Algengar lausnaraðferðir og ábendingar

Vífill Sverrisson

January 23, 2019

Keppnisforritun
Háskóli Íslands

Algengar lausnaraðferðir

- Complete Search eða Brute Force
- Backtracking
- Greedy approach
- Divide and Conquer
- Dynamic Programming

Lausnaraðferðir sem koma fyrir í hverri forritunarkeppni

Í þessum fyrirlestri

- Förum yfir:
  - Complete Search
  - Backtracking
  - Greedy approach
- Atli mun fjalla um 'DP' og Divide and Conquer í næstu viku

Complete search

Complete search

- Þegar við höfum takmarkað lausnarrúm
- Viljum finna það stak í rúminu sem uppfyllir gefin skilyrði
  - eða finna **öll** þau stök!
- Tiltölulega einfalt! Rennum í gegnum öll vænleg stök og athugum hver af þeim uppfylla skilyrðin með því að prófa
- Ekki beint hagkvæmt...
- En ef það tekst innan gefins tímaramma þá getum við, ásamt dómnefnd, verið sátt með lausnina
- Fyrsta aðferðin sem maður ætti að íhuga þegar nálgast er vandamál

Kostir og gallar

Kostir

- Einföld lausn
- Yfirleitt einfalt og fljótlegt í útfærslu
- Getur ekki klikkað!

Gallar

- Drullu hægt!
- Gæti verið engin fýsileg aðferð til að renna yfir lausnarrúmið
- Lausnarrúmið gæti verið ótakmarkað

Complete search

- Greinum vandamálið
    - Getum við skilgreint lausnarrúmið?
    - Hvað er lausnarúmið stórt?
        - Allar umraðanir? Öll hlutmengi? Veldismengi? Mengið sjálft?
        - $n!$, 2^n, $n^?$, n?
    - Leyfir tímaskorðurnar mér að skoða allt rúmið?
    - Stundum getur staðfestingin á lausn falið í sér of mikla tímaflækju
- Hvernig eigum við að renna yfir allt lausnarrúmið?
- Skoðum nokkur dæmi!

Renna í gegnum allar umraðanir

- Nú þegar útfært sem hluti af staðalbúnaði margra forritunarmála
    - `next_permutation` í C++
    - `itertools.permutations` í Python

```
int n = 5;
vector<int> perm(n);
for (int i = 0; i < n; i++) perm[i] = i + 1;

do {
    for (int i = 0; i < n; i++) {
        cout << perm[i] << endl;
    }

} while (next_permutation(perm.begin(), perm.end()));
```

Renna í gegnum allar umraðanir

- Athugum að fjöldi umraðana á mengi með n stökum er $n!$, þannig það er yfirleitt ekki vænlegt að renna í gegnum rúmið nema $n <= 11$, þ.e. $11! \approx 10^8$
  - Mögulega getum við takmarkað það einhvað en annars skulum við leita að lausn á öðrum miðum

Renna í gegnum öll hlutmengi

- Munið þið eftir samsvöruninni milli tvíundatalna og hlutmengja sem var fjallað um í síðasta fyrirlestri?
- Sérhver tala milli 0 upp í $2^n - 1$ lýsir ákveðnu hlutmengi af menginu $\{1, 2, \ldots, n\}$
- Við rennum bara í gegnum heiltölurnar

```
int n = 5;
for (int subset = 0; subset < (1 << n); subset++) {
    for (int i = 0; i < n; i++) {
        if ((subset & (1 << i)) != 0) {
            cout << i+1 << " ";
        }
    }
    cout << endl;
}
```

Sýnidæmi

- Gefnar 3 heiltölur A, B og C $(1 \leq A, B, C \leq 10000)$, finnum 3 frábrugðnar heiltölur x, y og z sem uppfylla jöfnurnar
    1. $x + y + z = A$
    2. $x * y * z = B$
    3. $x^2 + y^2 + z^2 = C$

- Hvert er lausnarrúmið?

