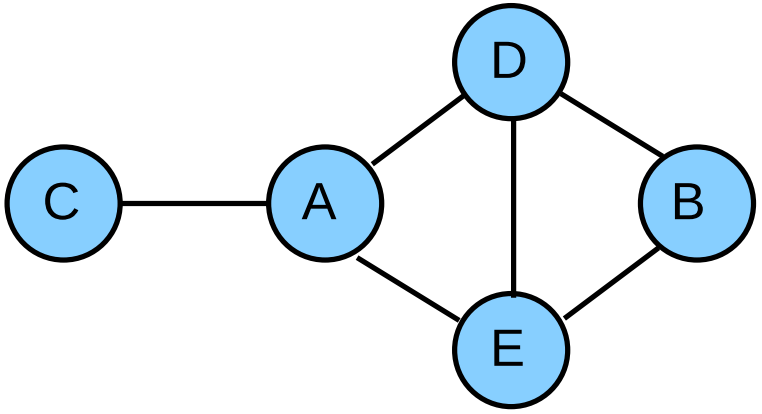
Další rozšíření backtrackingu

Problémy skoku zpět

- Při skoku zpět zapomínáme už udělanou práci
- Příklad:

  Obarvěte graf třemi barvami tak, že mají sousední vrcholy různou barvu

  (uvedené hodnoty barev jsme už vyzkoušeli)



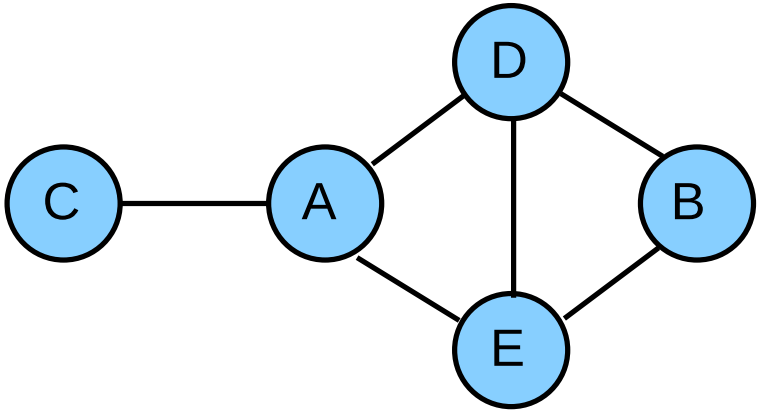
Vrchol	Barva
A	1
B	2
C	1 2
D	1 2 3
E	1 2 3

Problémy skoku zpět

- Při skoku zpět zapomínáme už udělanou práci
- Příklad:

  Obarvěte graf třemi barvami tak, že mají sousední vrcholy různou barvu

  (uvedené hodnoty barev jsme už vyzkoušeli)



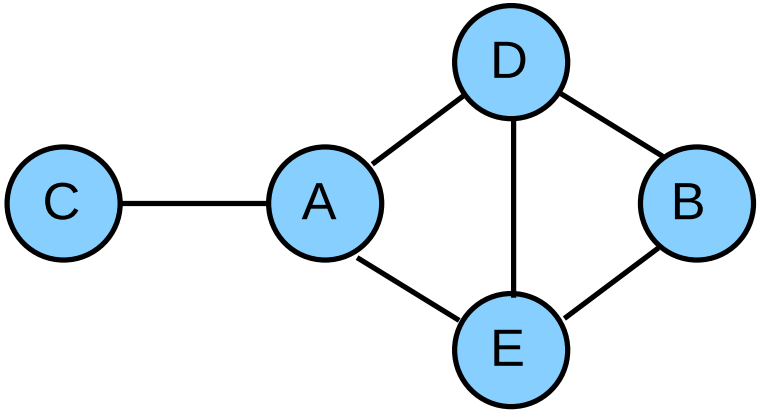
Vrchol	Barva
A	1
B	2
C	1 2
D	1 2 3
E	1 2 3 ⇒ zpět na D

Problémy skoku zpět

- Při skoku zpět zapomínáme už udělanou práci
- Příklad:

  Obarvěte graf třemi barvami tak, že mají sousední vrcholy různou barvu

  (uvedené hodnoty barev jsme už vyzkoušeli)



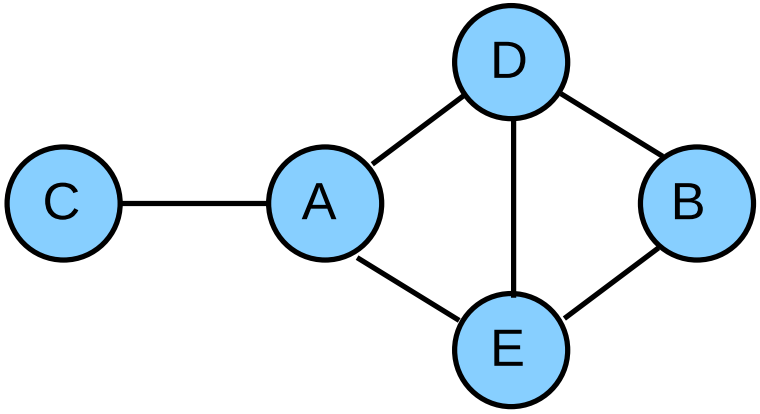
Vrchol	Barva	
A	1	1
B	2	2 1
C	1 2	1 2
D	1 2 3 ⇒ skok na B	1 2
E	1 2 3 ⇒ zpět na D	1 2 3

Problémy skoku zpět

- Při skoku zpět zapomínáme už udělanou práci
- Příklad:

  Obarvěte graf třemi barvami tak, že mají sousední vrcholy různou barvu

  (uvedené hodnoty barev jsme už vyzkoušeli)



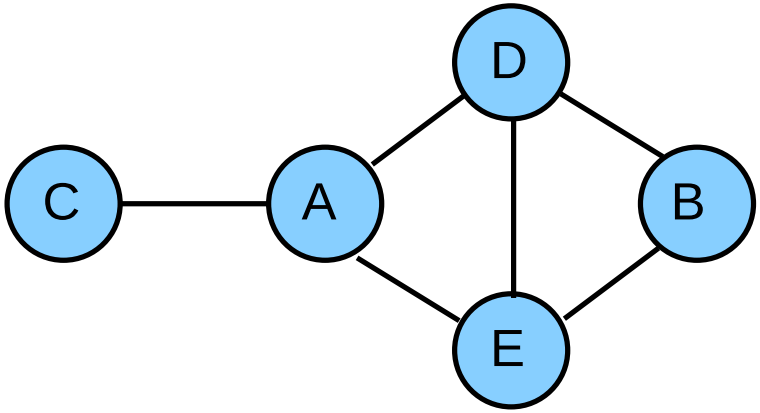
Vrchol	Barva	
A	1	1
B	2	2 1
C	1 2	1 2
D	1 2 3 ⇒ skok na B	1 2
E	1 2 3 ⇒ zpět na D	1 2 3

Při druhém ohodnocení C děláme zbytečnou práci,

stačilo nechat původní hodnotu 2, změnou B se nic neporušilo

Dynamický backtracking: příklad

- Stejný graf (A,B,C,D,E), stejné barvy (1,2,3), ale jiný postup



Skok zpět

\+ pamatování si důvodu konfliktu

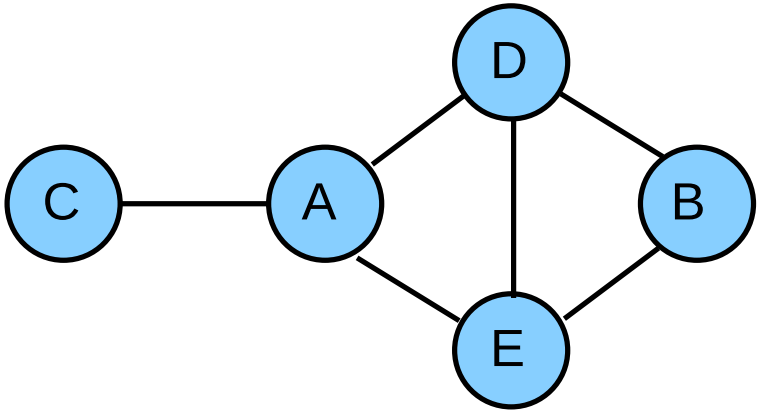
\+ přenos důvodu konfliktu

\+ změna pořadí proměnných

= **dynamický backtracking**

Dynamický backtracking: příklad

- Stejný graf (A,B,C,D,E), stejné barvy (1,2,3), ale jiný postup



Skok zpět

+ pamatování si důvodu konfliktu

+ přenos důvodu konfliktu

+ změna pořadí proměnných

= **dynamický backtracking**

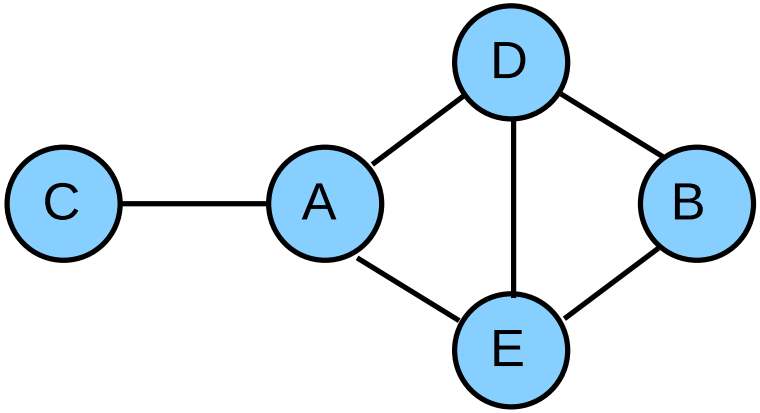
Vrchol	1	2	3
A	o		
B		o	
C	A	o	
D	A	B	o
E	A	B	D

vybraná barva: o

důvod konfliktu: AB

Dynamický backtracking: příklad

- Stejný graf (A,B,C,D,E), stejné barvy (1,2,3), ale jiný postup



Skok zpět

+ pamatování si důvodu konfliktu

+ přenos důvodu konfliktu

+ změna pořadí proměnných

= **dynamický backtracking**

Vrchol	1	2	3	Vrchol	1	2	3
A	o			A	o		
B		o		B		o	
C	A	o		C	A	o	
D	A	B	o	D	A	B	AB
E	A	B	D	E	A	B	

skok zpět (na D)

