

Kombinatorisk sökning

Douglas Wikström

KTH Stockholm

`popup-help@csc.kth.se`

Nordiska mästerskapen i programmering (NCPC)

- ▶ **När:** 4 oktober kl 10:00.
- ▶ **Var:** på flera nordiska universitet, bland annat KTH.
- ▶ **Vad står på spel:**
 - ▶ Betygspoäng till kursen.
 - ▶ KTH-mästare: bäst på KTH.
 - ▶ Svenska mästare: bäst i Sverige.
 - ▶ Nordiska mästare: bäst bland alla lag.
- ▶ Länk till **anmälningssidan finns på kurshemsidan.**
- ▶ Anmäl dig innan **26 september 08:00 CEST** så får du antagligen T-shirt och mat.

Kombinatorisk sökning

- ▶ Kombinatorisk sökning är ett fint uttryck för någon form av uttömmande sökning efter en lösning.
- ▶ Ibland är det enda (kända) alternativet för att lösa ett problem kombinatorisk sökning.
- ▶ Exempel: *M*-färgning av graf, knäcka kryptosystem, rubiks kub, osv...

Kombinatorisk sökning

Varför är det intressant att studera problem som vi inte tror vi kan lösa effektivt?

Kombinatorisk sökning

Varför är det intressant att studera problem som vi inte tror vi kan lösa effektivt?

- ▶ Vi kanske måste lösa problemet approximativt i praktiken.

Kombinatorisk sökning

Varför är det intressant att studera problem som vi inte tror vi kan lösa effektivt?

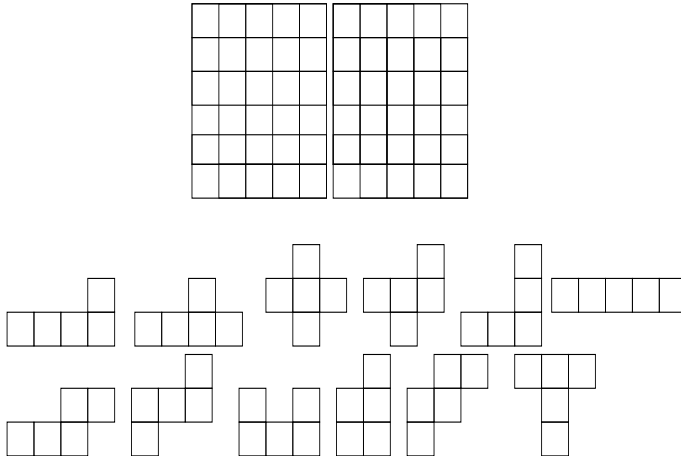
- ▶ Vi kanske måste lösa problemet approximativt i praktiken.
- ▶ Även om alla kända algoritmer är ineffektiva kan de vara mer eller mindre ineffektiva, vilket är av både teoretiskt och praktiskt intresse.

Kombinatorisk sökning

Varför är det intressant att studera problem som vi inte tror vi kan lösa effektivt?

- ▶ Vi kanske måste lösa problemet approximativt i praktiken.
- ▶ Även om alla kända algoritmer är ineffektiva kan de vara mer eller mindre ineffektiva, vilket är av både teoretiskt och praktiskt intresse.
- ▶ Tumregler, heuristiker, och tricks som fungerar på ett problem är ofta användbara eller kan tjäna som inspiration för att lösa ett annat problem bättre än naivt.

Pentatriss



Uttömmande sökning efter lösning i Pentatriss

- ▶ När vi söker en lösning försöker vi först fylla i första brädet och sedan det andra.
- ▶ Varje gång vi lägger en bit placeras den så att den täcker den översta vänstraste icke-använda rutan på det aktuella brädet. Vi kallar denna ruta "målrutan".
- ▶ Givet en partiell lösning provar vi varje möjlig oanvänd bit och dess speglingar och rotationer och gör en rekursiv sökning med den utvidgade partiella lösningen.

Sökning efter lösning i Pentatriss

För att minska antalet alternativ:

- ▶ Hitta alla symmetrier för varje bit.
- ▶ Hitta alla rutor på en bit som kan användas för att täcka "målrutan".

Implementationstricks:

- ▶ Använd bitmasker för att representera tillstånd. Det reducerar minnet som krävs, och gör det enkelt att kolla om en bit passar.
- ▶ När en bit passar, så xor-ar vi in den på den partiella lösningen, och om den rekursiva sökningen misslyckades xor-ar vi bort den igen.

Dubbel-DES

Data Encryption Standard (DES) är ett symmetriskt kryptosystem med 56-bitars nyckel som standardiserades 1976.

$DES_k(m)$ betyder kryptera meddelandet m med nyckeln k och $DES_k^{-1}(c)$ betyder dekryptera kryptotexten c med nyckeln k .

Dubbel-DES definieras enligt: $DDES_{k_1, k_1}(m) = DES_{k_1}(DES_{k_1}(m))$.

Hur hittar vi nyckeln till Dubbel-DES om vi känner till ett (långt) klartext-cryptotext par (m, c) ?

