Kvik bestun

Bergur Snorrason

6. febrúar 2023

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

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

  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.

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.

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

- 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];
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 fertugustu 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 fertugustu 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
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
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 *neðansækna kvika 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 niður flóknu dæmin 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 niður flóknu dæmin 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 niður flóknu dæmin 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 niður flóknu dæmin 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.

```
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;
}
```

- Neðansækin kvik bestun hefur sýna kosti.

- Neðansækin kvik bestun hefur sýna kosti.
- Það er stundum hægt að bæta tímaflækjuna, til dæmis með sniðugri gagnagrind.

- Neðansækin kvik bestun hefur sýna kosti.
- Það er stundum hægt að bæta tímaflækjuna, til dæmis með sniðugri gagnagrind.
- Sumar þessara bætinga krefjast þess að útfærsla sé neðansækin.

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 $1 \leq n, m \leq 10^3$.

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 $1 \leq n, m \leq 10^3$.
- 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 $1 \leq n, m \leq 10^3$.
- 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

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

```
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));
}

int lcs(char *a, char *b)
{
    int i, j, n = strlen(a) - 1, m = strlen(b) - 1;
    strcpy(s, a), strcpy(t, b);
    for (i = 0; i < n + 1; i++) for (j = 0; j < m + 1; j++) d[i][j] = -1;
    return dp_lookup(n, m);
}
```

- Það er þessi virði að bera saman `dp_lookup(...)` fallið í forritinu og $f(i,j)$ af glærunni að 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 $\mathcal{O}(\quad)$.

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

Skiptimyntadæmið

- Skoðum aftur Skiptimyntadæmið.

Skiptimyntadæmið

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

Skiptimyntadæmið

- Skoðum aftur Skiptimyntadæmið.
- Þú ert með ótakmarkað magn af m mismunandi myntum.
- Þær eru virði $x_1, x_2, \ldots, x_m$.

Skiptimyntadæmið

- Skoðum aftur Skiptimyntadæmið.
- 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 $x_1 = 1$.

Skiptimyntadæmið

- Skoðum aftur Skiptimyntadæmið.
- 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 sem þú þarft 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
int n, m, a[MAXM];
int d[MAXN];
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF;
    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];
}
```

- Þetta dæmi má þó hæglega gera neðansækið.

```c
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 $x_1, x_2, ..., x_m$ (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
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

```
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

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

```c
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);
}
```

- Það er léttara að ákvarða tímaflækjurnar á neðansæknu lausnunum.

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

- Það er léttara að ákvarða 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 $\mathcal{O}(\quad)$.

- Það er léttara að ákvarða 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;
    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 ofansæknu 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;
    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 $\mathcal{O}(\, m\,)$ tíma.

```
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF;
    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;
    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 ofansæknu 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 dp_lookup(int x)
{
    int i;
    if (x < 0) return INF;
    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 ofansæknu 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 ofansæknu 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 ofansæknu 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 ofansækknu 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);
}
```

- 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?
- 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?
- 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`.
- Okkur nægir því 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?
- 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`.
- Okkur nægir því 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?
- 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`.
- Okkur nægir því 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 hægt að þræða sig í gegn eftir á og finna klinkið sem þarf.

Finnur bara fjöldann

```c
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
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;
}
```

- Í stað þess að geyma besta skrefið getum við skoðað öll skrefin og valið það besta.

```c
int dp_lookup(int x)
{
    int i;
    if (x < 0) return INF;
    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 n, int *r)
{
    int i = 0, mn, mni, j;
    while (n > 0)
    {
        for (mn = INF, j = 0; j < m; j++) if (mn > dp_lookup(n - a[j]) + 1)
            mn = dp_lookup(n - a[j]) + 1, mni = j;
        r[i++] = a[mni];
        n -= a[mni];
    }
    return i;
}
```

- 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 kemur bara til með að gera nógu góða lausn hæga ef það þarf að reikna fyrir mörg gildi.

- 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 kemur bara til með að gera nógu góða lausn hæga ef það þarf að reikna fyrir mörg gildi.
- Skoðum nú hvernig við getum nýtt þetta til að finna eina af lengstu sameiginlegu hlutrunum tveggja strengja.

