Deila og drottna og kvik bestun

Bergur Snorrason

6. febrúar 2021

Almennar nálganir lausna

- Þegar við leysum dæmi í keppnisforritun notumst við oftast við eina af eftirfarandi aðferðum:

Almennar nálganir lausna

- Þegar við leysum dæmi í keppnisforritun notumst við oftast við eina af eftirfarandi aðferðum:
  - *Ad hoc*.

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*),

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*),

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*),

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*).

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 vikum fjölluðum við um Ad hoc dæmi, tæmandi leit og gráðug reiknirit.

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 vikum fjölluðum við um Ad hoc dæmi, tæmandi leit og gráðug reiknirit.
- Í þessari viku fjöllum við um deila og drottna reiknirit og kvika bestun.

Deila og drottna

- Sum dæmi má endurkvæmt skipta upp þangað til þau verða fáfengileg.

Deila og drottna

- Sum dæmi má endurkvæmt skipta upp þangað til þau verða fáfengileg.
- Síðan má líma fáfengilegu lausnirnar saman í heildarlausn í lokinn.

Deila og drottna

- Sum dæmi má endurkvæmt skipta upp þangað til þau verða fáfengileg.
- Síðan má líma fáfengilegu lausnirnar saman í heildarlausn í lokinn.
- Slík reiknirit kallast *deila og drottna* reiknirit.

Deila og drottna

- Sum dæmi má endurkvæmt skipta upp þangað til þau verða fáfengileg.
- Síðan má líma fáfengilegu lausnirnar saman í heildarlausn í lokinn.
- Slík reiknirit kallast *deila og drottna* reiknirit.
- Þessi flokkur er sjaldgæfastur.

Deila og drottna

- Sum dæmi má endurkvæmt skipta upp þangað til þau verða fáfengileg.
- Síðan má líma fáfengilegu lausnirnar saman í heildarlausn í lokinn.
- Slík reiknirit kallast *deila og drottna* reiknirit.
- Þessi flokkur er sjaldgæfastur.
- Það eru þó mörg þekkt reiknirit sem nýta sér deila og drottna.

Deila og drottna, þekkt dæmi

- Mergesort.

Deila og drottna, þekkt dæmi

- Mergesort.
- Helmingunarleit (e. *binary search*).

Deila og drottna, þekkt dæmi

- Mergesort.
- Helmingunarleit (e. *binary search*).
- Þriðjungaleit (e. *ternary search*).

Deila og drottna, þekkt dæmi

- Mergesort.
- Helmingunarleit (e. *binary search*).
- Þriðjungaleit (e. *ternary search*).
- Margföldunar reiknirit Karatsuba.

Deila og drottna, þekkt dæmi

- Mergesort.
- Helmingunarleit (e. *binary search*).
- Þriðjungaleit (e. *ternary search*).
- Margföldunar reiknirit Karatsuba.
- Margföldunar reiknirit Strassen.

Deila og drottna, þekkt dæmi

- Mergesort.
- Helmingunarleit (e. *binary search*).
- Þriðjungaleit (e. *ternary search*).
- Margföldunar reiknirit Karatsuba.
- Margföldunar reiknirit Strassen.
- Nálægustu punktar í plani.

Deila og drottna, þekkt dæmi

- Mergesort.
- Helmingunarleit (e. *binary search*).
- Þriðjungaleit (e. *ternary search*).
- Margföldunar reiknirit Karatsuba.
- Margföldunar reiknirit Strassen.
- Nálægustu punktar í plani.
- Fourier ummyndun (e. *fast Fourier transform* (`FFT`)).

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.
- Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.
- Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c.
- Endurtökum þangað til a eða b er tómur.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.
- Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c.
- Endurtökum þangað til a eða b er tómur.
- Skeytum svo því sem er eftir aftan á c.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.
- Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c.
- Endurtökum þangað til a eða b er tómur.
- Skeytum svo því sem er eftir aftan á c.
- Nú inniheldur c þau stök sem voru í a og b áður.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.
- Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c.
- Endurtökum þangað til a eða b er tómur.
- Skeytum svo því sem er eftir aftan á c.
- Nú inniheldur c þau stök sem voru í a og b áður.
- Einnig er c raðaður.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.
- Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c.
- Endurtökum þangað til a eða b er tómur.
- Skeytum svo því sem er eftir aftan á c.
- Nú inniheldur c þau stök sem voru í a og b áður.
- Einnig er c raðaður.
- Ef fjöldi staka í a og b er n þá er þetta $\mathcal{O}(\quad)$.

Merge

- Gerum ráð fyrir að við séum með tvo raðaða lista, a og b.
- Búum til nýjan, tóman lista c.
- Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c.
- Endurtökum þangað til a eða b er tómur.
- Skeytum svo því sem er eftir aftan á c.
- Nú inniheldur c þau stök sem voru í a og b áður.
- Einnig er c raðaður.
- Ef fjöldi staka í a og b er n þá er þetta $\mathcal{O}(n)$.

```
a = [1, 2, 5, 6, 8, 9]
b = [0, 3, 4, 7, 10]
c = []
```

```
a = [1, 2, 5, 6, 8, 9]
b = [3, 4, 7, 10]
c = [0]
```

```
a = [2, 5, 6, 8, 9]
b = [3, 4, 7, 10]
c = [0, 1]
```

```
a = [5, 6, 8, 9]
b = [3, 4, 7, 10]
c = [0, 1, 2]
```

```
a = [5, 6, 8, 9]
b = [4, 7, 10]
c = [0, 1, 2, 3]
```

```
a = [5, 6, 8, 9]
b = [7, 10]
c = [0, 1, 2, 3, 4]
```

```
a = [6, 8, 9]
b = [7, 10]
c = [0, 1, 2, 3, 4, 5]
```

```
a = [8, 9]
b = [7, 10]
c = [0, 1, 2, 3, 4, 5, 6]
```

```
a = [8, 9]
b = [10]
c = [0, 1, 2, 3, 4, 5, 6, 7]
```

```
a = [9]
b = [10]
c = [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
a = []
b = [10]
c = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
a = []
b = []
c = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Mergesort

- Við getum notað þessa aðferð til að raða almennum lista.

Mergesort

- Við getum notað þessa aðferð til að raða almennum lista.
- Við skiptum listanum okkar í tvo jafna hluta og köllum endurkvæmt á fallið okkar þangað til við erum með tómann lista.

Mergesort

- Við getum notað þessa aðferð til að raða almennum lista.
- Við skiptum listanum okkar í tvo jafna hluta og köllum endurkvæmt á fallið okkar þangað til við erum með tómann lista.
- Á leiðinni upp úr endurkvæmninni sameinum við svo helmingana eins og rætt var á undan.

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <assert.h>

void merge(int* a, int l, int m, int r)
{
    int i = l, j = m, b[r - l], c = 0;
    while (i < m && j < r)
    {
        if (a[j] < a[i]) b[c++] = a[j++];
        else b[c++] = a[i++];
    }
    while (i < m) b[c++] = a[i++];
    while (j < r) b[c++] = a[j++];
    for (i = l; i < r; i++) a[i] = b[i - l];
}

void mergesort(int* a, int l, int r)
{
    if (r - l < 2) return;
    int m = (l + r)/2;
    mergesort(a, l, m), mergesort(a, m, r);
    merge(a, l, m, r);
}

int main()
{
    srand(time(NULL));
    int i, n;
    scanf("%d", &n);
    int a[n];
    for (i = 0; i < n; i++) a[i] = rand()%(10*n);
    mergesort(a, 0, n);
    for (i = 0; i < n - 1; i++) assert(a[i] <= a[i + 1]);
    return 0;
}
```

Mergesort

- Mergesort er sígilt dæmi um deila og drottna reiknirit.

Mergesort

- Mergesort er sígilt dæmi um deila og drottna reiknirit.
- Við helmingum alltaf listann og tökum svo saman í línulegum tíma.

Mergesort

- Mergesort er sígilt dæmi um deila og drottna reiknirit.
- Við helmingum alltaf listann og tökum svo saman í línulegum tíma.
- Hvert stak kemur fyrir í $\mathcal{O}(\log n)$ sameiningum, svo reikniritið er $\mathcal{O}(\qquad)$.

Mergesort

- Mergesort er sígilt dæmi um deila og drottna reiknirit.
- Við helmingum alltaf listann og tökum svo saman í línulegum tíma.
- Hvert stak kemur fyrir í $\mathcal{O}(\log n)$ sameiningum, svo reikniritið er $\mathcal{O}(n \log n)$.

Mergesort

- Mergesort er sígilt dæmi um deila og drottna reiknirit.
- Við helmingum alltaf listann og tökum svo saman í línulegum tíma.
- Hvert stak kemur fyrir í $\mathcal{O}(\log n)$ sameiningum, svo reikniritið er $\mathcal{O}(n \log n)$.
- Þetta er mjög algeng tímaflækja í deila og drottna reikniritum.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- Skoðum það fyrst og alhæfum svo.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- Skoðum það fyrst og alhæfum svo.
- Látum a vera raðaðan lista af n tölum.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- Skoðum það fyrst og alhæfum svo.
- Látum a vera raðaðan lista af n tölum.
- Gerum ráð fyrir að við viljum finna t í listanum.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- Skoðum það fyrst og alhæfum svo.
- Látum a vera raðaðan lista af n tölum.
- Gerum ráð fyrir að við viljum finna t í listanum.
- Látum $m = \lfloor n/2 \rfloor$.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- Skoðum það fyrst og alhæfum svo.
- Látum a vera raðaðan lista af n tölum.
- Gerum ráð fyrir að við viljum finna t í listanum.
- Látum $m = \lfloor n/2 \rfloor$.
- Ef m-ta stakið í a er stærra en t þá getur t ekki verið í seinni helming listans.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- Skoðum það fyrst og alhæfum svo.
- Látum a vera raðaðan lista af n tölum.
- Gerum ráð fyrir að við viljum finna t í listanum.
- Látum $m = \lfloor n/2 \rfloor$.
- Ef m-ta stakið í a er stærra en t þá getur t ekki verið í seinni helming listans.
- Ef m-ta stakið í a er minna en t þá getur t ekki verið í fyrri helming listans.

Helmingunarleit

- Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- Skoðum það fyrst og alhæfum svo.
- Látum a vera raðaðan lista af n tölum.
- Gerum ráð fyrir að við viljum finna t í listanum.
- Látum $m = \lfloor n/2 \rfloor$.
- Ef m-ta stakið í a er stærra en t þá getur t ekki verið í seinni helming listans.
- Ef m-ta stakið í a er minna en t þá getur t ekki verið í fyrri helming listans.
- Svo við getum útilokað helming listans í hverri ítrun.

```c
#include <stdio.h>

int bs(int* a, int t, int l, int r)
{
    if (r - l == 1) return a[l] == t ? l : -1;
    int m = (l + r)/2;
    if (a[m] <= t) return bs(a, t, m, r);
    else return bs(a, t, l, m);
}

int main()
{
    int i, n, q, x;
    scanf("%d%d", &n, &q);
    int a[n];
    for (i = 0; i < n; i++) scanf("%d", &a[i]);
    while (q-- != 0)
    {
        scanf("%d", &x);
        printf("%d\n", bs(a, x, 0, n));
    }
    return 0;
}
```

- Helmingunarleit er $\mathcal{O}(\qquad)$, þar sem við helmingum stærð listans í hverri ítrun.

- Helmingunarleit er $\mathcal{O}(\log n)$, þar sem við helmingum stærð listans í hverri ítrun.

- Helmingunarleit er $\mathcal{O}(\log n)$, þar sem við helmingum stærð listans í hverri ítrun.
- Góð æfing í helmingurleit er að útfæra leitina þannig að hún skili vísi á fyrstu (eða síðustu) endurtekningu staksins.

- Helmingunarleit er $\mathcal{O}(\log n)$, þar sem við helmingum stærð listans í hverri ítrun.
- Góð æfing í helmingurleit er að útfæra leitina þannig að hún skili vísi á fyrstu (eða síðustu) endurtekningu staksins.
- Slíkar útgáfur að helmingunarleit nýtast þegar við förum að nota helmingurleit í almennari mynd.

- Gerum ráð fyrir að við séum með vaxandi fall $f\colon [a,b] \to \mathbb{R}$ og $y \in \mathbb{R}$.

- Gerum ráð fyrir að við séum með vaxandi fall $f\colon [a,b] \to \mathbb{R}$ og $y \in \mathbb{R}$.
- Getum við fundið $x \in [a,b]$ þannig að $f(x) = y$?

- Gerum ráð fyrir að við séum með vaxandi fall $f\colon [a,b] \to \mathbb{R}$ og $y \in \mathbb{R}$.
- Getum við fundið $x \in [a,b]$ þannig að $f(x) = y$?
- Slíkt x þarf ekki að vera til, en ef f er samfellt er það til ef $y \in [f(a), f(b)]$.

- Gerum ráð fyrir að við séum með vaxandi fall $f\colon [a,b] \to \mathbb{R}$ og $y \in \mathbb{R}$.
- Getum við fundið $x \in [a,b]$ þannig að $f(x) = y$?
- Slíkt x þarf ekki að vera til, en ef f er samfellt er það til ef $y \in [f(a), f(b)]$.
- En hvernig finnum við slíkt x?

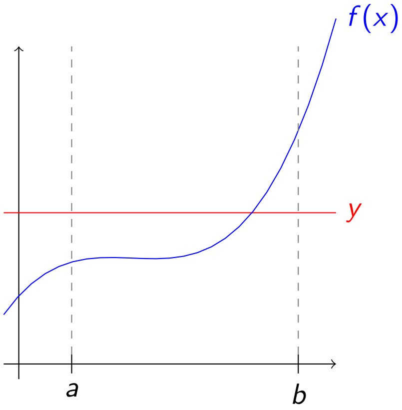
- Gerum ráð fyrir að við séum með vaxandi fall $f\colon [a, b] \to \mathbb{R}$ og $y \in \mathbb{R}$.
- Getum við fundið $x \in [a, b]$ þannig að $f(x) = y$?
- Slíkt x þarf ekki að vera til, en ef f er samfellt er það til ef $y \in [f(a), f(b)]$.
- En hvernig finnum við slíkt x?
- Við getum notað nákvæmlega sömu hugmynd og í helmingurleit í lista.

- Gerum ráð fyrir að við séum með vaxandi fall $f : [a, b] \to \mathbb{R}$ og $y \in \mathbb{R}$.
- Getum við fundið $x \in [a, b]$ þannig að $f(x) = y$?
- Slíkt x þarf ekki að vera til, en ef f er samfellt er það til ef $y \in [f(a), f(b)]$.
- En hvernig finnum við slíkt x?
- Við getum notað nákvæmlega sömu hugmynd og í helmingurleit í lista.
- Látum $m = (a + b)/2$.

