Algoritmická matematika 3

Backtracking

Petr Osička

DATA ANALYSIS AND MODELING LAB
Univerzita Palackého v Olomouci

Zimní semestr 2013

Základní idea

Metoda hrubé síly = „zkus všechny možnosti.“

Optimalizační problém

- algoritmus vygeneruje všechna přípustná řešení a pak z nich vybere to optimální.
- Příklad úlohy batohu, kde chceme najít podmnožinu vah takovou, že její suma je maximální ze všech podmnožin, jejíchž suma je menší než kapacita, algoritmus používající hrubou sílu nejdříve vygeneruje všechny podmnožiny vah a pak najde tu nejlepší.

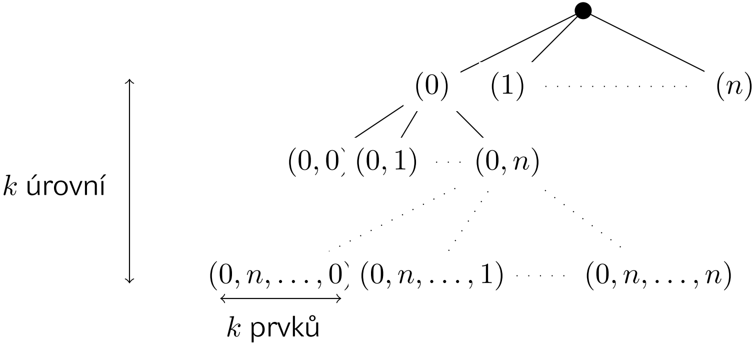
Rozhodovací problém

- Pokud pro danou instanci x rozhodovacího problému L (zde chápaného jako jazyk L), existuje vhodný certifikát, na jehož základě víme, že $x \in L$ (tedy, odpověd je ano), algoritmus fungující hrubou silou vygeneruje všechny možné kandidáty na takový certifikát a poté se jej v této množině pokusí najít.
- Příklad: SAT, certifikát je ohodnocení proměnných.

Metoda hrubé síly

Generování všech možností $\approx$ generování základních kombinatorických struktur.

Předpokládejme, že máme míčky n barev označených jako $\{0, 1, \ldots, n-1\}$ a chceme vytáhnout k míčků



Metoda hrubé síly

procedure GENERATE(X, $\langle a_1, \ldots, a_i \rangle$, k)
  if $i = k$ **then** ▷ Sekvence má k prvků, skonči rekurzi
    Zpracuj $a_1, \ldots, a_i$
  end if
  $S \leftarrow$ FILTER$(X, \langle a_1, \ldots, a_i \rangle)$ ▷ Vyfiltruj prvky, které lze dosadit za a_{i+1}
  for $x \in S$ **do**
    GENERATE$(X, \langle a_1, \ldots, a_i, x \rangle, k)$ ▷ Doplň sekvenci o x a pokračuj v rekurzi
  end for
end procedure

- Pokud FILTER vždy vrátí X, GENERATE generuje k-prvkové variace s opakováním,
- pokud FILTER vrátí $X - \{a_1, \ldots, a_i\}$, GENERATE generuje variace bez opakování.
- Generování spustíme zavoláním GENERATE s prázdnou sekvencí a.

Backtracking

Když algoritmus nalezne jednu sekvenci (dostane se do listu stromu), **vystoupí z rekurze o úroveň nahoru** a generuje další sekvence rekurzivním voláním na řádku 7.

Ořezání rekurzivního stromu

I. test na řádku 2 můžeme nahradit testem, který rozhodne, zda-li je $a_1, \dots, a_i$ už řešením, nebo se dá rychle doplnit na řešení (rychle většinou znamená libovolným výběrem zbylých prvků sekvence).

II. před rekurzivním voláním na řádku 7. můžeme testovat, jestli $a_1, \dots, a_i, x$ je prefixem sekvence, kterou chceme vygenerovat (tj. to, jestli má smysl pokračovat v generování zbytku sekvence). Pokud ne, Generate už rekurzivně nevoláme.