```c
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));
}

int lcs(char *a, char *b, char *r)
{
    int i, j, k, n = strlen(a) - 1, m = strlen(b) - 1, x, y;
    strcpy(s, a), strcpy(t, b), memset(r, '\0', MAXN);
    for (i = 0; i < n + 1; i++) for (j = 0; j < m + 1; j++) d[i][j] = -1;
    for (k = dp_lookup(n, m), x = n, y = m; x > 0 && y > 0; )
    {
        if (s[x - 1] == t[y - 1]) r[--k] = s[x - 1], x--, y--;
        else (dp_lookup(x, y - 1) > dp_lookup(x - 1, y)) ? y-- : x--;
    }
    return dp_lookup(n, m);
}
```

- Seinni skiptimyntadæmið er náskylt *hlutmengjasummudæminu* (e. *Subset Sum Problem*).

- Seinni skiptimyntadæmið er náskylt *hlutmengjasummudæminu* (e. *Subset Sum Problem*).
- Dæmið er einfalt.

- Seinni skiptimyntadæmið er náskylt *hlutmengjasummudæminu* (e. *Subset Sum Problem*).
- Dæmið er einfalt.
- Gefnar eru n tölur $a_1, \ldots, a_n$ ásamt tölu c.

- Seinni skiptimyntadæmið er náskylt *hlutmengjasummudæminu* (e. *Subset Sum Problem*).
- Dæmið er einfalt.
- Gefnar eru n tölur $a_1, \ldots, a_n$ ásamt tölu c.
- Hvaða hlutruna af tölum gefur hæstu summuna án þess að fara yfir c.

```
                    c = 15
                    b = 0
a: 1  2  7  9

s: 0
```

```
                      c = 15
                      b = 1
a: 1   2   7   9
   ^
s: 1
```

```
                      c = 15
                      b = 2
a: 1   2   7   9
       ^
s: 2
```

```
                         c = 15
                         b = 7
a: 1   2   7   9
           ^
s: 7
```

```
                    c = 15
                    b = 9
a: 1   2   7   9
               ^
s: 9
```

```
                c = 15
                b = 9
a: 1  2  7  9
   ^  ^
s: 3
```

```
                    c = 15
                    b = 9
a: 1   2   7   9
   ^       ^
s: 8
```

```
                  c = 15
                  b = 10
a: 1   2   7   9
   ^           ^
s: 10
```

```
                        c = 15
                        b = 10
a: 1   2   7   9
       ^   ^
s: 9
```

```
                    c = 15
                    b = 11
a: 1   2   7   9
       ^       ^
s: 11
```

```
                    c = 15
                    b = 11
a: 1   2   7   9
           ^   ^
s: 16
```

```
                  c = 15
                  b = 11
a: 1   2   7   9
   ^   ^   ^
s: 10
```

```
                    c = 15
                    b = 12
a: 1   2   7   9
   ^   ^       ^
s: 12
```

```
                    c = 15
                    b = 12
a: 1   2   7   9
   ^       ^   ^
s: 17
```

```
                c = 15
                b = 12
a: 1  2  7  9
      ^  ^  ^
s: 18
```

```
                    c = 15
                    b = 12
a: 1   2   7   9
   ^   ^   ^   ^
s: 19
```

- Látum

$$f(i,j) = \begin{cases} 1, & \text{ef til er hlutruna af } a_1, \ldots, a_i \text{ sem hefur summu } j, \\ 0, & \text{annars.} \end{cases}$$

- Látum

$$f(i,j) = \begin{cases} 1, & \text{ef til er hlutruna af } a_1, \dots, a_i \text{ sem hefur summu } j, \\ 0, & \text{annars.} \end{cases}$$

- Við getum nú umritað

$$f(i,j) = \begin{cases} 1, & \text{ef } i = 0 \text{ og } j = 0, \\ 0, & \text{ef } i = 0 \text{ og } j \neq 0, \text{ eða } j < 0, \\ \min(1, f(i-1,j) & \\ \quad + f(i-1, j-a_i)), & \text{ef } i \neq 0. \end{cases}$$

- Látum

$$f(i,j) = \begin{cases} 1, & \text{ef til er hlutruna af } a_1, \ldots, a_i \text{ sem hefur summu } j, \\ 0, & \text{annars.} \end{cases}$$

- Við getum nú umritað

$$f(i,j) = \begin{cases} 1, & \text{ef } i = 0 \text{ og } j = 0, \\ 0, & \text{ef } i = 0 \text{ og } j \neq 0 \text{, eða } j < 0, \\ \min(1, f(i-1,j) \\ \quad + f(i-1, j-a_i)), & \text{ef } i \neq 0. \end{cases}$$

- Svarið er þá stærsta $\ell \leq c$ þannig að $f(n, \ell)$ er einn.

```c
int d[MAXN][MAXC], b[MAXN];
int dp_lookup(int x, int y)
{
    if (x < 0) return y == 0;
    if (y < 0) return 0;
    if (d[x][y] != -1) return d[x][y];
    return d[x][y] = dp_lookup(x - 1, y) || dp_lookup(x - 1, y - b[x]);
}

