Seminár z algoritmizácie a programovania 1

Martin Bobák
Ústav informatiky
Slovenská akadémia vied

Obsah prednášky

Backtracking

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8

Backtracking
Prehľadávanie s návratom

Úloha:

- riešenie náročného problému tým, že rekurzívne prechádzame (systematicky) prípustné riešenia (spĺňajúce nejakú podmienku) a vyberieme to najlepšie
  - pri rekurzii skončíme výpočet, keď dosiahneme základný prípad
  - pri backtrackingu využijeme rekurziu (ako nástroj) na prehľadanie
- (potenciálne) riešenie konštruujeme inkrementálne

Backtracking
Prehľadávanie s návratom

v mnohých prípadoch vieme prechádzanie optimalizovať a niektoré vetvy vyhodnotiť ako nedosiahnuteľné

- modifikované prehľadávanie do šírky
- strom výpočtu prechádzame cez inorder

Backtracking

```
void backtracking(n, param):
    if (nasiel_som_najlepšie_riesenie):
        vypis_riesenie();
        return;

    for (riesenie = prve az posledne):
        if (je_pripustne(riesenie, n)) :
            zapamataj(riesenie, n);
            backtracking(n+1, param);
            zabudni(riesenie, n);
```

Motivácia

Aplikácie:

- matematická optimalizácia, teoretická informatika
- grafové algoritmy
- kombinatorika
- logické programovanie

Backtracking

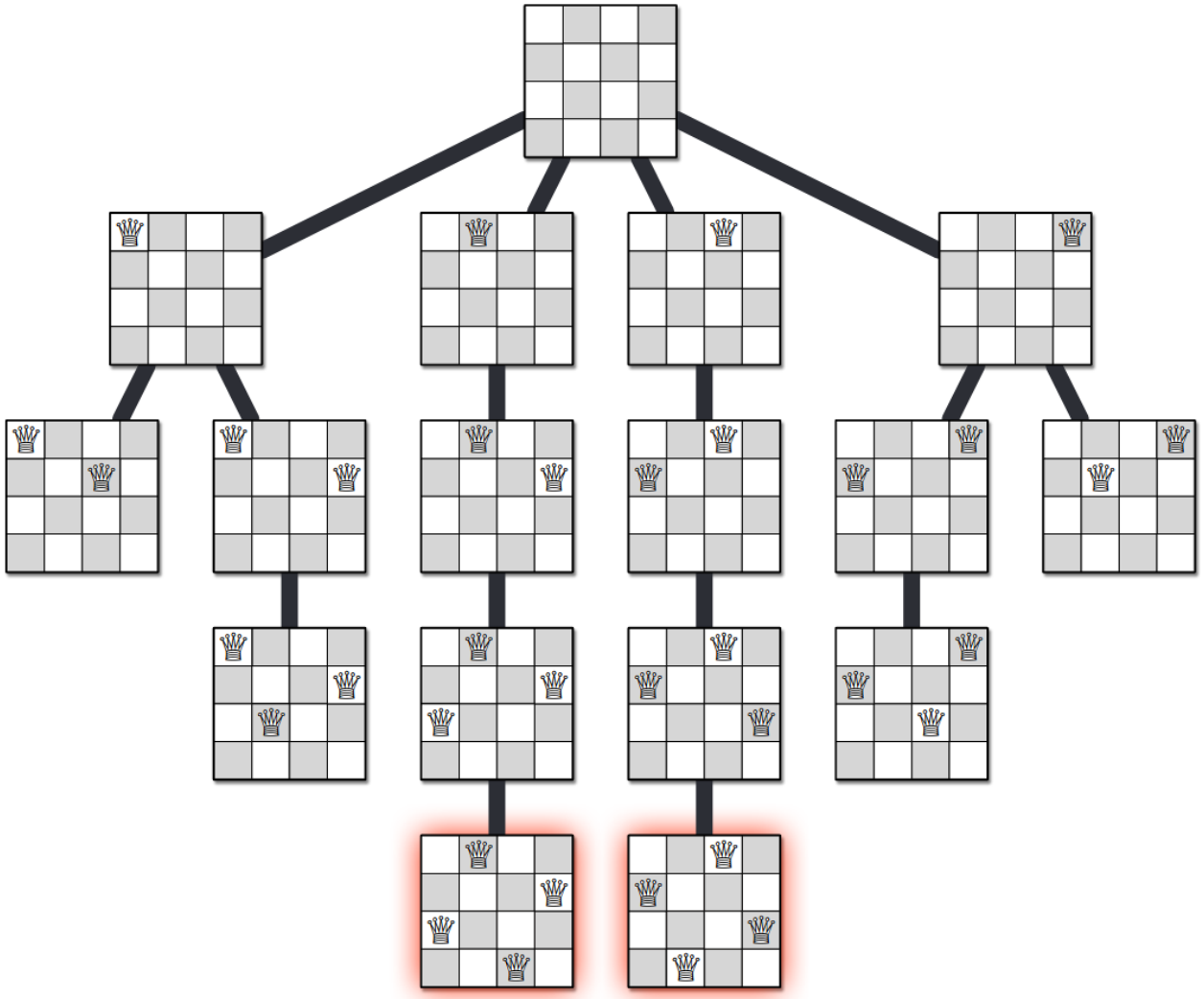
- pomerne všeobecný prístup (skúsim všetky potenciálne riešenia a vyberiem to najlepšie z nich)
- v "surovej" forme je neefektívny

Časová zložitosť: exponenciálna (napr. $O(2^n)$)

Triedenie:

skúšanie všetkých permutácií vs ("naivný") buble sort

Backtracking



Zdroj: Jeff Erickson. Algorithms. University of Illinois.

https://jeffe.cs.illinois.edu/teaching/algorithms

Problém N dám

Úloha: Majme šachovnicu s rozmermi NxN, chceme umiestniť N dám tak, aby sa vzájomne neohrozovali.