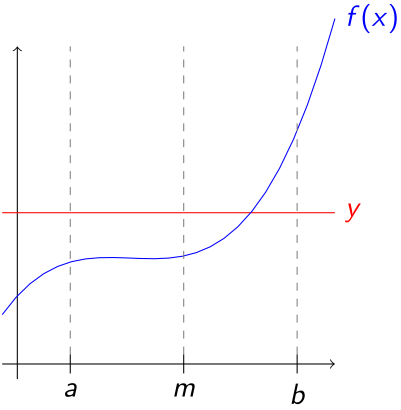
- Gerum ráð fyrir að við séum með vaxandi fall $f\colon [a,b] \to \mathbb{R}$ og $y \in \mathbb{R}$.
- Getum við fundið $x \in [a,b]$ þannig að $f(x) = y$?
- Slíkt x þarf ekki að vera til, en ef f er samfellt er það til ef $y \in [f(a), f(b)]$.
- En hvernig finnum við slíkt x?
- Við getum notað nákvæmlega sömu hugmynd og í helmingurleit í lista.
- Látum $m = (a+b)/2$.
- Ef $f(m) > t$ þá þarf $x \in [a,m]$.

- Gerum ráð fyrir að við séum með vaxandi fall $f : [a, b] \to \mathbb{R}$ og $y \in \mathbb{R}$.
- Getum við fundið $x \in [a, b]$ þannig að $f(x) = y$?
- Slíkt x þarf ekki að vera til, en ef f er samfellt er það til ef $y \in [f(a), f(b)]$.
- En hvernig finnum við slíkt x?
- Við getum notað nákvæmlega sömu hugmynd og í helmingurleit í lista.
- Látum $m = (a + b)/2$.
- Ef $f(m) > t$ þá þarf $x \in [a, m]$.
- Ef $f(m) < t$ þá þarf $x \in [m, b]$.

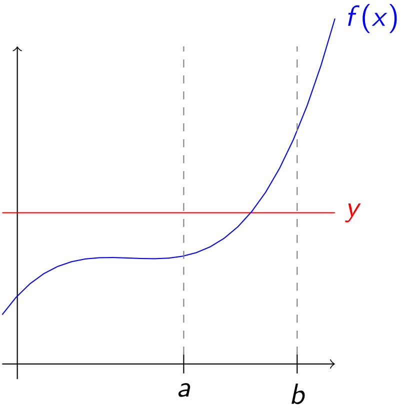
Sýnidæmi



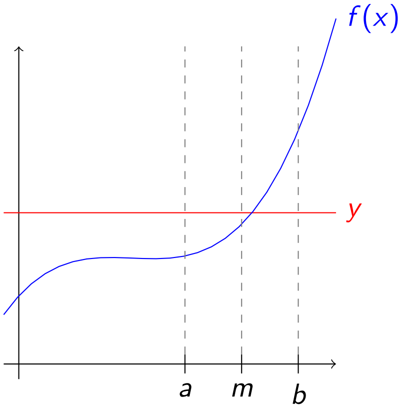
Sýnidæmi



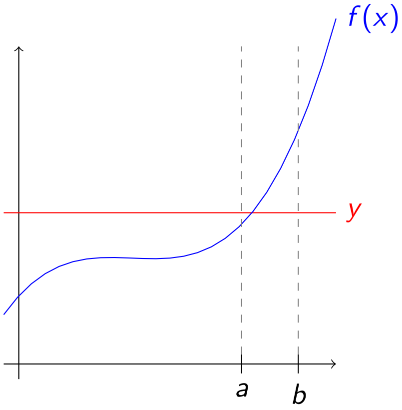
Sýnidæmi



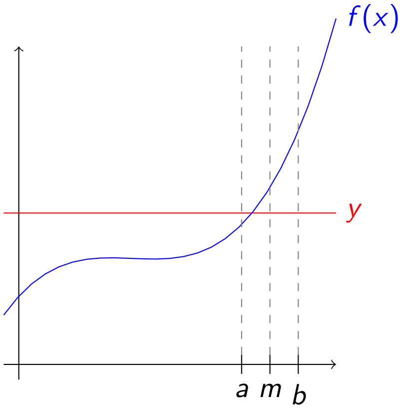
Sýnidæmi



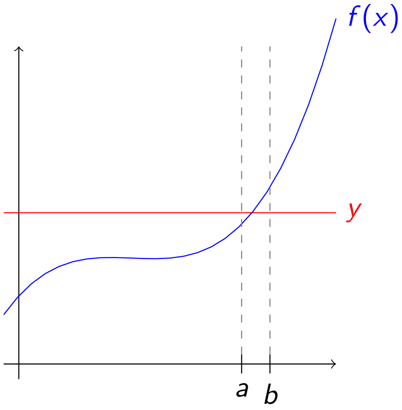
Sýnidæmi



Sýnidæmi



Sýnidæmi



- Takið eftir við munum ekki beint finna $x \in [a, b]$ þannig að $f(x) = y$.

- Takið eftir við munum ekki beint finna $x \in [a, b]$ þannig að $f(x) = y$.
- Það sem við finnum er $x \in [a, b]$ þannig að $|f(x) - y| < \varepsilon$, fyrir hvaða ε sem vera skal.

- Takið eftir við munum ekki beint finna $x \in [a, b]$ þannig að $f(x) = y$.
- Það sem við finnum er $x \in [a, b]$ þannig að $|f(x) - y| < \varepsilon$, fyrir hvaða ε sem vera skal.
- Það er þó aldrei ætlast til annars í keppnisforritun og skekkjan er alltaf gefin í úttakslýsingu dæma.

- Við getum alhæft frekar.

- Við getum alhæft frekar.
- Tökum dæmi.

- Við getum alhæft frekar.
- Tökum dæmi.
- Þú átt k ketti og n bæli fyrir kettina þína, með $k \leq n$.

- Við getum alhæft frekar.
- Tökum dæmi.
- Þú átt k ketti og n bæli fyrir kettina þína, með $k \leq n$.
- Öll bælin eru staðsett á gangi í íbúðinni þinni.

- Við getum alhæft frekar.
- Tökum dæmi.
- Þú átt k ketti og n bæli fyrir kettina þína, með $k \leq n$.
- Öll bælin eru staðsett á gangi í íbúðinni þinni.
- Ganginum má lýsa sem talnalínu og staðsetningar kattabælanna eru þá tölurnar $0 < x_1 < x_2 < \ldots < x_n < 10^9$ á talnalínunni.

- Við getum alhæft frekar.
- Tökum dæmi.
- Þú átt k ketti og n bæli fyrir kettina þína, með $k \leq n$.
- Öll bælin eru staðsett á gangi í íbúðinni þinni.
- Ganginum má lýsa sem talnalínu og staðsetningar kattabælanna eru þá tölurnar $0 < x_1 < x_2 < \ldots < x_n < 10^9$ á talnalínunni.
- En kettir eru einfarar svo þeir vilja hafa sem mesta fjarlægð í næsta kött.

- Við getum alhæft frekar.
- Tökum dæmi.
- Þú átt k ketti og n bæli fyrir kettina þína, með $k \leq n$.
- Öll bælin eru staðsett á gangi í íbúðinni þinni.
- Ganginum má lýsa sem talnalínu og staðsetningar kattabælanna eru þá tölurnar $0 < x_1 < x_2 < \ldots < x_n < 10^9$ á talnalínunni.
- En kettir eru einfarar svo þeir vilja hafa sem mesta fjarlægð í næsta kött.
- Þú átt að raða köttum á bælin þannig að nálægustu kettirnir eru sem lengst frá hvorum öðrum.

- Þetta dæmi má leysa með helmingunarleit.

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
  - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
  - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?
- Þetta dæmi má leysa gráðugt.

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
  - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?
- Þetta dæmi má leysa gráðugt.
- Við töpum aldrei á því að setja kött á bælið staðsett á x_1.

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
  - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?
- Þetta dæmi má leysa gráðugt.
- Við töpum aldrei á því að setja kött á bælið staðsett á x_1.
- Við megum þá ekki setja kött á bæli sem liggja í $[x_1, x_1 + r]$.

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
    - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?
- Þetta dæmi má leysa gráðugt.
- Við töpum aldrei á því að setja kött á bælið staðsett á x_1.
- Við megum þá ekki setja kött á bæli sem liggja í $[x_1, x_1 + r]$.
- Útilokum þau og veljum minnsta bælið sem er eftir, og endurtökum.

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
    - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?
- Þetta dæmi má leysa gráðugt.
- Við töpum aldrei á því að setja kött á bælið staðsett á x_1.
- Við megum þá ekki setja kött á bæli sem liggja í $[x_1, x_1 + r]$.
- Útilokum þau og veljum minnsta bælið sem er eftir, og endurtökum.
- Ef við komum fyrir k, eða fleiri, köttum svona þá er svarið við nýju spurningunni "já", en annars "nei".

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
  - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?
- Þetta dæmi má leysa gráðugt.
- Við töpum aldrei á því að setja kött á bælið staðsett á x_1.
- Við megum þá ekki setja kött á bæli sem liggja í $[x_1, x_1 + r]$.
- Útilokum þau og veljum minnsta bælið sem er eftir, og endurtökum.
- Ef við komum fyrir k, eða fleiri, köttum svona þá er svarið við nýju spurningunni "já", en annars "nei".
- Þetta tekur $\mathcal{O}(\qquad)$.

- Þetta dæmi má leysa með helmingunarleit.
- Byrjum á að svara annari spurningu:
  - Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r?
- Þetta dæmi má leysa gráðugt.
- Við töpum aldrei á því að setja kött á bælið staðsett á x_1.
- Við megum þá ekki setja kött á bæli sem liggja í $[x_1, x_1 + r]$.
- Útilokum þau og veljum minnsta bælið sem er eftir, og endurtökum.
- Ef við komum fyrir k, eða fleiri, köttum svona þá er svarið við nýju spurningunni "já", en annars "nei".
- Þetta tekur $\mathcal{O}(n + k)$.

- Tökum þó eftir að ef við komum fyrir öllum köttunum með fjarlægð r_0 þá gerum við það líka fyrir $r < r_0$.

- Tökum þó eftir að ef við komum fyrir öllum köttunum með fjarlægð r_0 þá gerum við það líka fyrir $r < r_0$.
- Skilgreinum fall

$$f(r) = \begin{cases} 1, & \text{ef koma má fyrir } k, \\ & \text{eða fleiri, köttum með fjarlægð } r \\ 0, & \text{annars.} \end{cases}$$

- Tökum þó eftir að ef við komum fyrir öllum köttunum með fjarlægð r_0 þá gerum við það líka fyrir $r < r_0$.
- Skilgreinum fall

$$f(r) = \begin{cases} 1, & \text{ef koma má fyrir } k, \\ & \text{eða fleiri, köttum með fjarlægð } r \\ 0, & \text{annars.} \end{cases}$$

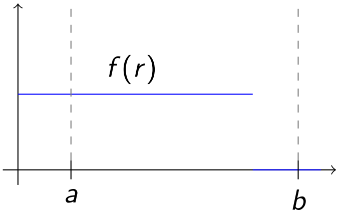
- Við getum nú umorðað upprunarlega dæmið sem: “Finnið stærsta r þannig að $f(r) = 1$”.

- Tökum þó eftir að ef við komum fyrir öllum köttunum með fjarlægð r_0 þá gerum við það líka fyrir $r < r_0$.
- Skilgreinum fall

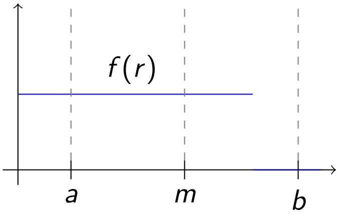
$$f(r) = \begin{cases} 1, & \text{ef koma má fyrir } k, \\ & \text{eða fleiri, köttum með fjarlægð } r \\ 0, & \text{annars.} \end{cases}$$

- Við getum nú umorðað upprunarlega dæmið sem: "Finnið stærsta r þannig að $f(r) = 1$".
- En nú er fallið f minnkandi (samkvæmt efsta punktinum á glærunni), svo við getum fundið slíkt gildi með helmingunar leit.

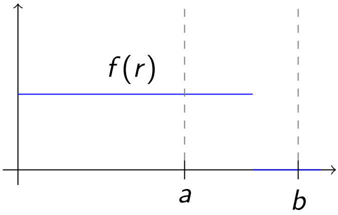
Sýnidæmi



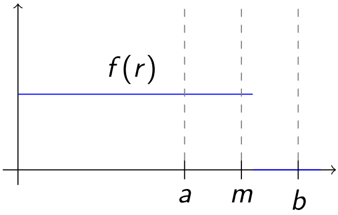
Sýnidæmi



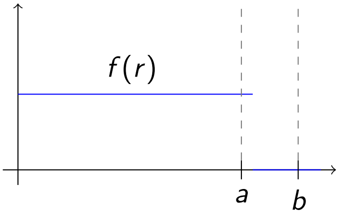
Sýnidæmi



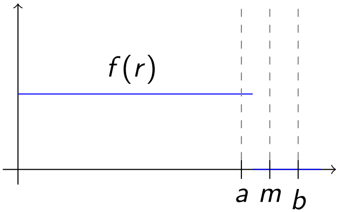
Sýnidæmi



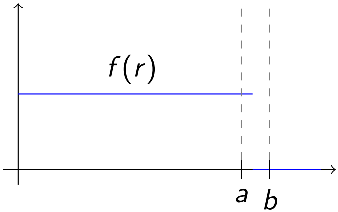
Sýnidæmi



Sýnidæmi



Sýnidæmi



- Ef við látum $M = 10^9/\varepsilon$, þar sem ε er leyfileg skekkja í dæminu, þá er lausnin $\mathcal{O}(\qquad\qquad)$.

- Ef við látum $M = 10^9/\varepsilon$, þar sem ε er leyfileg skekkja í dæminu, þá er lausnin $\mathcal{O}((k + n)\log M)$.

- Ef við látum $M = 10^9/\varepsilon$, þar sem ε er leyfileg skekkja í dæminu, þá er lausnin $\mathcal{O}((k+n)\log M)$.
- Hér gerum við ekki ráð fyrir að svarið sé heiltala.

- Ef við látum $M = 10^9/\varepsilon$, þar sem ε er leyfileg skekkja í dæminu, þá er lausnin $\mathcal{O}((k+n)\log M)$.
- Hér gerum við ekki ráð fyrir að svarið sé heiltala.
- Ef við gerum ráð fyrir því verður tímaflækjan eins, nema með $M = 10^9$.

- Það sem við gerðum í raun var að breyta dæminu úr “finnið minnsta/stærsta gildið þannig að...” yfir í “tökum ákveðið gildi og athugum hvort að...”.

- Það sem við gerðum í raun var að breyta dæminu úr “finnið minnsta/stærsta gildið þannig að...” yfir í “tökum ákveðið gildi og athugum hvort að...”.
- Þetta er algengasta notkunin á helmingunarleit í keppnisforritun.

