Algoritmické strategie

Prohledávání s návratem (Backtracking)

Prohledávání s návratem (backtracking)

- Systematický průchod všemi možnými konfiguracemi prostoru
- Obvykle prohledávání stromu do hloubky (DFS) - menší spotřeba paměti než při prohledávání do šířky (BFS)
- Pokud nelze postupovat dál směrem k listu, návrat k poslednímu rozhodovacímu místu a odtud jiný postup - strategie pokusu a omylu
- Typické úlohy: nejčastěji permutační, herní nebo grafové - všechny možné permutace nebo podmnožiny objektů, průchod všech koster grafu, všech cest mezi 2 vrcholy

Prohledávání s návratem (backtracking)

- Základní idea prohledávání s návratem:
- Jdi kupředu, dokud můžeš, a pamatuj si všechna rozhodnutí, která při tom děláš. Pokud již nemůžeš dál, vrať se na místo posledního rozhodnutí, rozhodni se jinak, a pokračuj stejným způsobem dál

Př.1: Pokrytí šachovnice

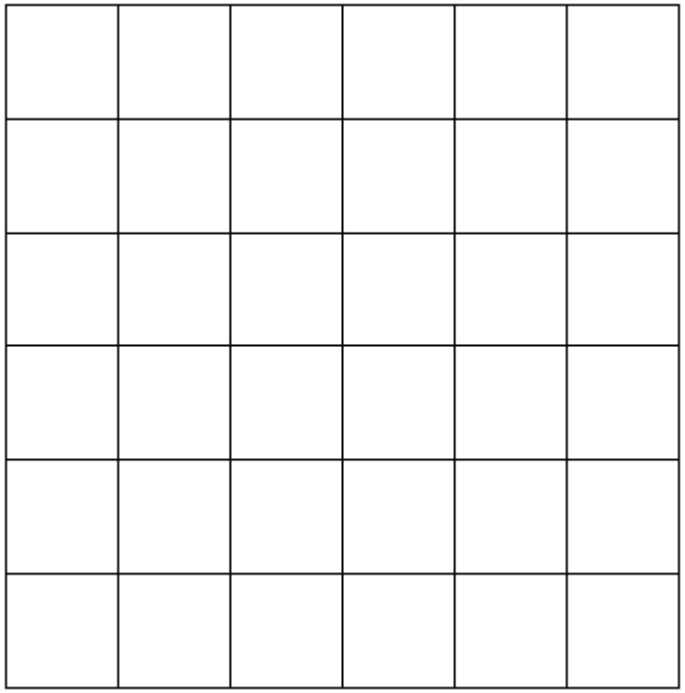
- Dána šachovnice a kůň v nějaké počáteční pozici, pokryjte šachovnici jeho tahy tak, aby každé políčko navštívil právě jednou
- Najděte alespoň 1 řešení, pokud existuje (pro některé velikosti šachovnice ne)

Řešení

Řešení: DFS a backtracking

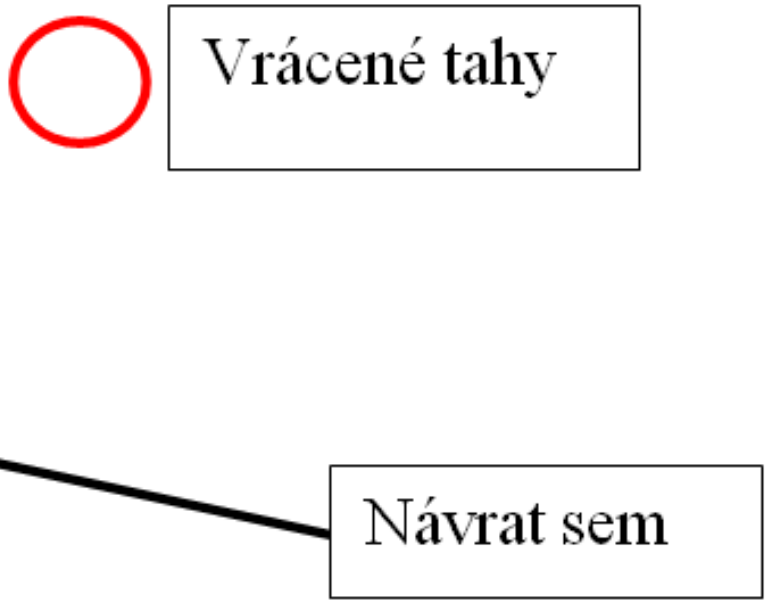
- Označovat tahy podle vzdálenosti od začátku pohybu
- Při uvíznutí návrat (backtracking)
- Na konci: očíslování dává cestu