- šachovnicu budeme reprezentovať ako maticu NxN
- 1 reprezentuje miesto, kde sa nachádza dáma. Na ostatných políčkach sa nachádza 0.

Základný algoritmus

- vygenerujeme si všetky možné umiestnenia dám na šachovnici (=konfigurácia)
- ak konfigurácia spĺňa podmienky (t.j. dámy sa neohrozujú), našli sme hľadané rozloženie dám

Backtracking

- postupne hľadáme miesto pre dámu v aktuálnej konfigurácii
  - skontrolujeme, či ju neohrozuje iná dáma
  - ak takáto pozícia neexistuje, vrátime sa o úroveň vyššie (danú vetvu prehľadávania ukončíme)
    - je zrejmé, že uvažovaním ďalšej dámy nám nijako nepomôže nájsť hľadané riešenie

Backtracking

```
bool backtracking(int board[N][N], int col) {
    // base case: vsetky damy sú na sachovnici
    if (col >= N)
        return true;

    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            //zapamatame si riesenie
            board[i][col] = 1;

            if (backtracking(board, col + 1))
                return true;

            //zabudneme zlu vetvu
            board[i][col] = 0;
        } }

    /* dámu nevieme nikam umiestnit */
    return false;
}
```

Backtracking

```
bool isSafe(int board[N][N], int row, int col){
    int i, j;

    // riadok
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    // horna diagonala
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    // spodna diagonala
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}
```

Sudoku

- mriežka 9x9
- vyplnenie políčok mriežky číslami z intervalu <1,9> tak, aby sa v každom riadku a stĺpci nachádzalo dané číslo práve raz.
- mriežka je rozdelená 9 menších štvorcov 3x3, v každom štvorci sa musí nachádzať dané číslo práve raz
- vstupná mriežka môže byť predvyplnená
  - číslo 0 reprezentuje prázdne políčko

Backtracking

- systematicky skúsime doplniť na každé miesto chýbajúce číslo a následne skontrolujeme, či sme našli riešenie.

Časová zložitosť: $O(9^{n*n})$

Pamäťová zložitosť: $O(n*n)$

Backtracking

```
bool backtracking(int grid[N][N]) {
    int row, col;
    // Zisti ci su policka vyplnene
    if (!FindUnassignedLocation(grid, row, col))
        return true; // success!

    for (int num = 1; num <= 9; num++){

        if (isSafe(grid, row, col, num)){

            // Zapamatame si potencialne riesenie
            grid[row][col] = num;

            if (backtracking(grid))
                return true;

            // Zabudneme zle riesenie
            grid[row][col] = 0; }}

    return false;}
```

Backtracking

```
bool FindUnassignedLocation(int grid[N][N], int row, int col) {
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == 0)
                return true;
    return false; }

bool UsedInRow(int grid[N][N], int row, int num) {
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false; }

bool UsedInCol(int grid[N][N], int col, int num) {
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

Backtracking

```
bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int
num) {
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row + boxStartRow]
                    [col + boxStartCol] ==
                                        num)
                return true;
    return false; }

bool isSafe(int grid[N][N], int row,
            int col, int num) {
    return !UsedInRow(grid, row, num)
           && !UsedInCol(grid, col, num)
           && !UsedInBox(grid, row - row % 3,
                         col - col % 3, num)
           && grid[row][col] == 0;
}
```

Bludisko

- mriežka NxN s hodnotami 0 alebo 1
- počiatočná pozícia: [0][0]
- cieľ: [N-1][N-1]
- pohyb: dopredu, dole (zjednodušená verzia)
- prekážka: na danej pozícii je 0

Backtracking

- postupne skúšame oba smery berúc do úvahy prekážky
- (rekurzívne) budujeme cestu z počiatočnej pozície do cieľa

Časová zložitosť: $O(2^{n*n})$

Pamäťová zložitosť: O(n*n)

Backtracking

```
bool backtracking(int maze[N][N], int x, int y, int sol[N][N]){
    // x, y je ciel
    if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {
        sol[x][y] = 1;
        return true;
    }

    if (isSafe(maze, x, y) == true) {
        sol[x][y] = 1;

        if (backtracking(maze, x + 1, y, sol) == true)
            return true;

        if (backtracking(maze, x, y + 1, sol) == true)
            return true;

        sol[x][y] = 0;
        return false;
    }
    return false;}
```

Backtracking

```
bool isSafe(int maze[N][N], int x, int y)
{
    if (x >= 0 && x < N &&
        y >= 0 && y < N && maze[x][y] == 1)

        return true;

    return false;
}
```

Námety na semestrálnu prácu

- nájdite si úlohu vhodnú pre prehľadávanie s návratom a porovnajte zložitosti jednotlivých prístupov

Zdroje

Problém N dám

https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/

Sudoku

https://www.geeksforgeeks.org/sudoku-backtracking-7/

Bludisko:

https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-2/

Ďakujem vám za pozornosť!

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8