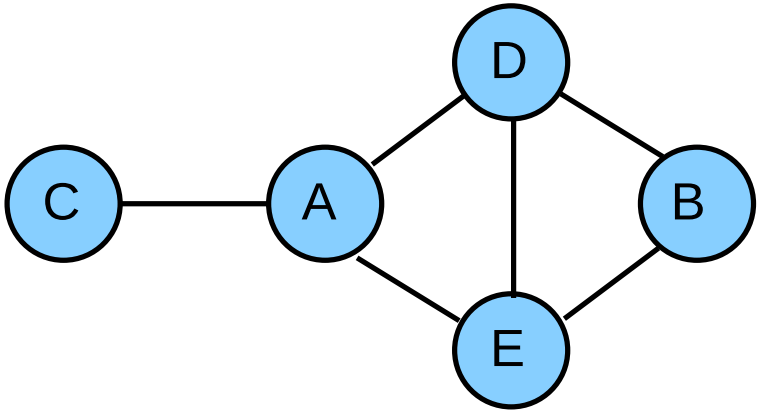
+ přenos důvodu chyby (AB)

vybraná barva: o

důvod konfliktu: AB

Dynamický backtracking: příklad

- Stejný graf (A,B,C,D,E), stejné barvy (1,2,3), ale jiný postup



Skok zpět

\+ pamatování si důvodu konfliktu

\+ přenos důvodu konfliktu

\+ změna pořadí proměnných

= **dynamický backtracking**

Vrchol	1	2	3
A	o		
B		o	
C	A	o	
D	A	B	o
E	A	B	D

skok zpět (na D)

\+ přenos důvodu chyby (AB)

Vrchol	1	2	3
A	o		
B		o	
C	A	o	
D	A	B	AB
E	A	B	

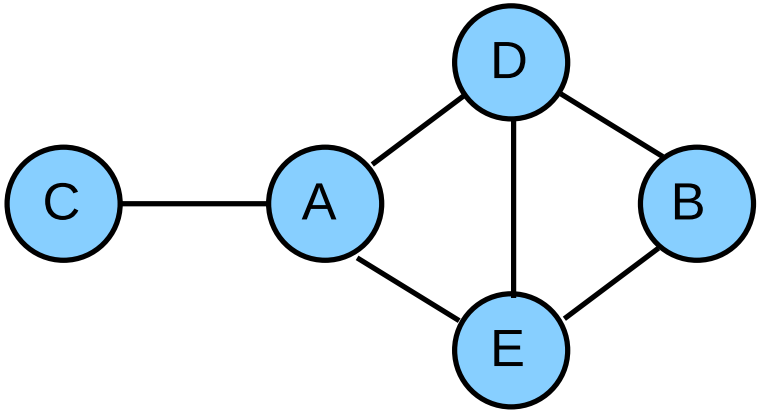
skok zpět (na B)

\+ přenos důvodu chyby (A) + změna pořadí (B,C)

Vrchol	1	2	3	
A	o			
C	A	o		vybraná barva: o
B	o	A		důvod konfliktu: AB
D	A	o		
E	A	D	o	

Dynamický backtracking: příklad

- Stejný graf (A,B,C,D,E), stejné barvy (1,2,3), ale jiný postup



Skok zpět

+ pamatování si důvodu konfliktu

+ přenos důvodu konfliktu

+ změna pořadí proměnných

= **dynamický backtracking**

Vrchol	1	2	3	Vrchol	1	2	3	Vrchol	1	2	3	
A	o			A	o			A	o			
B		o		B		o		C	A	o		vybraná barva: o
C	A	o		C	A	o		B	o	A		důvod konfliktu: AB
D	A	B	o	D	A	B	AB	D	A	o		
E	A	B	D	E	A	B		E	A	D	o	

skok zpět (na D)
+ přenos důvodu chyby (AB)

skok zpět (na B)
+ přenos důvodu chyby (A) + změna pořadí (B,C)

- Vrchol C, resp. celý graf, který na něm případně visel není nutno přebarvovat

Dynamický backtracking: algoritmus

procedure DB(Variables, Constraints)

Labelled := ∅; Unlabelled := Variables

while Unlabelled <> ∅ do

    vyber X z Unlabelled

    ValuesX := DX - hodnoty nekonzistentní s Labelled použitím Constraints

Dynamický backtracking: algoritmus

procedure DB(Variables, Constraints)

Labelled := ∅; Unlabelled := Variables

while Unlabelled <> ∅ do

    vyber X z Unlabelled

    ValuesX := DX - hodnoty nekonzistentní s Labelled použitím Constraints

    if ValuesX = ∅

    then nechť E je vysvětlení konfliktu (množina konfliktních proměnných)

        if E = ∅ then failure

        else nechť Y je nejbližší proměnná v E

            zruš přiřazení Y (z Labelled) s vysvětlením E-Y

            smaž všechna vysvětlení zahrnujícící Y

Dynamický backtracking: algoritmus

```
procedure DB(Variables, Constraints)
Labelled := ∅; Unlabelled := Variables
while Unlabelled <> ∅ do
    vyber X z Unlabelled
    ValuesX := DX - hodnoty nekonzistentní s Labelled použitím Constraints
    if ValuesX = ∅
    then nechť E je vysvětlení konfliktu (množina konfliktních proměnných)
        if E = ∅ then failure
        else nechť Y je nejbližší proměnná v E
            zruš přiřazení Y (z Labelled) s vysvětlením E-Y
            smaž všechna vysvětlení zahrnujícící Y
    else vyber V z ValuesX
        Unlabelled := Unlabelled - {X}
        Labelled := Labelled  ∪ {X/V}
return Labelled
```

Dynamický backtracking: algoritmus

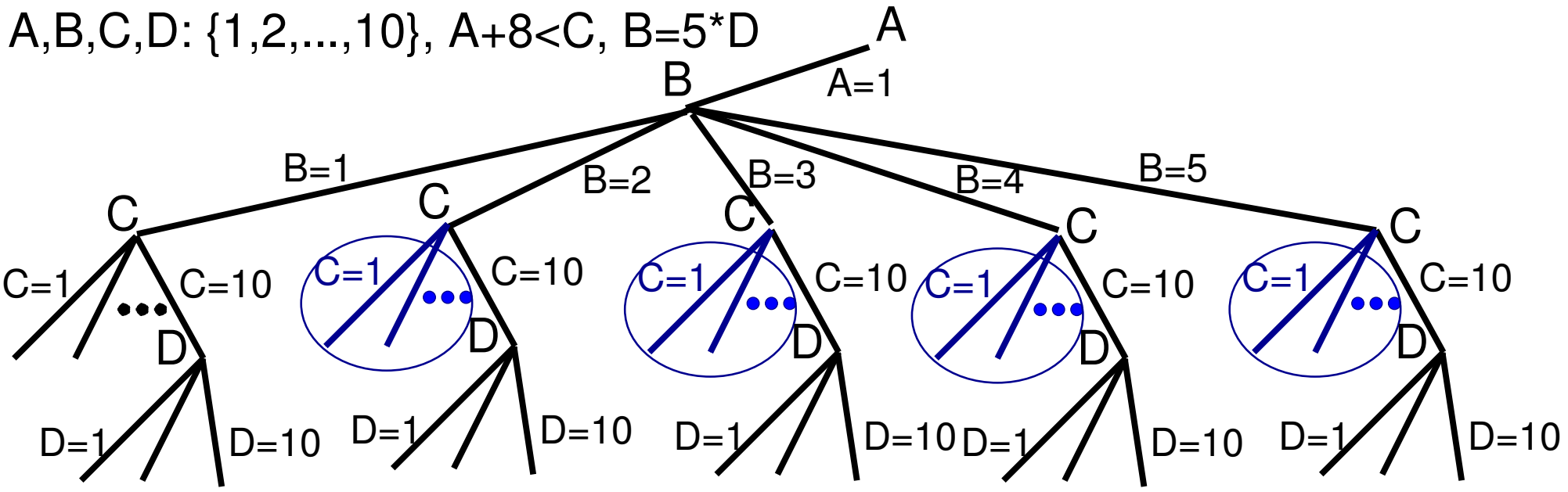
```
procedure DB(Variables, Constraints)
Labelled := ∅; Unlabelled := Variables
while Unlabelled <> ∅ do
    vyber X z Unlabelled
    ValuesX := DX - hodnoty nekonzistentní s Labelled použitím Constraints
    if ValuesX = ∅
    then nechť E je vysvětlení konfliktu (množina konfliktních proměnných)
        if E = ∅ then failure
        else nechť Y je nejbližší proměnná v E
            zruš přiřazení Y (z Labelled) s vysvětlením E-Y
            smaž všechna vysvětlení zahrnujícící Y
    else vyber V z ValuesX
        Unlabelled := Unlabelled - {X}
        Labelled := Labelled  ∪ {X/V}
return Labelled
```