Backtracking pro SAT

SAT	
Instance:	formule výrokové logiky v konjunktivní normální formě φ
Řešení:	1 pokud je φ splnitelná, jinak 0.

- K sestavení algoritmu pro SAT stačí upravit Generate.
- Algoritmus generuje postupně všechna možná ohodnocení výrokových proměnných.
- Uvažme formuli φ, která je konjunkcí m literálů $C_1, \ldots, C_m$ a obsahuje k výrokových proměnných $x_1, \ldots, x_k$. Ohodnocení těchto proměnných můžeme chápat jako sekvenci $\langle a_1, \ldots, a_k \rangle$ složenou z 1 a 0, přitom a_i je ohodnocení proměnné x_i.

Backtracking pro SAT

Dále si pro klauzuli C_j definujeme následující dva predikáty

- $\mathcal{F}(C_j, \langle a_1, \ldots, a_i \rangle)$ je pravdivý, právě když proměnné obsažené v C_j patří do $\{x_1, \ldots, x_i\}$ a vzhledem k ohodnocení $\langle a_1, \ldots, a_i \rangle$ neobsahuje C_j žádný pravdivý literál,
- $\mathcal{T}(C_j, \langle a_1, \ldots, a_i \rangle)$ je pravdivý, pokud literál alespoň jedné proměnné z C_j patřící do $\{x_1, \ldots, x_i\}$ je při ohodnocení $\langle a_1, \ldots, a_i \rangle$ pravdivý.

Ořezání stromu rekurze

- formule φ je pravdivá, právě když jsou pravdivé všechny klausule, tj. právě když je v každé klausuli alespoň jeden pravdivý literál.
- Pokud je $\mathcal{T}(C_j, \langle a_1, \ldots, a_i \rangle)$ pravdivý pro všechny klausule, víme, že φ je pravdivá pro libovolné doplnění $\langle a_1, \ldots, a_i \rangle$.
- Naopak, pokud je splněno $\mathcal{F}(C_j, \langle a_1, \ldots, a_i \rangle)$ alespoň pro jednu klausuli, je tato klausule pro libovolné doplnění $\langle a_1, \ldots, a_i \rangle$ nepravdivá, v důsledku čehož je nepravdivá i φ.

Backtracking pro SAT

procedure ES$(\varphi, \langle a_1, \ldots, a_i \rangle, k)$
    $E \leftarrow$ TRUE
    for $j \leftarrow 1$ to m **do**
        if not $\mathcal{T}(C_j, \langle a_1, \ldots, a_i \rangle)$ **then**
            $E \leftarrow$ FALSE
            break
        end if
    end for
    if E **then**
        return TRUE
    end if
    for $x \in \{0, 1\}$ **do**
        $E \leftarrow$ TRUE
        for $j \leftarrow 1$ to m **do**
            if $\mathcal{F}(C_j, \langle a_1, \ldots, a_i, x \rangle)$ **then**
                $E \leftarrow$ FALSE
                break
            end if
        end for
        if E **then**
            if ES$(\varphi, \langle \ldots, a_i, x \rangle, k)$ **then**
                return TRUE
            end if
        end if
    end for
    return FALSE
end procedure

Úloha n dam

Úkolem je spočítat kolika způsoby lze umístit n dam na šachovnici $n \times n$ tak, aby se vzájemně neohrožovaly.

