Tæmandi leit og gráðug reiknirit

Bergur Snorrason

25. janúar 2021

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.
- Við styttum oft „tæmandi leit" í `CS` (fyrir „Complete search") og „kvik bestun" í `DP`.

Tæmandi leit

- Safn allra lausna dæmis kallast *lausnarrúm* dæmisins.
- *Tæmandi leit* felur í sér að leita í gegnum allt lausnarrúmið.
- 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 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 alltaf 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 $1, 2, ..., n$.
- Sérhvert hlutmengi einkennist af því hvort stak k sé í hlutmenginu eða ekki, fyrir öll k í $1, 2, ..., n$.
- Við fáum þá 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 $0, 1, ..., 2^n - 1$.
- 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 og kennir A er $2^n - 1$.

- Þegar kemur að því að not bitakenni í forritun notum við okkur eftirfarandi:

Kennir k-ta einstökungs	`1 << k`
Kennir fyllimengis kennis	`~A`
Kennir samengis tveggja kenna	`A|B`
Kennir sniðmengis tveggja kenna	`A&B`
Kennir samhverfs mismunar tveggja kenna	`A^B`
Kennir mismunar tveggja kenna	`A&(~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`.

Lausn á dæminu

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

Lausn

```c
#include <stdio.h>

int main()
{
    int n, i, j;
    scanf("%d", &n);
    int a[n];
    for (i = 0; i < n; i++) scanf("%d", &(a[i]));

    int mx = 0, mask;
    for (i = 0; i < (1 << n); i++)
    {
        int s[n], c = 0;
        for (j = 0; j < n; j++) if (((1 << j)&i) != 0) s[c++] = j;
        for (j = 1; j < c; j++) if (a[s[j]] < a[s[j - 1]]) break;
        if (j == c && c > mx) mx = c, mask = i;
    }

    printf("%d\n", mx);
    for (i = 0; i < n; i++) if (((1 << i)&mask) != 0) printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```

- 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 $n! = n \cdot (n-1) \cdot ... \cdot 2 \cdot 1$ vegu.
- Tökum mjög einfalt dæmi:
- Gefið er n. Prentið allar umraðanir á $1, 2, ..., n$ í vaxandi stafrófsröð, hver á sinni línu.
- Við getum notað okkur innbyggða fallið `next_permutation(...)` í C++.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, i;
    cin >> n;
    vector<int> p;
    for (i = 0; i < n; i++) p.push_back(i + 1);
    do {
        for (i = 0; i < n; i++) cout << p[i] << ' ';
        cout << '\n';
    } while (next_permutation(p.begin(), p.end()));
    return 0;
}
```

- Mikilvægt er að $\mathtt{p}$ 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 sjálfsögðu leysa með innbyggðum röðunarföllum, en við viljum leysa þetta með tæmandi leit.
- Við getum notað sama forrit og áðan, með smávægilegum breytingum.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int x, n, i;
    cin >> n;
    vector<int> p, a, b(n);
    for (i = 0; i < n; i++)
    {
        cin >> x;
        a.push_back(x);
        p.push_back(i);
    }
    do {

        for (i = 0; i < n; i++) b[i] = a[p[i]];
        for (i = 0; i < n - 1; i++) if (b[i] > b[i + 1]) break;
        if (i == n - 1) break;
    } while (next_permutation(p.begin(), p.end()));
    for (i = 0; i < n; i++) cout << a[p[i]] << ' ';
    cout << endl;
    return 0;
}
```

- 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.

```c
#include <stdio.h>
#define SWAP(E, F) { int swap = (E); (E) = (F); (F) = swap; }
int perm(int* a, int n, int x)
{
    int i;
    if (x == n)
    {
        for (i = 0; i < n - 1; i++) if (a[i] > a[i + 1]) break;
        return i < n - 1 ? 0 : 1;
    }
    for (i = x; i < n; i++)
    {
        SWAP(a[x], a[i]);
        if (perm(a, n, x + 1)) return 1;
        SWAP(a[x], a[i]);
    }
    return 0;
}

int main()
{
    int i, n;
    scanf("%d", &n);
    int a[n];
    for (i = 0; i < n; i++) scanf("%d", &a[i]);
    perm(a, n, 0);
    for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

