

Tæmandi leit

Bergur Snorrason

24. janúar 2022

Almennar nálganir lausna

- ▶ Þegar við leysum dæmi í keppnisforritun notumst við oftast við eina af eftirfarandi aðferðum:
 - ▶ *Ad hoc*,
 - ▶ *Tæmandi leit* eða *ofbeldis aðferðin* (e. *complete search*, *brute force*),
 - ▶ *Gráðug reiknirit* (e. *greedy algorithms*),
 - ▶ *Deila og drottna* (e. *divide and conquer*),
 - ▶ *Kvik bestun* (e. *dynamic programming*).
- ▶ Í síðustu viku fjölluðum við um *Ad hoc* dæmi.
- ▶ Í þessari viku fjöllum við um *tæmandi leit* og *gráðug reiknirit*.

Tæmandi leit

- ▶ Safn allra lausna dæmis kallast *lausnarrúm* dæmisins.
- ▶ *Tæmandi leit* felur í sér að leita í gegnum allt lausnarrúmið.
- ▶ Takið eftir að lausnarrúmið inniheldur líka rangar lausnir.
- ▶ Tökum dæmi.

Dæmi

- ▶ Gefinn er listi a af n mismunandi heiltölum á bilinu $[0, m]$.
- ▶ Hver þeirra er stærst?
- ▶ Til að nota tæmandi leit þurfum við að byrja á að ákvarða lausnarrúmið.
- ▶ Hér er það einfaldlega heiltölurnar á bilinu $[0, m]$.
- ▶ Okkur nægir að ítra öfugt í gegnum heiltölurnar á bilinu $[0, m]$ þangað til við lendum á tölu sem er í a .
- ▶ Við getum athugað hvort tiltekin tala sé í a með því að ítra í gegnum a , sem tekur $\mathcal{O}(n)$ tíma.
- ▶ Þessi aðferð er því $\mathcal{O}(nm)$.
- ▶ Hvaða gagnagrind úr síðasta fyrirlestri mætti nota til að bæta þessa tímaflækju?

- ▶ Almennt fáum við að ef lausnarrúmið er af stærð S og við getum athugað hverja lausn í $\mathcal{O}(T(k))$ þá er tæmandi leit $\mathcal{O}(S \cdot T(k))$.
- ▶ Gildin $n!$ og 2^n eru algengar stærðir á lausnarrúmum.
- ▶ Þar af leiðandi eru $\mathcal{O}(n \cdot n!) = \mathcal{O}((n+1)!)$ og $\mathcal{O}(n2^n)$ algengar tímaflækjur.
- ▶ Oft má á þægilegan hátt breyta slíkum lausnum í $\mathcal{O}(n!)$ og $\mathcal{O}(2^n)$.

Tæmandi leit, öll hlutmengi

- ▶ Dæmið hér á undan gæti leitt ykkur til að halda að tæmandi leit sé einföld.
- ▶ Svo er ekki alltaf.
- ▶ Tökum annað dæmi.
- ▶ Gefin er runa af n tölum. Hver er lengsta vaxandi hlutruna gefnu rununnar?
- ▶ Ef við viljum leysa þetta dæmi með tæmandi leit þá þurfum við að skoða sérhvert hlutmengi gefnu rununnar.

Útúrdúr um hlutmengi

- ▶ Ef við erum með endanlegt mengi A af stærð n getum við númerað öll stökin með tölunum $1, 2, \dots, n$.
- ▶ Sérhvert hlutmengi einkennist af því hvort stak k sé í hlutmenginu eða ekki, fyrir öll k í $\{1, 2, \dots, n\}$.
- ▶ Við fáum því að fjöldi hlutmengja í A er 2^n .

Útúrdúr um bitaframsetingu talna

- ▶ Fyrir hlutmengi H í A er til ótvírætt ákvörðuð tala b sem hefur 1 í k -ta sæti bitaframsetningar sinnar þá og því aðeins að k -ta stak A sé í H .
- ▶ Þetta gefur okkur gagntæka samsvörun milli hlutmengja A og talnanna $0, 1, \dots, 2^n - 1$.
- ▶ Talan b er vanalega kölluð *bitakennir* eða *kennir* (e. *bitmask*, *mask*) hlutmengisins H .
- ▶ Sem dæmi, ef $A = \{1, 2, 3, 4, 5, 6\}$ og $H = \{1, 3, 5, 6\}$ þá er $b = 110101_2 = 53$.
- ▶ Kennir tómamengisins er alltaf $0 = 00 \dots 00_2$ og kennir A er $2^n - 1 = 11 \dots 11_2$.