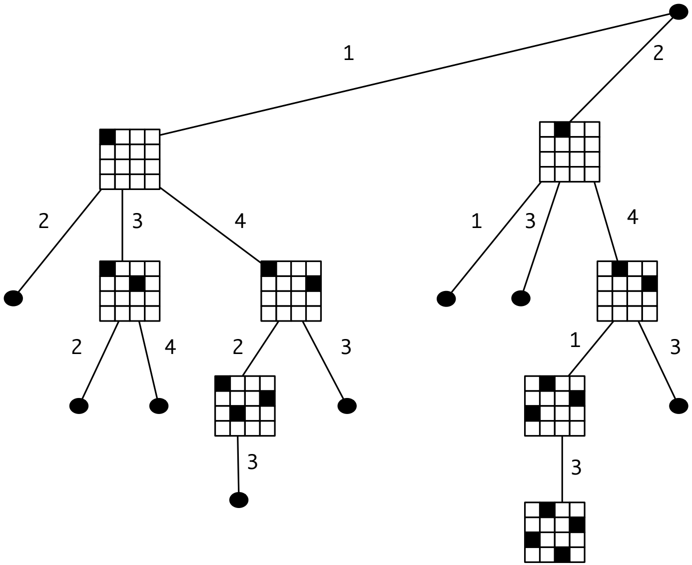
Naivní přístup k řešení tohoto problému by bylo vygenerovat všechna možná rozmístění dam na šachovnici a poté pro každé rozmístění otestovat, jestli je dámy neohrožují. Tento přístup je ale velmi neefektivní, protože takových rozmístění je $\binom{n^2}{n} = \frac{n^2!}{n!(n^2-n)!}$.

Backtracking

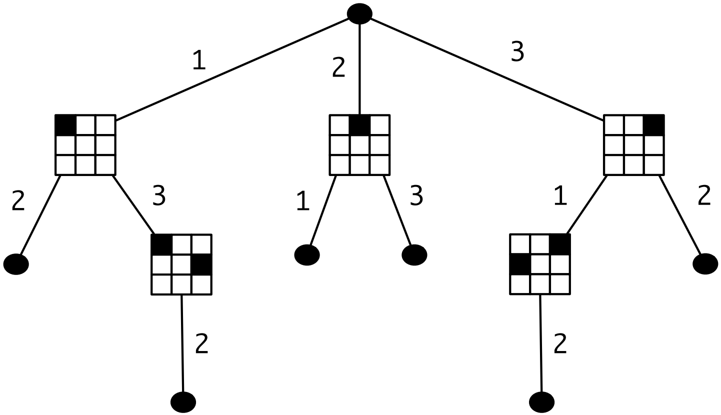
- Sloupce a řádky šachovnice si označíme čísly $1 \ldots n$.
- Pozice tedy můžeme generovat jako sekvence $\langle a_1, \ldots, a_n \rangle$, kde a_i je sloupec, ve kterém se nachází dáma na i-tém řádku.
- Protože v každém sloupci může být právě jedna dáma, můžeme jak kostru algoritmu využít Generate pro generování permutací.
- Jediná věc, kterou můžeme přidat, je test odpovídají podmínce II. Pro $a_1, \ldots, a_i$ otestujeme, jestli jestli se dámy v prvních i řádcích neohrožují diagonálně.

```
procedure Queens(⟨a_1, ..., a_i⟩, n)
    if i=n then
        return 1
    end if
    S ← {1, ..., n} \ {a_1, ..., a_i}
    c ← 0
    for a_{i+1} ∈ S do
        p ← TRUE
        for j ← 1 to i do
            if |j − (i + 1)| = |a_j − a_{i+1}| then
                p ← FALSE
                break
            end if
        end for
        if p then
            c ← c + Queens(⟨a_1, ..., a_i, a_{i+1}⟩, n)
        end if
    end for
    return c
```

$n = 4$



$n = 3$



Branch and Bound

Řekněmě, že hledáme řešení instance I minimalizačního problému.

- Idea: že budeme generovat prvky $x \in sol(I)$, počítat pro ně cenu $cost(x, I)$ a vybereme ten optimální.
- v průběhu algoritmu si pamatujeme cenu zatím nejlepšího nalezeného řešení,
- podmínkou je monotonie funkce $cost$ vzhledem k tomu, jak rozšiřujeme hledanou sekvenci. To znamená, že vždycky platí

$$cost(\langle a_1, \dots, a_i \rangle, I) \leq cost(\langle a_1, \dots, a_i, a_{i+1} \rangle, I).$$

- pokud $cost(\langle a_1, \dots, a_i \rangle, I)$ je větší než cena doposud nejlepšího nalezeného řešení, tak $\langle a_1, \dots, a_i \rangle$ není možné doplnit na optimální řešení.

