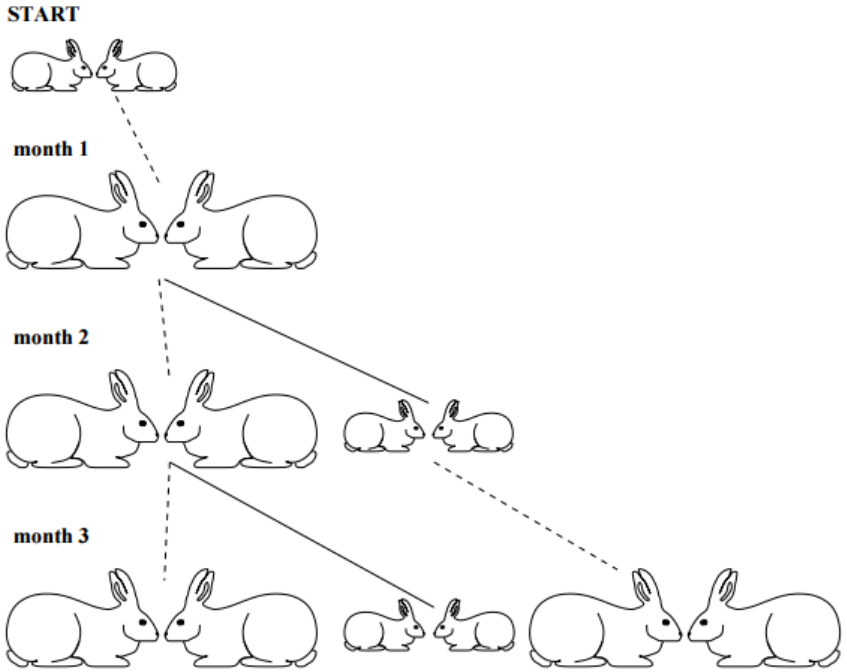
DP

How Many Pairs of Rabbits?



1,1,2,3,5,8,13,....

Fibonacci

- Fib_0 = Fib_1 = 1
- Fib_i = Fib_(i - 1) + Fib(i - 2)
- (1, 1, 2, 3, 5, 8, 13, 21, 34, 55)

```python
def fib(n):
    if n <= 1:
        return 1
    return fib(n - 1) + fib(n - 2)

n = 6
print(fib(n))
```

#stdin #stdout 0.04s 9580KB

stdin

```
Standard input is empty
```

stdout

```
13
```

```cpp
#include <iostream>

using namespace std;

int fib(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }
    return fib(n - 1) + fib(n - 2);
}
int main() {
    cout << fib(10);
}
```

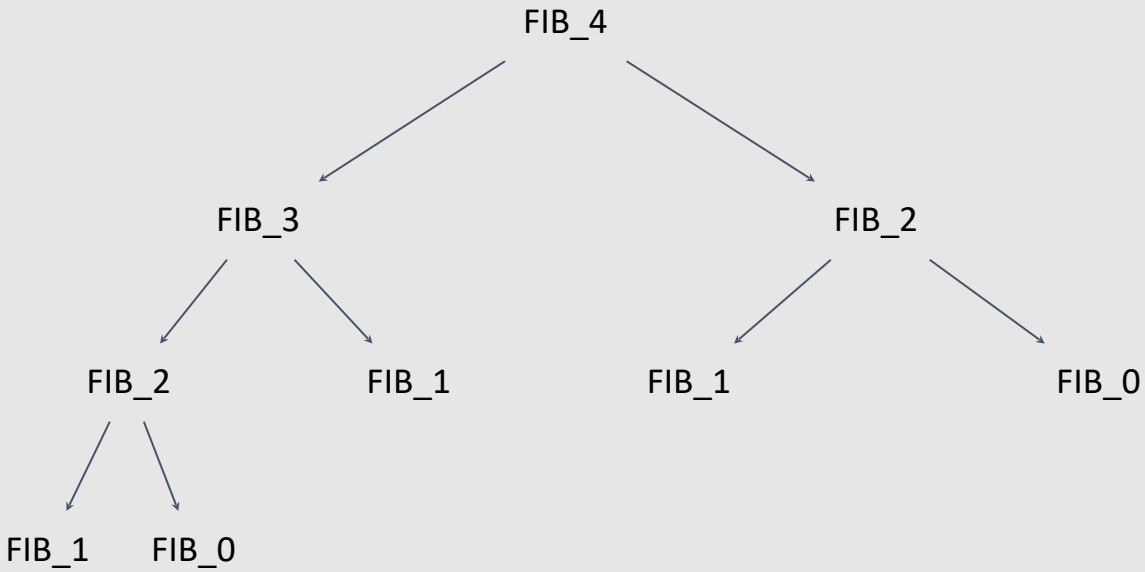
#stdin #stdout 0.01s 5352KB

stdin

```
Standard input is empty
```

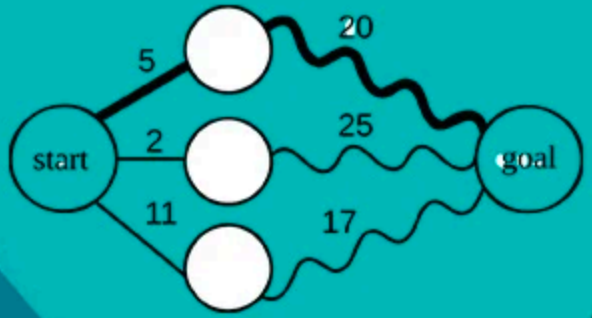
stdout

```
89
```


FIB_4
FIB_3
FIB_2
FIB_2
FIB_1
FIB_1
FIB_0
FIB_1
FIB_0

Asymptotics

- We calculate Fib_n in O(Fib_n).
- Let's prove this using recursion: We compute Fib_1 in O(1), which is in O(Fib_1).
- Fib_i is computed in O(Fib_(i - 1)) + O(Fib_(i - 2)), which is in O(Fib_i).
- Fib_n ≈ O(2^n)



Dynamic Programming

DP(Dynamic programming)

- What do we consider a state of dynamics (e.g. dp[i])?
- What values of dp do we know?
- What is the order in which dp is counted?
- Which formula is used to calculate dp?
- What's the answer to our problem?

