buďto vyhrajeme v prvním synu, nebo ve druhém synu, nebo ve třetím...
- V k-tém synu vyhrajeme, jestliže protihráč prohraje v prvním synu a současně ve druhém synu a současně ve třetím synu... tohoto stavu.
- Prohraje tam, jestliže my (pro dotyčný stav) umíme vyhrát buďto v prvním synu, nebo ve druhém, anebo ve třetím...
- Podmínky AND a OR se stále střídají, proto AND-OR strom.