- Það sem við gerðum í raun var að breyta dæminu úr “finnið minnsta/stærsta gildið þannig að...” yfir í “tökum ákveðið gildi og athugum hvort að...”.
- Þetta er algengasta notkunin á helmingunarleit í keppnisforritun.
- Algengt er helmingurleit af þessum toga sé hluti af erfiðum dæmum.

Þriðjungaleit

- Gerum ráð fyrir að við séum með kúpt fall $f : [a, b] \to \mathbb{R}$.

Þriðjungaleit

- Gerum ráð fyrir að við séum með kúpt fall $f : [a, b] \to \mathbb{R}$.
- Munið að fall er kúpt ef $f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2)$, fyrir öll $t \in [0, 1]$ og $x_1, x_2 \in [a, b]$.

Þriðjungaleit

- Gerum ráð fyrir að við séum með kúpt fall $f : [a, b] \to \mathbb{R}$.
- Munið að fall er kúpt ef $f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2)$, fyrir öll $t \in [0, 1]$ og $x_1, x_2 \in [a, b]$.
- Hvernig finnum við útgildi (há- og lággildi) fallsins?

Þriðjungaleit

- Gerum ráð fyrir að við séum með kúpt fall $f : [a, b] \to \mathbb{R}$.
- Munið að fall er kúpt ef $f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2)$, fyrir öll $t \in [0, 1]$ og $x_1, x_2 \in [a, b]$.
- Hvernig finnum við útgildi (há- og lággildi) fallsins?
- Auðvelt er að finna hágildi.

Þriðjungaleit

- Gerum ráð fyrir að við séum með kúpt fall $f : [a, b] \to \mathbb{R}$.
- Munið að fall er kúpt ef $f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$, fyrir öll $t \in [0, 1]$ og $x_1, x_2 \in [a, b]$.
- Hvernig finnum við útgildi (há- og lággildi) fallsins?
- Auðvelt er að finna hágildi.
- Gerum ráð fyrir að $f(a) \leq f(b)$.

Þriðjungaleit

- Gerum ráð fyrir að við séum með kúpt fall $f\colon [a,b] \to \mathbb{R}$.
- Munið að fall er kúpt ef
  $f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2)$, fyrir öll $t \in [0,1]$
  og $x_1, x_2 \in [a,b]$.
- Hvernig finnum við útgildi (há- og lággildi) fallsins?
- Auðvelt er að finna hágildi.
- Gerum ráð fyrir að $f(a) \le f(b)$.
- Fyrir öll $x \in [a,b]$ gildir þá að

$$f(x) = f(at + (1-t)b) \le tf(a) + (1-t)f(b)$$

$$\le tf(b) + (1-t)f(b) = f(b),$$

með $t = (x-b)/(a-b)$.

Þriðjungaleit

- Gerum ráð fyrir að við séum með kúpt fall $f : [a, b] \to \mathbb{R}$.
- Munið að fall er kúpt ef $f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2)$, fyrir öll $t \in [0, 1]$ og $x_1, x_2 \in [a, b]$.
- Hvernig finnum við útgildi (há- og lággildi) fallsins?
- Auðvelt er að finna hágildi.
- Gerum ráð fyrir að $f(a) \leq f(b)$.
- Fyrir öll $x \in [a, b]$ gildir þá að

$$f(x) = f(at + (1 - t)b) \leq tf(a) + (1 - t)f(b)$$

$$\leq tf(b) + (1 - t)f(b) = f(b),$$

  með $t = (x - b)/(a - b)$.
- Svo hágildi fæst í endapuntkunum a eða b.

Þriðjungaleit

- Við getum fundið lággildið tölulega, svipað og með helmingunarleit.

Þriðjungaleit

- Við getum fundið lággildið tölulega, svipað og með helmingunarleit.
- Við veljum punkta $m_1, m_2 \in [a, b]$ þannig að bilin $[a, m_1]$, $[m_1, m_2]$ og $[m_2, b]$ séu jafn löng.

Þriðjungaleit

- Við getum fundið lággildið tölulega, svipað og með helmingunarleit.
- Við veljum punkta $m_1, m_2 \in [a, b]$ þannig að bilin $[a, m_1]$, $[m_1, m_2]$ og $[m_2, b]$ séu jafn löng.
- Við skoðum svo fallgildin $f(m_1)$ og $f(m_2)$.

Þriðjungaleit

- Við getum fundið lággildið tölulega, svipað og með helmingunarleit.
- Við veljum punkta $m_1, m_2 \in [a, b]$ þannig að bilin $[a, m_1]$, $[m_1, m_2]$ og $[m_2, b]$ séu jafn löng.
- Við skoðum svo fallgildin $f(m_1)$ og $f(m_2)$.
- Ef $f(m_1) < f(m_2)$ þá getur lággildið ekki legið á bilinu $[m_2, b]$, svo við getum útilokað það bil í leitinni.

Þriðjungaleit

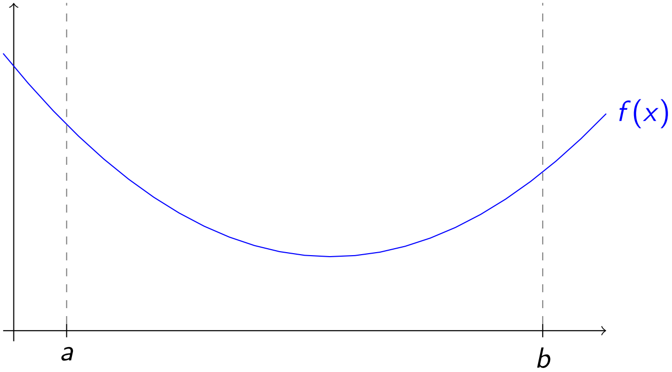
- Við getum fundið lággildið tölulega, svipað og með helmingunarleit.
- Við veljum punkta $m_1, m_2 \in [a, b]$ þannig að bilin $[a, m_1]$, $[m_1, m_2]$ og $[m_2, b]$ séu jafn löng.
- Við skoðum svo fallgildin $f(m_1)$ og $f(m_2)$.
- Ef $f(m_1) < f(m_2)$ þá getur lággildið ekki legið á bilinu $[m_2, b]$, svo við getum útilokað það bil í leitinni.
- Ef $f(m_2) < f(m_1)$ þá getur lággildið ekki legið á bilinu $[a, m_1]$, svo við getum útilokað það bil í leitinni.

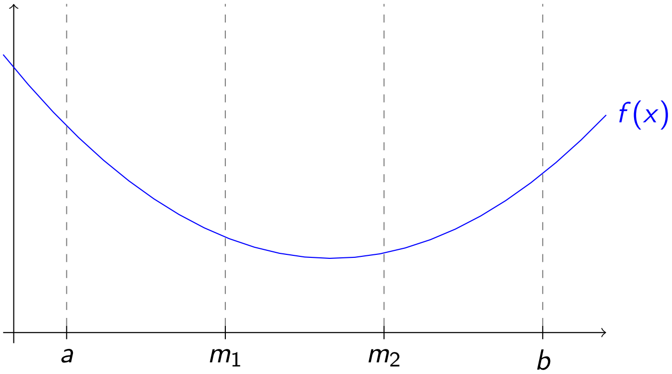
Þriðjungaleit

- Við getum fundið lággildið tölulega, svipað og með helmingunarleit.
- Við veljum punkta $m_1, m_2 \in [a, b]$ þannig að bilin $[a, m_1]$, $[m_1, m_2]$ og $[m_2, b]$ séu jafn löng.
- Við skoðum svo fallgildin $f(m_1)$ og $f(m_2)$.
- Ef $f(m_1) < f(m_2)$ þá getur lággildið ekki legið á bilinu $[m_2, b]$, svo við getum útilokað það bil í leitinni.
- Ef $f(m_2) < f(m_1)$ þá getur lággildið ekki legið á bilinu $[a, m_1]$, svo við getum útilokað það bil í leitinni.
- Ef $f(m_2) = f(m_1)$ þá þarf lággildið að liggja á bilinu $[m_1, m_2]$.

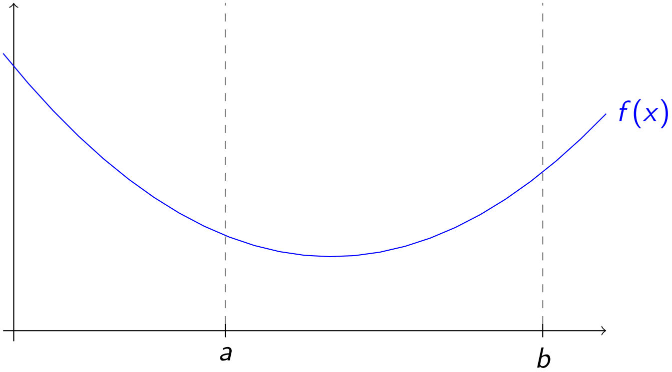
Þriðjungaleit

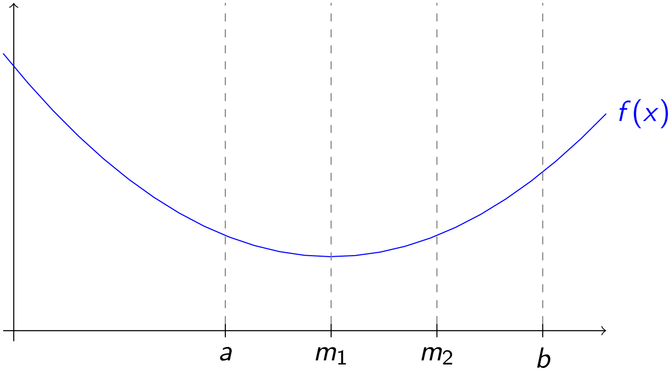
- Við getum fundið lággildið tölulega, svipað og með helmingunarleit.
- Við veljum punkta $m_1, m_2 \in [a, b]$ þannig að bilin $[a, m_1]$, $[m_1, m_2]$ og $[m_2, b]$ séu jafn löng.
- Við skoðum svo fallgildin $f(m_1)$ og $f(m_2)$.
- Ef $f(m_1) < f(m_2)$ þá getur lággildið ekki legið á bilinu $[m_2, b]$, svo við getum útilokað það bil í leitinni.
- Ef $f(m_2) < f(m_1)$ þá getur lággildið ekki legið á bilinu $[a, m_1]$, svo við getum útilokað það bil í leitinni.
- Ef $f(m_2) = f(m_1)$ þá þarf lággildið að liggja á bilinu $[m_1, m_2]$.
- Þetta stafar allt af því að kúpt föll taka hágildi í öðru hvorum endapunkta sinna.


$f(x)$
a
b

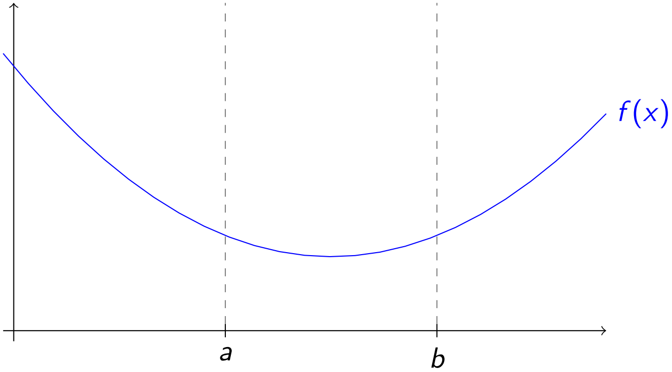


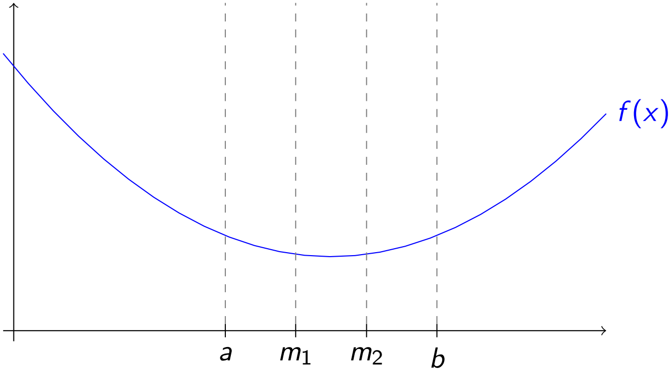
f(x)
a
m1
m2
b

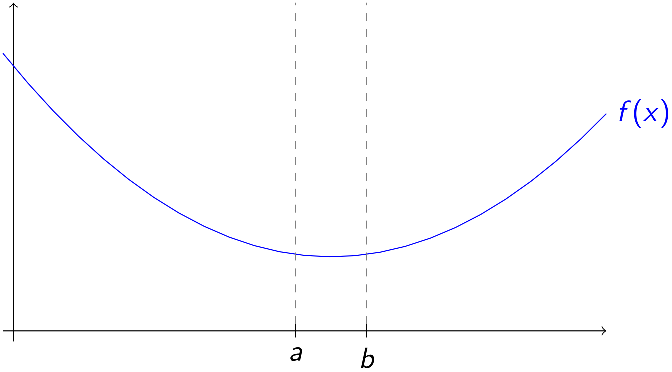

$f(x)$
a
b

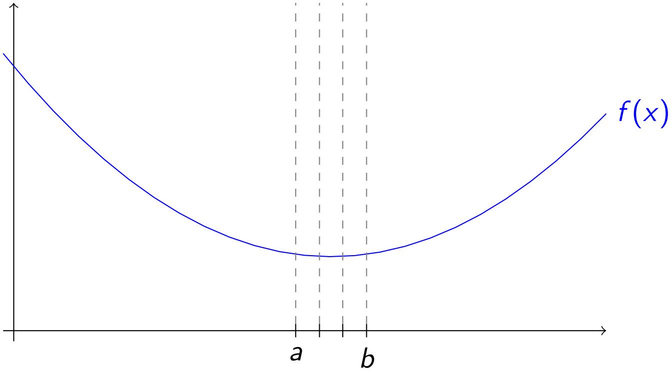


$f(x)$
a
m_1
m_2
b

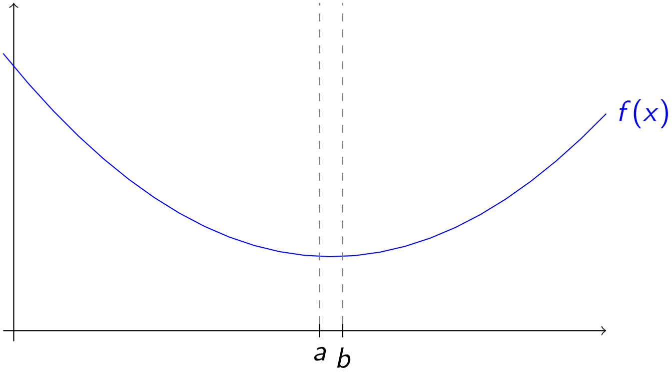

$f(x)$
a
b


$f(x)$
a
m_1
m_2
b


$f(x)$
a
b



f(x)
a
b



$f(x)$
a
b

- Við notum okkur oft að tvídiffranlegt fall er kúpt ef og aðeins ef önnur afleiðan er jákvæð.

