Kvik bestun

Bergur Snorrason

10. febrúar 2021

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 öll $n > k$.
- 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 $S = s_1 s_2 ... s_n$ og $T = t_1 t_2 ... t_m$ vera strengi af lengd n og m, þannig að $1 \leq n, m \leq 10^4$.

Lengsta sameiginlega hlutruna

- Tökum annað dæmi.
- Látum $S = s_1 s_2 ... s_n$ og $T = t_1 t_2 ... t_m$ 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 $s_1 s_2 \ldots s_i$ og $t_1 t_2 \ldots t_j$.

- 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.

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

```
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 $\mathcal{O}(\qquad)$.

- 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 $x_1, x_2, \ldots, x_m$. 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 $x_1, x_2, \ldots, x_m$. 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

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

```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 að andvirði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).

- Breytum dæminu örlítið.
- Núna höfum við takmarkað magn af hverju klinki.
- Nánar tiltekið höfum við m klink að andvirði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver er 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 að andvirði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver er 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 að andvirði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver er 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 að andvirði $x_1, x_2, \ldots, x_m$ (núna geta verið endurtekin gildi).
- Hver er 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 höfum 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 höfum 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 höfum við um tvennt að velja fyrir hvern pening.
- Annaðhvort notum við hann, eða ekki.
- Svo við látum $f(n, j)$ 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 höfum við um tvennt að velja fyrir hvern pening.
- Annaðhvort notum við hann, eða ekki.
- Svo við látum $f(n, j)$ 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 neðansæknu lausnunum.
- Þær eru báðar tvöfaldar `for`-lykkjur, sú ytri af lengd m og innri af lengd $\mathcal{O}(n)$.
- Svo tímaflækjurnar eru $\mathcal{O}(\qquad)$.

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

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

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

```
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 ofansækknu 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 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 ofansækknu 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 $\mathcal{O}(n \cdot m)$.

```c
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 ofansækknu lausninni á hinu dæminu þarf að reikna, allt að, $(n+1)\cdot(m+1)$ fallgildi.

```
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);
}
```

- Í ofansækknu 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 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);
}
```

- Í ofansækinu 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 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);
}
```

- Í ofansækn 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 $\mathcal{O}(\qquad)$.

```
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);
}
```

- Í ofansækninni 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 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);
}
```

- Þessi dæmi eru oft kölluð *hlutmengjasummudæmi* (e. *Subset Sum Problem*).

- Þessi dæmi eru oft kölluð *hlutmengjasummudæmi* (e. *Subset Sum Problem*).
- Í sinni einföldustu mynd eru þau: "Er hægt að finna hlutrunu í gefinni talnarunu þannig að hlutrunan summast upp í t?".

- Þessi dæmi eru oft kölluð *hlutmengjasummudæmi* (e. *Subset Sum Problem*).
- Í sinni einföldustu mynd eru þau: “Er hægt að finna hlutrunu í gefinni talnarunu þannig að hlutrunan summast upp í t?”.
- Eins og við sáum, er bæði hægt að skoða dæmið með og án endurtekninga, ásamt öðrum kröfum.

- Þessi dæmi eru oft kölluð *hlutmengjasummudæmi* (e. *Subset Sum Problem*).
- Í sinni einföldustu mynd eru þau: “Er hægt að finna hlutrunu í gefinni talnarunu þannig að hlutrunan summast upp í t?”.
- Eins og við sáum, er bæði hægt að skoða dæmið með og án endurtekninga, ásamt öðrum kröfum.
- Fræga *bakpokadæmið* (e. *Knapsack Problem*) er dæmi um hlutmengjasummu dæmi.

- 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`.
- Fyrri aðferðin 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`.
- Fyrri aðferðin 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`.
- Fyrri aðferðin 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 besta skrefið er ákvarðað í $\mathcal{O}(\quad)$ tíma.

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

- Helsti kostur fyrri aðferðarinnar er að besta skrefið er 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 besta skrefið er 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 besta skrefið er 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 besta skrefið er 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 (tvisvar í upphafsstöðuna) 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 (tvisvar í upphafsstöðuna) 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 (tvisvar í upphafsstöðuna) 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 (tvisvar í upphafsstöðuna) 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 $f(\qquad)$.

- 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 $n \cdot 2^{n-1}$ 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 $n \cdot 2^{n-1}$ 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 $n \cdot 2^{n-1}$ 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 $n \cdot 2^{n-1}$ 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 $n \cdot 2^{n-1}$ 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 $n \cdot 2^{n-1}$ 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 $n \cdot 2^{n-1}$ 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 $n \cdot 2^{n-1}$ 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 $n \leq 18$.
- Við náum 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 $n \cdot 2^{n-1}$ 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 $n \leq 18$.
- Við náum 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;
}
```