int subsetsum(int *a, int n, int c)
{
    int i, j;
    for (i = 0; i < n; i++) for (j = 0; j < c + 1; j++) d[i][j] = -1;
    for (i = 0; i < n; i++) b[i] = a[i];
    while (!dp_lookup(n - 1, c)) c--;
    return c;
}
```

- Við reiknum hvert gildi á $f(i,j)$ í $\mathcal{O}(\quad)$.

- Við reiknum hvert gildi á $f(i, j)$ í $\mathcal{O}(1)$.

- Við reiknum hvert gildi á $f(i, j)$ í $\mathcal{O}(1)$.
- Við þurfum í versta falli að reinka $n \cdot (c + 1)$ slíka gildi.

- Við reiknum hvert gildi á $f(i,j)$ í $\mathcal{O}(\,1\,)$.
- Við þurfum í versta falli að reinka $n \cdot (c+1)$ slíka gildi.
- Svo það tekur okkur $\mathcal{O}(\qquad)$ tíma að reikna $f(i,j)$.

- Við reiknum hvert gildi á $f(i,j)$ í $\mathcal{O}(1)$.
- Við þurfum í versta falli að reinka $n \cdot (c+1)$ slíka gildi.
- Svo það tekur okkur $\mathcal{O}(n \cdot c)$ tíma að reikna $f(i,j)$.

- Við reiknum hvert gildi á $f(i,j)$ í $\mathcal{O}(1)$.
- Við þurfum í versta falli að reinka $n \cdot (c+1)$ slíka gildi.
- Svo það tekur okkur $\mathcal{O}(n \cdot c)$ tíma að reikna $f(i,j)$.
- En sökum minnunar tekur það $\mathcal{O}(q + n \cdot c)$ tíma að reikna q sinnum fallgildi fallsins f.

- Við reiknum hvert gildi á $f(i,j)$ í $\mathcal{O}(1)$.
- Við þurfum í versta falli að reinka $n \cdot (c+1)$ slíka gildi.
- Svo það tekur okkur $\mathcal{O}(n \cdot c)$ tíma að reikna $f(i,j)$.
- En sökum minnunar tekur það $\mathcal{O}(q + n \cdot c)$ tíma að reikna q sinnum fallgildi fallsins f.
- Við þurfum að reikna það, í versta falli, c sinnum, svo forritið er $\mathcal{O}(\quad)$.

- Við reiknum hvert gildi á $f(i, j)$ í $\mathcal{O}(1)$.
- Við þurfum í versta falli að reinka $n \cdot (c + 1)$ slíka gildi.
- Svo það tekur okkur $\mathcal{O}(n \cdot c)$ tíma að reikna $f(i, j)$.
- En sökum minnunar tekur það $\mathcal{O}(q + n \cdot c)$ tíma að reikna q sinnum fallgildi fallsins f.
- Við þurfum að reikna það, í versta falli, c sinnum, svo forritið er $\mathcal{O}(n \cdot c)$.

- Ein algeng hagnýting á þessu er tvískipting talna.

- Ein algeng hagnýting á þessu er tvískipting talna.
- Látum $a_1, \ldots, a_n$ vera heiltölur.

- Ein algeng hagnýting á þessu er tvískipting talna.
- Látum $a_1, \dots, a_n$ vera heiltölur.
- Hvernig er best að skipta þeim í tvo hópa þannig að mismunur summa hvors hóps sé sem minnstur.

- Ein algeng hagnýting á þessu er tvískipting talna.
- Látum $a_1, \ldots, a_n$ vera heiltölur.
- Hvernig er best að skipta þeim í tvo hópa þannig að mismunur summa hvors hóps sé sem minnstur.
- Ef tölurnar eru $(10, 2, 10, 30, 15, 2, 30, 10)$ þá skiptum við í $(2, 2, 10, 10, 30)$ og $(10, 15, 30)$.

- Ein algeng hagnýting á þessu er tvískipting talna.
- Látum $a_1, \ldots, a_n$ vera heiltölur.
- Hvernig er best að skipta þeim í tvo hópa þannig að mismunur summa hvors hóps sé sem minnstur.
- Ef tölurnar eru $(10, 2, 10, 30, 15, 2, 30, 10)$ þá skiptum við í $(2, 2, 10, 10, 30)$ og $(10, 15, 30)$.
- Summurnar eru 54 og 55, og mismunur þeirra er 1.

- Hvernig getum við leyst þetta?

- Hvernig getum við leyst þetta?
- Látum T vera summu allra talnanna.

- Hvernig getum við leyst þetta?
- Látum T vera summu allra talnanna.
- Við getum nú notað `subsetsum(...)` með $c = \lfloor T/2 \rfloor$.

- Hvernig getum við leyst þetta?
- Látum T vera summu allra talnanna.
- Við getum nú notað `subsetsum(...)` með $c = \lfloor T/2 \rfloor$.
- Það gefur okkur annan hópinn og hinn hópurinn verður afgangurinn.

- Hvernig getum við leyst þetta?
- Látum T vera summu allra talnanna.
- Við getum nú notað `subsetsum(...)` með $c = \lfloor T/2 \rfloor$.
- Það gefur okkur annan hópinn og hinn hópurinn verður afgangurinn.
- Okkur vantar þó að finna hvaða tölur eiga að vera í hvorum hóp.