Duální princip platí pro nerostoucí cenovou funkci a maximalizační problémy.

Set Cover

Jsou dány množina X a systém jejích podmnožin $\mathcal{S}$, který tuto množinu pokrývá (tj. platí $\bigcup \mathcal{S} = X$). Úkolem je nalézt co nejmenší podmnožinu $\mathcal{S}$ tak, aby stále pokrývala X.

Set Cover	
Instance:	konečná množina X, systém podmnožin $\mathcal{S} = \{S_i \mid S_i \subseteq X\}$ takový, že $\bigcup_{S_i \in \mathcal{S}} S_i = X$
Přípustná řešení:	$\mathcal{C} \subseteq \mathcal{S}$ takové, že $\bigcup_{S_i \in \mathcal{C}} S_i = X$
Cena řešení:	$cost(X, \mathcal{S}, \mathcal{C}) = \lvert\mathcal{C}\rvert$
Cíl:	minimum

Set Cover

Uvažme množinu Y. Pak pro každou podmnožinu Z této množiny můžeme definovat její **charakteristickou funkci** $\mu_Z : Y \to \{0,1\}$ jako

$$\mu_Z(y) = \begin{cases} 0 & y \notin Z \\ 1 & y \in Z \end{cases}$$

Je-li dána charakteristická funkce $\mu : Y \to \{0,1\}$ pak množinu $Set(\mu)$, která tuto funkci indukuje nalezneme jako

$$Set(\mu) = \{y \in Y \mid \mu(y) = 1\}.$$

Zafixujeme-li pořadí prvků, dá se každá podmnožina $Z \subseteq X$ zapsat jako sekvence

$$\langle \mu_Z(x_1), \mu_Z(x_2), \ldots, \mu_Z(x_n) \rangle$$

Označme $Set(\langle a_1, \ldots, a_i \rangle)$ podmnožinu, která odpovídá sekvenci $\langle a_1, \ldots, a_i \rangle$ doplněné na konci potřebným počtem 0.

Idea algoritmu pro Set Cover

Ke vygenerování všech podmnožin $\mathcal{S}$ tedy stačí generovat všechny n-prvkové sekvence nad $\{0,1\}$.

Ořezání stromu rekurze
Uvažujme situaci, kdy máme vygenerovánu sekvenci $\langle a_1, \dots, a_i \rangle$. (Pozor! $Set(\langle a_1, \dots, a_i \rangle)$ je podmnožina $\mathcal{S}$).

- Pokud $\bigcup Set(\langle a_1, \dots, a_i \rangle) \cup \bigcup(\mathcal{S} \setminus \{S_1, \dots, S_i\}) \neq X$, pak můžeme rekurzi ukončit, protože $\langle a_1, \dots, a_i \rangle$ není možné doplnit tak, aby pokryla celou množinu X.
- Pokud $\bigcup Set(\langle a_1, \dots, a_i \rangle) = X$, tak končíme rekurzi, protože hledáme minimální pokrytí X a přidávat další prvky do $Set(\langle a_1, \dots, a_i \rangle)$ proto nemá smysl.
- Pokud je $|Set(\langle a_1, \dots, a_i \rangle)|$ větší než velikost doposud nejmenšího nalezeného pokrytí, končíme rekurzi. Hledáme minimální pokrytí a pokračovat v přidávání prvků do $Set(\langle a_1, \dots, a_i \rangle)$ nemá smysl.