Nevýhody algoritmu:

přeuspořádáváním proměnných rušíme efekt heuristik výběru proměnných

Redundance backtrackingu

- **Redundantní práce:** opakování výpočtu, jehož výsledek už máme k dispozici

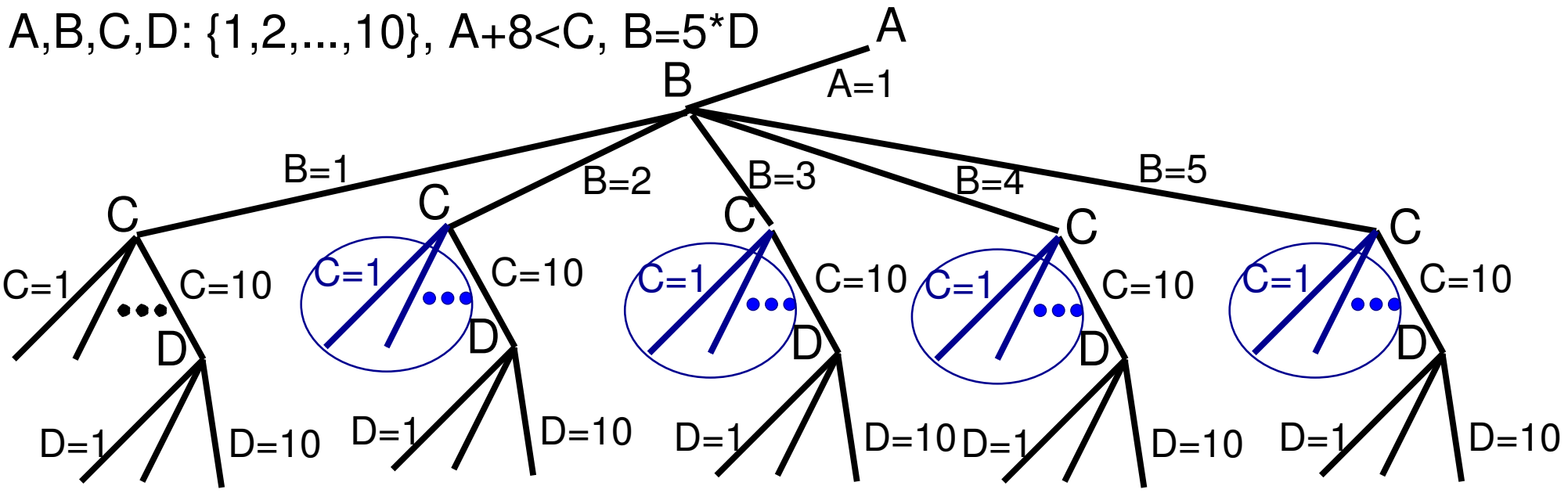


Změna B neovlivňuje hodnotu C ⇒

není potřeba znova procházet podstromy C=1,. . . ,C=9

Redundance backtrackingu

- **Redundantní práce:** opakování výpočtu, jehož výsledek už máme k dispozici



Změna B neovlivňuje hodnotu C ⇒

není potřeba znova procházet podstromy C=1,. . . ,C=9

- **Backmarking**
  - pamatuje si, kde testy na konzistenci neuspěly
  - eliminuje opakování dříve provedených konzistenčních testů

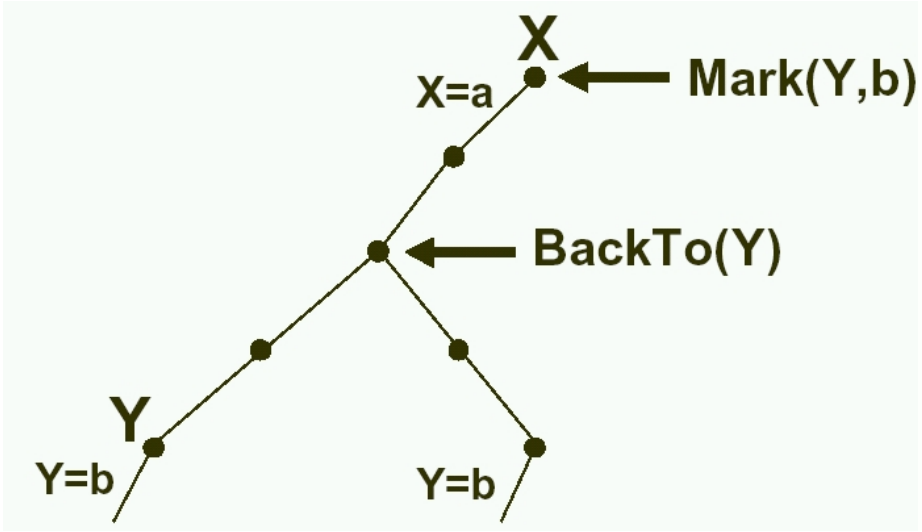
Základy backmarkingu

- **Mark(Y, b)**: u každé hodnoty b z domény Y pamatuje nejvzdálenější konflikt
  - konflikt = proměnná x_p taková, že $\vec{a_p}$ je v konfliktu s $Y = b$
- **BackTo(Y)**: u každé proměnné Y pamatuje nejvzdálenější návrat
  - návrat = proměnná, jejíž hodnota se změnila od poslední instanciace Y

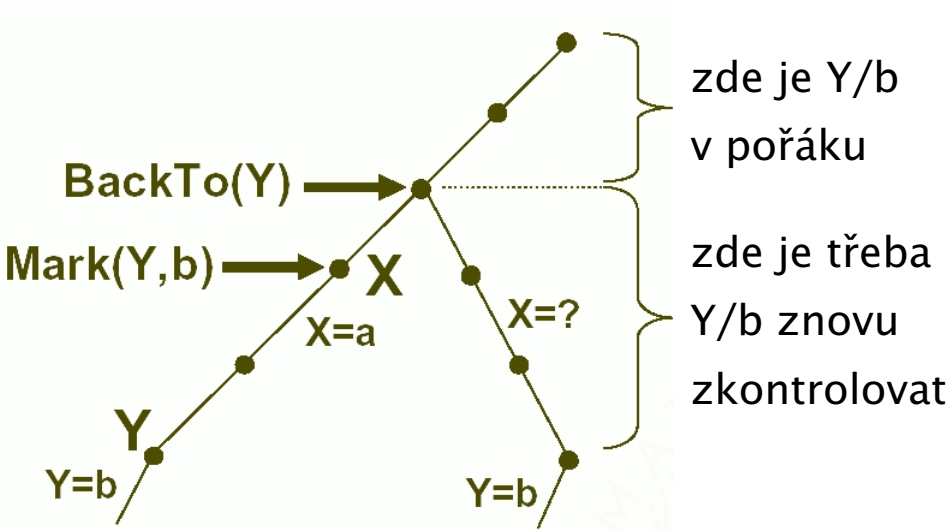
Základy backmarkingu

- **Mark(Y, b)**: u každé hodnoty b z domény Y pamatuje nejvzdálenější konflikt
  - konflikt = proměnná x_p taková, že $\vec{a_p}$ je v konfliktu s $Y = b$
- **BackTo(Y)**: u každé proměnné Y pamatuje nejvzdálenější návrat
  - návrat = proměnná, jejíž hodnota se změnila od poslední instanciace Y

Mark<BackTo



Mark ≥ BackTo



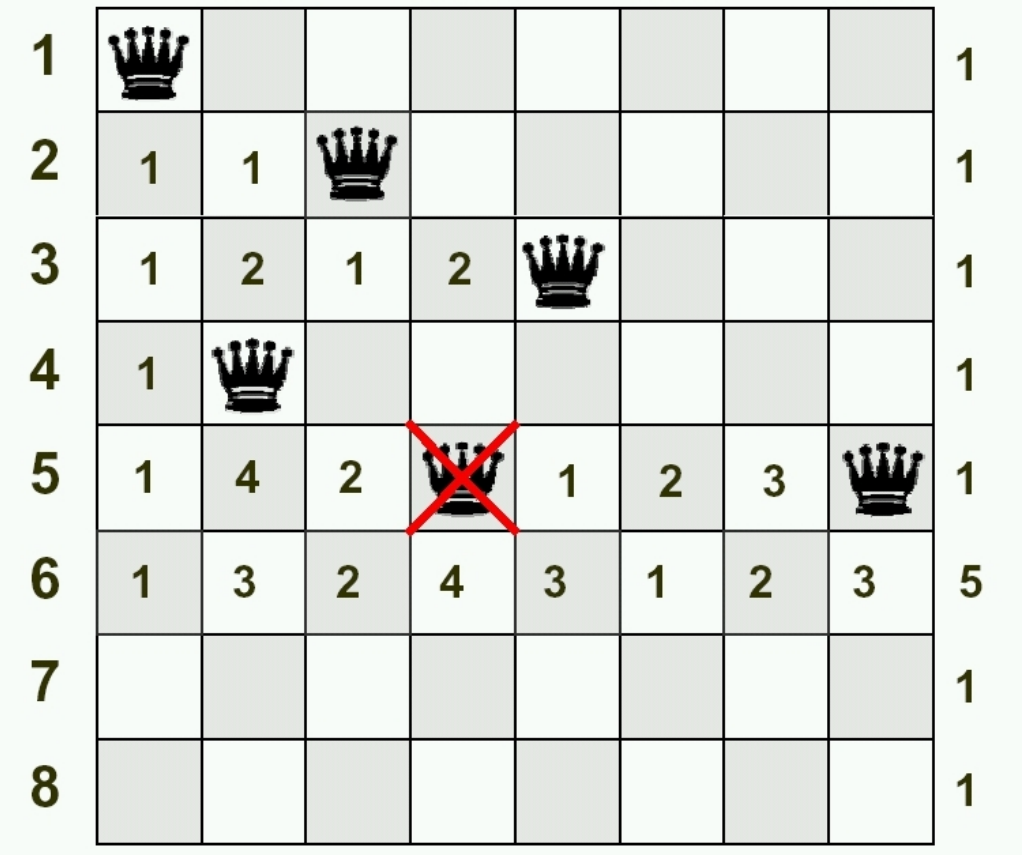
Y/b je nekonzistentní s X/a (konzistentní se vším nad X)