```c
int d[MAXN][MAXC], b[MAXN];
int dp_lookup(int x, int y)
{
    if (x < 0) return y == 0;
    if (y < 0) return 0;
    if (d[x][y] != -1) return d[x][y];
    return d[x][y] = dp_lookup(x - 1, y) || dp_lookup(x - 1, y - b[x]);
}

void partition(int *a, int *r, int n)
{
    int i, j, t = 0, c;
    for (i = 0; i < n; i++) t += a[i], r[i] = 0, b[i] = a[i];
    c = t/2;
    for (i = 0; i < n; i++) for (j = 0; j < c + 1; j++) d[i][j] = -1;
    while (!dp_lookup(n - 1, c)) c--;
    printf(">> %d\n", c);
    for (i = n - 1, j = c; i >= 0 && j > 0; i--)
        if (dp_lookup(i - 1, j - a[i])) r[i] = 1, j -= a[i];
}
```

- Látum T vera summu allra talnanna.

- Látum T vera summu allra talnanna.
- Þá er tímaflækjan $\mathcal{O}(\qquad)$.

- Látum T vera summu allra talnanna.
- Þá er tímaflækjan $\mathcal{O}(n \cdot T)$.

- Látum T vera summu allra talnanna.
- Þá er tímaflækjan $\mathcal{O}(n \cdot T)$.
- Það er til skemmtileg leið til að fá tímaflækjuna $\mathcal{O}(n \cdot w)$, þar sem w er stærsta talan í inntakinu.

- Látum T vera summu allra talnanna.
- Þá er tímaflækjan $\mathcal{O}(n \cdot T)$.
- Það er til skemmtileg leið til að fá tímaflækjuna $\mathcal{O}(n \cdot w)$, þar sem w er stærsta talan í inntakinu.
- Takið eftir að tímaflækjan $\mathcal{O}(n \cdot T)$ er í raun sú sama og $\mathcal{O}(n^2 \cdot w)$.

- Látum T vera summu allra talnanna.
- Þá er tímaflækjan $\mathcal{O}(n \cdot T)$.
- Það er til skemmtileg leið til að fá tímaflækjuna $\mathcal{O}(n \cdot w)$, þar sem w er stærsta talan í inntakinu.
- Takið eftir að tímaflækjan $\mathcal{O}(n \cdot T)$ er í raun sú sama og $\mathcal{O}(n^2 \cdot w)$.
- Við skoðum hana kannski seinna.

- Hlutmengjadæmið er í raun sértilfelli af *bakpokadæminu* (e. *Knapsack Problem*).

- Hlutmengjadæmið er í raun sértilfelli af *bakpokadæminu* (e. *Knapsack Problem*).
- Gerum ráð fyrir að við séum með n hluti sem allir hafa einhverja vigt og verðgildi.

- Hlutmengjadæmið er í raun sértilfelli af *bakpokadæminu* (e. *Knapsack Problem*).
- Gerum ráð fyrir að við séum með n hluti sem allir hafa einhverja vigt og verðgildi.
- Við erum einnig með bakpoka sem þolir tiltekna samtals vigt.

- Hlutmengjadæmið er í raun sértilfelli af *bakpokadæminu* (e. *Knapsack Problem*).
- Gerum ráð fyrir að við séum með n hluti sem allir hafa einhverja vigt og verðgildi.
- Við erum einnig með bakpoka sem þolir tiltekna samtals vigt.
- Verkefnið snýst þá um að hámarka heildar verðgildi hluta sem hægt er að setja í bakpokann.

- Hlutmengjadæmið er í raun sértilfelli af *bakpokadæminu* (e. *Knapsack Problem*).
- Gerum ráð fyrir að við séum með n hluti sem allir hafa einhverja vigt og verðgildi.
- Við erum einnig með bakpoka sem þolir tiltekna samtals vigt.
- Verkefnið snýst þá um að hámarka heildar verðgildi hluta sem hægt er að setja í bakpokann.
- Ef sérhver hlutur hefur sömu vigt og verðgildi þá erum við komin með hlutmengjasummudæmið.

- Nánar, við höfum tölur $v_1, \ldots, v_n$, $w_1, \ldots, w_n$ og c.

- Nánar, við höfum tölur $v_1, \ldots, v_n$, $w_1, \ldots, w_n$ og c.
- Við viljum ákvarða tölur $b_1, \ldots, b_n \in \{0, 1\}$ þannig að

$$\sum_{i=1}^{n} b_i \cdot w_i \leq c \quad \text{og} \quad \sum_{i=1}^{n} b_i \cdot v_i \text{ sé hámarkað.}$$

- Nánar, við höfum tölur $v_1, \ldots, v_n$, $w_1, \ldots, w_n$ og c.
- Við viljum ákvarða tölur $b_1, \ldots, b_n \in \{0, 1\}$ þannig að

$$\sum_{i=1}^{n} b_i \cdot w_i \leq c \quad \text{og} \quad \sum_{i=1}^{n} b_i \cdot v_i \text{ sé hámarkað.}$$

- Látum $f(i, j)$ tákna hámarks verðgildi sem má fá úr fyrstu i hlutunum með bakpoka sem þolir þyngd j.

- Nánar, við höfum tölur $v_1, \ldots, v_n$, $w_1, \ldots, w_n$ og c.
- Við viljum ákvarða tölur $b_1, \ldots, b_n \in \{0, 1\}$ þannig að

$$\sum_{i=1}^{n} b_i \cdot w_i \leq c \quad \text{og} \quad \sum_{i=1}^{n} b_i \cdot v_i \text{ sé hámarkað.}$$

- Látum $f(i, j)$ tákna hámarks verðgildi sem má fá úr fyrstu i hlutunum með bakpoka sem þolir þyngd j.
- Við höfum þá

$$f(i,j) = \begin{cases} -\infty, & \text{ef } j < 0, \\ 0, & \text{annars, ef } i = 0, \\ \max(f(i-1,j) & \\ \quad +f(i-1,j-w_i)+v_i), & \text{annars.} \end{cases}$$