- Gerum ráð fyrir að $n = 5$ og gefnu tölrunar 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.

```c
#include <stdio.h>
#define SWAP(E, F) { int swap = (E); (E) = (F); (F) = swap; }
int perm(int* a, int n, int x)
{
    int i;
    if (x == n) return 1;
    for (i = x; i < n; i++) if (x == 0 || a[x - 1] <= a[i])
    {
        SWAP(a[x], a[i]);
        if (perm(a, n, x + 1)) return 1;
        SWAP(a[x], a[i]);
    }
    return 0;
}

int main()
{
    int i, n;
    scanf("%d", &n);
    int a[n];
    for (i = 0; i < n; i++) scanf("%d", &a[i]);
    perm(a, n, 0);
    for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

- Tímaflækjan eftir þessa breytingu er ekki augljós.
- 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.

Gráðug reiknirit

- Reiknirit sem tekur í hverju skrefi ákvörðun byggða á því hvað lítur best út á þeim tímapunkti kallast *gráðugt*.
- Höfum í huga að það er alls ekki sjálfsagt að gráðugt reiknirit skili réttri lausn.
- Tökum dæmi.

Dæmi

- Þú vinnur í sjoppu of þarft að gefa n krónur í afgang. Í boði er ótakmarkað magn af 1 krónum, 5 krónum og 10 krónum. Hver er minnsti fjöldi af klinki sem þú getur gefið?
- Dæmi um lausn er:
- Látum k tákna dýrasta klinkið sem er ekki dýrara en n. Gefum til baka k og endurtökum fyrir $n - k$ þangað til n er 0.
- Tökum til dæmis $n = 24$. Þá myndum við gefa $10, 10, 1, 1, 1, 1$ til baka.
- Það vill svo skemmtilega til að þessi gráðuga lausn virkar fyrir öll n.
- En hvað ef við breytum aðeins dæminu?

Dæmi

- Þú vinnur í sjoppu of þarft að gefa n krónur í afgang. Í boði er ótakmarkað magn af 1 krónum, 8 krónum og 20 krónum. Hver er minnsti fjöldi af klinki sem þú getur gefið?
- Tökum til dæmis $n = 24$. Þá myndum aðferðin í glærunni á undan gefa $20, 1, 1, 1, 1$ til baka.
- En þetta er ekki besta leiðin. Betra væri að gefa $8, 8, 8$.
- Svo gráðuga aðferðin virkar bara stundum.
- Þetta dæmi er þekkt sem Skiptimyntadæmið (e. *The Coin Change Problem*) og það er ekki augljóst hvenær gráðuga lausnin virkar.

Dæmi

- Þú ert yfirmaður hjá leigubíla fyrirtæki. Í dag mættu n bílstjórar í vinnuna og það eru m leigubílar til að skipa bílstjóra á. Ekki eru þó allir bílstjórar og leigubílar skapaðir jafnir. Bíll i er h_i hestöfl og bílstjóri j getur keyrt bíla sem eru g_j hestöfl eða minna. Hver er mesti fjöldi bíla sem þú getur skipað á bílstjóra þannig að hver bílstjóri fær mest einn bíl, hver bíll er með mest einn bílstjóra og enginn bílstjóri fær bíl sem hann ræður ekki við.
- Tökum eftir að við viljum að bíll sé keyrður af þeim bílstjóra sem er óreyndastur (en þó nógu hæfur til að keyra hann).
- Byrjum á að raða bílunum og bílstjórunum í vaxandi röð.
- Við getum núna í línulegum tíma fundið þann bílstjóra sem getur keyrt fyrsta bíl.
- Til að finna bílstjóra fyrir næstu bíla leitum við aftur línulega, en höldum áfram þaðan sem við hættum áðan.

Gráðug reiknirit, annað (hefðbundnara dæmi)

Að neðan er útfærsla á lausninni á glærunum hér á undan. Fyrstu tvær tölurnar á inntakinu eru n og m. Næstu n tölur lýsa getu bílstjóranna. Síðustu m tölurnar eru hestöfl bílanna.

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, m, i, j, x, r;
    cin >> n >> m;
    vector<int> a(n), b(m);
    for (i = 0; i < n; i++) cin >> a[i];
    for (i = 0; i < m; i++) cin >> b[i];
    sort(a.begin(), a.end());
    sort(b.begin(), b.end());
    r = j = 0;
    for (i = 0; i < m; i++)
    {
        while (j < n && a[j] < b[i]) j++;
        if (j < n) r++, j++;
    }
    cout << r << endl;
    return 0;
}
```

Lausn

- Þessi lausn er $\mathcal{O}(n + m)$.

- Það fyrsta sem manni dettur í hug þegar maður les dæmi er of gráðug lausn.
- Það getur samt verið erfitt að sanna að gráðug lausn sé rétt.
- Það dugar þó að sannfæra sjálfan sig (og liðsfélag í liðakeppnum).
- Ef maður útfærir og sendir inn gráðuga lausn og fær `WA` getur verið erfitt að átta sig á því hvort maður hafi gert villu eða hvort gráðuga lausnin sé röng.

Samantekt

- Tæmandi leit er oft auðvelta að bera kennsl á, en á það til að vera of hæg. Hér má búast við TLE, en eitthvað furðulegt hefur gerst ef maður fær WA.
- Það er oft létt að semja gráðug reiknirit, en það getur verið mikil vinna að sanna að lausnin sem maður fær sé alltaf rétt. Gráðug reiknirit eiga ekki að fá TLE, en algengt er að fá WA.