Y/b je s X/a stále nekonzistentní, Y=b tedy nezkoušíme přiřazovat

Y/b je nekonzistení s X/a (ale je konzistentní se vším předtím)

Backmarking: příklad

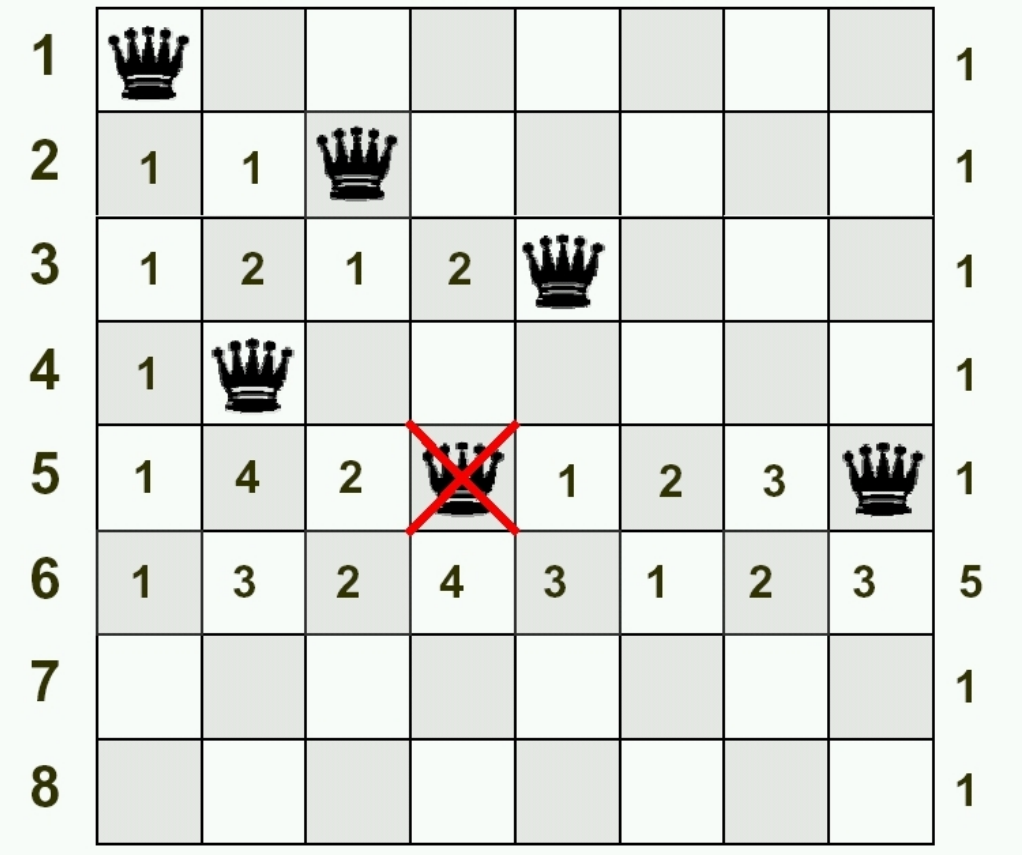
Problém N královen



1. Dámy přiřazujeme po řádcích, tj. pro každou dámu hledáme sloupec
2. Vedle šachovnice píšeme úrovně návratu (BackTo). Na začátku všude 1.
3. Do políčka zapisujeme čísla nejvzdálenějších konfliktních dam (Mark). Na začátku všude 1.

Backmarking: příklad

Problém N královen



1. Dámy přiřazujeme po řádcích, tj. pro každou dámu hledáme sloupec
2. Vedle šachovnice píšeme úrovně návratu (BackTo). Na začátku všude 1.
3. Do políčka zapisujeme čísla nejvzdálenějších konfliktních dam (Mark). Na začátku všude 1.
4. Dámu v řádku 6 nelze přiřadit.
5. Vracíme se na 5, opravíme BackTo.
6. Když znova přijdeme na 6, všechny pozice jsou stále špatné (Mark<BackTo)

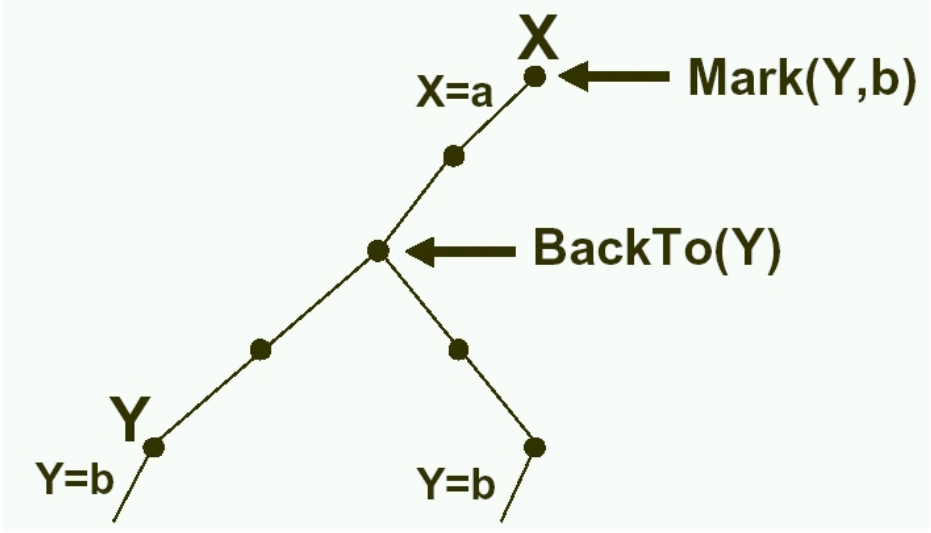
Backmarking lze kombinovat s backjumpingem (zdarma)

Algoritmus backmarkingu

```
procedure Backmarking((X,D,C))                                  rozdíly od backtrackingu
Mark(x_i,v) := 0,   BackTo(x_i) := 0 pro ∀i a ∀v   (inicializace datových struktur)
i := 1                                             (inicializace čítače proměnných)
D'_i := D_i                                        (kopírování domény)
while 1 ≤ i ≤ n
      přiřazení x_i := Select-Value-Backmarking
      if x_i is null                               (žádná hodnota nebyla vrácena)
         for ∀j : i < j ≤ n                        (úprava BackTo pro budoucí proměnné)
             if i < BackTo(x_j) then BackTo(x_j) := i   (i je nový nejvzdálenější návrat)
         BackTo(x_i) := i − 1
         i := i − 1                                (zpětná fáze)
      else i := i + 1                              (dopředná fáze)
           D'_i := D_i
if i = 0 return ,,nekonzistentní''
else return přiřazené hodnoty {x_1,...,x_n}
end Backmarking
```

Uspořádání hodnot pro backmarking

```
procedure Select-Value-Backmarking
smaž z D'_i všechna ν taková, že Mark(x_i,ν)<BackTo(x_i)
while D'_i is not empty
      vyber a smaž libovolný ν ∈ D'_i
      consistent := true
      k := BackTo(x_i)
      while k < i ∧ consistent
            if not Consistent(ā_k, x_i = ν)
               Mark(x_i,ν) := k
               consistent := false
            else k := k + 1
      if consistent
         Mark(x_i,ν) := i
         return ν
return null
```

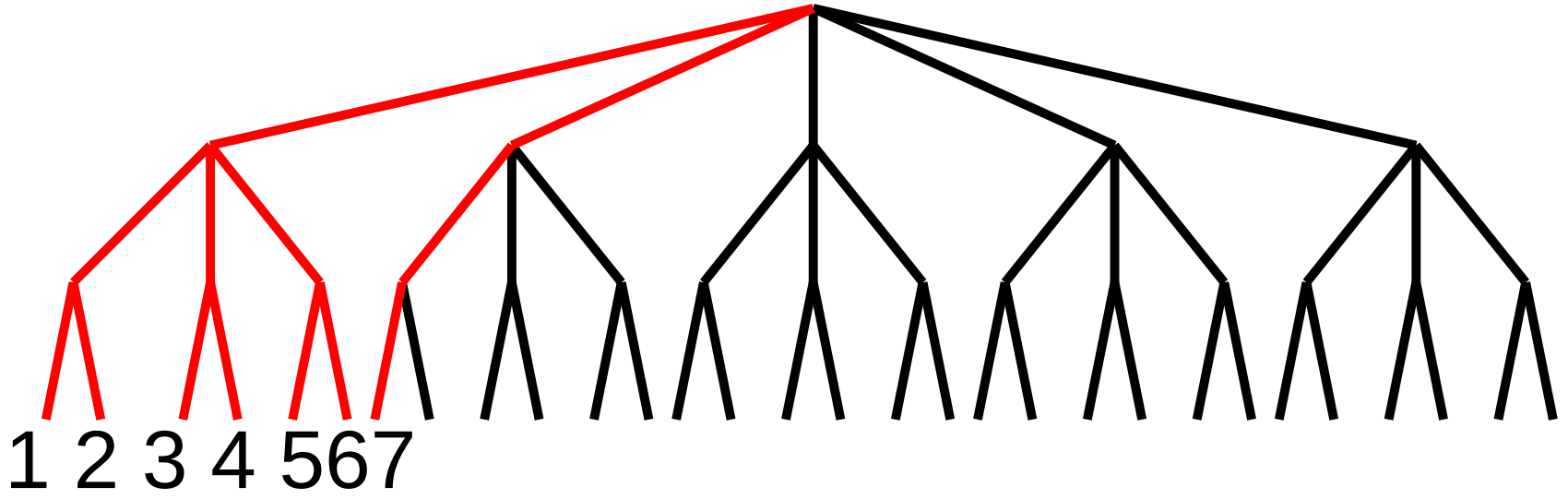


Neúplná stromová prohledávání

Neúplné prohledávání do hloubky

- ***Depth first search (DFS)***: odpovídá algoritmu **backtrackingu**
- Reálné problémy mají často tak velké prostory možných ohodnocení, že není možné je celé prohledat.
- Je možné prohledat jen omezený prostor