```c
int d[MAXN][MAXC], a[MAXN], b[MAXN];
int dp_lookup(int x, int y)
{
    if (y < 0) return -INF;
    if (x < 0) return 0;
    if (d[x][y] != -1) return d[x][y];
    return d[x][y] = max(dp_lookup(x - 1, y),
                         dp_lookup(x - 1, y - b[x]) + a[x]);
}

void knapsack(int *v, int *w, int *r, int n, int c)
{
    int i, j, s[MAXN], ss;
    for (i = 0; i < n; i++) for (j = 0; j <= c; j++) d[i][j] = -1;
    for (i = 0; i < n; i++) a[i] = v[i], b[i] = w[i], r[i] = 0;
    for (j = c, i = n - 1; i >= 0; i--)
        if (dp_lookup(i - 1, j) < dp_lookup(i - 1, j - w[i]) + v[i])
            j -= w[i], r[i] = 1;
}
```

- Við þurfum að reikna $n \cdot (c+1)$ fallgildi fallsins f.

- Við þurfum að reikna $n \cdot (c+1)$ fallgildi fallsins f.
- Hvert fallgildi er reiknað í $\mathcal{O}(\quad)$.

- Við þurfum að reikna $n \cdot (c + 1)$ fallgildi fallsins f.
- Hvert fallgildi er reiknað í $\mathcal{O}(1)$.

- Við þurfum að reikna $n \cdot (c+1)$ fallgildi fallsins f.
- Hvert fallgildi er reiknað í $\mathcal{O}(1)$.
- Í heildina er þetta því $\mathcal{O}(\qquad)$.

- Við þurfum að reikna $n \cdot (c + 1)$ fallgildi fallsins f.
- Hvert fallgildi er reiknað í $\mathcal{O}(1)$.
- Í heildina er þetta því $\mathcal{O}(n \cdot c)$.

- 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 stystu leiðina til að fara í allar stöður $P \setminus A$ 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 stystu leiðina til að fara í allar stöður $P \setminus A$ nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.
- Rakningarformúla fyrir f fæst því með

$$f(s, A) = \begin{cases} 0, & \text{ef } s = 1 \text{ og } A = P \\ \infty, & \text{ef } s = 1 \text{ og } A \neq \emptyset \\ \min_{e \in P \setminus A}(d_{se} + f(e, A \setminus e)), & \text{annars.} \end{cases}$$

- 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 stystu leiðina til að fara í allar stöður $P \setminus A$ nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.
- Rakningarformúla fyrir f fæst því með

$$f(s, A) = \begin{cases} 0, & \text{ef } s = 1 \text{ og } A = P \\ \infty, & \text{ef } s = 1 \text{ og } A \neq \emptyset \\ \min_{e \in P \setminus A}(d_{se} + f(e, A \setminus e)), & \text{annars.} \end{cases}$$

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

- 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 stystu leiðina til að fara í allar stöður $P \setminus A$ nákvæmlega einu sinni frá s og enda í fyrstu stöðunni.
- Rakningarformúla fyrir f fæst því með

$$f(s, A) = \begin{cases} 0, & \text{ef } s = 1 \text{ og } A = P \\ \infty, & \text{ef } s = 1 \text{ og } A \neq \emptyset \\ \min_{e \in P \setminus A}(d_{se} + f(e, A \setminus e)), & \text{annars.} \end{cases}$$

- Svarið við dæminu fæst svo með $f(1, \emptyset)$.

- 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 $\mathcal{O}(\quad)$.

- 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 $n \leq 18$.
- Við náum bara $n \leq$ með augljósu endurkvæmnu 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 $n \leq 18$.
- Við náum bara $n \leq 10$ með augljósu endurkvæmnu lausninni.

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

int tsp(int *a, int n)
{
    int i, j;
    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++) g[i][j] = a[i*MAXN + j];
    return dp_lookup(0, 0);
}
```

- Tökum nú dæmi sem er ekki jafn staðlað.