Pokrytí šachovnice - BT



Pokrytí šachovnice BT

18	3	16		20	5
15		19	4	27	
2	17	14		6	21
13			24	9	26
	1		11	22	7
	12	23	8	25	10



Pokrytí šachovnice BT

18	3	16	29	20	5
15		19	4		
2	17	14	27	6	21
13	30		24	9	26
	1	28	11	22	7
29	12	23	8	25	10

Po několika tazích opět uvíznutí a návrat sem

Pokrytí šachovnice BT

- backtraking 7 tahů ….
- Několik dalších tahů zjistí, že tahy 24 a 25 se musí změnit

18	3	16	29	20	5
15	30	19	4		28
2	17	14	27	6	21
13		31	24	9	26
32	1		11	22	7
	12	23	8	25	10

Pokrytí šachovnice BT

- Možné řešení

18	3	16	27	20	5
15	26	19	4	35	28
2	17	14	29	6	21
13	30	25	36	9	34
24	1	32	11	22	7
31	12	23	8	33	10

Pokrytí šachovnice BT

Poznámky k řešení:

- Označovat neúspěšnou cestu, aby nedošlo k zacyklení
- Upřednostňovat rohy a kraje
- Tu a tam nechat prázdné místo kvůli pohyblivosti na konci

Snadné prohledávání s návratem (Backtrack)

N	Počet řešení	Počet testovaných pozic dámy		Zrychlení
		Hrubá síla (N^N)	Backtrack	
4	2	256	240	1.07
5	10	3 125	1 100	2.84
6	4	46 656	5 364	8.70
7	40	823 543	25 088	32.83
8	92	16 777 216	125 760	133.41
9	352	387 420 489	651 402	594.75
10	724	10 000 000 000	3 481 500	2 872.33
11	2 680	285 311 670 611	19 873 766	14 356.20
12	14 200	8 916 100 448 256	121 246 416	73 537.00

Rychlosti řešení problému osmi dam

Prohledávání s návratem (backtracking)

- Jedná se o vylepšení hledání řešení hrubou silou v tom, že velké množství potenciálních řešení může být vyloučeno bez přímého vyzkoušení. Algoritmus je založen na prohledávání do hloubky možných řešení

Prohledávání s návratem (backtracking)

- Můžeme si to hezky představit jako strom řešení, kde každý vrchol je částečné řešení; navrchu (v kořeni stromu) je počáteční stav, kdy nevíme vůbec nic, a listy stromu jsou všechna možná úplná řešení (z nichž většina je špatných). Když zjistíme, že některé částečné řešení je špatné, uřízneme celou jeho větev a vrátíme se ve stromu o úroveň výše

Backtracking

- Vhodné také pro paralelní implementaci (spousta práce, málo komunikace)
- Množina všech možných řešení se rozdělí na podmnožiny a ty se paralelně prohledávají
- Pokud nějaký procesor hotov a jiný ne, přerozdělí se dosud nehotová část množiny řešení mezi hotové procesory
- Kromě přerozdělování je komunikace nutná jen pro nalezení nejlepšího řešení z nejlepších řešení jednotlivých procesorů