Dubbel-DES

Hur lång tid tar uttömmande sökning?

Dubbel-DES

Vi har $(2^{56})^2 = 2^{112}$ olika nycklar på formen (k_1, k_2) , och behöver testa om $\text{DDES}_{k_1, k_1}(m) = c$ för varje sådan lösning.

Dubbel-DES

Vi har $(2^{56})^2 = 2^{112}$ olika nycklar på formen (k_1, k_2) , och behöver testa om $\text{DDES}_{k_1, k_1}(m) = c$ för varje sådan lösning.

Vad kan vi göra istället?

Attack mot Dubbel-DES

1. Vi räknar ut en tabeller enligt:

k	$\text{DES}_k(m)$
0	c_0
1	c_1
2	c_2
$\vdots$	$\vdots$
$2^{56} - 1$	$c_{2^{56}-1}$

$\text{DES}_k^{-1}(c)$	k
c'_0	0
c'_1	1
c'_2	2
$\vdots$	$\vdots$
$c'_{2^{56}-1}$	$2^{56} - 1$

Attack mot Dubbel-DES

1. Vi räknar ut en tabeller enligt:

k	$\text{DES}_k(m)$	$\text{DES}_k^{-1}(c)$	k
0	c_0	c'_0	0
1	c_1	c'_1	1
2	c_2	c'_2	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2^{56} - 1$	$c_{2^{56}-1}$	$c'_{2^{56}-1}$	$2^{56} - 1$

2. Försöker hitta par av kryptotexter i de två tabellerna så att $c_i = c'_j$. I så fall är (k_i, k_j) en bra gissning på Dubbel-DES-nnyckeln.

Notera att vi inte behöver inte skapa den andra tabellen explicit. Alltså är Dubbel-DES inte mycket bättre än DES!

Bastekniker

- ▶ Bredden-först-sökning (BFS). Optimal lösning, men stor kö.
- ▶ Kostnadslikformig sökning ("Uniform cost search"). Kan ses som BFS för viktade grafer. Optimal lösning, men stor kö.
- ▶ Djupet-först-sökning (DFS). Litet minne, men inte optimal lösning.
- ▶ Iterativt djupnande DFS (ID-DFS). Kör DFS men med ett maxdjup. Om ingen lösning hittas börjar vi om med ett lite större maxdjup (normalt ett större). Litet minne, optimal, men lite tar längre tid.

Dubbelriktad sökning

När kortaste vägen mellan två kända tillstånd söks, kan man gå halva vägen från varje tillstånd och försöka hitta en kollision.

- ▶ Kör BFS med ett maxdjup från ett av tillstånden och lagra elementen som förekommer i kön upp till maxdjupet i en datastruktur för snabba uppslagningar (Hashtabell, binärt träd, sorterad lista etc).
- ▶ Kör sökning från andra tillståndet och leta kollision med köelementen från BFS.

Förgrena och uppskatta ("Branch and bound")

Ibland kan man se att ingen bättre lösning på sökproblemet kan hittas i en sökning i en delmängd.

Kappsäcksproblemet:

- ▶ Värde/vikt-par $(v_1, w_1), \dots, (v_k, w_k)$ och kapacitet W .
- ▶ Sortera så att $\frac{v_1}{w_1} \geq \dots \geq \frac{v_k}{w_k}$.
- ▶ Gör totalsökning från vänster. b_i kodar om w_i väljs eller inte.
 $\text{Search}(b_1, \dots, b_{i-1}) =$
 $\max\{\text{Search}(b_1, \dots, b_{i-1}, 1), \text{Search}(b_1, \dots, b_{i-1}, 0)\}.$
- ▶ Om $V(b_1, \dots, b_{i-1}) + \frac{v_i}{w_i}(W - W(b_1, \dots, b_{i-1}))$ är mindre än bästa lösningen hittills, så kan vi ignorera delsökningen.

Bästaförstsökning

För alla typer av uttömmande sökning kan det vara bra att **välja** ordningen för hur man söker igenom delrymder.

Även om man inte kan bevisa att en viss ordning är bättre än en annan så visar sig detta vara avgörande för att lösa så stora problem som möjligt i praktiken.

A*

Hittar väg av lägsta kostnad från ett starttillstånd till ett måltillstånd. Idén är att prova vägar som verkar bra först genom att uppskatta kostnaden att nå målet.

- ▶ $g(x)$ är avståndet från start till x .
- ▶ $h(x)$ är en **underskattning** av avståndet från x till måltillståndet.
- ▶ $f(x) = g(x) + h(x)$.
- ▶ Fungerar som BFS, men kön ersätts av en prioritetskö med f som värdefunktion.

ID-A*

Precis som för BFS, kan storleken på prioritetsskön bli ett problem.

Precis som BFS kan ersättas med ID-DFS, så kan A^* ersättas med ID- A^* .

Enda skillnaden mot ID-DFS är att vi kan unna oss att utöka djupet med den lägsta uppskattningen av återstående väglängd istället för bara 1 i varje steg.