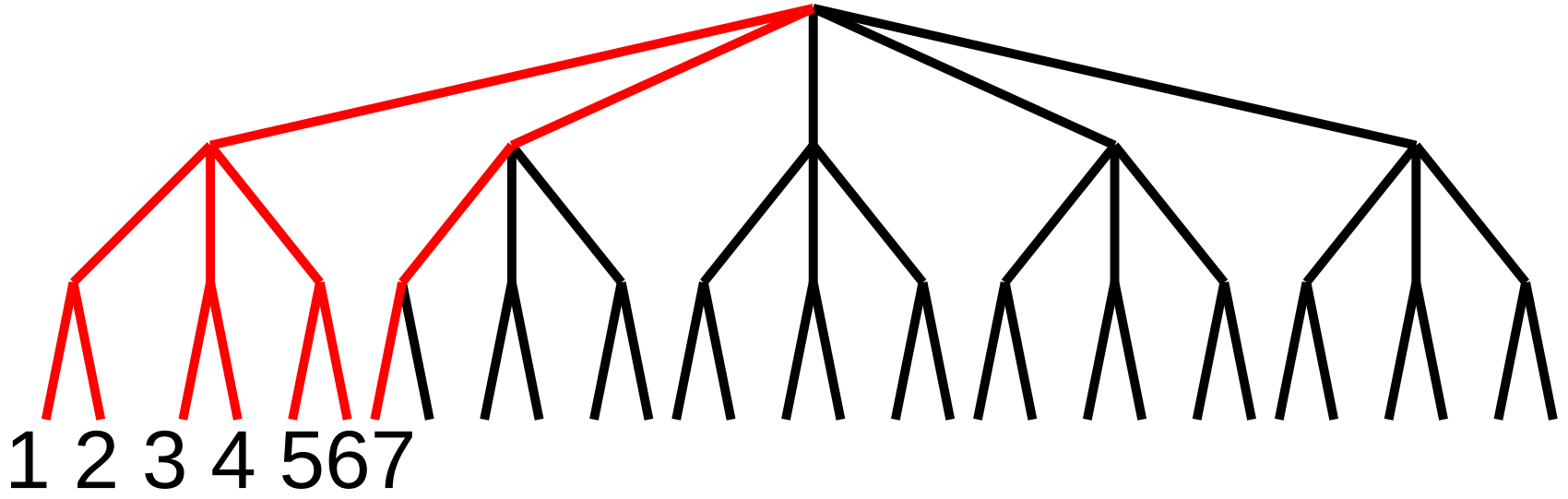
  ⇒ Neúplná stromová prohledávání
- **Neúplné prohledávání do hloubky**



Neúplné prohledávání do hloubky

- ***Depth first search (DFS)***: odpovídá algoritmu **backtrackingu**
- Reálné problémy mají často tak velké prostory možných ohodnocení, že není možné je celé prohledat.
- Je možné prohledat jen omezený prostor

  ⇒ Neúplná stromová prohledávání
- **Neúplné prohledávání do hloubky**



Neúplná stromová prohledávání

- Neprohledáváme celý stavový prostor
- Nemáme záruku, že řešení neexistuje, i když ho algoritmus nenalezne
  - ztráta úplnosti
  - u některých algoritmů lze *obecně* zaručit úplnost,
    i když s vyšší složitostí než měl původní algoritmus
- V řadě případů najdeme řešení rychleji

Neúplná stromová prohledávání

- Neprohledáváme celý stavový prostor
- Nemáme záruku, že řešení neexistuje, i když ho algoritmus nenalezne
  - ztráta úplnosti
  - u některých algoritmů lze *obecně* zaručit úplnost,
    i když s vyšší složitostí než měl původní algoritmus
- V řadě případů najdeme řešení rychleji
- Neúplné algoritmy často odvozeny od algoritmu úplného (DFS)
  - **přerušení běhu algoritmu (*cutoff*)**
    - po vyčerpání přiděleného **prostředku** (čas, počet návratů, . . .) algoritmus přerušíme
    - prostředek může být **globální** (pro celý strom) i **lokální** (pro daný podstrom nebo uzel)
  - **opakovaní běhu algoritmu (*restart*)**
    - běh předešlého neúplného prohledávání opakujeme s jiným nastavením parametrů
    - při opakování běhu lze využívat algoritmy učení

Randomizovaný backtracking

- **Časově omezený backtracking (přerušení)**
  - běh (úplného) algoritmu ukončíme po zadaném časovém intervalu (prostředek=čas)
  - časový interval lze pro další běhy zvětšit

    ⇒ při dostatečném počtu kroků máme úplný algoritmus
- **Náhodný výběr hodnot a proměnných (opakování)**
  - pokud máme možnost volby při výběru hodnot nebo proměnných (vzhledem k dané heuristice uspořádání hodnot a proměnných) náhodně zvolíme některou z nich

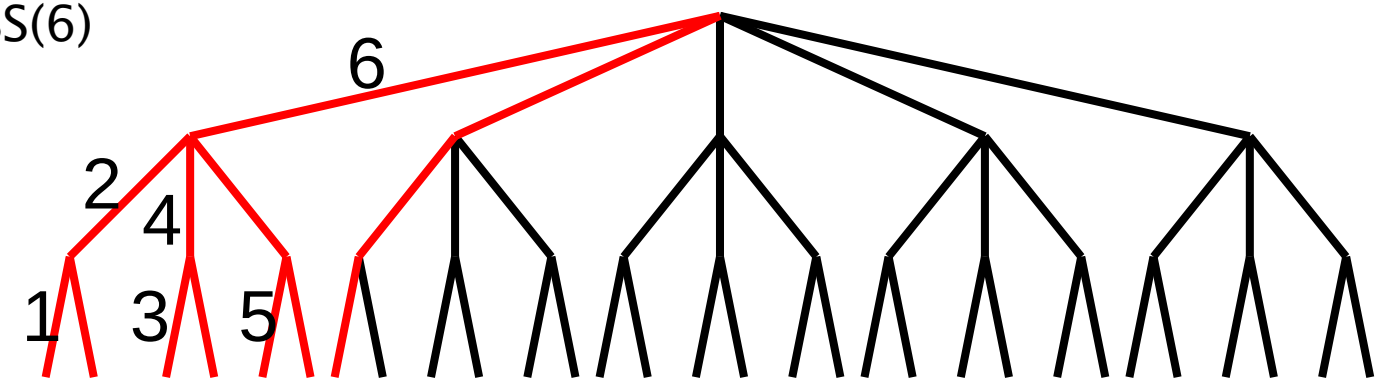
Randomizovaný backtracking

- **Časově omezený backtracking (přerušení)**
  - běh (úplného) algoritmu ukončíme po zadaném časovém intervalu (prostředek=čas)
  - časový interval lze pro další běhy zvětšit

    ⇒ při dostatečném počtu kroků máme úplný algoritmus
- **Náhodný výběr hodnot a proměnných (opakování)**
  - pokud máme možnost volby při výběru hodnot nebo proměnných (vzhledem k dané heuristice uspořádání hodnot a proměnných) náhodně zvolíme některou z nich
- **Randomizovaný backtracking s učením**
  - chybná přiřazení předchozích běhů uchováme a využíváme
  - takto lze také dosáhnout úplnosti, protože chybných přiřazení je konečně mnoho

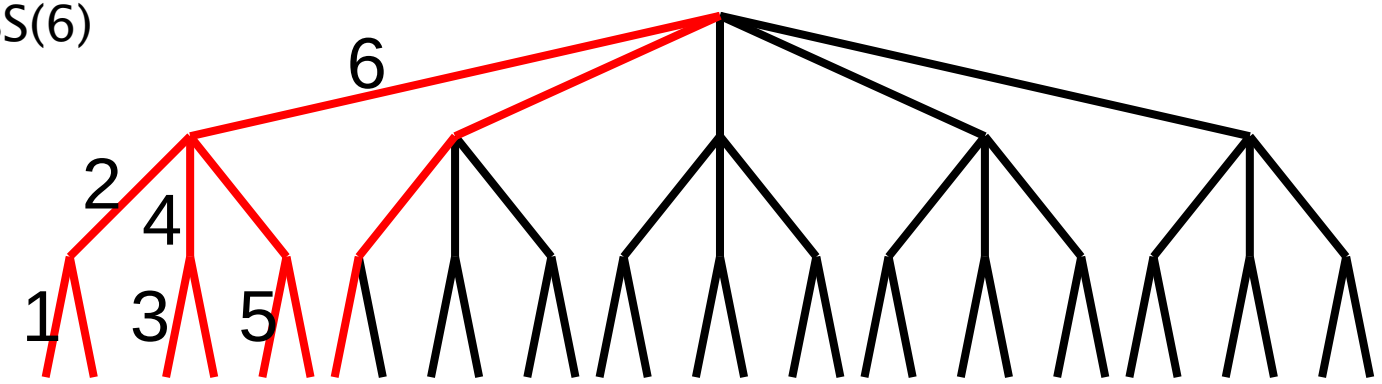
Omezení počtu návratů

- ***Bounded-backtrack search*** **(BBS)**
- Omezený počet návratů **(přerušení)**
  - návrat do bodů volby, kde už nelze vybrat novou hodnotu nezapočítáváme
  - „omezený počet navštívených listů”
- Pro úplnost: při neúspěchu zvětšíme počet návratů o jedna **(opakování)**
- Příklad: BBS(6)