Fibonacci

- dp[i] - i_th Fibonacci number?
- dp[0] = dp[1] = 1
- We need to count from the lower Fibonacci number to the higher one, i.e. from index 2 to index n
- dp[i] = dp[i - 1] + dp[i - 2], because i-th Fibonacci number = i-1 Fibonacci number + i - 2 number
- dp[n] - answer

```python
n = 50
dp = [0] * n
dp[0] = dp[1] = 1
for i in range(2, n):
    dp[i] = dp[i - 1] + dp[i - 2]
print(dp[n - 1])
```

#stdin #stdout 0.04s 9572KB

stdin

```
Standard input is empty
```

stdout

```
12586269025
```

```cpp
#include <iostream>

using namespace std;

int main() {
    int n = 50;
    long long dp[n];
    dp[0] = dp[1] = 1;
    for (int i = 2; i < n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    cout << dp[n - 1];
}
```

#stdin #stdout 0.01s 5448KB

stdin

```
Standard input is empty
```

stdout

```
12586269025
```

Prefix Sum

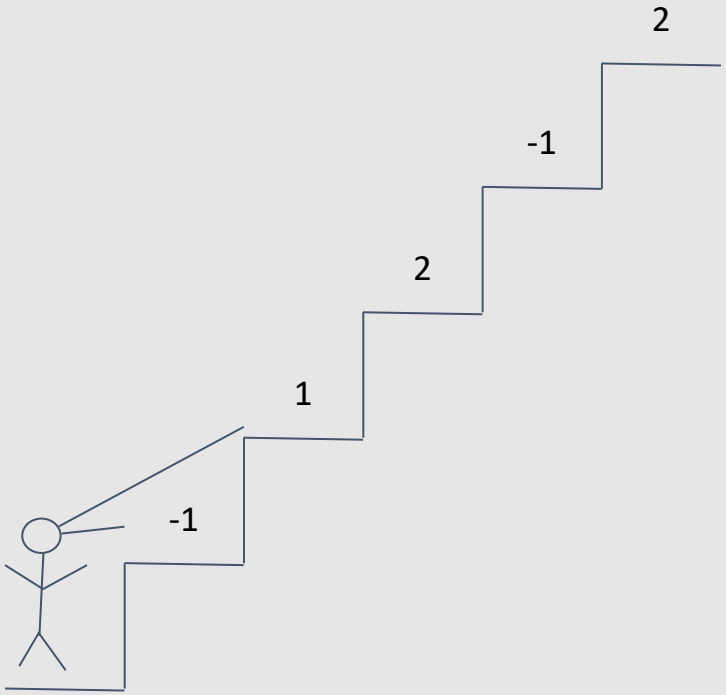
- dp[i] - sum(a[j]) for j = 0..(i - 1)
- dp[0] = 0
- We need to count from the lower index to the highest
- dp[i] = dp[i - 1] + a[i - 1]
- dp[n] - sum(a[i]) for i = 0..(n - 1)

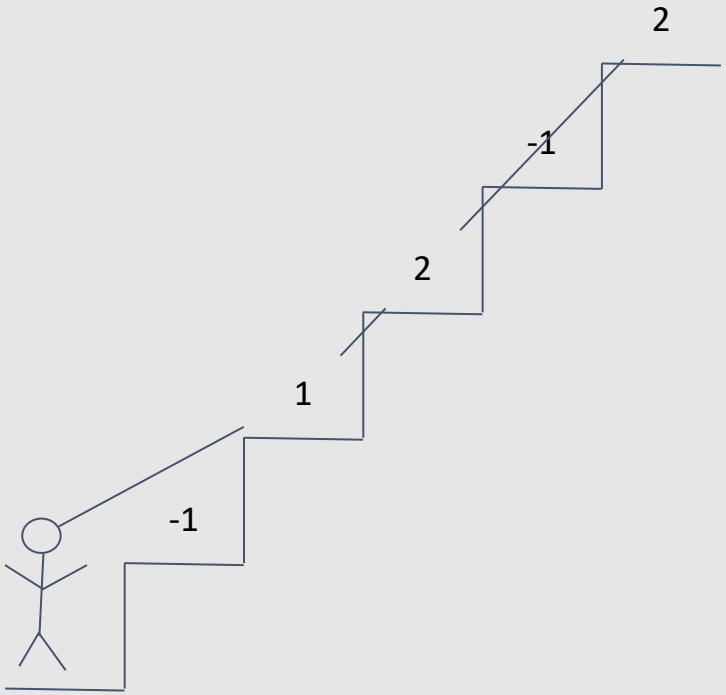
Scanline

- dp[i] - results for i first events
- dp[0] = 0
- We need to count from the lower index to the highest
- dp[i] = dp[i - 1] + event[i - 1].type
- dp[n] - result for the last event

Stairs

- We are standing initially on floor 0
- We can climb to the next floor or skip one floor
- Each floor has either a penalty or a reward written on it.
- We need to get to the final step with as much money as possible.


2
-1
2
1
-1


2
-1
2
1
-1

DP(Stairs)

- dp[i] - maximum result for the i-th floor
- dp[0] = 0
- We need to count from the lower floor to the higher one, i.e. from index 1 to index n
- dp[i] = max(dp[i - 1], dp[i - 2]) + a[i]
- dp[n] - answer

```cpp
#include <iostream>

using namespace std;

int main() {
    int n = 6;
    int dp[n];
    int a[n] = {0, -1, 1, 2, -1, 2};
    dp[0] = 0;
    dp[1] = a[1];
    for (int i = 2; i < n; i++) {
        dp[i] = max(dp[i - 1], dp[i - 2]) + a[i];
    }
    cout << dp[n - 1];
}
```

#stdin #stdout 0.01s 5436KB

stdin

```
Standard input is empty
```

stdout

```
5
```

```python
n = 6
dp = [0] * n
a = [0, -1, 1, 2, -1, 2]
dp[0] = 0
dp[1] = a[1]
for i in range(2, n):
    dp[i] = max(dp[i - 1], dp[i - 2]) + a[i]
print(dp[n - 1])
```

#stdin #stdout 0.04s 9488KB

stdin

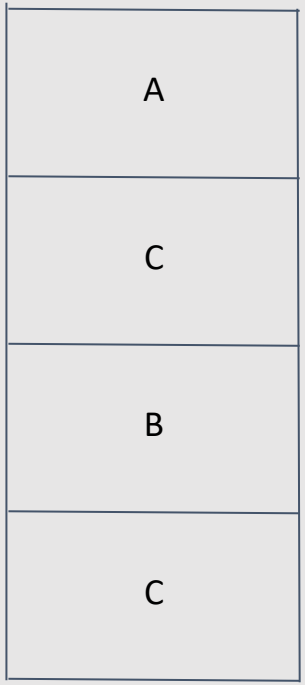
```
Standard input is empty
```

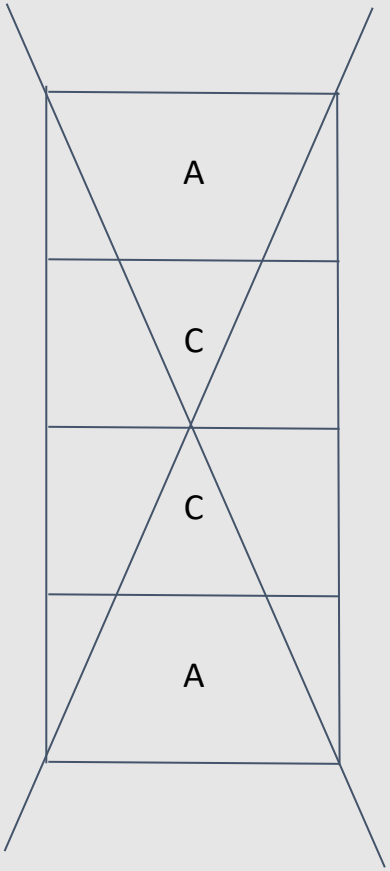
stdout

```
5
```

Barrels

- We have three types of barrels - A(safe), B(dangerous), C(very dangerous).
- We need to put n barrels in a column
- We can't put a C barrel on another C barrel
- How many ways are there to put the barrels in?