- Við notum okkur oft að tvídiffranlegt fall er kúpt ef og aðeins ef önnur afleiðan er jákvæð.
- Umfjöllunin okkar heimfærist á eðlilegan hátt yfir á hvelf föll.

- Við notum okkur oft að tvídiffranlegt fall er kúpt ef og aðeins ef önnur afleiðan er jákvæð.
- Umfjöllunin okkar heimfærist á eðlilegan hátt yfir á hvelf föll.
- Þriðjungaleit er algeng í rúmfræði því Evklíðska firðinni er kúpt.

```c
#include <stdio.h>
#define EPS 1e-9

double f(double x)
{
    return 5.0 - 1.2*x + 0.1*x*x;
}

int main()
{
    double a = 1.0, b = 10.0, m1, m2;
    while (b - a > EPS)
    {
        m1 = (a + a + b)/3.0;
        m2 = (a + b + b)/3.0;
        if (f(m1) > f(m2)) a = m1;
        else b = m2;
    }
    printf("f(%.2f) = %.2f\n", a, f(a));
    return 0;
}
```

- Tökum eitt hefðbundið sýnidæmi í lokinn.

- Tökum eitt hefðbundið sýnidæmi í lokinn.
- Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.

- Tökum eitt hefðbundið sýnidæmi í lokinn.
- Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.
- Nánar, þú ætlar að fá lánaðar 100 danskar krónur á einhverjum degi og skipta þeim um leið í íslenskar krónur.

- Tökum eitt hefðbundið sýnidæmi í lokinn.
- Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.
- Nánar, þú ætlar að fá lánaðar 100 danskar krónur á einhverjum degi og skipta þeim um leið í íslenskar krónur.
- Eftir einhvern fjölda daga þarft þú svo að kaupa 100 danskar krónur til að borga lánið, ásamt því að borga K íslenskar krónur á dag í lánakostnað.

- Tökum eitt hefðbundið sýnidæmi í lokinn.
- Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.
- Nánar, þú ætlar að fá lánaðar 100 danskar krónur á einhverjum degi og skipta þeim um leið í íslenskar krónur.
- Eftir einhvern fjölda daga þarft þú svo að kaupa 100 danskar krónur til að borga lánið, ásamt því að borga K íslenskar krónur á dag í lánakostnað.
- Hver er mesti fjöldi íslenska króna sem þú getur grætt?

- Tökum eitt hefðbundið sýnidæmi í lokinn.
- Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.
- Nánar, þú ætlar að fá lánaðar 100 danskar krónur á einhverjum degi og skipta þeim um leið í íslenskar krónur.
- Eftir einhvern fjölda daga þarft þú svo að kaupa 100 danskar krónur til að borga lánið, ásamt því að borga K íslenskar krónur á dag í lánakostnað.
- Hver er mesti fjöldi íslenska króna sem þú getur grætt?
- Fyrsta lína inntaksins inniheldur tvær tölur, $1 \leq n \leq 10^5$ og $1 \leq k \leq 100$.

- Tökum eitt hefðbundið sýnidæmi í lokinn.
- Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.
- Nánar, þú ætlar að fá lánaðar 100 danskar krónur á einhverjum degi og skipta þeim um leið í íslenskar krónur.
- Eftir einhvern fjölda daga þarft þú svo að kaupa 100 danskar krónur til að borga lánið, ásamt því að borga K íslenskar krónur á dag í lánakostnað.
- Hver er mesti fjöldi íslenska króna sem þú getur grætt?
- Fyrsta lína inntaksins inniheldur tvær tölur, $1 \leq n \leq 10^5$ og $1 \leq k \leq 100$.
- Næsta lína inniheldur n heiltölur $1 \leq x_1, x_2, ..., x_n \leq 10^5$.

- Tökum eitt hefðbundið sýnidæmi í lokinn.
- Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.
- Nánar, þú ætlar að fá lánaðar 100 danskar krónur á einhverjum degi og skipta þeim um leið í íslenskar krónur.
- Eftir einhvern fjölda daga þarft þú svo að kaupa 100 danskar krónur til að borga lánið, ásamt því að borga K íslenskar krónur á dag í lánakostnað.
- Hver er mesti fjöldi íslenska króna sem þú getur grætt?
- Fyrsta lína inntaksins inniheldur tvær tölur, $1 \leq n \leq 10^5$ og $1 \leq k \leq 100$.
- Næsta lína inniheldur n heiltölur $1 \leq x_1, x_2, ..., x_n \leq 10^5$.
- Hér tákna x_i fjölda íslenska króna sem ein dönsk króna kostar á i-ta degi.

- Skoðum sýniinntök.

- Skoðum sýniinntök.

```
Sample input 1                  Sample output 1
5 10                            98950
1000 980 960 940 10


Sample input 2                  Sample output 2
5 100                           100
100 100 100 103 100
```

- Skoðum sýniinntök.

```
Sample input 1                  Sample output 1
5 10                            98950
1000 980 960 940 10


Sample input 2                  Sample output 2
5 100                           100
100 100 100 103 100
```

- Í fyrra tilfellinu viljum við taka lánið á fyrsta degi og borga það á síðasta degi.

- Skoðum sýniinntök.

```
Sample input 1
5 10
1000 980 960 940 10


Sample input 2
5 100
100 100 100 103 100
```

```
Sample output 1
98950


Sample output 2
100
```

- Í fyrra tilfellinu viljum við taka lánið á fyrsta degi og borga það á síðasta degi.
- Við fáum $100 \cdot 1000 = 10^5$ íslenskar krónur á fyrst degi og borgum $100 \cdot 10 = 10^3$ íslenskar krónur á síðasta degi.

- Skoðum sýniinntök.

```
Sample input 1                    Sample output 1
5 10                              98950
1000 980 960 940 10


Sample input 2                    Sample output 2
5 100                             100
100 100 100 103 100
```

- Í fyrra tilfellinu viljum við taka lánið á fyrsta degi og borga það á síðasta degi.
- Við fáum $100 \cdot 1000 = 10^5$ íslenskar krónur á fyrst degi og borgum $100 \cdot 10 = 10^3$ íslenskar krónur á síðasta degi.
- Við borgum svo $5 \cdot 10 = 50$ íslenskar krónur í lánakostnað.

- Skoðum sýniinntök.

```
Sample input 1                      Sample output 1
5 10                                98950
1000 980 960 940 10


Sample input 2                      Sample output 2
5 100                               100
100 100 100 103 100
```

- Í fyrra tilfellinu viljum við taka lánið á fyrsta degi og borga það á síðasta degi.
- Við fáum $100 \cdot 1000 = 10^5$ íslenskar krónur á fyrst degi og borgum $100 \cdot 10 = 10^3$ íslenskar krónur á síðasta degi.
- Við borgum svo $5 \cdot 10 = 50$ íslenskar krónur í lánakostnað.
- Svo við endum með $10^5 - 10^3 - 50 = 98950$ íslenskar krónur.

- Skoðum sýniinntök.

```
Sample input 1
5 10
1000 980 960 940 10


Sample input 2
5 100
100 100 100 103 100
```

```
Sample output 1
98950




Sample output 2
100
```

- Í seinna tilfellinu viljum við taka lánið á fjórða degi og borga það á síðasta degi.

- Skoðum sýniinntök.

```
Sample input 1                      Sample output 1
5 10                                98950
1000 980 960 940 10


Sample input 2                      Sample output 2
5 100                               100
100 100 100 103 100
```

- Í seinna tilfellinu viljum við taka lánið á fjórða degi og borga það á síðasta degi.
- Við fáum $100 \cdot 103 = 10300$ íslenskar krónur á fyrst degi og borgum $100 \cdot 100 = 10^4$ íslenskar krónur á síðasta degi.

- Skoðum sýniinntök.

```
Sample input 1                    Sample output 1
5 10                              98950
1000 980 960 940 10


Sample input 2                    Sample output 2
5 100                             100
100 100 100 103 100
```

- Í seinna tilfellinu viljum við taka lánið á fjórða degi og borga það á síðasta degi.
- Við fáum $100 \cdot 103 = 10300$ íslenskar krónur á fyrst degi og borgum $100 \cdot 100 = 10^4$ íslenskar krónur á síðasta degi.
- Við borgum svo $2 \cdot 100 = 200$ íslenskar krónur í lánakostnað.

- Skoðum sýniinntök.

```
Sample input 1
5 10
1000 980 960 940 10


Sample input 2
5 100
100 100 100 103 100
```

```
Sample output 1
98950

Sample output 2
100
```

- Í seinna tilfellinu viljum við taka lánið á fjórða degi og borga það á síðasta degi.
- Við fáum $100 \cdot 103 = 10300$ íslenskar krónur á fyrst degi og borgum $100 \cdot 100 = 10^4$ íslenskar krónur á síðasta degi.
- Við borgum svo $2 \cdot 100 = 200$ íslenskar krónur í lánakostnað.
- Svo við endum með $10300 - 10^4 - 200 = 100$ íslenskar krónur.

- Til að nota deila og drottna þurfum við að taka eftir einu.

- Til að nota deila og drottna þurfum við að taka eftir einu.
- Gerum ráð fyrir að engin lánakostnaður sé greiddur síðasta.

- Til að nota deila og drottna þurfum við að taka eftir einu.
- Gerum ráð fyrir að engin lánakostnaður sé greiddur síðasta.
- Það einfaldar reikninga og við getum alltaf bætt honum við eftir á.

- Til að nota deila og drottna þurfum við að taka eftir einu.
- Gerum ráð fyrir að engin lánakostnaður sé greiddur síðasta.
- Það einfaldar reikninga og við getum alltaf bætt honum við eftir á.
- Táknum með $f(i, j)$ þann gróða (eða tap) sem fæst með því að taka lánið á i-ta degi og borga það á j-degi degi, og $g(i)$ vera gengið á i-ta degi.

- Til að nota deila og drottna þurfum við að taka eftir einu.
- Gerum ráð fyrir að engin lánakostnaður sé greiddur síðasta.
- Það einfaldar reikninga og við getum alltaf bætt honum við eftir á.
- Táknum með $f(i, j)$ þann gróða (eða tap) sem fæst með því að taka lánið á i-ta degi og borga það á j-degi degi, og $g(i)$ vera gengið á i-ta degi.
- Við fáum nú að $f(i, j) = 100 \cdot g(i) - 100 \cdot g(j) - (j - i) \cdot k$.

- Til að nota deila og drottna þurfum við að taka eftir einu.
- Gerum ráð fyrir að engin lánakostnaður sé greiddur síðasta.
- Það einfaldar reikninga og við getum alltaf bætt honum við eftir á.
- Táknum með $f(i,j)$ þann gróða (eða tap) sem fæst með því að taka lánið á i-ta degi og borga það á j-degi degi, og $g(i)$ vera gengið á i-ta degi.
- Við fáum nú að $f(i,j) = 100 \cdot g(i) - 100 \cdot g(j) - (j-i) \cdot k$.
- Ef a, b og c eru heiltölur þannig að $1 \leq a < b < c \leq n$ þá fæst

$$
\begin{aligned}
f(a,b) + f(b,c) =& 100 \cdot (g(a) - g(b)) - (b-a) \cdot k \\
& + 100 \cdot (g(b) - g(c)) - (c-b) \cdot k \\
=& 100 \cdot (g(a) - g(c)) - (c-a) \cdot k \\
=& f(a,c).
\end{aligned}
$$

- Látum nú $m = \lfloor n/2 \rfloor$.

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:
    - Báðir endapunktar bestu lausnarinnar liggja á $[1, m-1]$.

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:
  - Báðir endapunktar bestu lausnarinnar liggja á $[1, m - 1]$.
  - Báðir endapunktar bestu lausnarinnar liggja á $[m, n]$.

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:
  - Báðir endapunktar bestu lausnarinnar liggja á $[1, m-1]$.
  - Báðir endapunktar bestu lausnarinnar liggja á $[m, n]$.
  - Annar endapunktur bestu lausnarinnar liggur á $[1, m-1]$ og hinn liggur á $[m, n]$.

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:
  - Báðir endapunktar bestu lausnarinnar liggja á $[1, m-1]$.
  - Báðir endapunktar bestu lausnarinnar liggja á $[m, n]$.
  - Annar endapunktur bestu lausnarinnar liggur á $[1, m-1]$ og hinn liggur á $[m, n]$.
- Fyrri tvö tilfellin má leysa með einfaldri endurkvæmni.

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:
  - Báðir endapunktar bestu lausnarinnar liggja á $[1, m-1]$.
  - Báðir endapunktar bestu lausnarinnar liggja á $[m, n]$.
  - Annar endapunktur bestu lausnarinnar liggur á $[1, m-1]$ og hinn liggur á $[m, n]$.
- Fyrri tvö tilfellin má leysa með einfaldri endurkvæmni.
- Fyrir síðasta tilfellið nýtum við gegnvirknina af fyrri glærunni.

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:
  - Báðir endapunktar bestu lausnarinnar liggja á $[1, m-1]$.
  - Báðir endapunktar bestu lausnarinnar liggja á $[m, n]$.
  - Annar endapunktur bestu lausnarinnar liggur á $[1, m-1]$ og hinn liggur á $[m, n]$.
- Fyrri tvö tilfellin má leysa með einfaldri endurkvæmni.
- Fyrir síðasta tilfellið nýtum við gegnvirknina af fyrri glærunni.
- Við viljum finna bestu lausnina sem liggur í gegnum m-ta stakið.

- Látum nú $m = \lfloor n/2 \rfloor$.
- Þá gildir eitt af þrennu, fyrir tiltekna lausn:
  - Báðir endapunktar bestu lausnarinnar liggja á $[1, m-1]$.
  - Báðir endapunktar bestu lausnarinnar liggja á $[m, n]$.
  - Annar endapunktur bestu lausnarinnar liggur á $[1, m-1]$ og hinn liggur á $[m, n]$.
- Fyrri tvö tilfellin má leysa með einfaldri endurkvæmni.
- Fyrir síðasta tilfellið nýtum við gegnvirknina af fyrri glærunni.
- Við viljum finna bestu lausnina sem liggur í gegnum m-ta stakið.
- Gegnvirknin segir þó að okkur nægir að finna fyrst bestu lausnina sem endar í m-ta stakinu, finna svo bestu lausnina sem byrjar í m-ta stakinu og sameina svo lausnirnar.

```c
#include <stdlib.h>
#include <stdio.h>
typedef long long ll;
ll max(ll a, ll b) { if (a < b) return b; return a; }

ll foo(ll* a, ll l, ll r, ll k)
{
    if (r - l < 5)
    {
        ll i, j, mx = 0;
        for (i = l; i < r; i++) for (j = i + 1; j < r; j++)
            mx = max(mx, 100*(a[i] - a[j]) - k*(j - i));
        return mx;
    }
    ll m = (l + r)/2, i, j;
    ll v1 = foo(a, l, m, k), v2 = foo(a, m, r, k), mx1 = 0, mx2 = 0;
    for (i = l; i < m; i++) mx1 = max(mx1, 100*(a[i] - a[m]) - k*(m - i));
    for (i = m; i < r; i++) mx2 = max(mx2, 100*(a[m] - a[i]) - k*(i - m));
    return max(max(v1, v2), mx1 + mx2);
}

int main()
{
    ll i, j;
    int x, n, k;
    scanf("%d%d", &n, &k);
    ll a[n];
    for (i = 0; i < n; i++)
    {
        scanf("%d", &x);
        a[i] = x;
    }
    printf("%lld\n", max(0, foo(a, 0, n, k) - k));
    return 0;
}
```

Rakningarvensl

- Talnarunan $a_1, a_2, \ldots$ kallast *k-ta stigs rakningarvensl* ef til er fall $f : \mathbb{R}^k \to \mathbb{R}$ þannig að

$$a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-k})$$

fyrir ölll $n > k$.