Omezení počtu návratů

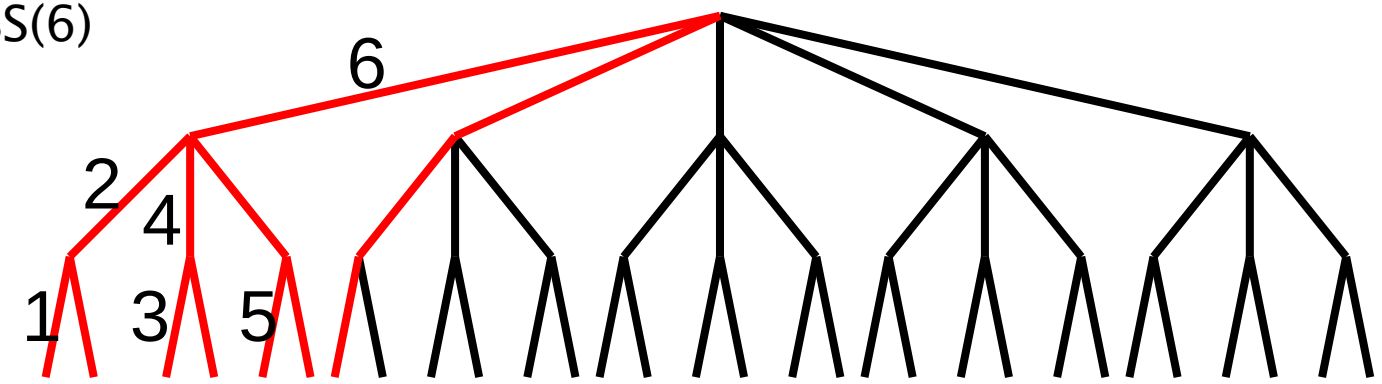
- ***Bounded-backtrack search*** **(BBS)**
- Omezený počet návratů **(přerušení)**
  - návrat do bodů volby, kde už nelze vybrat novou hodnotu nezapočítáváme
  - „omezený počet navštívených listů”
- Pro úplnost: při neúspěchu zvětšíme počet návratů o jedna **(opakování)**
- Příklad: BBS(6)



- Implementace

Omezení počtu návratů

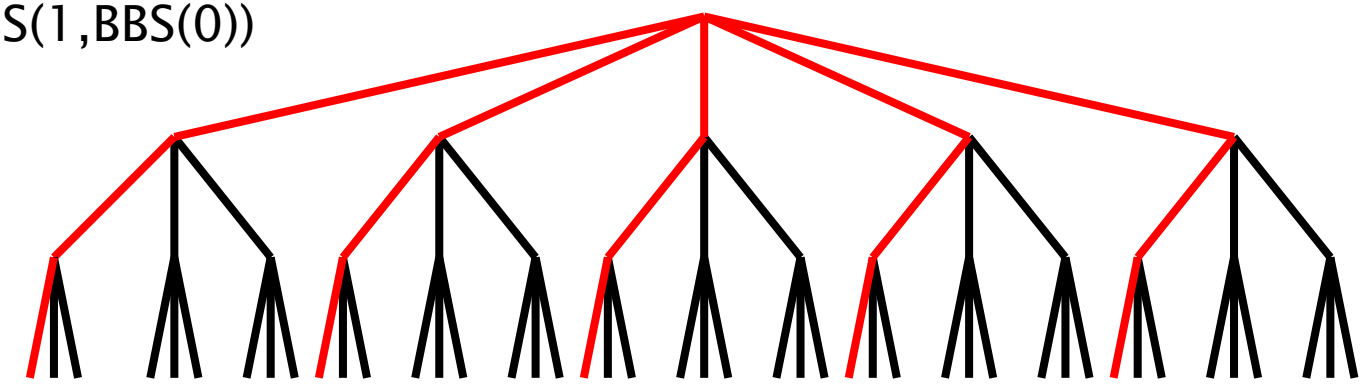
- ***Bounded-backtrack search*** **(BBS)**
- Omezený počet návratů **(přerušení)**
  - návrat do bodů volby, kde už nelze vybrat novou hodnotu nezapočítáváme
  - „omezený počet navštívených listů”
- Pro úplnost: při neúspěchu zvětšíme počet návratů o jedna **(opakování)**
- Příklad: BBS(6)



- Implementace
  - počítáme počet návratů (neúspěchů)
  - při překročení meze se prohledávání ukončí

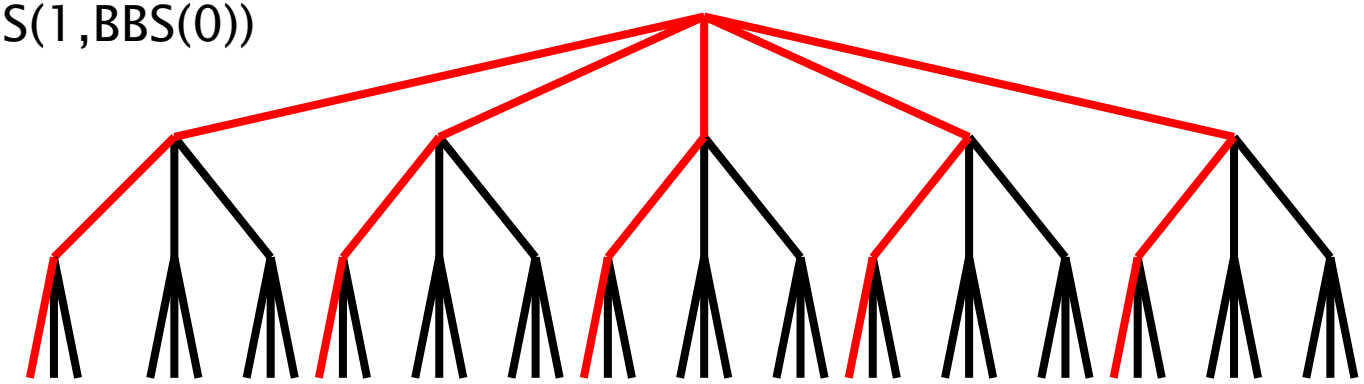
Omezení hloubky

- ***Depth-bounded search*** **(DBS)**
- Omezíme hloubku prohledávaného stromu **(přerušení)**
  - do dané hloubky stromu se zkouší všechny alternativy
  - ve zbytku stromu se může použít jiná neúplná metoda
- Pro úplnost: při neúspěchu zvětšíme hloubku prohledávání o jedna **(opakování)**
- Příklad: DBS(1,BBS(0))



Omezení hloubky

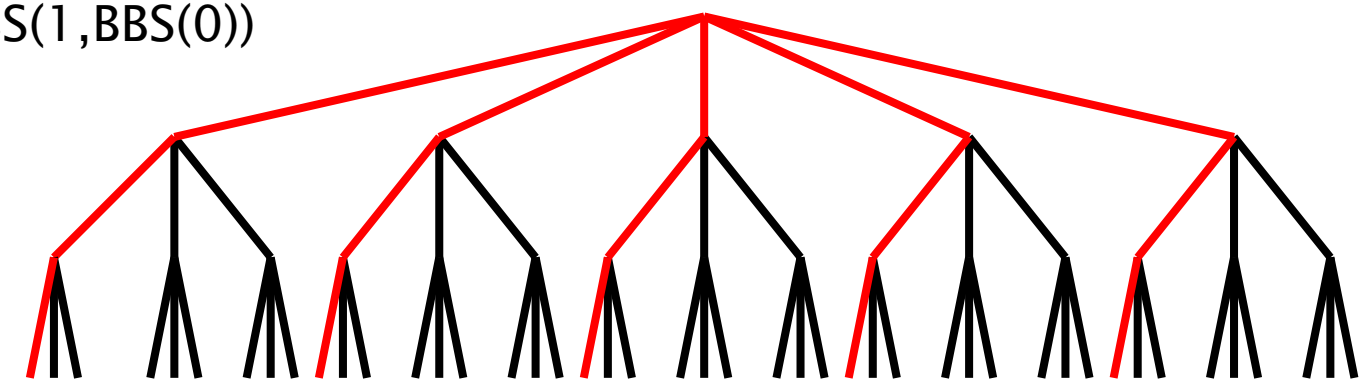
- ***Depth-bounded search*** **(DBS)**
- Omezíme hloubku prohledávaného stromu **(přerušení)**
  - do dané hloubky stromu se zkouší všechny alternativy
  - ve zbytku stromu se může použít jiná neúplná metoda
- Pro úplnost: při neúspěchu zvětšíme hloubku prohledávání o jedna **(opakování)**
- Příklad: DBS(1,BBS(0))



- Implementace

Omezení hloubky

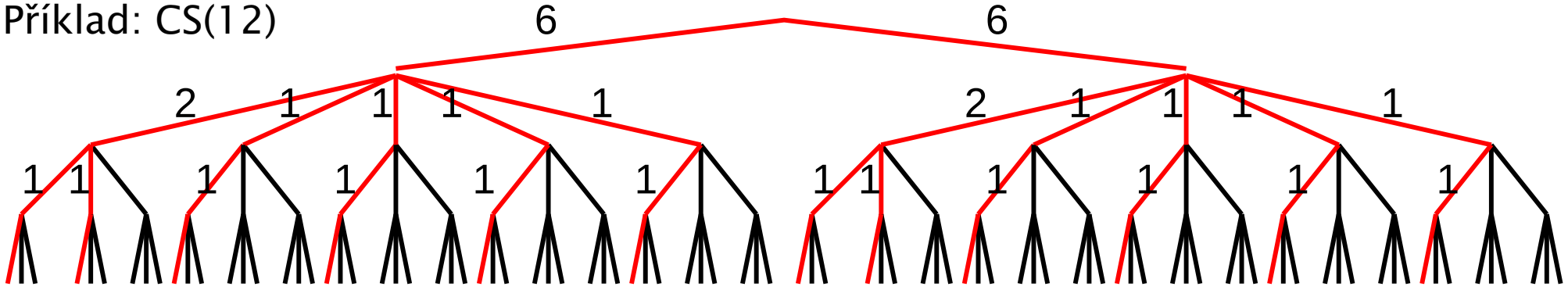
- ***Depth-bounded search*** **(DBS)**
- Omezíme hloubku prohledávaného stromu **(přerušení)**
  - do dané hloubky stromu se zkouší všechny alternativy
  - ve zbytku stromu se může použít jiná neúplná metoda
- Pro úplnost: při neúspěchu zvětšíme hloubku prohledávání o jedna **(opakování)**
- Příklad: DBS(1,BBS(0))