- ▶ Þegar kemur að því að nota bitakenni í forritun notum við okkur eftirfarandi:

Kennir k -ta einstökungs	$1 \ll k$
Kennir fyllimengis kennis	$\sim A$
Kennir samengis tveggja kenna	$A B$
Kennir sniðmengis tveggja kenna	$A \& B$
Kennir samhverfs mismunar tveggja kenna	$A \sim B$
Kennir mismunar tveggja kenna	$A \& (\sim B)$

- ▶ NB: Vegna forgang aðgerða í flestum forritunarmálum er góður vani að nota nóg af svigum þegar unnið er með bitaaðgerðir.
- ▶ Til dæmis er $A \& B == 0$ jafngilt $A \& (B == 0)$ í C/C++, þó við viljum yfirleitt $(A \& B) == 0$.
- ▶ Takið eftir að kennir fyllimengisins bætir við bitum utan þeirra sem við höfum áhuga því við erum í rauninni að taka fyllimengi með stærra óðal.
- ▶ Til að allt sé rétt þarf notum við $(\sim A) \& ((1 \ll n) - 1)$.

Lausn á dæminu

- ▶ Við getum nú leyst dæmið.
- ▶ Til að ítra í gegnum öll hlutmengi ítrum við í gegnum alla bitakenni mengisins.

Lausn

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int n, i, j;
6     scanf("%d", &n);
7     int a[n];
8     for (i = 0; i < n; i++) scanf("%d", &(a[i]));
9
10    int mx = 0, mask;
11    for (i = 0; i < (1 << n); i++)
12    {
13        int s[n], c = 0;
14        for (j = 0; j < n; j++) if (((1 << j)&i) != 0) s[c++] = j;
15        for (j = 1; j < c; j++) if (a[s[j]] < a[s[j - 1]]) break;
16        if (j == c && c > mx) mx = c, mask = i;
17    }
18
19    printf("%d\n", mx);
20    for (i = 0; i < n; i++) if (((1 << i)&mask) != 0) printf("%d ", a[i]);
21    printf("\n");
22
23    return 0;
24 }
```

- ▶ Hér er lausnarrúmið af stærð 2^n og við erum $\mathcal{O}(n)$ að ganga úr skugga um hvort tiltekin lausn sé í raun rétt, svo reikniritið er $\mathcal{O}(n2^n)$.

Tæmandi leit, allar umraðanir

- ▶ Við höfum oft áhuga á að ítra í gegnum allar umraðanir á lista talna.
- ▶ Munum að, ef við höfum n ólíkar tölur þá getum við raðað þeim á $n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$ vegu.
- ▶ Tökum mjög einfalt dæmi:
- ▶ Gefið er n . Prentið allar umraðanir á $1, 2, \dots, n$ í vaxandi stafrófsröð, hverja á sinni línu.
- ▶ Við getum notað okkur innbyggða fallið `next_permutation(...)` í C++.

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     int n, i;
7     cin >> n;
8     vector<int> p;
9     for (i = 0; i < n; i++) p.push_back(i + 1);
10    do {
11        for (i = 0; i < n; i++) cout << p[i] << ' ';
12        cout << '\n';
13    } while (next_permutation(p.begin(), p.end()));
14    return 0;
15 }
```

- ▶ Mikilvægt er að p sé vaxandi í upphafi því lykkjan hættir þegar það lendir á síðustu umröðunni, í stafrófsröð.
- ▶ Þessi lausn er $\mathcal{O}((n + 1)!)$.