A
C
B
C


A
C
C
A

DP(Barrels) - bad

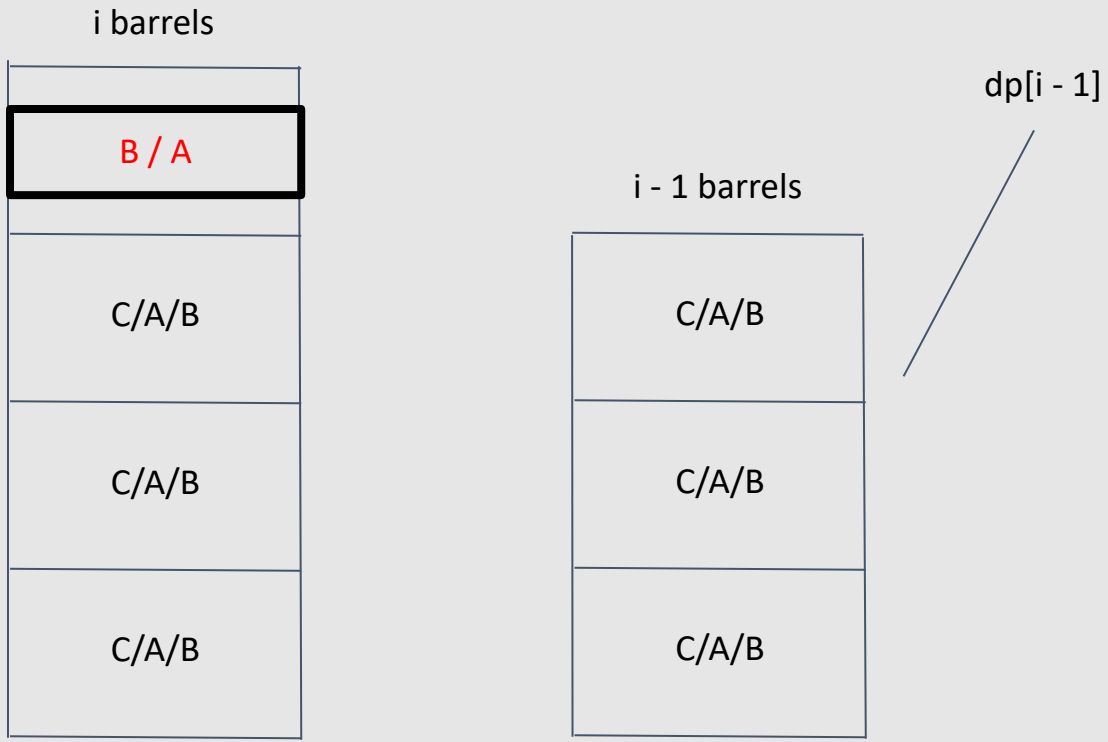
- dp[i][j] - Number of variants of i barrels arrangement where last one is j
- dp[1][0] = dp[1][1] = dp[1][2] = 1
- We need to count from 1 to n barrels, i.e. from index 1 to index n
- dp[i][0/1] = dp[i - 1][0] + dp[i - 1][1] + dp[i - 1][2]
- dp[i][2] = dp[i - 1][0] + dp[i - 1][1]
- dp[n][0] + dp[n][1] + dp[n][2] - answer

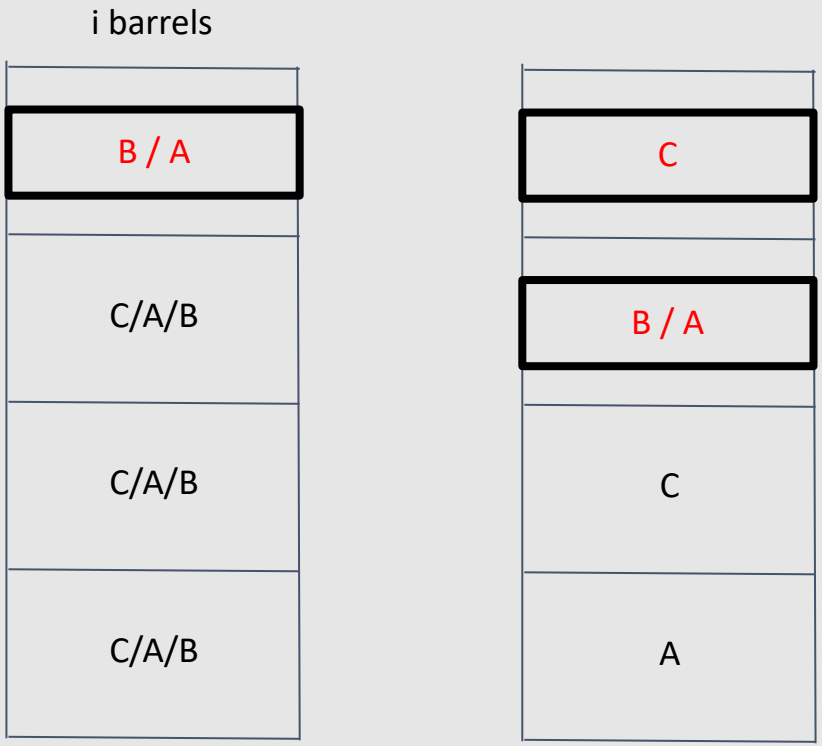
Example

- dp[1][0] = dp[1][1] = dp[1][2] = 1
- dp[2][0] = 3, dp[2][1] = 3, dp[2][2] = 2
- dp[3][0] = 8, dp[3][1] = 8, dp[3][2] = 6

DP(Barrels)

- dp[i] - Number of variants of i barrels arrangement
- dp[0] = 1, dp[1] = 3
- We need to count from 1 to n barrels, i.e. from index 1 to index n
- dp[i] = 2 * dp[i - 1] + 1 * 2 * dp[i - 2]
- dp[n] - answer


i barrels
B / A
C/A/B
C/A/B
C/A/B
i - 1 barrels
C/A/B
C/A/B
C/A/B
dp[i - 1]


i barrels
B / A
C/A/B
C/A/B
C/A/B
C
B / A
C
A

```python
n = 10
dp = [0] * n
dp[0] = 1
dp[1] = 3
for i in range(2, n):
    dp[i] = 2 * (dp[i - 1] + dp[i - 2])
    print(dp[i])
```

#stdin #stdout 0.03s 9520KB

stdin

```
Standard input is empty
```

stdout

```
8
22
60
164
448
1224
3344
9136
```

```cpp
#include <iostream>

using namespace std;

int main() {
    int n = 10;
    int dp[n];
    dp[0] = 1;
    dp[1] = 3;
    for (int i = 2; i < n; i++) {
        dp[i] = 2 * (dp[i - 1] + dp[i - 2]);
        cout << dp[i] << "\n";
    }
}
```