- Implementace
  - udržujeme pořadové číslo přiřazované proměnné
  - je-li pořadové číslo větší než daná mez, zkouší se pouze jedna alternativa – BBS(0)

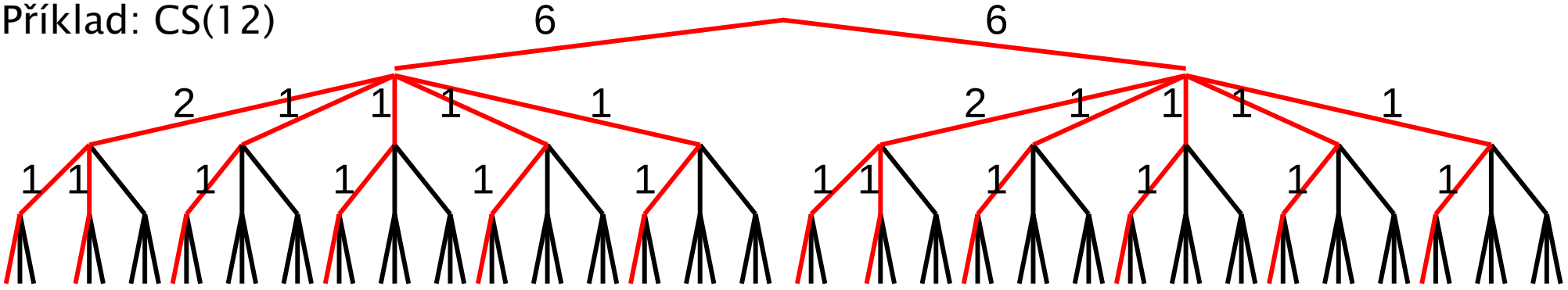
Prohledávání s kreditem

- ***Credit search* (CS)**
- Omezený kredit (počet návratů) pro prohledávání **(přerušení)**
  - kredit se rovnoměrně dělí mezi alternativní větve prohledávání
  - jednotkový kredit zakazuje možnost volby (hodnoty), tj. pokračujeme pouze bez návratů
- Pro úplnost: při neúspěchu navýšíme kredit o jedna **(opakování)**
- Příklad: CS(12)



Prohledávání s kreditem

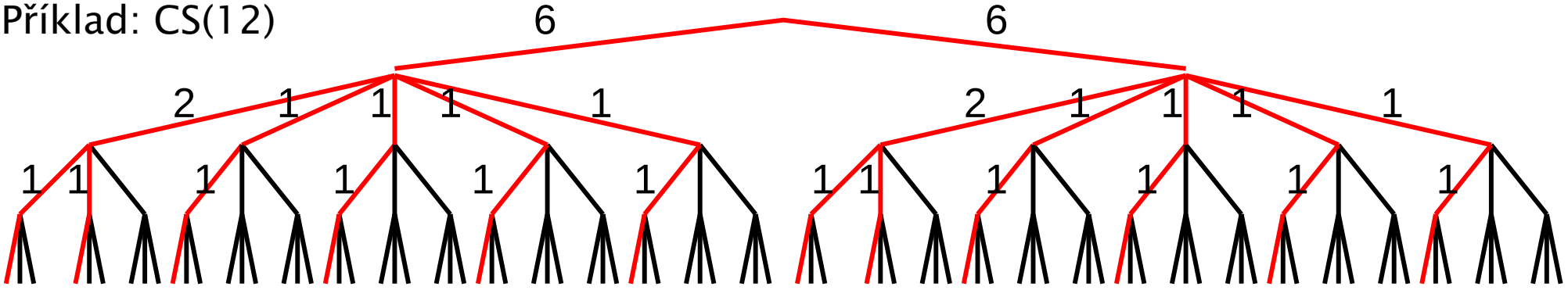
- ***Credit search* (CS)**
- Omezený kredit (počet návratů) pro prohledávání **(přerušení)**
  - kredit se rovnoměrně dělí mezi alternativní větve prohledávání
  - jednotkový kredit zakazuje možnost volby (hodnoty), tj. pokračujeme pouze bez návratů
- Pro úplnost: při neúspěchu navýšíme kredit o jedna **(opakování)**
- Příklad: CS(12)



- Implementace

Prohledávání s kreditem

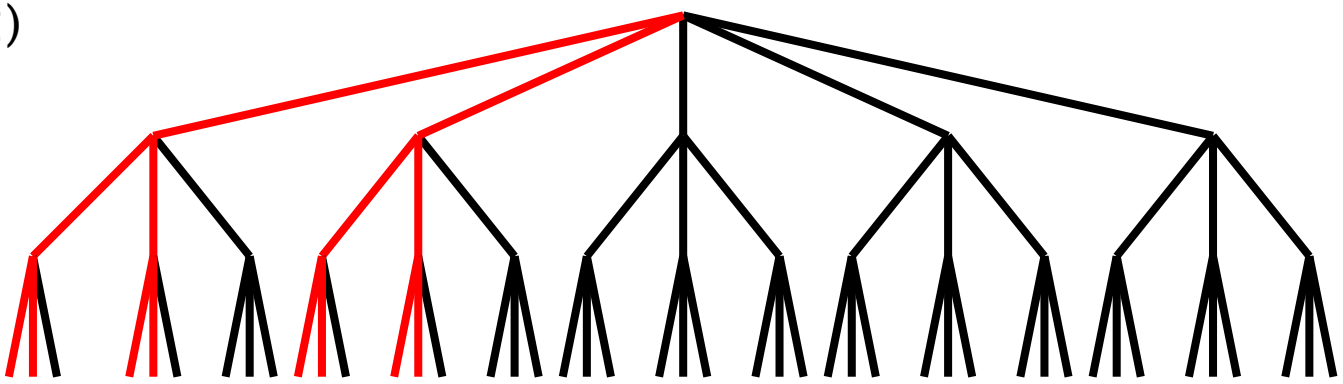
- ***Credit search* (CS)**
- Omezený kredit (počet návratů) pro prohledávání **(přerušení)**
  - kredit se rovnoměrně dělí mezi alternativní větve prohledávání
  - jednotkový kredit zakazuje možnost volby (hodnoty), tj. pokračujeme pouze bez návratů
- Pro úplnost: při neúspěchu navýšíme kredit o jedna **(opakování)**
- Příklad: CS(12)



- Implementace
  - v každém uzlu se nejednotkový kredit (rovnoměrně) rozdělí mezi alternativní podstromy
  - při jednotkovém kreditu se neberou alternativy (tj. při neúspěchu končíme)

Iterativní rozšiřování

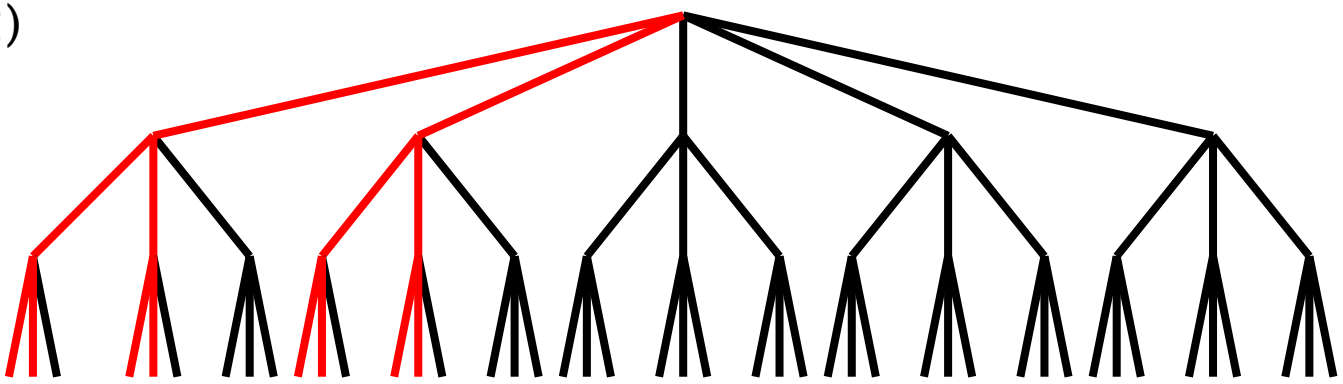
- ***Iterative broadening*** **IB**
- Omezený maximální počet voleb (hodnot)
  při každém výběru proměnné **(přerušení)**
  - v každém bodě volby větvení omezeno konstantou
  - při překročení max. počtu voleb pokračujeme předchozím bodem volby
- Úplnost: při neúspěchu zvýšíme povolený počet voleb o jedna **(opakování)**
- Příklad: IB(2)



- Implementace

Iterativní rozšiřování

- ***Iterative broadening*** **IB**
- Omezený maximální počet voleb (hodnot)
  při každém výběru proměnné **(přerušení)**
  - v každém bodě volby větvení omezeno konstantou
  - při překročení max. počtu voleb pokračujeme předchozím bodem volby
- Úplnost: při neúspěchu zvýšíme povolený počet voleb o jedna **(opakování)**
- Příklad: IB(2)



- Implementace
  - po výběru proměnné umožníme pouze výběr určeného počtu jejích hodnot