Backtracking pro Set Cover

$\mathcal{C} \leftarrow \mathcal{S}$

procedure OptimalSetCover($\langle a_1, \ldots, a_i \rangle, \mathcal{F}, X$)

  $Y \leftarrow \bigcup Set(\langle a_1, \ldots, a_i \rangle)$

  if $Y = X$ **then**

    $\mathcal{C} \leftarrow Set(\langle a_1, \ldots, a_i \rangle)$

    return

  end if

  if $Y \cup \bigcup \mathcal{F} \neq X$ **or** $|Set(\langle a_1, \ldots, a_i \rangle)| = |\mathcal{C}| - 1$ **then**

    return

  end if

  OptimalSetCover $(\langle a_1, \ldots, a_i, 1 \rangle, \mathcal{F} \setminus \{S_{i+1}\}, X)$

  OptimalSetCover $(\langle a_1, \ldots, a_i, 0 \rangle, \mathcal{F} \setminus \{S_{i+1}\}, X)$

end procedure

Úloha batohu

Backtracking se dá zkombinovat s algoritmem navrženým jiným způsobem a vylepšit tak jeho výkon (tak, že algoritmus spočítá řešení, které je v průměrném případě bližší optimálnímu řešení).

Úloha batohu	
Instance:	$\{(b, w_1, \dots, w_n) \mid b, w_1, \dots, w_n \in \mathbb{N}\}$
Přípustná řešení:	$sol(b, w_1, \dots, w_n) = \{C \subseteq \{1, \dots, n\} \mid \sum_{i \in C} w_i \leq b\}$
Cena řešení:	$cost(C, (b, w_1, \dots, w_n)) = \sum_{i \in C} w_i$
Cíl:	maximum

Úloha batohu

Idea algoritmu:

1. Pro $k \leq n$ nejdříve vygenerujeme všechny podmnožiny množiny $\{1, \dots, n\}$ do velikosti k, takové, že pro každou z nich je suma odpovídajících prvků menší než b.
2. každou z podmnožin se pokusíme rozšířit tak, že budeme greedy způsobem přidávat další prvky $\{1, \dots, n\}$. Vybereme takovou rozšířenou množinu, suma jejíchž prvků je největší

Poznámky

- Algoritmus určitě vrátí lepší řešení než samotný greedy algoritmus (proc?)
- První krok realizujeme pomocí backtrackingu

Krok 1

procedure GenerateCandidates($\langle a_1, \ldots, a_i \rangle, (b, w_1, \ldots, w_n), k$)
    if $\sum_{i \in Set(\langle a_1, \ldots, a_i \rangle)} w_i > b$ **then**
        return $\emptyset$
    end if
    if $|Set(\langle a_1, \ldots, a_i \rangle)| = k$ **or** $i = n$ **then**
        return $\{Set(\langle a_1, \ldots, a_i \rangle)\}$
    end if
    $X \leftarrow$ GenerateCandidates($\langle a_1, \ldots, a_i, 1 \rangle, (b, w_1, \ldots, w_n), k$)
    $Y \leftarrow$ GenerateCandidates($\langle a_1, \ldots, a_i, 0 \rangle, (b, w_1, \ldots, w_n), k$)
    return $X \cup Y$
end procedure

Krok 2

procedure EXTENDCANDIDATE$(C, (b, w_1, \ldots, w_n))$
    Vytvoř přiritní frontu Q z prvků $1, \ldots, n$ uspořádanou sestupně podle odpovídajících prvků $w_1, \ldots, w_n$
    while Q není prázdná **do**
        Odeber z Q prvek x
        if $x \notin C$ **and** $\sum_{i \in C} + w_x \leq b$ **then**
            $C \leftarrow C \cup \{x\}$
        end if
    end while
    return C
end procedure

Celý algoritmus

procedure KnapsakScheme$((b, w_1, \ldots, w_n), k)$
    $X \leftarrow$GenerateCandidates$(\langle\rangle, (b, w_1, \ldots, w_n), k)$
    $B \leftarrow \emptyset$
    for $x \in X$ **do**
        $A \leftarrow$ExtendCandidate$(x, (b, w_1, \ldots, w_n))$
        if $\sum_{x \in A} w_x > \sum_{x \in B} w_x$ **then**
            $B \leftarrow A$
        end if
    end for
    return B
end procedure