#stdin #stdout 0.01s 5516KB

stdin

```
Standard input is empty
```

stdout

```
8
22
60
164
448
1224
3344
9136
```

Turtle

- We have a field of size n by m.
- Each cell in the field has an amount (either a penalty or a bonus) written on it for that cell.
- You need to get from cell (0, 0) to cell (n, m), moving only to the right and down, getting the maximum amount of money

	1	-1	1
2	-1	2	3
-1	2	1	-1
3	4	-5	1

	1	-1	1
2	-1	2	3
-1	2	1	-1
3	4	-5	1

DP(Turtle)

- dp[i][j] - maximum amount that can be collected by reaching cell (i, j)
- dp[0][0] = 0
- We need to count from 0 to n in rows, and from 0 to m in columns
- dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + a[i][j]
- dp[n][m] - answer

```python
n = 4
m = 4
dp = []
for i in range(m):
    dp.append([0] * n)
a = [[0, 1, -1, 1], [2, -1, 2, 3], [-1, 2, 1, -1], [3, 4, -5, 1]]
dp[0][0] = 0;
for j in range(1, m):
    dp[0][j] = dp[0][j - 1] + a[0][j]

for i in range(1, n):
    dp[i][0] = dp[i - 1][0] + a[i][0]
    for j in range(1, m):
        dp[i][j] = max(dp[i][j - 1], dp[i - 1][j]) + a[i][j]
for i in range(0, n):
    for j in range(0, m):
        print(dp[i][j], end=' ')
    print("")
```

#stdin #stdout 0.04s 9440KB

stdin

```
Standard input is empty
```

stdout

```
0 1 0 1
2 1 3 6
1 3 4 5
4 8 3 6
```

```cpp
#include <iostream>

using namespace std;

const int n = 4, m = 4;

int main() {
    int dp[n][m];
    int a[n][m] = {{0, 1, -1, 1}, {2, -1, 2, 3}, {-1, 2, 1, -1}, {3, 4, -5, 1}};
    dp[0][0] = 0;
    for (int j = 1; j < m; j++) {
        dp[0][j] = dp[0][j - 1] + a[0][j];
    }
    for (int i = 1; i < n; i++) {
        dp[i][0] = dp[i - 1][0] + a[i][0];
        for (int j = 1; j < m; j++) {
            dp[i][j] = max(dp[i][j - 1], dp[i - 1][j]) + a[i][j];
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << dp[i][j] << " ";
        }
        cout << "\n";
    }
}
```

#stdin #stdout 0.01s 5464KB

comments ()

stdin

copy

```
Standard input is empty
```

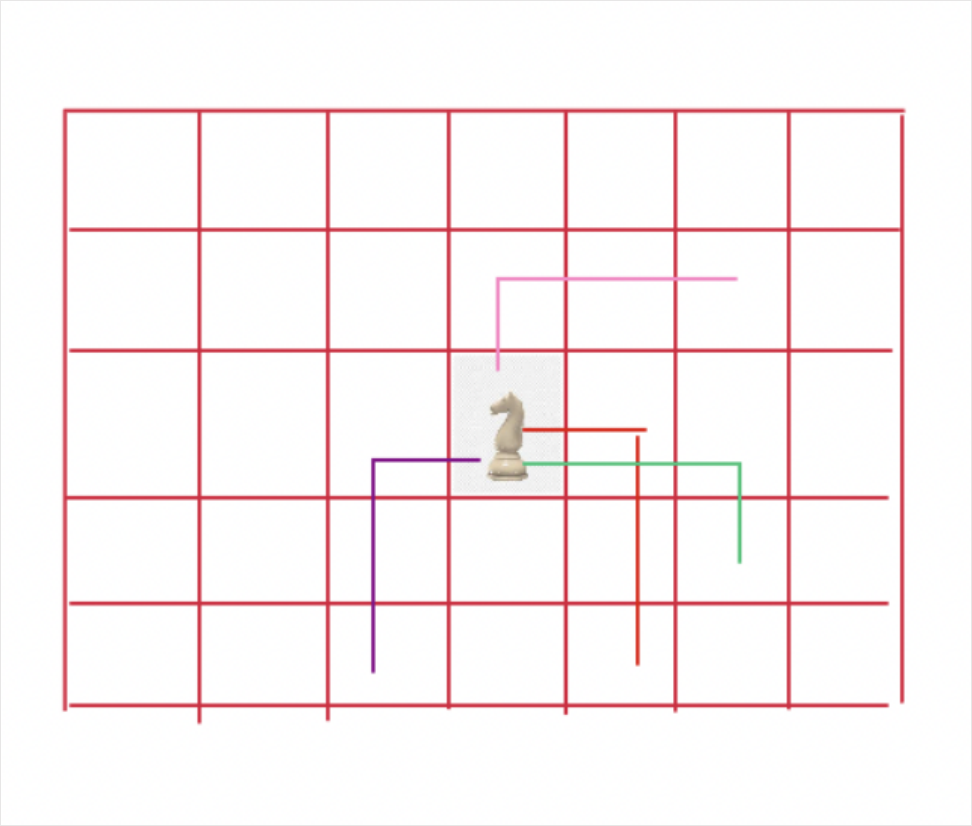
stdout

copy

```
0 1 0 1
2 1 3 6
1 3 4 5
4 8 3 6
```

Knight

- We have a field of size n by m.
- You need to get from cell (0, 0) to cell (n, m), moving only by moves of the chess knight, which will be in the picture below
- How many ways are there to do this?