Rakningarvensl

- Talnarunan $a_1, a_2, \ldots$ kallast *k-ta stigs rakningarvensl* ef til er fall $f : \mathbb{R}^k \to \mathbb{R}$ þannig að

$$a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-k})$$

  fyrir ölll $n > k$.
- Frægasta dæmið um rakningarvensl er *Fibonacci runan*.

Rakningarvensl

- Talnarunan $a_1, a_2, \ldots$ kallast *k-ta stigs rakningarvensl* ef til er fall $f : \mathbb{R}^k \to \mathbb{R}$ þannig að

$$a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-k})$$

  fyrir ölll $n > k$.
- Frægasta dæmið um rakningarvensl er *Fibonacci runan*.
- Hún er   stigs rakningarvensl gefin með fallinu $f(x, y) = x + y$.

Rakningarvensl

- Talnarunan $a_1, a_2, \ldots$ kallast *k-ta stigs rakningarvensl* ef til er fall $f: \mathbb{R}^k \to \mathbb{R}$ þannig að
$$a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-k})$$
fyrir ölll $n > k$.
- Frægasta dæmið um rakningarvensl er *Fibonacci runan*.
- Hún er annars stigs rakningarvensl gefin með fallinu $f(x, y) = x + y$.

Rakningarvensl

- Talnarunan $a_1, a_2, \ldots$ kallast *k-ta stigs rakningarvensl* ef til er fall $f : \mathbb{R}^k \to \mathbb{R}$ þannig að

$$a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-k})$$

  fyrir ölll $n > k$.
- Frægasta dæmið um rakningarvensl er *Fibonacci runan*.
- Hún er annars stigs rakningarvensl gefin með fallinu $f(x, y) = x + y$.
- Reikna má upp úr þessum venslum endurkvæmt.

```c
#include <stdio.h>

int fib(int x)
{
    if (x < 3) return 1;
    return fib(x - 1) + fib(x - 2);
}

int main()
{
    int n;
    scanf("%d", &n);
    printf("%d\n", fib(n));
    return 0;
}
```

- Í hverju skrefi skiptist endurkvæmnin í tvennt svo þetta forrit hefur tímaflækju $\mathcal{O}(\quad)$.

- Í hverju skrefi skiptist endurkvæmnin í tvennt svo þetta forrit hefur tímaflækju $\mathcal{O}(2^n)$.

- Í hverju skrefi skiptist endurkvæmnin í tvennt svo þetta forrit hefur tímaflækju $\mathcal{O}(2^n)$.
- Við getum þó bætt þetta til muna með því að geyma niðurstöðuna úr hverju kalli.

- Í hverju skrefi skiptist endurkvæmnin í tvennt svo þetta forrit hefur tímaflækju $\mathcal{O}(2^n)$.
- Við getum þó bætt þetta til muna með því að geyma niðurstöðuna úr hverju kalli.
- Þá nægir að reikna hvert gildi einu sinni.

- Í hverju skrefi skiptist endurkvæmnin í tvennt svo þetta forrit hefur tímaflækju $\mathcal{O}(2^n)$.
- Við getum þó bætt þetta til muna með því að geyma niðurstöðuna úr hverju kalli.
- Þá nægir að reikna hvert gildi einu sinni.
- Þessi viðbót kallast *minnun* (e. *memoization*).

```c
#include <stdio.h>
#define MAXN 1000000

int d[MAXN]; // Hér geymum við skilagildi fib(...).
             // Ef d[i] == -1 þá eigum við eftir að reikna fib(i).
int fib(int x)
{
    if (d[x] != -1) return d[x];
    if (x < 2) return 1;
    return d[x] = fib(x - 1) + fib(x - 2);
}

int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 0; i < n; i++) d[i] = -1;
    printf("%d\n", fib(n - 1));
    return 0;
}
```

Ofansækin kvik bestun

- Nú reiknum við hvert gildi aðeins einu sinni.

Ofansækin kvik bestun

- Nú reiknum við hvert gildi aðeins einu sinni.
- Við þurfum að reikna n gildi og hvert gildi má reikna í $\mathcal{O}(\quad)$ tíma, svo í heildina er forritið $\mathcal{O}(\quad)$.

Ofansækin kvik bestun

- Nú reiknum við hvert gildi aðeins einu sinni.
- Við þurfum að reikna n gildi og hvert gildi má reikna í $\mathcal{O}(1)$ tíma, svo í heildina er forritið $\mathcal{O}(\quad)$.

Ofansækin kvik bestun

- Nú reiknum við hvert gildi aðeins einu sinni.
- Við þurfum að reikna n gildi og hvert gildi má reikna í $\mathcal{O}(1)$ tíma, svo í heildina er forritið $\mathcal{O}(n)$.

Ofansækin kvik bestun

- Nú reiknum við hvert gildi aðeins einu sinni.
- Við þurfum að reikna n gildi og hvert gildi má reikna í $\mathcal{O}(1)$ tíma, svo í heildina er forritið $\mathcal{O}(n)$.
- Án minnunar náum við með erfiðum að reikna fertugust Fibonacci töluna (því eframatið $\mathcal{O}(2^n)$ mætti bæta ögn) en með minnun náum við hæglega að reikna milljónustu Fibonacci töluna (hún mun þó ekki einu sinni passa í 64 bita).

Ofansækin kvik bestun

- Nú reiknum við hvert gildi aðeins einu sinni.
- Við þurfum að reikna n gildi og hvert gildi má reikna í $\mathcal{O}(1)$ tíma, svo í heildina er forritið $\mathcal{O}(n)$.
- Án minnunar náum við með erfiðum að reikna fertugust Fibonacci töluna (því eframatið $\mathcal{O}(2^n)$ mætti bæta ögn) en með minnun náum við hæglega að reikna milljónustu Fibonacci töluna (hún mun þó ekki einu sinni passa í 64 bita).
- Ef lausnin okkar er endurkvæm með minnun kallast hún *ofansækin kvik bestun* (e. *top down dynamic programming*).

Neðansækin kvik bestun

- Það er þó lítið mál að breyta endurkvæmnu lausninni okkar í ítraða lausn.

Neðansækin kvik bestun

- Það er þó lítið mál að breyta endurkvæmnu lausninni okkar í ítraða lausn.
- Eina sem við þurfum að passa er að reikna gildin í vaxandi röð.

Neðansækin kvik bestun

- Það er þó lítið mál að breyta endurkvæmnu lausninni okkar í ítraða lausn.
- Eina sem við þurfum að passa er að reikna gildin í vaxandi röð.

```c
#include <stdio.h>

int main()
{
    int n, i;
    scanf("%d", &n);
    int a[n];
    a[0] = a[1] = 1;
    for (i = 2; i < n; i++) a[i] = a[i - 1] + a[i - 2];
    printf("%d\n", a[n - 1]);
    return 0;
}
```

Neðansækin kvik bestun

- Það er þó lítið mál að breyta endurkvæmnu lausninni okkar í ítraða lausn.
- Eina sem við þurfum að passa er að reikna gildin í vaxandi röð.

```c
#include <stdio.h>

int main()
{
    int n, i;
    scanf("%d", &n);
    int a[n];
    a[0] = a[1] = 1;
    for (i = 2; i < n; i++) a[i] = a[i - 1] + a[i - 2];
    printf("%d\n", a[n - 1]);
    return 0;
}
```

- Þegar ofansækin kvik bestunar lausn er útfærð með ítrun köllum við það *neðansækin kvik bestun* (e. *bottom up dymanic programming*).

- Í neðansækinni kvikri bestun byrjum við með grunntilfellin og smíðum flóknari lausnirnar út frá þeim.

- Í neðansækinni kvikri bestun byrjum við með grunntilfellin og smíðum flóknari lausnirnar út frá þeim.
- Í ofansækinni kvikri bestun brjótum við fyrst flóknu dæmin niður í smærri dæmi sem við vitum svarið við og reiknum svo út úr því.

- Í neðansækinni kvikri bestun byrjum við með grunntilfellin og smíðum flóknari lausnirnar út frá þeim.
- Í ofansækinni kvikri bestun brjótum við fyrst flóknu dæmin niður í smærri dæmi sem við vitum svarið við og reiknum svo út úr því.
- Ef endurkvæmnafallið okkar er háð k breytum þá segjum við að lausnin okkar sé *k víð kvik bestun*.

- Í neðansækinni kvikri bestun byrjum við með grunntilfellin og smíðum flóknari lausnirnar út frá þeim.
- Í ofansækinni kvikri bestun brjótum við fyrst flóknu dæmin niður í smærri dæmi sem við vitum svarið við og reiknum svo út úr því.
- Ef endurkvæmnafallið okkar er háð k breytum þá segjum við að lausnin okkar sé *k víð kvik bestun*.
- Ofansækin kvik bestun hentar þegar við erum að vinna með fleiri en eina vídd.

- Í neðansækinni kvikri bestun byrjum við með grunntilfellin og smíðum flóknari lausnirnar út frá þeim.
- Í ofansækinni kvikri bestun brjótum við fyrst flóknu dæmin niður í smærri dæmi sem við vitum svarið við og reiknum svo út úr því.
- Ef endurkvæmnafallið okkar er háð k breytum þá segjum við að lausnin okkar sé *k víð kvik bestun*.
- Ofansækin kvik bestun hentar þegar við erum að vinna með fleiri en eina vídd.
- Þá getur verið erfitt að ítra í gegnum stöðurnar í "réttri röð".

- Annar kostur ofansækinnar kvikrar bestunar er að lausnirnar geta verið nokkuð einsleitar.

- Annar kostur ofansækinnar kvikrar bestunar er að lausnirnar geta verið nokkuð einsleitar.

```c
#include <stdio.h>
#define MAXN 1000000

int d[MAXN];
int dp_lookup(int x)
{
    if (d[x] != -1) return d[x];
    if (/* Er þetta grunntilfelli? */)
    {
        /* Skila tilheyrandi grunnsvari */
    }
    /* Reikna d[x] */
    return d[x];
}

int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 0; i < MAXN; i++) d[i] = -1;
    printf("%d\n", dp_lookup(n));
    return 0;
}
```

Lengsta sameiginlega hlutruna

- Tökum annað dæmi.

Lengsta sameiginlega hlutruna

- Tökum annað dæmi.
- Látum $S = s_1 s_2 \ldots s_n$ og $T = t_1 t_2 \ldots t_m$ vera strengi af lengd n og m, þannig að $1 \leq n, m \leq 10^4$.

Lengsta sameiginlega hlutruna

- Tökum annað dæmi.
- Látum $S = s_1 s_2 \dots s_n$ og $T = t_1 t_2 \dots t_m$ vera strengi af lengd n og m, þannig að $1 \leq n, m \leq 10^4$.
- Hver er lengd lengsta strengs X þannig að hann sé hlutruna í bæði S og T?

Lengsta sameiginlega hlutruna

- Tökum annað dæmi.
- Látum $S = s_1 s_2 \ldots s_n$ og $T = t_1 t_2 \ldots t_m$ vera strengi af lengd n og m, þannig að $1 \leq n, m \leq 10^4$.
- Hver er lengd lengsta strengs X þannig að hann sé hlutruna í bæði S og T?
- Takið eftir að "12" og "13" eru hlutrunur í "123" en "21" er það ekki.

- Getum við sett upp dæmið með þægilegum rakningarvenslum?

- Getum við sett upp dæmið með þægilegum rakningarvenslum?
- Ef svo er þá getum við notað kvika bestun.

- Getum við sett upp dæmið með þægilegum rakningarvenslum?
- Ef svo er þá getum við notað kvika bestun.
- Það er yfirleitt þægilegast að hugsa um rakningarvenslin sem fall, frekar en runu.

- Getum við sett upp dæmið með þægilegum rakningarvenslum?
- Ef svo er þá getum við notað kvika bestun.
- Það er yfirleitt þægilegast að hugsa um rakningarvenslin sem fall, frekar en runu.
- Látum $f(i, j)$ tákna lengstu sameiginlegu hlutrunu strengjanna $s_1 s_2 ... s_i$ og $t_1 t_2 ... t_j$.

- Getum við sett upp dæmið með þægilegum rakningarvenslum?
- Ef svo er þá getum við notað kvika bestun.
- Það er yfirleitt þægilegast að hugsa um rakningarvenslin sem fall, frekar en runu.
- Látum $f(i, j)$ tákna lengstu sameiginlegu hlutrunu strengjanna $s_1 s_2 \ldots s_i$ og $t_1 t_2 \ldots t_j$.
- Okkur mun svo nægja að reikna $f(n, m)$.

- Við vitum að $f(0, i) = f(j, 0) = 0$.

- Við vitum að $f(0, i) = f(j, 0) = 0$.
- Þetta munu vera grunntilfellin okkar.

- Við vitum að $f(0, i) = f(j, 0) = 0$.
- Þetta munu vera grunntilfellin okkar.
- Almennt gildir að ef við erum að reikna $f(i, j)$ og $s_i = t_j$ þá getum við látið þann staf vera aftastan í sameiginlegu hlutrununni.

- Við vitum að $f(0, i) = f(j, 0) = 0$.
- Þetta munu vera grunntilfellin okkar.
- Almennt gildir að ef við erum að reikna $f(i, j)$ og $s_i = t_j$ þá getum við látið þann staf vera aftastan í sameiginlegu hlutrununni.
- Svo $f(i, j) = f(i-1, j-1) + 1$ ef $s_i = t_j$.

- Við vitum að $f(0, i) = f(j, 0) = 0$.
- Þetta munu vera grunntilfellin okkar.
- Almennt gildir að ef við erum að reikna $f(i, j)$ og $s_i = t_j$ þá getum við látið þann staf vera aftastan í sameiginlegu hlutrununni.
- Svo $f(i, j) = f(i - 1, j - 1) + 1$ ef $s_i = t_j$.
- Ef $s_i \neq t_j$ þá verður annað stakið (eða bæði stökin) að vera ekki í hlutrununni.