Sýnidæmi

```
bool sol = false; int x, y, z;
for (int x = -100; x <= 100; x++)
  for (int y = -100; y <= 100; y++)
    for (int z = -100; z <= 100; z++)
      if (x != y && x != z && y != z && x + y + z == A &&
          x * y * z == B && x * x + y * y + z * z == C) {
          if (!sol) printf("%d %d %d\n", x, y, z);
          sol = true;
      }
```

Complete Search

- En ef við höfum ekki tíma til að reikna allt lausnarrúmið en það er samt ekki mjög stórt?
- Forreiknum!
- Skrifum forrit, sem við keyrum síðan á okkar eign vél óhamlað af tímamörkum, sem reiknar út allt lausnarrúmið
- Harðkóðum síðan forrreiknaða lausnarrúmið inni í lausnina okkar og notum uppflettingu
- Yfirleitt ekki lausnin sem dæmahöfundur var með í huga en virkar þó og stundum vænlegasta leiðin

Ábendingar til að hraða upp kóða

- Heildstæð leit er yfirleitt einföld lausn með einföldum kóða sem er keyrður oft
- Því getur lítil hraðaaukning í critical region af kóðanum skilað sér í mikilum tímasparnaði á heildarlausninni

Ábendingar til að hraða upp kóða

1. Sía (filter) vs. Framkalla (generate)
2. Snyrta strax burt óvænlega kima leitarrúmsin
3. Nýta sér samhverfur
4. Forreikna algenga og dýra útreikninga
5. Reyna að vinna sig að lausn aftan frá
6. Skrifa hagkvæman kóða
7. Nota bestu reiknirit og gagnaskipan fyrir vandamálið

Ábendingar til að hraða upp kóða

- Nokkrar ábendinga til að besta kóða
  1. Nota C++...
  2. ios_base::sync_with_stdio(false);
  3. Sækja gögn úr fylkjum röð eftir röð, skilgreina upphafstærð á vector ofl.
  4. Bit manipulation er hax hratt
  5. Nota einföldustu gagnatög sem unnt er að nota að hverju sinni
  6. StringBuilder fyrir Java notendur og char[] fyrir C++
  7. Skilgreina gagnatög einu sinni og yfirskrifa frekar en að skilgreina aftur
  8. Auðveldara að aflúsa rennslukóða en endurkvæman kóða
  9. Frekar en að sækja sama gildið margoft úr fylki þá er betra að færa það inn í breytu

Backtracking

- Höfum nú skoðað nokkrar aðferðir til að renna í gegnum allt lausnarrúmið
- Voru kannski aðeins sérhæfð tilfelli
- Væri betra að hafa einhverja almennari leið

- Kynnum Backtracking til sögunnar!

Backtracking

- Basic hugmyndin:
  1. Byrjum með "tóma" lausn
  2. Notum síðan endurkvæmni til að ferðast skref fyrir skref í gegnum lausnarrúmeð með því einfaldlega að prófa alla valmöguleika á hverju stigi
  3. Metum á hverju stigi hvort við séum með fýsilega lausn eða ekki, annars drepum við þá grein - þ.e. tökum ekki fleiri skref þaðan
  4. Þær greinar sem komast á botn endurkvæmninnar leiða til lausnar!

Backtracking

- Sauðakóði á útfærslu

```
state S;

void generate() {
    if (!is_valid(S))
        return;

    if (is_complete(S))
        print(S);

    foreach (possible next move P) {
        apply move P;
        generate();
        undo move P;
    }
}

S = empty state;
generate();
```

n drottningar

- Finnum leið til að setja n drottningar á $n \times n$ skákborð þannig að engin ógni annari drottningu
- Frekar sérhæft lausnarrúm sem við þurfum að renna í gegnum og gæti því reynst vændræðasamt að útfæra það þó það sé hægt
- Backtracking!

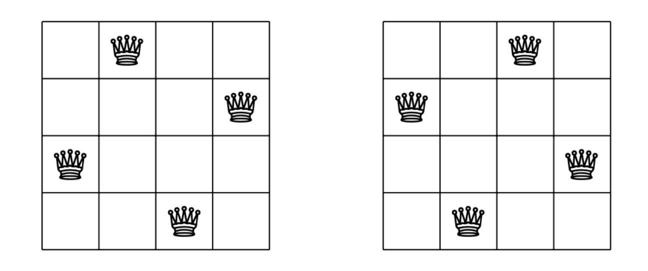
n queens

Strategían

- Vinnum okkur línu fyrir línu, byrjum efst og vinnum okkur niður borðið
- Prófum að setja niður drottningu í hverjum reit í þeirri línu og vinnum okkur þaðan niður borðið
- Leið og við leggjum niður drottningu sem ógnar annari getum við hætt við að skoða tilfelli sem leiða af því.
- Þau tilfelli sem komast á botn borðsins eru lausnir