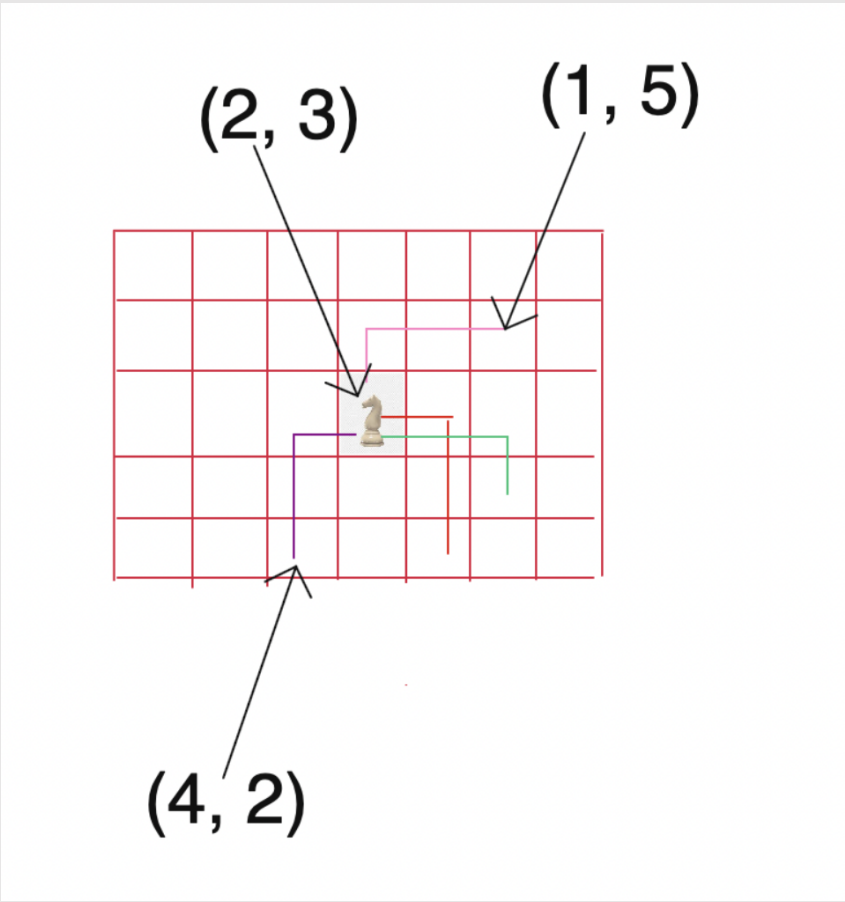
DP(Knight)

- dp[i][j] - number of ways to get to cell (i, j)
- dp[0][0] = 1
- Next slide
- dp[i][j] = dp[i - 2][j + 1] + dp[i + 1][j - 2] + dp[i - 2][j - 1] + dp[i - 1][j - 2]
- dp[n][m] - answer

DP(Knight) - Order

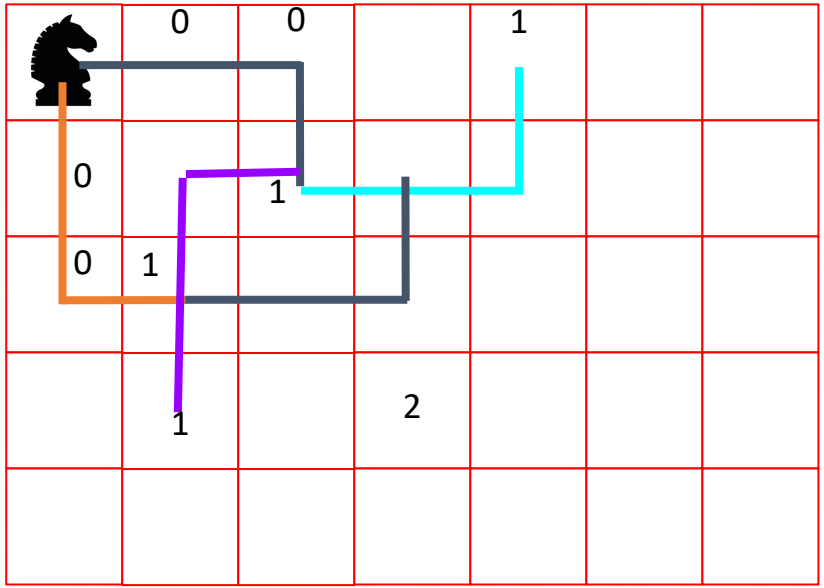
- for i = 0..n for j = 0..m - no
- for j = 0..m for i = 0..n - no

Why?


(2, 3)
(1, 5)
(4, 2)

DP(Knight) - Order

- for diag = 0..(n + m) for i = 0..n


0
0
1
0
1
0
1
1
2

```python
n = 4
m = 4
dp = []
for i in range(20):
    dp.append([0] * 20)

dp[0][0] = 1;
for diag in range(1, n + m + 1):
    for i in range(0, n + 1):
        j = diag - i
        dp[i][j] = 0;
        if j >= 2:
            dp[i][j] += dp[i + 1][j - 2]
            if i >= 1:
                dp[i][j] += dp[i - 1][j - 2]
        if i >= 2:
            dp[i][j] += dp[i - 2][j + 1]
            if j >= 1:
                dp[i][j] += dp[i - 2][j - 1]
for i in range(0, n):
    for j in range(0, m):
        print(dp[i][j], end=' ')
    print("")
```

#stdin #stdout 0.04s 9376KB

stdin

```
Standard input is empty
```

stdout

```
1 0 0 0
0 0 1 1
0 1 0 2
0 1 2 2
```

```cpp
const int n = 4, m = 4;

int main() {
    int dp[20][20];
    dp[0][0] = 1;
    for (int diag = 1; diag <= n + m; diag++) {
        for (int i = 0; i <= n; i++) {
            int j = diag - i;
            dp[i][j] = 0;
            if (j >= 2) {
                dp[i][j] += dp[i + 1][j - 2];
                if (i >= 1) {
                    dp[i][j] += dp[i - 1][j - 2];
                }
            }
            if (i >= 2) {
                dp[i][j] += dp[i - 2][j + 1];
                if (j >= 1) {
                    dp[i][j] += dp[i - 2][j - 1];
                }
            }
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << dp[i][j] << " ";
        }
        cout << "\n";
    }
}
```

#stdin #stdout 0.01s 5464KB

comments ()

stdin

copy

```
Standard input is empty
```

stdout

copy

```
1 0 0 0
0 0 1 1
0 1 0 2
0 1 2 2
```

Lazy DP

- Let’s solve Fibonacci another way?

редактировать fork скачать

```python
FIB = [0] * 200

def fib(n):
    if n <= 1:
        return 1
    if FIB[n] != 0:
        return FIB[n]
    FIB[n] = fib(n - 1) + fib(n - 2)
    return FIB[n]

n = 150
print(fib(n))
```

#stdin #stdout 0.03s 9684KB

stdin

```
Standard input is empty
```

stdout

```
1613053142490458141579790738634 9
```

редактировать fork скачать

```
1. def fib(n):
2.     if n <= 1:
3.         return 1
4.     return fib(n - 1) + fib(n - 2)
5.
6. n = 40
7. print(fib(n))
```

Time limit exceeded

#stdin #stdout 5s 9424KB

stdin

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```cpp
#include <iostream>

using namespace std;

int FIB[200], MOD = 10000;

int fib(int n) {
    if (FIB[n] != 0) {
        return FIB[n];
    }
    if (n == 0 || n == 1) {
        return 1;
    }
    return FIB[n] = (fib(n - 1) + fib(n - 2)) % MOD;
}
int main() {
    cout << fib(150);
}
```

stdin #stdout 0.01s 5316KB

stdin