- Tökum nú dæmi sem er ekki jafn staðlað.
- Gerum ráð fyrir að við séum með n reiti fyrir framan okkur, hver merktur með tölu a_i.

- Tökum nú dæmi sem er ekki jafn staðlað.
- Gerum ráð fyrir að við séum með n reiti fyrir framan okkur, hver merktur með tölu a_i.
- Reitirnir liggja jafndreift á beinni línu og þú byrjar á reit númer eitt.

- Tökum nú dæmi sem er ekki jafn staðlað.
- Gerum ráð fyrir að við séum með n reiti fyrir framan okkur, hver merktur með tölu a_i.
- Reitirnir liggja jafndreift á beinni línu og þú byrjar á reit númer eitt.
- Þú vilt síðan ferðast á reit n með því að hoppa á milli reitanna.

- Tökum nú dæmi sem er ekki jafn staðlað.
- Gerum ráð fyrir að við séum með n reiti fyrir framan okkur, hver merktur með tölu a_i.
- Reitirnir liggja jafndreift á beinni línu og þú byrjar á reit númer eitt.
- Þú vilt síðan ferðast á reit n með því að hoppa á milli reitanna.
- Þú færð a_i stig ef þú lendir á i-ta reitnum og heildarstigafjöldinn er summa stiganna fyrir öll hoppin.

- Tökum nú dæmi sem er ekki jafn staðlað.
- Gerum ráð fyrir að við séum með n reiti fyrir framan okkur, hver merktur með tölu a_i.
- Reitirnir liggja jafndreift á beinni línu og þú byrjar á reit númer eitt.
- Þú vilt síðan ferðast á reit n með því að hoppa á milli reitanna.
- Þú færð a_i stig ef þú lendir á i-ta reitnum og heildarstigafjöldinn er summa stiganna fyrir öll hoppin.
- Hvert stökk má þó ekki vera lengra en stökkið á undan.

- Tökum nú dæmi sem er ekki jafn staðlað.
- Gerum ráð fyrir að við séum með n reiti fyrir framan okkur, hver merktur með tölu a_i.
- Reitirnir liggja jafndreift á beinni línu og þú byrjar á reit númer eitt.
- Þú vilt síðan ferðast á reit n með því að hoppa á milli reitanna.
- Þú færð a_i stig ef þú lendir á i-ta reitnum og heildarstigafjöldinn er summa stiganna fyrir öll hoppin.
- Hvert stökk má þó ekki vera lengra en stökkið á undan.
- Hver er mesti fjöldi stiga sem þú getur fengið.

- Tökum nú dæmi sem er ekki jafn staðlað.
- Gerum ráð fyrir að við séum með n reiti fyrir framan okkur, hver merktur með tölu a_i.
- Reitirnir liggja jafndreift á beinni línu og þú byrjar á reit númer eitt.
- Þú vilt síðan ferðast á reit n með því að hoppa á milli reitanna.
- Þú færð a_i stig ef þú lendir á i-ta reitnum og heildarstigafjöldinn er summa stiganna fyrir öll hoppin.
- Hvert stökk má þó ekki vera lengra en stökkið á undan.
- Hver er mesti fjöldi stiga sem þú getur fengið.
- Takið eftir að $1 \leq n \leq 3 \cdot 10^3$ og $-10^9 \leq a_i \leq 10^9$.

- Finnum rakningarformúlu sem lýsir svarinu.

- Finnum rakningarformúlu sem lýsir svarinu.
- Látum $f(i, j)$ vera mesta stigafjöldan sem má fá með því að stökkva frá i-ta reitnum með stökkum sem eru ekki lengri en j.

- Finnum rakningarformúlu sem lýsir svarinu.
- Látum $f(i,j)$ vera mesta stigafjöldan sem má fá með því að stökkva frá i-ta reitnum með stökkum sem eru ekki lengri en j.
- Við fáum þá

$$f(i,j) = \begin{cases} a_n, & \text{ef } i = n, \\ \max\limits_{1 \leq k \leq \min(j, n-i)} f(i+k, k) + a_i, & \text{annars.} \end{cases}$$

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

- Við þurfum að reikna $n \cdot n$ fallgildi og hvert fallgildi er reiknað í $\mathcal{O}(\quad)$.

- Við þurfum að reikna $n \cdot n$ fallgildi og hvert fallgildi er reiknað í $\mathcal{O}(n)$.

- Við þurfum að reikna $n \cdot n$ fallgildi og hvert fallgildi er reiknað í $\mathcal{O}(\ n\ )$.
- Svo heildartímaflækjan er $\mathcal{O}(\quad)$.