- Við vitum að $f(0, i) = f(j, 0) = 0$.
- Þetta munu vera grunntilfellin okkar.
- Almennt gildir að ef við erum að reikna $f(i, j)$ og $s_i = t_j$ þá getum við látið þann staf vera aftastan í sameiginlegu hlutrununni.
- Svo $f(i, j) = f(i-1, j-1) + 1$ ef $s_i = t_j$.
- Ef $s_i \neq t_j$ þá verður annað stakið (eða bæði stökin) að vera ekki í hlutrununni.
- Við veljum að sjálfsögðu að sleppa þeim sem gefur okkur betra svar, það er að segja $f(i, j) = \max(f(i-1, j), f(i, j-1))$.

- Við vitum að $f(0,i) = f(j,0) = 0$.
- Þetta munu vera grunntilfellin okkar.
- Almennt gildir að ef við erum að reikna $f(i,j)$ og $s_i = t_j$ þá getum við látið þann staf vera aftastan í sameiginlegu hlutrununni.
- Svo $f(i,j) = f(i-1, j-1) + 1$ ef $s_i = t_j$.
- Ef $s_i \neq t_j$ þá verður annað stakið (eða bæði stökin) að vera ekki í hlutrununni.
- Við veljum að sjálfsögðu að sleppa þeim sem gefur okkur betra svar, það er að segja $f(i,j) = \max(f(i-1,j), f(i,j-1))$.
- Við getum svo sett allt saman og fengið

$$f(i,j) = \begin{cases} 0, & \text{ef } i = 0 \text{ eða } j = 0 \\ f(i-1, j-1) + 1, & \text{annars, og ef } s_i = t_j \\ \max(f(i-1,j), f(i,j-1)), & \text{annars.} \end{cases}$$

```c
#include <stdio.h>
#include <string.h>
#define MAXN 10001
int max(int a, int b) { if (a < b) return b; return a; }

char s[10001], t[10001];
int d[MAXN][MAXN];
int dp_lookup(int x, int y)
{
    if (d[x][y] != -1) return d[x][y];
    if (x == 0 || y == 0) return 0;
    if (s[x - 1] == t[y - 1]) return d[x][y] = dp_lookup(x - 1, y - 1) + 1;
    return d[x][y] = max(dp_lookup(x - 1, y), dp_lookup(x, y - 1));
}

int main()
{
    int n, m, i, j;
    fgets(s, MAXN, stdin);
    fgets(t, MAXN, stdin);
    n = strlen(s) - 1;
    m = strlen(t) - 1;
    for (i = 0; i < n + 1; i++) for (j = 0; j < m + 1; j++) d[i][j] = -1;
    printf("%d\n", dp_lookup(n, m));
    return 0;
}
```

- Það er þessi virði að bera saman `dp_lookup(...)` fallið í forritinu og $f(i,j)$ af glærunni í framan.

$$f(i,j) = \begin{cases} 0, & \text{ef } i = 0 \text{ eða } j = 0 \\ f(i-1,j-1)+1, & \text{annars, og ef } s_i = t_j \\ \max(f(i-1,j), f(i,j-1)), & \text{annars.} \end{cases}$$

```c
int dp_lookup(int x, int y)
{
    if (d[x][y] != -1) return d[x][y];
    if (x == 0 || y == 0) return 0;
    if (s[x - 1] == t[y - 1]) return d[x][y] = dp_lookup(x - 1, y - 1) + 1;
    return d[x][y] = max(dp_lookup(x - 1, y), dp_lookup(x, y - 1));
}
```

- Forritið okkar þarf í versta falli að reikna öll möguleg gildi á $f(i, j)$, sem eru $(n+1) \cdot (m+1)$ talsins.

- Forritið okkar þarf í versta falli að reikna öll möguleg gildi á $f(i, j)$, sem eru $(n+1) \cdot (m+1)$ talsins.
- En hvert gildi má reikna í $\mathcal{O}(\quad)$ tíma.

- Forritið okkar þarf í versta falli að reikna öll möguleg gildi á $f(i, j)$, sem eru $(n+1) \cdot (m+1)$ talsins.
- En hvert gildi má reikna í $\mathcal{O}(1)$ tíma.

- Forritið okkar þarf í versta falli að reikna öll möguleg gildi á $f(i, j)$, sem eru $(n+1) \cdot (m+1)$ talsins.
- En hvert gildi má reikna í $\mathcal{O}(1)$ tíma.
- Svo forritið hefur tímaflækjuna $\mathcal{O}(\quad)$.

- Forritið okkar þarf í versta falli að reikna öll möguleg gildi á $f(i, j)$, sem eru $(n+1) \cdot (m+1)$ talsins.
- En hvert gildi má reikna í $\mathcal{O}(\,1\,)$ tíma.
- Svo forritið hefur tímaflækjuna $\mathcal{O}(n \cdot m)$.

Skiptimyndadæmið

- Skoðum aftur Skiptimyntadæmið úr síðust viku.

Skiptimyndadæmið

- Skoðum aftur Skiptimyntadæmið úr síðust viku.
- Þú ert með ótakmarkað magn af m mismunandi myntum.

Skiptimyndadæmið

- Skoðum aftur Skiptimyntadæmið úr síðust viku.
- Þú ert með ótakmarkað magn af m mismunandi myntum.
- Þær eru virði $x_1, x_2, ..., x_m$. Til þæginda gerum við ráð fyrir því að $x_1 = 1$.

Skiptimyndadæmið

- Skoðum aftur Skiptimyntadæmið úr síðust viku.
- Þú ert með ótakmarkað magn af m mismunandi myntum.
- Þær eru virði $x_1, x_2, ..., x_m$. Til þæginda gerum við ráð fyrir því að $x_1 = 1$.
- Hver er minnsti nauðsynlegi fjöldi af klinki ef þú vilt gefa n krónur til baka.

- Gerum ráð fyrir að við byrjum að gefa til baka x_j krónur.

- Gerum ráð fyrir að við byrjum að gefa til baka x_j krónur.
- Þá erum við búin að smækka dæmið niður í $n - x_j$.

- Gerum ráð fyrir að við byrjum að gefa til baka x_j krónur.
- Þá erum við búin að smækka dæmið niður í $n - x_j$.
- Við getum því skoðað öll mögulega gildi x_j og séð hvað er best.

- Gerum ráð fyrir að við byrjum að gefa til baka x_j krónur.
- Þá erum við búin að smækka dæmið niður í $n - x_j$.
- Við getum því skoðað öll mögulega gildi x_j og séð hvað er best.
- Við viljum því reikna gildin á fallinu

$$f(x) = \begin{cases} \infty, & \text{ef } x < 0 \\ 0, & \text{ef } x = 0 \\ \min_{j=1,2,\ldots,m} f(x - x_j) + 1, & \text{annars.} \end{cases}$$

```c
#include <stdio.h>
#define MAXN 10001
#define MAXM 10001
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int n, m, a[MAXM];
int d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF; // Þessi lína þarf að vera fremst!
    if (d[x] != -1) return d[x];
    if (x == 0) return 0;
    d[x] = INF;
    for (i = 0; i < m; i++) d[x] = min(d[x], dp_lookup(x - a[i]) + 1);
    return d[x];
}

int main()
{
    int i;
    scanf("%d%d", &n, &m);
    for (i = 0; i < n + 1; i++) d[i] = -1;
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    printf("%d\n", dp_lookup(n));
    return 0;
}
```

- Þetta dæmi má þó hæglega gera neðansækið.

```c
#include <stdio.h>
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int main()
{
    int i, j, n, m;
    scanf("%d%d", &n, &m);
    int d[n + 1], a[m];
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    for (i = 0; i < n + 1; i++) d[i] = INF;
    d[0] = 0;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n + 1 - a[i]; j++) if (d[j] < INF)
        {
            d[j + a[i]] = min(d[j + a[i]], d[j] + 1);
        }
    }

    printf("%d\n", d[n]);
    return 0;
}
```

- Breytum dæminu örlítið.

- Breytum dæminu örlítið.
- Núna höfum við takmarkað magn af hverju klinki.

- Breytum dæminu örlítið.
- Núna höfum við takmarkað magn af hverju klinki.
- Nánar tiltekið höfum við m klink hvert að virði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).

- Breytum dæminu örlítið.
- Núna höfum við takmarkað magn af hverju klinki.
- Nánar tiltekið höfum við m klink hvert að virði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver minnsti fjöldi að klinki sem þarf til að gefa til baka n krónur, ef það er á annað borð hægt.

- Breytum dæminu örlítið.
- Núna höfum við takmarkað magn af hverju klinki.
- Nánar tiltekið höfum við m klink hvert að virði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver minnsti fjöldi að klinki sem þarf til að gefa til baka n krónur, ef það er á annað borð hægt.
- Nú er óþarfi að gera ráð fyrir því að $x_1 = 1$.

- Breytum dæminu örlítið.
- Núna höfum við takmarkað magn af hverju klinki.
- Nánar tiltekið höfum við m klink hvert að virði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver minnsti fjöldi að klinki sem þarf til að gefa til baka n krónur, ef það er á annað borð hægt.
- Nú er óþarfi að gera ráð fyrir því að $x_1 = 1$.
- Hvernig mætti breyta neðansæknu lausninni til að höndla þetta?

- Breytum dæminu örlítið.
- Núna höfum við takmarkað magn af hverju klinki.
- Nánar tiltekið höfum við m klink hvert að virði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver minnsti fjöldi að klinki sem þarf til að gefa til baka n krónur, ef það er á annað borð hægt.
- Nú er óþarfi að gera ráð fyrir því að $x_1 = 1$.
- Hvernig mætti breyta neðansæknu lausninni til að höndla þetta?
- Skoðum aftur neðansæknu lausnina.

Hefðbundna skiptimyntadæmið

```c
#include <stdio.h>
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int main()
{
    int i, j, n, m;
    scanf("%d%d", &n, &m);
    int d[n + 1], a[m];
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    for (i = 0; i < n + 1; i++) d[i] = INF;
    d[0] = 0;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n + 1 - a[i]; j++) if (d[j] < INF)
        {
            d[j + a[i]] = min(d[j + a[i]], d[j] + 1);
        }
    }

    printf("%d\n", d[n]);
    return 0;
}
```

Nýja dæmið

```c
#include <stdio.h>
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int main()
{
    int i, j, n, m;
    scanf("%d%d", &n, &m);
    int d[n + 1], a[m];
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    for (i = 0; i < n + 1; i++) d[i] = INF;
    d[0] = 0;
    for (i = 0; i < m; i++)
    {
        for (j = n - a[i]; j >= 0; j--) if (d[j] < INF)
        {
            d[j + a[i]] = min(d[j + a[i]], d[j] + 1);
        }
    }

    printf("%d\n", d[n]);
    return 0;
}
```

- Skoðum báðar aðferðirnar á litlu sýnidæmi.

- Skoðum báðar aðferðirnar á litlu sýnidæmi.
- Skoðum fyrst með endurtekningum og síðan án endurtekningar.

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
         ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2, -1, -1, -1, -1, -1, -1, -1, -1]
         ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2, -1, -1, -1, -1, -1, -1, -1, -1]
             ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3, -1, -1, -1, -1, -1, -1, -1]
             ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,   1,   2,   3,  -1,  -1,  -1,  -1,  -1,  -1,  -1]
                    ^    |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4, -1, -1, -1, -1, -1, -1]
                 ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  3,  4, -1, -1, -1, -1, -1, -1]
                     ^    |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5, -1, -1, -1, -1, -1]
                     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,   1,   2,   3,   4,   5,  -1,  -1,  -1,  -1,  -1]
                                  ^    |
```

```
n = 10
a = [1, 3, 5]
     ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  3,  4,  5,  6, -1, -1, -1, -1]
                             ^    |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6, -1, -1, -1, -1]
                             ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6,  7, -1, -1, -1]
                             ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6,  7, -1, -1, -1]
                                 ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6,  7,  8, -1, -1]
                                     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6,  7,  8, -1, -1]
                                     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  -1]
                                                ^    |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6,  7,  8,  9, -1]
                                         ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]
                                         ^   |
```

```
n = 10
a = [1, 3, 5]
        ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]
     ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  4,  5,  6,  7,  8,  9, 10]
      ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  4,  5,  6,  7,  8,  9, 10]
         ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  5,  6,  7,  8,  9, 10]
         ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  5,  6,  7,  8,  9, 10]
                ^              |
```

```
n = 10
a = [1, 3, 5]
        ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  6,  7,  8,  9, 10]
               ^                 |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  6,  7,  8,  9, 10]
                 ^              |
```

```
n = 10
a = [1, 3, 5]
        ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  3,  2,  7,  8,  9, 10]
                 ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  2,  7,  8,  9, 10]
                          ^              |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  2,  3,  8,  9, 10]
                     ^              |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  2,  3,  8,  9, 10]
                              ^              |
```

```
n = 10
a = [1, 3, 5]
        ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  3,  2,  3,  4,  9, 10]
                         ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

       0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  2,  3,  4,  9, 10]
                                      ^                 |
```

```
n = 10
a = [1, 3, 5]
        ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  3,  2,  3,  4,  3, 10]
                             ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  2,  3,  4,  3, 10]
                                         ^            |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  3,  2,  3,  4,  3,  4]
                                         ^           |
```

```
n = 10
a = [1, 3, 5]
           ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,   1,   2,   1,   2,   3,   2,   3,   4,   3,   4]
     ^                        |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  1,  2,  3,  4,  3,  4]
     ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  1,  2,  3,  4,  3,  4]
         ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  1,  2,  3,  4,  3,  4]
         ^                   |
```

```
n = 10
a = [1, 3, 5]
         ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,   1,   2,   1,   2,   1,   2,   3,   4,   3,   4]
               ^                        |
```

```
n = 10
a = [1, 3, 5]
           ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  1,  2,  3,  4,  3,  4]
               ^                        |
```

```
n = 10
a = [1, 3, 5]
           ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  1,  2,  3,  4,  3,  4]
                   ^                        |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1,  2,  1,  2,  1,  2,  3,  2,  3,  4]
                 ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  1,  2,  3,  2,  3,  4]
                     ^                        |
```

```
n = 10
a = [1, 3, 5]
            ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  1,  2,  3,  2,  3,  4]
                         ^                        |
```

```
n = 10
a = [1, 3, 5]
           ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,   1,   2,   1,   2,   1,   2,   3,   2,   3,   4]
                              ^                         |
```

```
n = 10
a = [1, 3, 5]
           ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  1,  2,  3,  2,  3,  2]
                                ^                        |
```

```
n = 10
a = [1, 3, 5]

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1,  2,  1,  2,  1,  2,  3,  2,  3,  2]
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                                         ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                                     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                                     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                             ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                         ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                 ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
             ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
         ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
     ^   |
```

```
n = 10
a = [1, 3, 5]
     ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,   1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
     ^    |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                                          ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                                ^                |
```

```
n = 10
a = [1, 3, 5]
        ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                         ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                     ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
                  ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
             ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
         ^           |
```

```
n = 10
a = [1, 3, 5]
        ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,   1,  -1,  -1,   2,  -1,  -1,  -1,  -1,  -1,  -1]
          ^              |
```