```
Standard input is empty
```

stdout

```
6349
```

Lazy vs Regular

Lazy:

1) use only needed things
2) sometimes faster and needs less memory
3) easier to create

Regular:

1) mostly faster
2) mostly better with memory
3) Easier to prove

Greatest common subsequence

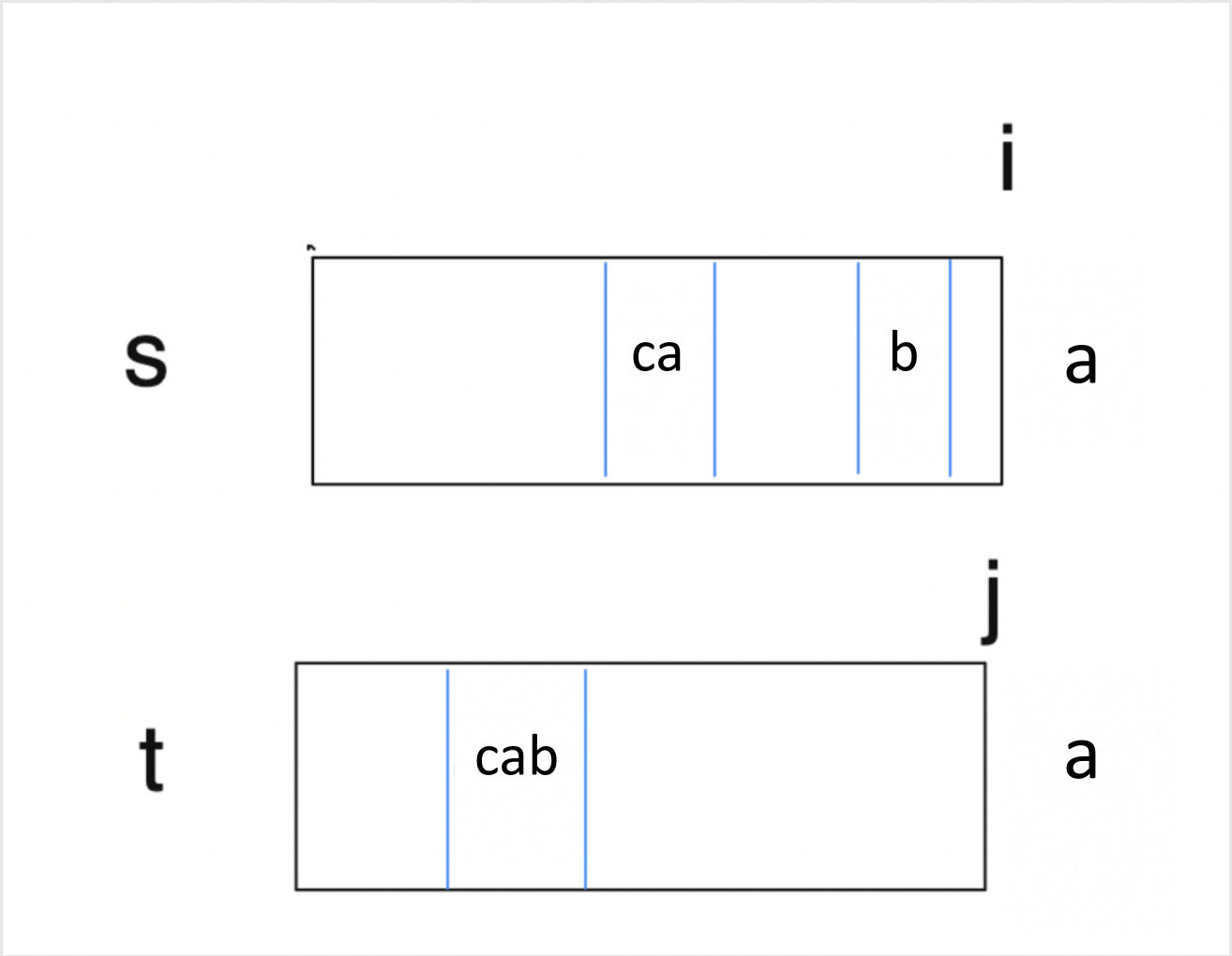
- We have two sequences, s of length n and t of length m.
- A subsequence is a set of ordered indices, e.g. aca is a subsequence of abcad, but adb is not.
- A common subsequence is such a subsequence which both strings have.
- We need to find the largest common subsequence.

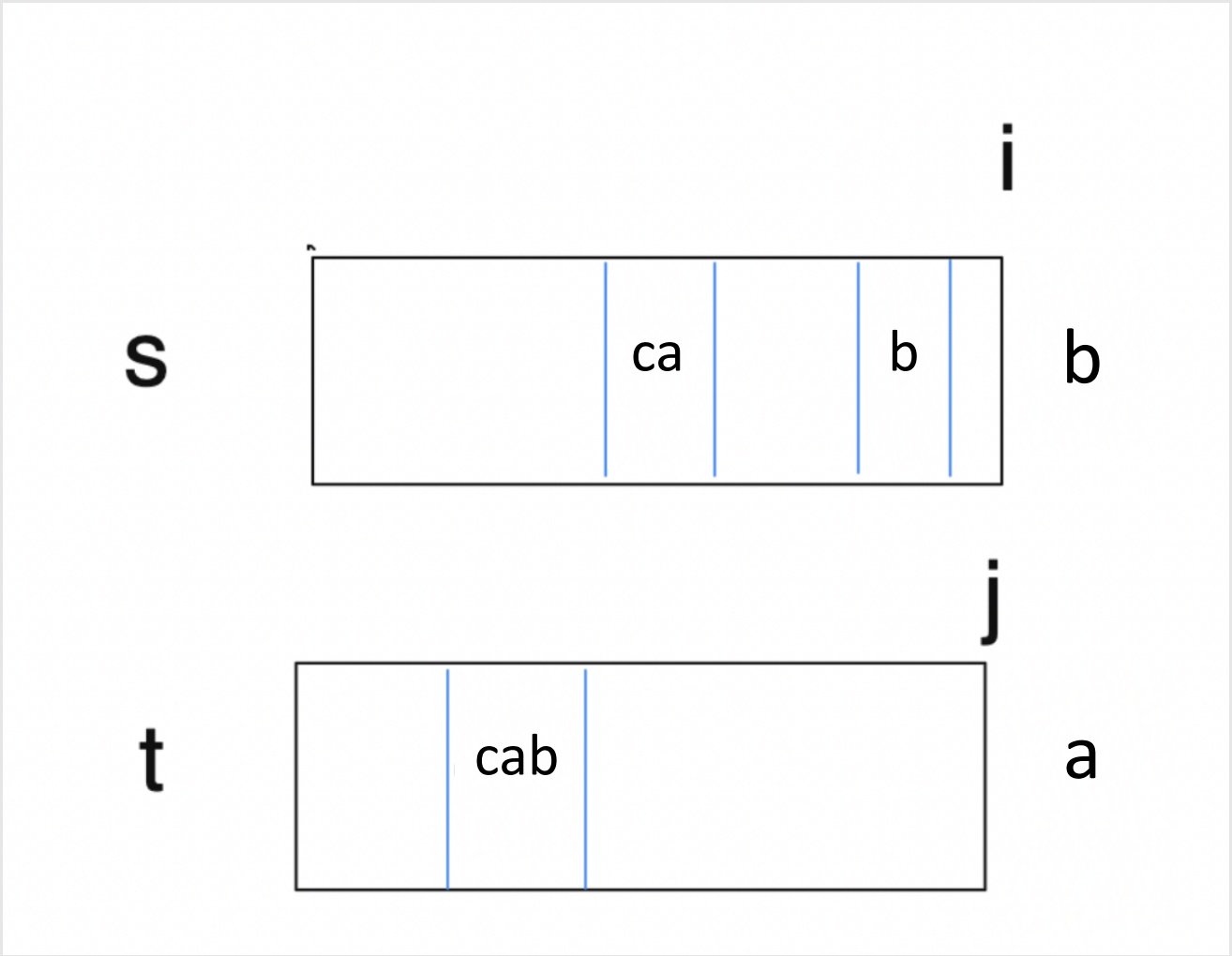
Greatest common subsequence

- s = gabacaba, t = abcdag
- common subsequences = {g, a, b, c, ab, ac, ba, ca, abc, cba, abca}
- greatest common subsequence = abca

DP(GCS)

- dp[i][j] - GCS(s[0..(i - 1)], t[0..(j - 1)])
- dp[0][0] = 0
- Any
- if (s[i] == t[j]) -> dp[i][j] = dp[i - 1][j - 1] + 1; else -> dp[i][j] = max(dp[i][j - 1], dp[i - 1][j])
- dp[n][m] - answer


i
s
ca
b
a
j
t
cab
a


i
s
ca
b
b
j
t
cab
a