- Við þurfum að reikna $n \cdot n$ fallgildi og hvert fallgildi er reiknað í $\mathcal{O}(n)$.
- Svo heildartímaflækjan er $\mathcal{O}(n^3)$.

- Við þurfum að reikna $n \cdot n$ fallgildi og hvert fallgildi er reiknað í $\mathcal{O}(\, n \,)$.
- Svo heildartímaflækjan er $\mathcal{O}(n^3)$.
- En nú má n vera, allt að, $3 \cdot 10^3$ og $(3 \cdot 10^3)^3 = 27 \cdot 10^9$.

- Við þurfum að reikna $n \cdot n$ fallgildi og hvert fallgildi er reiknað í $\mathcal{O}(n)$.
- Svo heildartímaflækjan er $\mathcal{O}(n^3)$.
- En nú má n vera, allt að, $3 \cdot 10^3$ og $(3 \cdot 10^3)^3 = 27 \cdot 10^9$.
- Svo þessi lausn er of hæg.

- Við þurfum að reikna $n \cdot n$ fallgildi og hvert fallgildi er reiknað í $\mathcal{O}(\, n\,)$.
- Svo heildartímaflækjan er $\mathcal{O}(n^3)$.
- En nú má n vera, allt að, $3 \cdot 10^3$ og $(3 \cdot 10^3)^3 = 27 \cdot 10^9$.
- Svo þessi lausn er of hæg.

Submission

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8387732	15:12:47	Decelerating Jump	✖ Time Limit Exceeded	> 1.00 s	C

- Útfærum nú lausnina neðansækið í von um að geta bætt hana síðan.

- Útfærum nú lausnina neðansækið í von um að geta bætt hana síðan.
- Munum að

$$f(i,j) = \begin{cases} a_n, & \text{ef } i = n, \\ \max\limits_{1 \le k \le \min(j, n-i)} f(i+k, k) + a_i, & \text{annars.} \end{cases}$$

- Útfærum nú lausnina neðansækið í von um að geta bætt hana síðan.
- Munum að

$$f(i,j) = \begin{cases} a_n, & \text{ef } i = n, \\ \max\limits_{1 \le k \le \min(j, n-i)} f(i+k, k) + a_i, & \text{annars.} \end{cases}$$

- Nú er $f(i,j)$ bara háð styttri stökkum eða jafn löngum stökkum sem byrja aftar.

- Útfærum nú lausnina neðansækið í von um að geta bætt hana síðan.
- Munum að

$$f(i,j) = \begin{cases} a_n, & \text{ef } i = n, \\ \max\limits_{1 \leq k \leq \min(j, n-i)} f(i+k, k) + a_i, & \text{annars.} \end{cases}$$

- Nú er $f(i,j)$ bara háð styttri stökkum eða jafn löngum stökkum sem byrja aftar.
- Svo við getum reiknað út $f(i,j)$ í vaxandi stökk röð, og minnkandi upphafsstöðum.

- Útfærum nú lausnina neðansækið í von um að geta bætt hana síðan.
- Munum að

$$f(i,j) = \begin{cases} a_n, & \text{ef } i = n, \\ \max\limits_{1 \le k \le \min(j, n-i)} f(i+k, k) + a_i, & \text{annars.} \end{cases}$$

- Nú er $f(i,j)$ bara háð styttri stökkum eða jafn löngum stökkum sem byrja aftar.
- Svo við getum reiknað út $f(i,j)$ í vaxandi stökk röð, og minnkandi upphafsstöðum.
- Með öðrum orðum látum við j vaxa, og i minnka.

```c
int main()
{
    ll i, j, k, n;
    scanf("%lld", &n);
    ll d[n][n], a[n];
    for (i = 0; i < n; i++) scanf("%lld", &a[i]);
    for (i = 0; i < n; i++) for (j = 0; j < n; j++) d[i][j] = -INF;
    d[n - 1][1] = a[n - 1];
    for (i = n - 2; i >= 0; i--) d[i][1] = d[i + 1][1] + a[i];
    for (i = 0; i < n; i++) d[n - 1][i] = a[n - 1];
    for (j = 2; j < n; j++)
        for (i = n - 2; i >= 0; i--)
            for (k = 1; k < min(j + 1, n - i); k++)
                d[i][j] = max(d[i][j], d[i + k][k] + a[i]);
    printf("%lld\n", d[0][n - 1]);
    return 0;
}
```

- Nú er þetta forrit lítið annað en þreföld `for`-lykkja, hver af lengd $\mathcal{O}(n)$.

- Nú er þetta forrit lítið annað en þreföld `for`-lykkja, hver af lengd $\mathcal{O}(n)$.
- Svo tímaflækjan er $\mathcal{O}(\quad)$.

- Nú er þetta forrit lítið annað en þreföld `for`-lykkja, hver af lengd $\mathcal{O}(n)$.
- Svo tímaflækjan er $\mathcal{O}(n^3)$.