```
n = 10
a = [1, 3, 5]
        ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1, -1,  2, -1, -1, -1, -1, -1, -1]
      ^              |
```

```
n = 10
a = [1, 3, 5]
        ^

     0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1,  1,  2, -1, -1, -1, -1, -1, -1]
     ^              |
```

```
n = 10
a = [1, 3, 5]
           ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1,  1,  2, -1, -1, -1, -1, -1, -1]
                              ^                         |
```

```
n = 10
a = [1, 3, 5]
           ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1,  1,  2, -1, -1, -1, -1, -1, -1]
                     ^                         |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1,  1,  2, -1, -1, -1, -1,  3, -1]
                     ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1,  1,  2, -1, -1, -1, -1,  3, -1]
                     ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1,  1,  2, -1, -1, -1,  2,  3, -1]
                 ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1,  1,  2, -1, -1, -1,  2,  3, -1]
              ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1,  1,  2, -1, -1, -1,  2,  3, -1]
         ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1,  1,  2, -1,  2, -1,  2,  3, -1]
         ^                      |
```

```
n = 10
a = [1, 3, 5]
           ^

     0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1,  1,  2, -1,  2, -1,  2,  3, -1]
     ^                   |
```

```
n = 10
a = [1, 3, 5]
           ^

      0   1   2   3   4   5   6   7   8   9  10
d = [0,  1, -1,  1,  2,  1,  2, -1,  2,  3, -1]
     ^                   |
```

```
n = 10
a = [1, 3, 5]

      0    1    2    3    4    5    6    7    8    9   10
d = [0,  1, -1,  1,  2,  1,  2, -1,  2,  3, -1]
```

- Hvernig myndum við þó leysa seinna dæmið með ofansækinni kvikri bestun?

- Hvernig myndum við þó leysa seinna dæmið með ofansækinni kvikri bestun?
- Við þurfum að hugsa það aðeins öðruvísi.

- Hvernig myndum við þó leysa seinna dæmið með ofansækinni kvikri bestun?
- Við þurfum að hugsa það aðeins öðruvísi.
- Nú hefum við um tvennt að velja fyrir hvern pening.

- Hvernig myndum við þó leysa seinna dæmið með ofansækinni kvikri bestun?
- Við þurfum að hugsa það aðeins öðruvísi.
- Nú hefum við um tvennt að velja fyrir hvern pening.
- Annaðhvort notum við hann, eða ekki.

- Hvernig myndum við þó leysa seinna dæmið með ofansækinni kvikri bestun?
- Við þurfum að hugsa það aðeins öðruvísi.
- Nú hefum við um tvennt að velja fyrir hvern pening.
- Annaðhvort notum við hann, eða ekki.
- Svo við látum $f(i, j)$ tákna minnsta fjölda af klinki sem þarf til að gefa til baka n krónur, ef við megum nota klink $x_j, x_{j+1}, \ldots, x_m$.

- Hvernig myndum við þó leysa seinna dæmið með ofansækinni kvikri bestun?
- Við þurfum að hugsa það aðeins öðruvísi.
- Nú hefum við um tvennt að velja fyrir hvern pening.
- Annaðhvort notum við hann, eða ekki.
- Svo við látum $f(i, j)$ tákna minnsta fjölda af klinki sem þarf til að gefa til baka n krónur, ef við megum nota klink $x_j, x_{j+1}, \ldots, x_m$.
- Þá fáum við að

$$
f(i,j) = \begin{cases} \infty, & \text{ef } i < 0 \\ \infty, & \text{ef } i \neq 0 \text{ og } j = m+1 \\ 0, & \text{ef } i = 0 \text{ og } j = m+1 \\ \min(f(i, j+1) & \\ +f(i - x_j, j+1) + 1), & \text{annars.} \end{cases}
$$

```c
#include <stdio.h>
#define MAXN 10001
#define MAXM 10001
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int n, m, a[MAXM];
int d[MAXN][MAXM];
int dp_lookup(int x, int y)
{
    if (x < 0) return INF;
    if (d[x][y] != -1) return d[x][y];
    if (y == m) return x == 0 ? 0 : INF;
    return d[x][y] = min(dp_lookup(x, y + 1), dp_lookup(x - a[y], y + 1) + 1);
}

int main()
{
    int i, j;
    scanf("%d%d", &n, &m);
    for (i = 0; i < n + 1; i++) for (j = 0; j < m + 1; j++) d[i][j] = -1;
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    printf("%d\n", dp_lookup(n, 0));
    for (i = 0; i < n + 1; i++)
        printf("%3d ", dp_lookup(i, 0) == INF ? -1 : dp_lookup(i, 0));
    printf("\n");
    return 0;
}
```

- Það er nokkuð létt að meta tímaflækjurnar á neðansæknu lausnunum.

- Það er nokkuð létt að meta tímaflækjurnar á neðansæknu lausnunum.
- Þær eru báðar tvöfaldar `for`-lykkjur, sú ytri af lengd m og innri af lengd $\mathcal{O}(n)$.

- Það er nokkuð létt að meta tímaflækjurnar á neðansækknu lausnunum.
- Þær eru báðar tvöfaldar `for`-lykkjur, sú ytri af lengd m og innri af lengd $\mathcal{O}(n)$.
- Svo tímaflækjurnar eru $\mathcal{O}(\quad)$.

- Það er nokkuð létt að meta tímaflækjurnar á neðansæknu lausnunum.
- Þær eru báðar tvöfaldar `for`-lykkjur, sú ytri af lengd m og innri af lengd $\mathcal{O}(n)$.
- Svo tímaflækjurnar eru $\mathcal{O}(n \cdot m)$.

- Í neðansækni lausninni á hefðbundna dæminu þarf að reikna, allt að, $n+1$ fallgildi.

```
int n, m, a[MAXM];
int d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF; // Þessi lína þarf að vera fremst!
    if (d[x] != -1) return d[x];
    if (x == 0) return 0;
    d[x] = INF;
    for (i = 0; i < m; i++) d[x] = min(d[x], dp_lookup(x - a[i]) + 1);
    return d[x];
}
```

- Í neðansækknu lausninni á hefðbundna dæminu þarf að reikna, allt að, $n+1$ fallgildi.
- Hvert gildi má reikna í $\mathcal{O}(\quad)$ tíma.

```
int n, m, a[MAXM];
int d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF; // Þessi lína þarf að vera fremst!
    if (d[x] != -1) return d[x];
    if (x == 0) return 0;
    d[x] = INF;
    for (i = 0; i < m; i++) d[x] = min(d[x], dp_lookup(x - a[i]) + 1);
    return d[x];
}
```

- Í neðansæknu lausninni á hefðbundna dæminu þarf að reikna, allt að, $n + 1$ fallgildi.
- Hvert gildi má reikna í $\mathcal{O}(\, m\,)$ tíma.

```
int n, m, a[MAXM];
int d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF; // Þessi lína þarf að vera fremst!
    if (d[x] != -1) return d[x];
    if (x == 0) return 0;
    d[x] = INF;
    for (i = 0; i < m; i++) d[x] = min(d[x], dp_lookup(x - a[i]) + 1);
    return d[x];
}
```

- Í neðansækknu lausninni á hefðbundna dæminu þarf að reikna, allt að, $n + 1$ fallgildi.
- Hvert gildi má reikna í $\mathcal{O}(\, m \,)$ tíma.
- Svo í heildina er hún $\mathcal{O}(\qquad)$.

```
int n, m, a[MAXM];
int d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF; // Þessi lína þarf að vera fremst!
    if (d[x] != -1) return d[x];
    if (x == 0) return 0;
    d[x] = INF;
    for (i = 0; i < m; i++) d[x] = min(d[x], dp_lookup(x - a[i]) + 1);
    return d[x];
}
```

- Í neðansækni lausninni á hefðbundna dæminu þarf að reikna, allt að, $n+1$ fallgildi.
- Hvert gildi má reikna í $\mathcal{O}(\ m\ )$ tíma.
- Svo í heildina er hún $\mathcal{O}(n \cdot m)$.

```
int n, m, a[MAXM];
int d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF; // Þessi lína þarf að vera fremst!
    if (d[x] != -1) return d[x];
    if (x == 0) return 0;
    d[x] = INF;
    for (i = 0; i < m; i++) d[x] = min(d[x], dp_lookup(x - a[i]) + 1);
    return d[x];
}
```

- Í neðansækni lausninni á hinu dæminu þarf að reikna, allt að, $(n+1) \cdot (m+1)$ fallgildi.

```
int n, m, a[MAXM];
int d[MAXN][MAXM];
int dp_lookup(int x, int y)
{
    if (x < 0) return INF;
    if (d[x][y] != -1) return d[x][y];
    if (y == m) return x == 0 ? 0 : INF;
    return d[x][y] = min(dp_lookup(x, y + 1), dp_lookup(x - a[y], y + 1) +
}

int main()
{
```

- Í neðansæknu lausninni á hinu dæminu þarf að reikna, allt að, $(n+1) \cdot (m+1)$ fallgildi.
- Hvert gildi má þó reikna í $\mathcal{O}(\quad)$ tíma.

```
int n, m, a[MAXM];
int d[MAXN][MAXM];
int dp_lookup(int x, int y)
{
    if (x < 0) return INF;
    if (d[x][y] != -1) return d[x][y];
    if (y == m) return x == 0 ? 0 : INF;
    return d[x][y] = min(dp_lookup(x, y + 1), dp_lookup(x - a[y], y + 1) +
}

int main()
{
```

- Í neðansæknu lausninni á hinu dæminu þarf að reikna, allt að, $(n+1)\cdot(m+1)$ fallgildi.
- Hvert gildi má þó reikna í $\mathcal{O}(1)$ tíma.

```
int n, m, a[MAXM];
int d[MAXN][MAXM];
int dp_lookup(int x, int y)
{
    if (x < 0) return INF;
    if (d[x][y] != -1) return d[x][y];
    if (y == m) return x == 0 ? 0 : INF;
    return d[x][y] = min(dp_lookup(x, y + 1), dp_lookup(x - a[y], y + 1) +
}

int main()
{
```

- Í neðansæknu lausninni á hinu dæminu þarf að reikna, allt að, $(n+1)\cdot(m+1)$ fallgildi.
- Hvert gildi má þó reikna í $\mathcal{O}(1)$ tíma.
- Svo í heildina er hún $\mathcal{O}(\quad)$.

```
int n, m, a[MAXM];
int d[MAXN][MAXM];
int dp_lookup(int x, int y)
{
    if (x < 0) return INF;
    if (d[x][y] != -1) return d[x][y];
    if (y == m) return x == 0 ? 0 : INF;
    return d[x][y] = min(dp_lookup(x, y + 1), dp_lookup(x - a[y], y + 1) +
}

int main()
{
```

- Í neðansæknu lausninni á hinu dæminu þarf að reikna, allt að, $(n+1)\cdot(m+1)$ fallgildi.
- Hvert gildi má þó reikna í $\mathcal{O}(1)$ tíma.
- Svo í heildina er hún $\mathcal{O}(n \cdot m)$.

```
int n, m, a[MAXM];
int d[MAXN][MAXM];
int dp_lookup(int x, int y)
{
    if (x < 0) return INF;
    if (d[x][y] != -1) return d[x][y];
    if (y == m) return x == 0 ? 0 : INF;
    return d[x][y] = min(dp_lookup(x, y + 1), dp_lookup(x - a[y], y + 1) +
}

int main()
{
```

- Hvað gerum við ef við viljum vita *hvaða* klink á að gefa til baka, ekki bara hversu mikið?

- Hvað gerum við ef við viljum vita *hvaða* klink á að gefa til baka, ekki bara hversu mikið?
- Það er yfirleitt farið aðra af tveimur leiðum.

- Hvað gerum við ef við viljum vita *hvaða* klink á að gefa til baka, ekki bara hversu mikið?
- Það er yfirleitt farið aðra af tveimur leiðum.
- Takið eftir að þegar við reiknum, til dæmis, `min(dp(x, y + 1), dp(x - a[y], y + 1) + 1)` þá erum við í raun að velja hvort er betra: `dp(x, y + 1)` eða `dp(x - a[y], y + 1) + 1`.

- Hvað gerum við ef við viljum vita *hvaða* klink á að gefa til baka, ekki bara hversu mikið?
- Það er yfirleitt farið aðra af tveimur leiðum.
- Takið eftir að þegar við reiknum, til dæmis, `min(dp(x, y + 1), dp(x - a[y], y + 1) + 1)` þá erum við í raun að velja hvort er betra: `dp(x, y + 1)` eða `dp(x - a[y], y + 1) + 1`.
- Sú fyrri felur í sér að geyma fyrir hvert inntak í `dp_lookup(...)` hver besta leiðin er.

- Hvað gerum við ef við viljum vita *hvaða* klink á að gefa til baka, ekki bara hversu mikið?
- Það er yfirleitt farið aðra af tveimur leiðum.
- Takið eftir að þegar við reiknum, til dæmis, `min(dp(x, y + 1), dp(x - a[y], y + 1) + 1)` þá erum við í raun að velja hvort er betra: `dp(x, y + 1)` eða `dp(x - a[y], y + 1) + 1`.
- Sú fyrri felur í sér að geyma fyrir hvert inntak í `dp_lookup(...)` hver besta leiðin er.
- Kvik bestun byggir á því að besta leiðin sé alltaf sú sama.

- Hvað gerum við ef við viljum vita *hvaða* klink á að gefa til baka, ekki bara hversu mikið?
- Það er yfirleitt farið aðra af tveimur leiðum.
- Takið eftir að þegar við reiknum, til dæmis, `min(dp(x, y + 1), dp(x - a[y], y + 1) + 1)` þá erum við í raun að velja hvort er betra: `dp(x, y + 1)` eða `dp(x - a[y], y + 1) + 1`.
- Sú fyrri felur í sér að geyma fyrir hvert inntak í `dp_lookup(...)` hver besta leiðin er.
- Kvik bestun byggir á því að besta leiðin sé alltaf sú sama.
- Síðan er eftir á hægt að þræða sig í gegn og finna klinkið sem þarf.

Finnur bara fjöldann

```c
#include <stdio.h>
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int main()
{
    int i, j, n, m, x;
    scanf("%d%d", &n, &m);
    int d[n + 1], a[m];
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    for (i = 0; i < n + 1; i++) d[i] = INF;
    d[0] = 0;
    for (i = 0; i < m; i++)
        for (j = 0; j < n + 1 - a[i]; j++)
            if (d[j] < INF && d[j + a[i]] > d[j] + 1)
    {
        d[j + a[i]] = d[j] + 1;

    }

    printf("%d\n", d[n]);



    return 0;
}
```

Finnur hvaða klink

```c
#include <stdio.h>
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int main()
{
    int i, j, n, m, x;
    scanf("%d%d", &n, &m);
    int d[n + 1], a[m], e[n + 1];
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    for (i = 0; i < n + 1; i++) d[i] = INF;
    d[0] = 0;
    for (i = 0; i < m; i++)
        for (j = 0; j < n + 1 - a[i]; j++)
            if (d[j] < INF && d[j + a[i]] > d[j] + 1)
    {
        d[j + a[i]] = d[j] + 1;
        e[j + a[i]] = a[i];
    }

    printf("%d\n", d[n]);
    x = n;
    while (x != 0)
    {
        printf("%d ", e[x]);
        x -= e[x];
    }
    printf("\n");
    return 0;
}
```

- Hin aðferðin hentar oft betur ef við erum með ofansækina kvika bestun.

- Hin aðferðin hentar oft betur ef við erum með ofansækina kvika bestun.
- Þá búum við til annað endurkvæmt fall sem notar `dp_lookup(...)` til að finna besta skrefið.

- Hin aðferðin hentar oft betur ef við erum með ofansækina kvika bestun.
- Þá búum við til annað endurkvæmt fall sem notar `dp_lookup(...)` til að finna besta skrefið.
- Þetta fall er auðvelt að smíða því það mun vera næstum eins og `dp_lookup(...)`.

- Hin aðferðin hentar oft betur ef við erum með ofansækina kvika bestun.
- Þá búum við til annað endurkvæmt fall sem notar `dp_lookup(...)` til að finna besta skrefið.
- Þetta fall er auðvelt að smíða því það mun vera næstum eins og `dp_lookup(...)`.
- Þegar það er búið að finna besta gildið prentar það hvert gildið er, og heldur svo áfram endurkvæmt.