N-Queen Problem

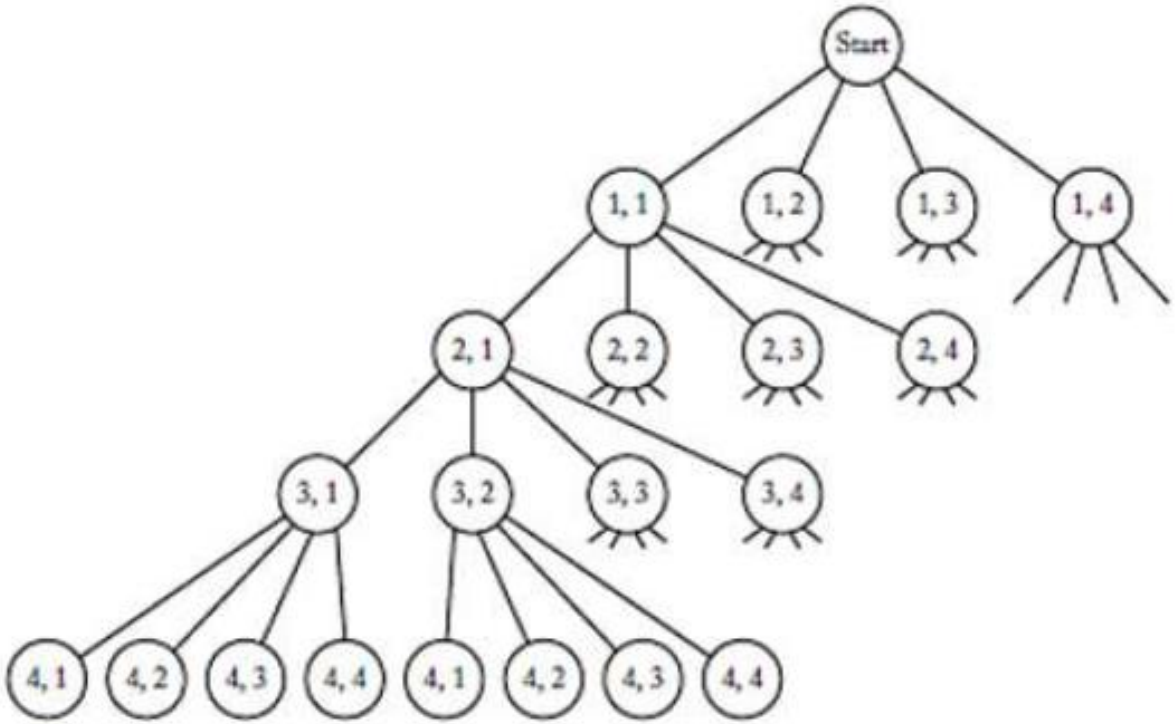
- Goal: position n queens on a n x n board such that no two queens threaten each other
  - No two queens may be in the same row, column, or diagonal
- Sequence: n positions where queens are placed
- Set: n^2 positions on the board
- Criterion: no two queens threaten each other

Construct State-Space Tree - Candidate Solutions

- Root – start node
- Column choices for first queen stored at level-1 nodes
- Column choices for second queen stored at level-2 nodes
- Etc.
- Path from root to a leaf is candidate solution
- Check each candidate solution in sequence starting with left-most path

Construct State-Space Tree –Candidate Solutions

- Example Figure below – no point in checking any candidate solutions from 2,1



Pruning

- DFS of state space tree
- Check to see if each node is promising
- If a node is non-promising, backtrack to node's parent
- Pruned state space tree – subtree consisting of visited nodes
- Promising function – application dependent
- Promising function n-queen problem: returns false if a node and any of the node's ancestors place queens in the same column or diagonal

Algorithm for N-Queens Problem

- All solutions to N-Queens problem
- Promising function:
  - 2 queens same row? col(i) == col(k)
  - 2 queens same diagonal? col(i) – col(k) == i – k || col(i) – col(k) == k - i

Analysis of queens theoretically difficult

- Upper bound on number of nodes checked in pruned state space tree by counting number of nodes in entire state space tree:
- 1 node level 0
- n nodes level 1
- n^2 nodes level 2 . . .
- N^n nodes level n

Total Number of nodes

$$1 + n + n^2 + n^3 + \ldots + n^n = (n^{n+1})/(n - 1)$$

Let n = 8

- Number of nodes in state space tree = 19173961
- Purpose of backtracking is to avoid checking many of these nodes
- Difficult to theoretically analyze savings by backtracking

Illustrate backtracking savings by executing code and counting nodes checked

- Algorithm 1 – DFS of state space tree without backtracking
- Algorithm 2 – checks no two queens same row or same column