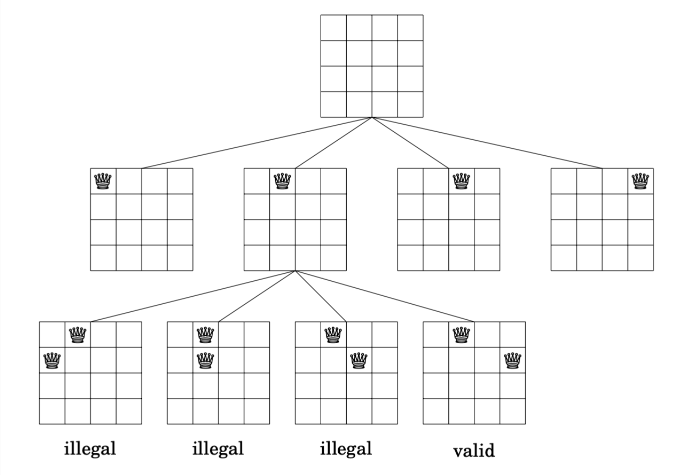
n queens

- Skoðum nú tilfellið fyrir $n = 4$
- Það eru til tvær lausnir á því:



n queens

- Sjáum nú fyrstu skrefin í endurkvæmninni



```
const int n = 4;
bool has_queen[n][n];
int queens_left = n;

// generate function

memset(has_queen, 0, sizeof(has_queen));
generate(0, 0);
```

n queens

```
void generate(int x, int y) {
    if (y == n) {
        generate(x+1, 0);
    } else if (x == n) {
        if (queens_left == 0) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    printf("%c", has_queen[i][j] ? 'Q' : '.');
                }
                printf("\n");
            }
        }
    } else {

        if (queens_left > 0 and no queen can attack cell (x,y)) {
            // try putting a queen on this cell
            has_queen[x][y] = true;
            queens_left--;

            generate(x, y+1);

            // undo the move
            has_queen[x][y] = false;
            queens_left++;
        }

        // try leaving this cell empty
        generate(x, y+1);
    }
}
```

Greedy

- Gráðugum reikniritum má lýsa þannig að þau fari þá leið á krossgötum sem lítur best út frá þeim stað, óháð því hvort það muni leiða í ógöngur seinna meir.
- Tekur aldrei til baka ákvarðanir og smíðar því lokalausnina beint
- Því mjög hagkvæmir!
- ...nema þeir skila ekki bestu lausn í öllum tilfellum

Sýnidæmi: Skiptimyntir

Tökum sem dæmi afgreiðslumann í verslun

- Hann rukkar viðskiptavininn um 66¢
- Viðskiptavinurinn gerðir með \$1 seðli
- Afgreiðslumaðurinn þarf að borga honum 34¢ til baka en vill gera það í sem fæstum myntum
- Hvernig er best að nálgast það vandamál?
- Myntirnar sem slegnar eru í þessu landi eru 1¢, 5¢, 10¢, 25¢ og 100¢

Sýnidæmi: Skiptimyntir

Afgreiðslumanns reikniritið:

```
// Látum x vera þá upphæð sem greiða þarf til baka
while(x > 0) {
     k <- stærsta myntin þ.a. k <= x;
     if (ekkert slíkt k er til) {
         return engin lausn til;
     }
     Greiddu út k¢ mynt;
     x <- x - k;
}
return 0;
```

Skilar rétt fyrir $x = 34$, þ.e. 25¢,5¢ og 4×1¢

Sýnidæmi: Skiptimyntir

- Skilar þetta okkur rétt í öllum tilfellum?

- Já!
- ...en reyndar bara fyrir þennan ákveðna myntslátt

Sýnidæmi: Skiptimyntir

Hvað ef við erum með myntirnar 1, 10, 21, 34, 70, 100?

- Þurfum að greiða til baka 140
  - Reikniritið skilar okkur 6×1, 34 og 100 Auðvitað er 2×70 betra!
- Þannig það fer eftir myntslættinum!
- Hvernig leysir maður samt ofangreint vandamál?

- Dynamic Programming!

Fin

Atli mun taka það fyrir í næsta fyrirlestri!