- ▶ Tökum nú annað dæmi.
- ▶ Gefnar eru n mismunandi heiltölur.
- ▶ Raðið þeim.
- ▶ Þetta má að sjálfsgöðu leysa með innbyggðum röðunarföllum, en við viljum leysa þetta með tæmandi leit.
- ▶ Við getum notað sama forrit og áðan, með smávægilegum breytingum.


```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     int x, n, i;
7     cin >> n;
8     vector<int> p, a, b(n);
9     for (i = 0; i < n; i++)
10     {
11         cin >> x;
12         a.push_back(x);
13         p.push_back(i);
14     }
15     do {
16
17         for (i = 0; i < n; i++) b[i] = a[p[i]];
18         for (i = 0; i < n - 1; i++) if (b[i] > b[i + 1]) break;
19         if (i == n - 1) break;
20     } while (next_permutation(p.begin(), p.end()));
21     for (i = 0; i < n; i++) cout << a[p[i]] << ' ';
22     cout << endl;
23     return 0;
24 }

```

- ▶ Tímaflækjan á þessari lausn er $\mathcal{O}((n+1)!)$.
- ▶ Við getum bætt hana.
- ▶ Byrjum á að leysa dæmið án þess að nota `next_permutation(...)`.
- ▶ Við getum gert það endurkvæmt.
- ▶ Í hverju skrefi í endurkvæmninni veljum við stak sem við höfum ekki valið áður, setjum það á hlaða og höldum áfram.

```

1 #include <stdio.h>
2 #define SWAP(E, F) { int swap = (E); (E) = (F); (F) = swap; }
3 int perm(int* a, int n, int x)
4 {
5     int i;
6     if (x == n)
7     {
8         for (i = 0; i < n - 1; i++) if (a[i] > a[i + 1]) break;
9         return i < n - 1 ? 0 : 1;
10    }
11    for (i = x; i < n; i++)
12    {
13        SWAP(a[x], a[i]);
14        if (perm(a, n, x + 1)) return 1;
15        SWAP(a[x], a[i]);
16    }
17    return 0;
18 }
19
20 int main()
21 {
22     int i, n;
23     scanf("%d", &n);
24     int a[n];
25     for (i = 0; i < n; i++) scanf("%d", &a[i]);
26     perm(a, n, 0);
27     for (i = 0; i < n; i++) printf("%d ", a[i]);
28     printf("\n");
29     return 0;
30 }

```

- ▶ Gerum ráð fyrir að $n = 5$ og gefnu tölurnar séu 3 2 5 4 1.
- ▶ Við byrjum með tómann hlaða, táknum hann með x x x x x.
- ▶ Við bætum fyrst við 3 og fáum 3 x x x x.
- ▶ Síðan bætum við 2 við og fáum 3 2 x x x.
- ▶ Næst prófum við allar umraðanir 5 4 1 þar fyrir aftan.
- ▶ En við sjáum strax að það mun aldrei verða raðað, því $3 > 2$.
- ▶ Svo við getum sleppt því að skoða dýpra.

```

1 #include <stdio.h>
2 #define SWAP(E, F) { int swap = (E); (E) = (F); (F) = swap; }
3 int perm(int* a, int n, int x)
4 {
5     int i;
6     if (x == n) return 1;
7     for (i = x; i < n; i++) if (x == 0 || a[x - 1] <= a[i])
8     {
9         SWAP(a[x], a[i]);
10        if (perm(a, n, x + 1)) return 1;
11        SWAP(a[x], a[i]);
12    }
13    return 0;
14 }
15
16 int main()
17 {
18     int i, n;
19     scanf("%d", &n);
20     int a[n];
21     for (i = 0; i < n; i++) scanf("%d", &a[i]);
22     perm(a, n, 0);
23     for (i = 0; i < n; i++) printf("%d ", a[i]);
24     printf("\n");
25     return 0;
26 }

```

- ▶ Tímaflækjan eftir þessa breytingu er ekki augljós.
- ▶ Munið að tölurnar eru allar ólíkar.
- ▶ Takið eftir að sérhvert hlutmengi mun koma fyrir í hlaðanum okkar.
- ▶ Hlaðinn er einnig alltaf raðaður, svo hann inniheldur aldrei sama mengið tvisvar.
- ▶ Svo tímaflækjan er $\mathcal{O}(2^n)$.

Tæmandi leit, kostir og gallar

- ▶ Það eru ýmsir kostir við tæmandi leit.
- ▶ Til að mynda er lausnin sem reikniritin skila alltaf rétt (við sjáum á eftir aðferðir þar sem það gildir ekki)
- ▶ Tæmandi leit á það til að vera auðveld í útfærslu (þegar maður er kominn með smá æfingu).
- ▶ Á keppnum er tæmandi leit yfirleitt í léttu dæmunum, ef hún er í keppninni á annað borð.
- ▶ Keppnir innihalda frekar dæmi þar sem tæmandi leit er aðeins hluti af lausninni.