```c
#include <stdio.h>
#define MAXN 10001
#define MAXM 10001
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }
int n, m, a[MAXM], d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF; // Þessi lína þarf að vera fremst!
    if (d[x] != -1) return d[x];
    if (x == 0) return 0;
    d[x] = INF;
    for (i = 0; i < m; i++) d[x] = min(d[x], dp_lookup(x - a[i]) + 1);
    return d[x];
}
int dp_traverse(int x)
{
    if (x < 0) return INF;
    if (x == 0) return 0;
    int i, mn = INF, mni;
    for (i = 0; i < m; i++) if (mn > dp_lookup(x - a[i]) + 1)
        mn = dp_lookup(x - a[i]) + 1, mni = i;
    printf("%d ", a[mni]), dp_traverse(x - a[mni]);
    return mn;
}
int main()
{
    int i;
    scanf("%d%d", &n, &m);
    for (i = 0; i < n + 1; i++) d[i] = -1;
    for (i = 0; i < m; i++) scanf("%d", &a[i]);
    printf("%d\n", dp_lookup(n));
    dp_traverse(n);
    printf("\n");
    return 0;
}
```

- Helsti kostur fyrri aðferðarinnar er að þar er besta skrefið ákvarðað í $\mathcal{O}(\quad)$ tíma.

- Helsti kostur fyrri aðferðarinnar er að þar er besta skrefið ákvarðað í $\mathcal{O}(1)$ tíma.

- Helsti kostur fyrri aðferðarinnar er að þar er besta skrefið ákvarðað í $\mathcal{O}(1)$ tíma.
- Í seinni aðferðinni tekur það jafnalangan tíma og `dp_lookup(...)` tekur að meta hverja stöðu.

- Helsti kostur fyrri aðferðarinnar er að þar er besta skrefið ákvarðað í $\mathcal{O}(1)$ tíma.
- Í seinni aðferðinni tekur það jafnalangan tíma og `dp_lookup(...)` tekur að meta hverja stöðu.
- Þetta má þó bæta með minnun.

- Helsti kostur fyrri aðferðarinnar er að þar er besta skrefið ákvarðað í $\mathcal{O}(1)$ tíma.
- Í seinni aðferðinni tekur það jafnalangan tíma og `dp_lookup(...)` tekur að meta hverja stöðu.
- Þetta má þó bæta með minnun.
- Þetta kemur bara til með að gera nógu góða lausn hæga ef það þarf að reikna fyrir mörg n.

- Helsti kostur fyrri aðferðarinnar er að þar er besta skrefið ákvarðað í $\mathcal{O}(1)$ tíma.
- Í seinni aðferðinni tekur það jafnalangan tíma og `dp_lookup(...)` tekur að meta hverja stöðu.
- Þetta má þó bæta með minnun.
- Þetta kemur bara til með að gera nógu góða lausn hæga ef það þarf að reikna fyrir mörg n.
- Skoðum nú hvernig við gætum nýtt þetta til að finna lengsta sameiginlega hlutstreng.

```c
#include <stdio.h>
#include <string.h>
#define MAXN 10001
int max(int a, int b) { if (a < b) return b; return a; }
char s[MAXN], t[MAXN];
int d[MAXN][MAXN];
int dp_lookup(int x, int y)
{
    if (d[x][y] != -1) return d[x][y];
    if (x == 0 || y == 0) return 0;
    if (s[x - 1] == t[y - 1]) return d[x][y] = dp_lookup(x - 1, y - 1) + 1;
    return d[x][y] = max(dp_lookup(x - 1, y), dp_lookup(x, y - 1));
}

void dp_traverse(int x, int y)
{
    if (x == 0 || y == 0) return;
    if (s[x - 1] == t[y - 1])
    {
        dp_traverse(x - 1, y - 1);
        printf("%c", s[x - 1]);
    }
    else if (dp_lookup(x - 1, y) > dp_lookup(x, y - 1)) dp_traverse(x - 1, y);
    else dp_traverse(x, y - 1);
}

int main()
{
    int n, m, i, j;
    fgets(s, MAXN, stdin), fgets(t, MAXN, stdin);
    n = strlen(s) - 1, m = strlen(t) - 1;
    for (i = 0; i < n + 1; i++) for (j = 0; j < m + 1; j++) d[i][j] = -1;
    dp_traverse(n, m);
    printf("\n");
    return 0;
}
```

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.
- Tökum vel þekkt dæmi.

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.
- Tökum vel þekkt dæmi.
- Gerum ráð fyrir að við séum með n stöður.

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.
- Tökum vel þekkt dæmi.
- Gerum ráð fyrir að við séum með n stöður.
- Gefið er tvívítt fylki $(d_{ij})_{1 \leq i,j \leq n}$, þar sem d_{ij} táknar tímann sem það tekur að fara úr i-tu stöðunni í j-tu stöðuna.

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.
- Tökum vel þekkt dæmi.
- Gerum ráð fyrir að við séum með n stöður.
- Gefið er tvívítt fylki $(d_{ij})_{1\leq i,j\leq n}$, þar sem d_{ij} táknar tímann sem það tekur að fara úr i-tu stöðunni í j-tu stöðuna.
- Við viljum nú ferðast í gegnum allar stöðurnar í einhverri röð þannig að við byrjum og endum í sömu stöðu, förum í hverja stöðu nákvæmlega einu sinni og tökum sem stystan tíma.

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.
- Tökum vel þekkt dæmi.
- Gerum ráð fyrir að við séum með n stöður.
- Gefið er tvívítt fylki $(d_{ij})_{1\leq i,j\leq n}$, þar sem d_{ij} táknar tímann sem það tekur að fara úr i-tu stöðunni í j-tu stöðuna.
- Við viljum nú ferðast í gegnum allar stöðurnar í einhverri röð þannig að við byrjum og endum í sömu stöðu, förum í hverja stöðu nákvæmlega einu sinni og tökum sem stystan tíma.
- Þetta er fræga *Farandsölumannadæmið* (e. *Travelling Salseman Problem*).

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.
- Tökum vel þekkt dæmi.
- Gerum ráð fyrir að við séum með n stöður.
- Gefið er tvívítt fylki $(d_{ij})_{1 \leq i,j \leq n}$, þar sem d_{ij} táknar tímann sem það tekur að fara úr i-tu stöðunni í j-tu stöðuna.
- Við viljum nú ferðast í gegnum allar stöðurnar í einhverri röð þannig að við byrjum og endum í sömu stöðu, förum í hverja stöðu nákvæmlega einu sinni og tökum sem stystan tíma.
- Þetta er fræga *Farandsölumannadæmið* (e. *Travelling Salseman Problem*).
- Sígilt er að leysa þetta dæmi endurkvæmt í $\mathcal{O}((n+1)!)$ tíma.

- Stundum getur verið erfitt að ákvarða hvernig stöðurúmið okkar á að líta út.
- Tökum vel þekkt dæmi.
- Gerum ráð fyrir að við séum með n stöður.
- Gefið er tvívítt fylki $(d_{ij})_{1 \leq i,j \leq n}$, þar sem d_{ij} táknar tímann sem það tekur að fara úr i-tu stöðunni í j-tu stöðuna.
- Við viljum nú ferðast í gegnum allar stöðurnar í einhverri röð þannig að við byrjum og endum í sömu stöðu, förum í hverja stöðu nákvæmlega einu sinni og tökum sem stystan tíma.
- Þetta er fræga *Farandsölumannadæmið* (e. *Travelling Salseman Problem*).
- Sígilt er að leysa þetta dæmi endurkvæmt í $\mathcal{O}((n+1)!)$ tíma.
- Við höfum nú tólin til að gera betur.

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.
- Við getum því gert ráð fyrir að við byrjum í fyrstu stöðunni.

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.
- Við getum því gert ráð fyrir að við byrjum í fyrstu stöðunni.
- Látum P tákna mengi alla staða, A vera eiginlegt hlutmengi þar í og s vera stak utan A.

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.
- Við getum því gert ráð fyrir að við byrjum í fyrstu stöðunni.
- Látum P tákna mengi alla staða, A vera eiginlegt hlutmengi þar í og s vera stak utan A.
- Við getum þá látið $f(s, A)$ vera stysta leiðin til að fara í allar stöður A nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.
- Við getum því gert ráð fyrir að við byrjum í fyrstu stöðunni.
- Látum P tákna mengi alla staða, A vera eiginlegt hlutmengi þar í og s vera stak utan A.
- Við getum þá látið $f(s, A)$ vera stysta leiðin til að fara í allar stöður A nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.
- Tökum eftir að $f(s, \emptyset) = d_{s1}$.

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.
- Við getum því gert ráð fyrir að við byrjum í fyrstu stöðunni.
- Látum P tákna mengi alla staða, A vera eiginlegt hlutmengi þar í og s vera stak utan A.
- Við getum þá látið $f(s, A)$ vera stysta leiðin til að fara í allar stöður A nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.
- Tökum eftir að $f(s, \emptyset) = d_{s1}$.
- Við getum nú sett fallið f fram endurkvæmt með

$$f(s, A) = \begin{cases} d_{s1}, & \text{ef } A = \emptyset \\ \min_{e \in A}(d_{se} + f(e, A \setminus e)), & \text{annars.} \end{cases}$$

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.
- Við getum því gert ráð fyrir að við byrjum í fyrstu stöðunni.
- Látum P tákna mengi alla staða, A vera eiginlegt hlutmengi þar í og s vera stak utan A.
- Við getum þá látið $f(s, A)$ vera stysta leiðin til að fara í allar stöður A nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.
- Tökum eftir að $f(s, \emptyset) = d_{s1}$.
- Við getum nú sett fallið f fram endurkvæmt með

$$f(s, A) = \begin{cases} d_{s1}, & \text{ef } A = \emptyset \\ \min_{e \in A}(d_{se} + f(e, A \setminus e)), & \text{annars.} \end{cases}$$

- Svarið við dæminu fæst svo með $f(\qquad\qquad)$.

- Tökum fyrst eftir að það skiptir ekki máli í hvaða stöðu við byrjum.
- Við getum því gert ráð fyrir að við byrjum í fyrstu stöðunni.
- Látum P tákna mengi alla staða, A vera eiginlegt hlutmengi þar í og s vera stak utan A.
- Við getum þá látið $f(s, A)$ vera stysta leiðin til að fara í allar stöður A nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.
- Tökum eftir að $f(s, \emptyset) = d_{s1}$.
- Við getum nú sett fallið f fram endurkvæmt með

$$f(s, A) = \begin{cases} d_{s1}, & \text{ef } A = \emptyset \\ \min_{e \in A}(d_{se} + f(e, A \setminus e)), & \text{annars.} \end{cases}$$

- Svarið við dæminu fæst svo með $f(1, P \setminus \{1\})$.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna falligildi á f, ef við erum með n stöður.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(\quad)$ tíma.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(\, n\,)$ tíma.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(n)$ tíma.
- Svo í heildina er forritið $\mathcal{O}(\qquad)$.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(\, n \,)$ tíma.
- Svo í heildina er forritið $\mathcal{O}(n^2 \cdot 2^n)$.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(n)$ tíma.
- Svo í heildina er forritið $\mathcal{O}(n^2 \cdot 2^n)$.
- Samkvæmt 10^8 reglunni náum við að leysa dæmi með $n \leq$.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(n)$ tíma.
- Svo í heildina er forritið $\mathcal{O}(n^2 \cdot 2^n)$.
- Samkvæmt 10^8 reglunni náum við að leysa dæmi með $n \leq 18$.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(\, n \,)$ tíma.
- Svo í heildina er forritið $\mathcal{O}(n^2 \cdot 2^n)$.
- Samkvæmt 10^8 reglunni náum við að leysa dæmi með $n \leq 18$.
- Í aftur á móti náum við bara $n \leq$ með augljósum endurkvæmu lausninni.

- Wikipedia kennir þessa aðferð við Held og Karp (1962) og segir að þetta sé með fyrstu hagnýtingum kvikrar bestunar.
- Í versta falli þurfum við að reikna $n \cdot 2^n$ falligildi á f, ef við erum með n stöður.
- Hvert fallgildi er reiknað í $\mathcal{O}(n)$ tíma.
- Svo í heildina er forritið $\mathcal{O}(n^2 \cdot 2^n)$.
- Samkvæmt 10^8 reglunni náum við að leysa dæmi með $n \leq 18$.
- I aftur á móti náum við bara $n \leq 10$ með augljósum endurkvæmu lausninni.

```c
#include <stdio.h>
#include <assert.h>
#define MAXN 18
#define INF (1 << 30)
int min(int a, int b) { if (a < b) return a; return b; }

int a[MAXN][MAXN], d[MAXN][1 << MAXN], n;
int dp_lookup(int x, int y)
{
    int i;
    if (d[x][y] != -1) return d[x][y];
    if (y == 0) return a[x][0];
    d[x][y] = INF;
    for (i = 0; i < n; i++) if ((y&(1 << i)) != 0)
        d[x][y] = min(d[x][y], dp_lookup(i, y - (1 << i)) + a[x][i]);
    return d[x][y];
}

int main()
{
    int i, j;
    scanf("%d", &n);
    for (i = 0; i < n; i++) for (j = 0; j < (1 << n); j++) d[i][j] = -1;
    for (i = 0; i < n; i++) for (j = 0; j < n; j++) scanf("%d", &a[i][j]);
    printf("%d\n", dp_lookup(0, (1 << n) - 2));
    return 0;
}
```