Stromová prohledávání a heuristiky

- Při řešení reálných problémů často existuje nápověda odvozená ze zkušeností s „ručním” řešením problému
- Heuristiky – radí, jak pokračovat v prohledávání
  - doporučují hodnotu pro přiřazení
  - často vedou přímo k řešení
- Co dělat, když heuristika neuspěje?
  - DFS se stará hlavně o konec prohledávání (spodní část stromu)
  - DFS tedy spíše opravuje poslední použité heuristiky než první
  - DFS předpokládá, že dříve použité heuristiky ho navedly dobře

Stromová prohledávání a heuristiky

- Při řešení reálných problémů často existuje nápověda odvozená ze zkušeností s „ručním” řešením problému
- Heuristiky – radí, jak pokračovat v prohledávání
  - doporučují hodnotu pro přiřazení
  - často vedou přímo k řešení
- Co dělat, když heuristika neuspěje?
  - DFS se stará hlavně o konec prohledávání (spodní část stromu)
  - DFS tedy spíše opravuje poslední použité heuristiky než první
  - DFS předpokládá, že dříve použité heuristiky ho navedly dobře
- Pozorování:
  - počet porušení heuristiky na úspěšné cestě je malý
  - heuristiky jsou méně spolehlivé na začátku prohledávání než na jeho konci (na konci máme více informací a méně možností)

Zotavení se z chyb heuristiky

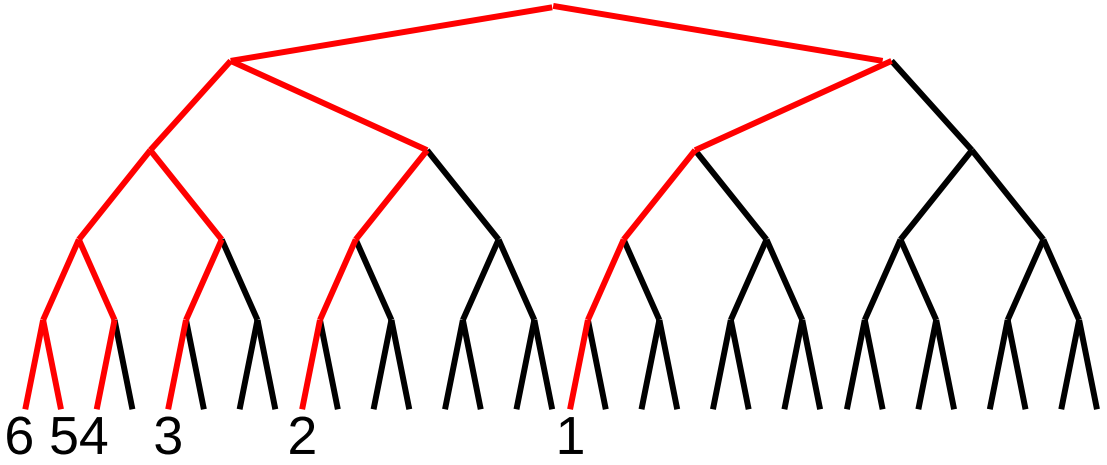
- Heuristika doporučuje hodnotu pro přiřazení
- **Diskrepance** = porušení heuristiky
  - použita jiná hodnota, než doporučila heuristika
- Pozorování: málo chyb heuristiky na cestě k řešení

  ⇒ cesty s méně diskrepancemi jsou prozkoumány dříve
- Pozorování: chyby heuristiky hlavně na začátku cesty

  ⇒ cesty s diskrepancemi na začátku jsou prozkoumány dříve

Omezené diskrepance

- ***Limited discrepancy search*** **(LDS)**
- Omezený maximální počet diskrepancí na cestě **(přerušení)**
  - cesty s diskrepancemi na začátku jsou prozkoumány dříve
- Při neúspěchu navýšíme počet povolených diskrepancí o jedna **(opakování)**
  - tj. nejprve jdeme tak, jak doporučuje heuristika; potom jdeme po cestách s maximálně jednou diskrepancí; pak maximálně se dvěma diskrepancemi, atd.
- Příklad: LDS-PROBE(1), heuristika doporučuje vydat se levou větví



- **Diskrepance pro nebinární domény**
  - nedoporučené hodnoty se berou jako jedna diskrepance (zde)
  - výběr každé další hodnoty proměnné je jedna diskrepance (tj. třetí hodnota = dvě diskrepance, čtvrtá hodnota = tři diskrepance, . . .)

Algoritmus LDS

```
procedure LDS(Variables,Constraints)
   for D=0 to |Variables| do                    % D určuje max. počet povolených diskrepancí
      R := LDS-PROBE(Variables,{},Constraints,D)
      if R ≠ fail then return R
return fail
```

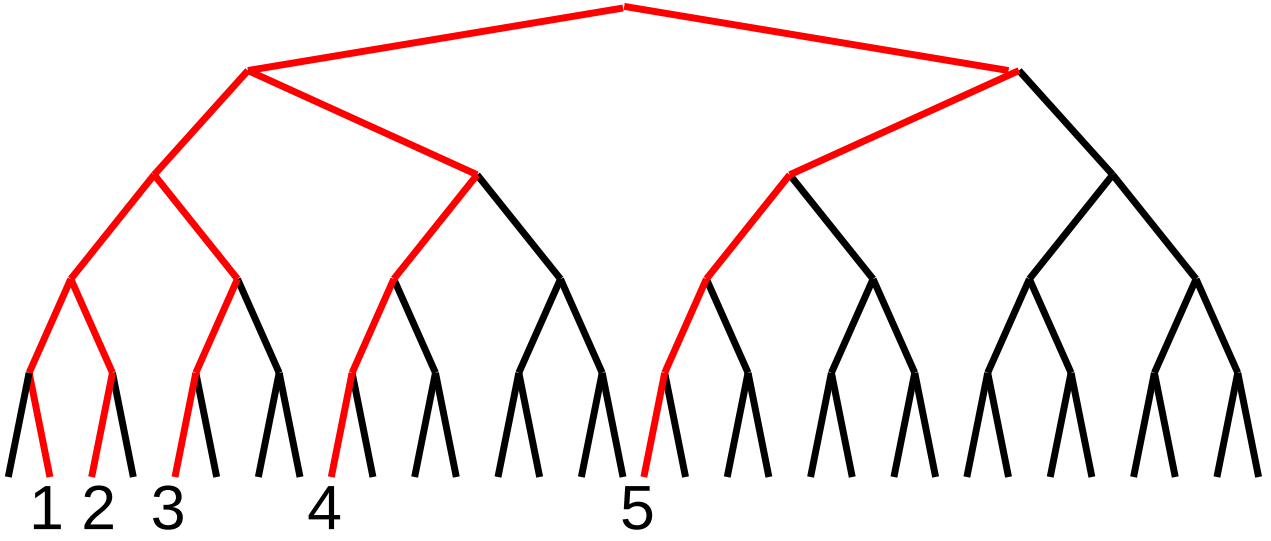
Algoritmus LDS

```
procedure LDS(Variables,Constraints)
  for D=0 to |Variables| do                    % D určuje max. počet povolených diskrepancí
    R := LDS-PROBE(Variables,{},Constraints,D)
    if R ≠ fail then return R
return fail

procedure LDS-PROBE(Unlabelled,Labelled,Constraints,D)
  if Unlabelled = {} then return Labelled
  select X ∈ Unlabelled
  Values_X := D_X - {values inconsistent with Labelled using Constraints}
  if Values_X = {} then return fail
  else select HV ∈ Values_X using heuristic
    if D>0 then
      for ∀V ∈ Values_X-{HV} do
        R := LDS-PROBE(Unlabelled-{X}, Labelled ∪ {X/V}, Constraints, D-1)
        if R ≠ fail then return R
    return LDS-PROBE(Unlabelled-{X}, Labelled ∪ {X/HV}, Constraints, D)
end LDS-PROBE
```

Omezené diskrepance – zlepšení

- LDS v každé další iteraci prochází i větve z předchozí iterace, tj. opakuje již provedený výpočet a navíc se v rámci iterace musí vracet do již prošlých částí
- ***Improved limited discrepancy search (ILDS)***
  - daný počet diskrepancí na cestě **(přerušení)**
    - cesty s diskrepancemi na konci prozkoumány dříve
- Při neúspěchu navýšíme počet diskrepancí o jedna **(opakování)**
- Příklad: ILDS-PROBE(1), heuristika doporučuje vydat se levou větví



Algoritmus ILDS

procedure ILDS(Variables,Constraints) % analogie LDS(Variables,Constraints)

procedure ILDS-PROBE(Unlabelled,Labelled,Constraints,D) Rozdíly od LDS

if Unlabelled = {} then return Labelled

select X $\in$ Unlabelled

Values$_X$:= D$_X$ - {values inconsistent with Labelled using Constraints}

if Values$_X$ = {} then return fail

else select HV $\in$ Values$_X$ using heuristic

if D<|Unlabelled| then

R := ILDS-PROBE(Unlabelled-{X}, Labelled $\cup$ {X/HV}, Constraints, D)

if R $\neq$ fail then return R

if D>0 then

for $\forall$V $\in$ Values$_X$-{HV} do

R := ILDS-PROBE(Unlabelled-{X}, Labelled $\cup$ {X/V}, Constraints, D-1)

if R $\neq$ fail then return R

end ILDS-PROBE

Hloubkou omezené diskrepance

- ILDS bere cesty s diskrepancemi na konci dříve
- ***Depth-bounded discrepancy search (DDS)***
- Diskrepance povoleny pouze do dané hloubky **(přerušení)**
  - v této hloubce je vždy diskrepance, tj. zabrání se procházení větví z předchozí iterace
  - hloubka zároveň omezuje maximální počet diskrepancí
  - cesty s diskrepancemi na začátku prozkoumány dříve
- Při neúspěchu navýšíme hloubku o jedna **(opakování)**
- Příklad: DDS(3), heuristika doporučuje vydat se levou větví