```python
x = input()
y = input()

DP = [[0]*(len(y)+1) for _ in range(len(x)+1)]

for x_i, x_elem in enumerate(x):
    for y_i, y_elem in enumerate(y):
        if x_elem == y_elem:
            DP[x_i][y_i] = DP[x_i - 1][y_i - 1] + 1
        else:
            DP[x_i][y_i] = max((DP[x_i][y_i - 1], DP[x_i - 1][y_i]))

print(DP[len(x) - 1][len(y) - 1])
```

Success #stdin #stdout 0.03s 9692KB

stdin

```
gabacaba
abcdag
```

stdout

```
4
```

```cpp
using namespace std;
int L[100][100];

int main() {
    string s, t;
    cin >> s >> t;
    int n = s.length(), m = t.length();

    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (i == 0 || j == 0) {
                L[i][j] = 0;
            }
            else if (s[i - 1] == t[j - 1]) {
                L[i][j] = L[i - 1][j - 1] + 1;
            }
            else {
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
            }
        }
    }
    cout << L[n][m];
}
```

Success #stdin #stdout 0.01s 5508KB

stdin

```
gabacaba
abcdag
```

stdout

```
4
```

Greatest increasing subsequence

- We have a sequence a of length n of numbers
- A subsequence is a set of ordered indices, e.g. aca is a subsequence of abcad, but adb is not.
- An ascending sequence is a sequence of numbers such that each number is greater than the previous number
- We need to find the largest increasing subsequence.

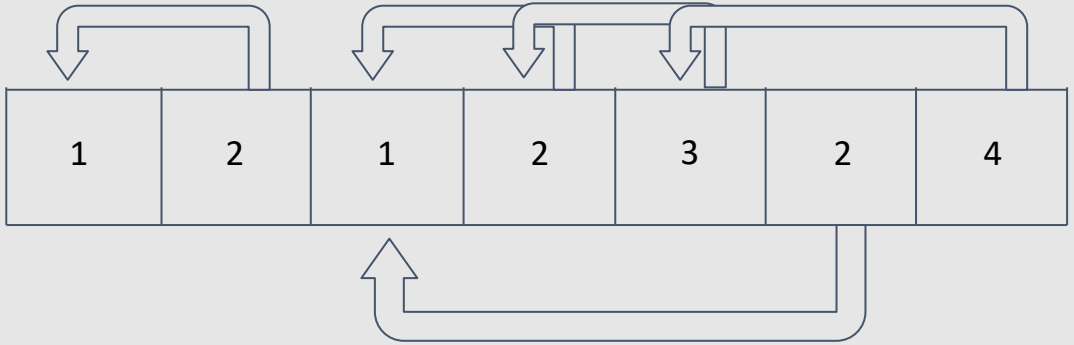
Greatest increasing subsequence

- a = {1, 3, -1, 2, 3, 0, 4}
- increasing subsequences = {{1}, {3}, {-1}, {2}, {0}, {4}, {1, 3}, {-1, 2}, {1, 2}, {-1, 0} .. , {1, 2, 3, 4}, {-1, 2, 3, 4}}
- greatest increasing subsequence = {1, 2, 3, 4}
- Answer is {1, 2, 3, 4} and length = 4

DP1(GIS)

- dp[i] - GIS(a[0..(i - 1)])
- dp[i] = 1 for all i
- from i = 0 to n - 1
- dp[i] = max(dp[j] + 1) for all a[j] < a[i] and j < i
- max(dp[i]) for all i - answer


1
3
-1
2
3
0
4


1
2
1
2
3
2
4

```cpp
int main() {
    int n;
    cin >> n;

    int a[n], dp[n];
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    int max_length = 0;

    for (int i = 0; i < n; i++) {
        dp[i] = 1;
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i]) {
                dp[i] = max(dp[j] + 1, dp[i]);
            }
        }
        max_length = max(max_length, dp[i]);
    }

    cout << max_length;
}
```

Success #stdin #stdout 0.01s 5444KB

stdin

```
7
1 3 -1 2 3 0 4
```

stdout

```
4
```

```python
n = int(input())

a = list(map(int, input().split(' ')))

dp = [0] * n
max_length = 0

for i in range(n):
    dp[i] = 1;
    for j in range(i):
        if a[j] < a[i]:
            dp[i] = max(dp[j] + 1, dp[i])
    max_length = max(max_length, dp[i])