- Nú er þetta forrit lítið annað en þreföld `for`-lykkja, hver af lengd $\mathcal{O}(n)$.
- Svo tímaflækjan er $\mathcal{O}(n^3)$.
- Þetta ætti því einnig að vera of hægt.

- Nú er þetta forrit lítið annað en þreföld `for` -lykkja, hver af lengd $\mathcal{O}(n)$.
- Svo tímaflækjan er $\mathcal{O}(n^3)$.
- Þetta ætti því einnig að vera of hægt.

Submission

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8387890	15:45:07	Decelerating Jump	✖ Time Limit Exceeded	> 1.00 s	C

- Nú er þetta forrit lítið annað en þreföld `for`-lykkja, hver af lengd $\mathcal{O}(n)$.
- Svo tímaflækjan er $\mathcal{O}(n^3)$.
- Þetta ætti því einnig að vera of hægt.

Submission

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8387890	15:45:07	Decelerating Jump	✖ Time Limit Exceeded	> 1.00 s	C

- Tökum þó eftir einu.

- Þegar við reiknum fallgildið $f(i,j)$ þá tökum við stærsta gildið á skálínunni $f(i+1,1), f(i+2,2), \ldots, f(i+k,k)$.

- Þegar við reiknum fallgildið $f(i,j)$ þá tökum við stærsta gildið á skálínunni $f(i+1,1), f(i+2,2), \ldots, f(i+k,k)$.
- Eftir að hafa reiknað fallgildin fyrir tiltekna stökklengd hefur eitt stak bæst við hverja skálínu.

- Þegar við reiknum fallgildið $f(i,j)$ þá tökum við stærsta gildið á skálínunni $f(i+1,1), f(i+2,2), \ldots, f(i+k,k)$.
- Eftir að hafa reiknað fallgildin fyrir tiltekna stökklengd hefur eitt stak bæst við hverja skálínu.
- Það er lítið mál að halda utan um stærsta stakið á hverri skálínu með því að uppfæra eftir að nýtt fallgildi er reiknað.

- Þegar við reiknum fallgildið $f(i,j)$ þá tökum við stærsta gildið á skálínunni $f(i+1,1), f(i+2,2), \ldots, f(i+k,k)$.
- Eftir að hafa reiknað fallgildin fyrir tiltekna stökklengd hefur eitt stak bæst við hverja skálínu.
- Það er lítið mál að halda utan um stærsta stakið á hverri skálínu með því að uppfæra eftir að nýtt fallgildi er reiknað.
- Þá getum við líka reiknað hvert fallgildi í föstum tíma.

```c
int main()
{
    ll i, j, k, n;
    scanf("%lld", &n);
    ll d[n][n], e[n], a[n];
    for (i = 0; i < n; i++) scanf("%lld", &a[i]);
    for (i = 0; i < n; i++) for (j = 0; j < n; j++) d[i][j] = -INF;
    d[n - 1][1] = e[n - 1] = a[n - 1];
    for (i = n - 2; i >= 0; i--) e[i] = d[i][1] = d[i + 1][1] + a[i];
    for (j = 2; j < n; j++)
    {
        e[n - j] = max(e[n - j], d[n - 1][j] = a[n - 1]);
        for (i = n - 2; i >= 0; i--)
        {
            d[i][j] = e[i + 1] + a[i];
            if (i >= j - 1) e[i - j + 1] = max(e[i - j + 1], d[i][j]);
        }
    }
    printf("%lld\n", d[0][n - 1]);
    return 0;
}
```

- Nú er þetta forrit lítið annað en tvöföld `for`-lykkja, hvor af lengd $\mathcal{O}(n)$.

- Nú er þetta forrit lítið annað en tvöföld `for`-lykkja, hvor af lengd $\mathcal{O}(\,n\,)$.
- Svo tímaflækjan er $\mathcal{O}(\quad)$.

- Nú er þetta forrit lítið annað en tvöföld `for`-lykkja, hvor af lengd $\mathcal{O}(\, n \,)$.
- Svo tímaflækjan er $\mathcal{O}(n^2)$.

- Nú er þetta forrit lítið annað en tvöföld `for`-lykkja, hvor af lengd $\mathcal{O}(n)$.
- Svo tímaflækjan er $\mathcal{O}(n^2)$.
- Þetta ætti sleppa, því $(3 \cdot 10^3)^2 = 9 \cdot 10^6 < 10^8$.

- Nú er þetta forrit lítið annað en tvöföld `for`-lykkja, hvor af lengd $\mathcal{O}(n)$.
- Svo tímaflækjan er $\mathcal{O}(n^2)$.
- Þetta ætti sleppa, því $(3 \cdot 10^3)^2 = 9 \cdot 10^6 < 10^8$.

Submission

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8388221	16:50:36	Decelerating Jump	✔ Accepted	0.10 s	C