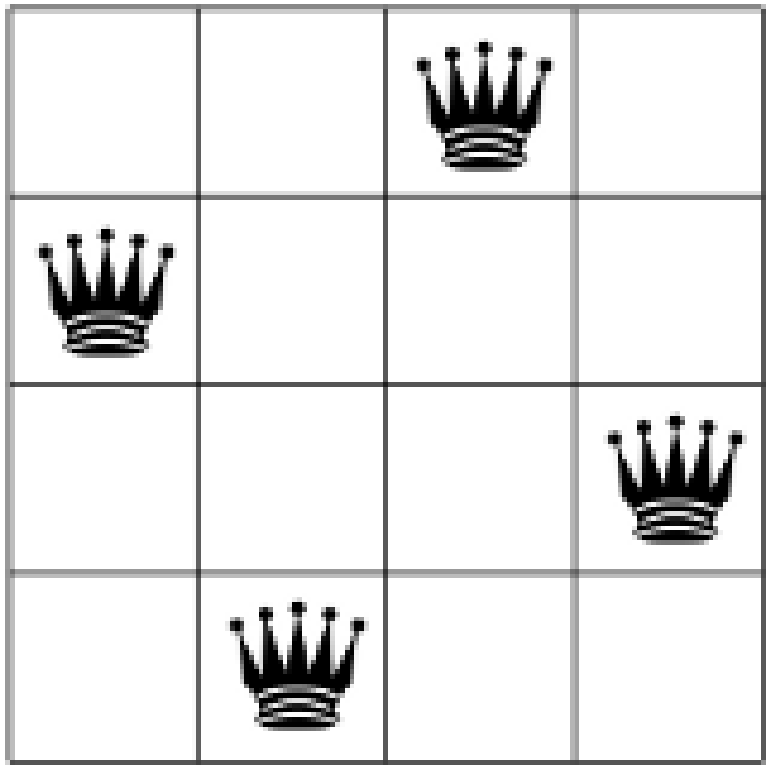
n	Number of Nodes Checked by Algorithm 1†	Number of Candidate Solutions Checked by Algorithm 2‡	Number of Nodes Checked by Backtracking	Number of Nodes Found Promising by Backtracking
4	341	24	61	17
8	19,173,961	40,320	15,721	2057
12	9.73×10^{12}	4.79×10^{8}	1.01×10^{7}	8.56×10^{5}
14	1.20×10^{16}	8.72×10^{10}	3.78×10^{8}	2.74×10^{7}

*Entries indicate numbers of checks required to find all solutions.

†Algorithm 1 does a depth-first search of the state space tree without backtracking.

‡Algorithm 2 generates the $n!$ candidate solutions that place each queen in a different row and column.

4-Queen Problem Solution



Typické aplikace BT

A) Optimalizační problémy, zejména komerční

- Obchodní cestující/vhodné linky pro autobusy
- Plánování (schůzky - sdružit seznam kandidátů se seznamem protikandidátů)
- Turnajové schéma (kandidáti i protikandidáti ze stejného seznamu)
- Výzkum trhu (testování výrobků)
- Sladění nabídky a poptávky
- Prohledávání adresáře s podadresáři

Typické aplikace BT

B) Hry

- Šach
- Dáma
- Piškvorky
- Reversi
- Go

Typické aplikace BT

C) Hádanky (a programátorské soutěže ☺)

- Křížovky
- Hádanky na stěhování a přesuny
- (Rubikova kostka, pokrývání šachovnice, stěhování piána)
- Solitéry (a: v co nejméně tazích)
- Vytváření bludiště a jeho řešení
- Sudoku
- Magické čtverce ...

Př.3: Odpory

- Zadání: Sestavte z daného souboru rezistorů odpor požadované hodnoty, pokud jde sestavit kombinací sériového a paralel. zapojení

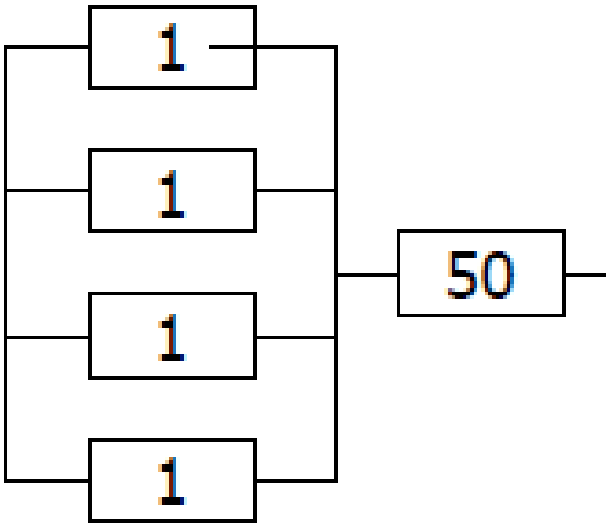
Př.3: Odpory

- Př.: Jsou k dispozici 2 rezistory o odporech 10 Ω a 20 Ω, požadovaný odpor je 50 Ω.
- Řešení: nelze sestavit

- Připomenutí:
- Sériově: $R=\Sigma R_i$
- Paralelně: $1/R=\Sigma 1/R_i$

Př.3: Odpory

- Př.: Jsou k dispozici rezistory o odporech (1 Ω, 1 Ω, 1 Ω ,1 Ω, 50 Ω), požadovaný odpor je 50.25 Ω
- Řešení:



Př.3: Odpory - řešení

- I pouze sériové kombinace - NP-úplný problém, takže použijeme BT
- Na začátku - pole nepoužitých odporů vstupních velikostí
- Z něj vybíráme nepoužité odpory a zkoušíme je spojovat sériově a paralelně
- Každým spojením označíme původní 2 odpory za použité a nový odpor přidáme na konec pole
- Pak rekurzivně znovu
- Pokud hledáme 1. možné řešení: Končíme s rekurzí, pokud dosažena požadovaná hodnota
- Pokud hledáme nejmenší počet rezistorů: pamatovat si řešení s nejmenším počtem rezistorů

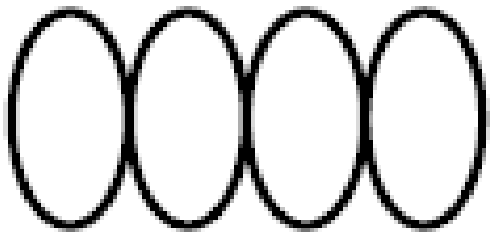
Př.3: Odpory - řešení

- Pokud už všechny odpory použity nebo už nelze dosáhnout požadované hodnoty (současné zapojení se sér.zapojením zbylých rezistorů má menší odpor než požadováno), návrat

- Zkuste doma:
- Vymyslete podrobný algoritmus řešící tuto úlohu

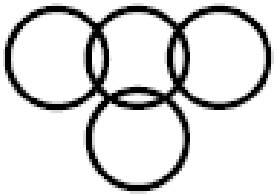
Př.4: Magický zapletenec

- Zkrácené znění: Řetízek má různě pospojované články, některé je třeba rozpojit, přesunout a jiné zas spojit, aby vznikl řetízek s 2 konci. Najděte řešení s minimem rozpojených článků
- Právě zadání hledat minimum dělá z úlohy kandidáta na BT

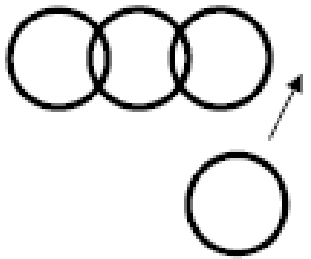


Př.4: Magický zapletenec

Příklad: 4 články spojené 1-2, 2-3, 2-4

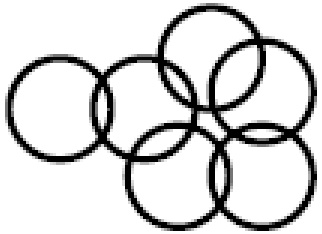


Řešení: rozpojit aspoň jeden článek, např. 4 připojit za 3

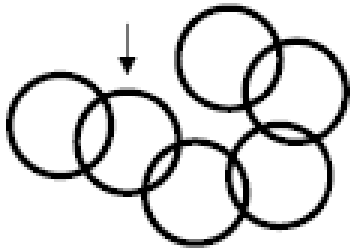


Př.4: Magický zapletenec

Příklad: 5 článků spojených 1-2, 2-3, 3-4, 4-1, 1-5



Řešení: rozpojit aspoň jeden článek, např. 1, 4 z něj pak můžeme vyvléknout



Př.4: Magický zapletenec

- Základní varianta bez urychlení:
- Vybereme množinu vrcholů a označíme je jako rozevřené
- Otestujeme, že nyní v grafu existují jen cesty (uzly stupně 2, bez kružnic) - pokud ne, vybraná množina není správná
- Pokud množina prošla 1. testem, zkontrolujeme, že máme dost rozevřených článků na to, aby šlo spojit v jeden řetěz
- Čas. složitost $O(2^{\text{počet vrcholů}})$

Př.4: Magický zapletenec

- Úkol pro vás: Vymyslete algoritmus, který najde nějaké řešení (ne nutně optimální) – rychlejší než předchozí