print(max_length)
```

Success #stdin #stdout 0.03s 9796KB

stdin

```
7
1 3 -1 2 3 0 4
```

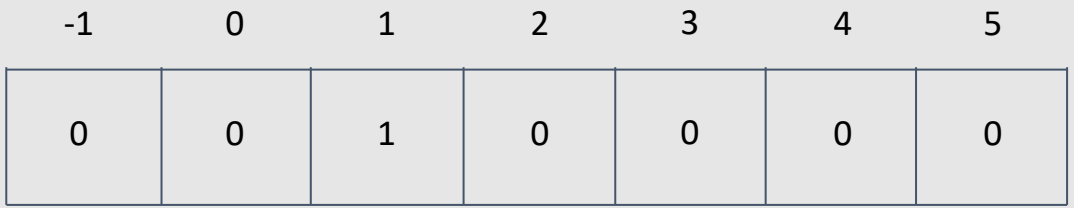
stdout

```
4
```

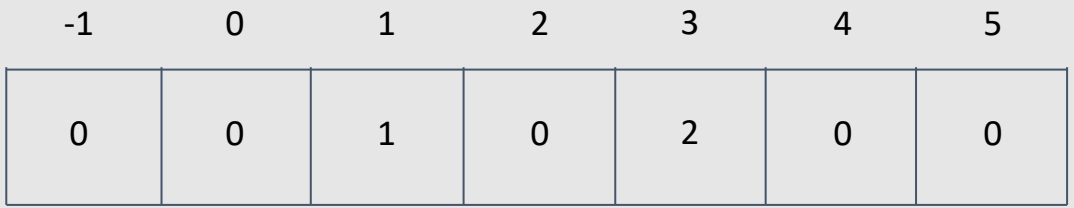
DP2(GIS)

- dp[i] - GIS(elems) for elem in elems : elem < i, elems.last = i
- dp[i] = 0 for all i, dp[a[i]] = 1 for all i
- from i = 0 to n - 1 : a[i]
- dp[a[i]] = max(dp[j] + 1) for all j < a[i]
- max(dp[i]) for all i - answer

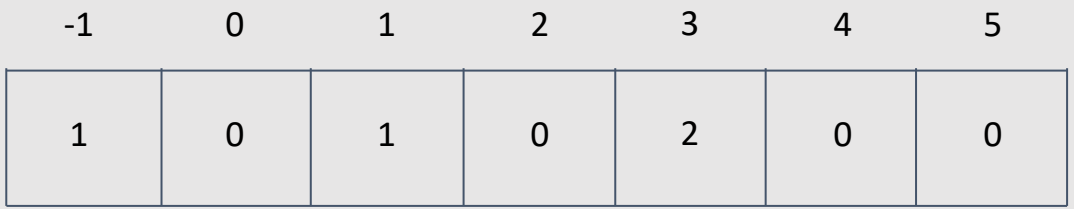

1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
0
0
1
0
0
0
0

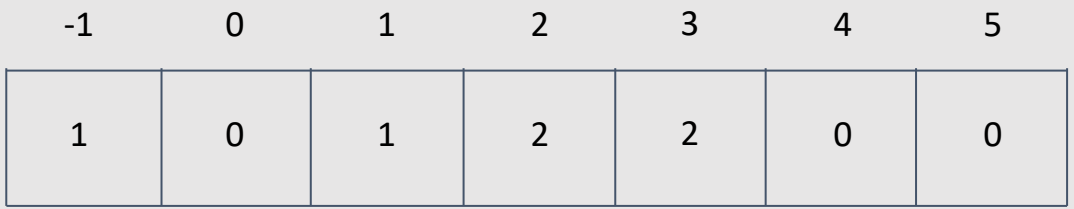

1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
0
0
1
0
2
0
0

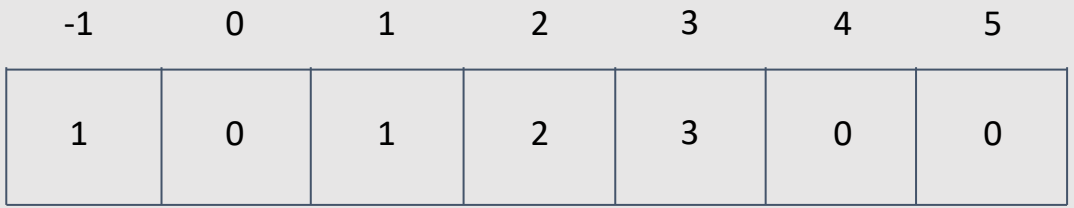

1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
1
0
1
0
2
0
0

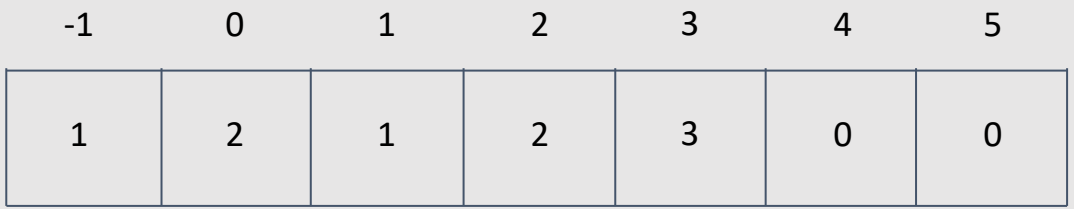

1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
1
0
1
2
2
0
0

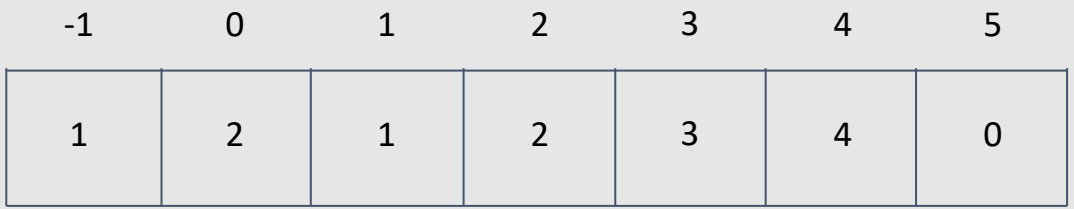

1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
1
0
1
2
3
0
0

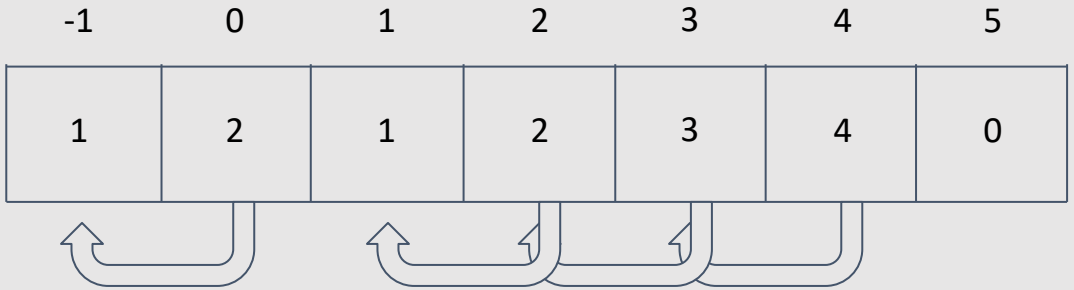

1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
1
2
1
2
3
0
0


1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
1
2
1
2
3
4
0

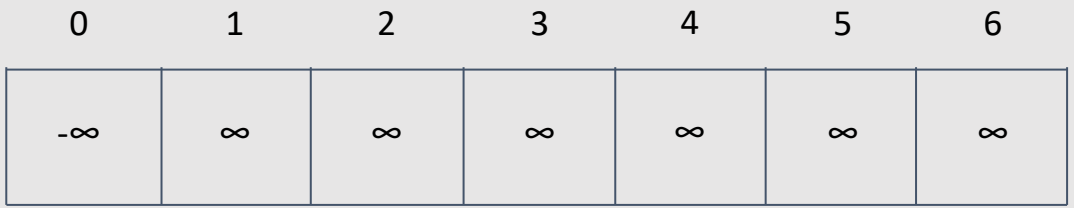

1
3
-1
2
3
0
4


-1
0
1
2
3
4
5
1
2
1
2
3
4
0

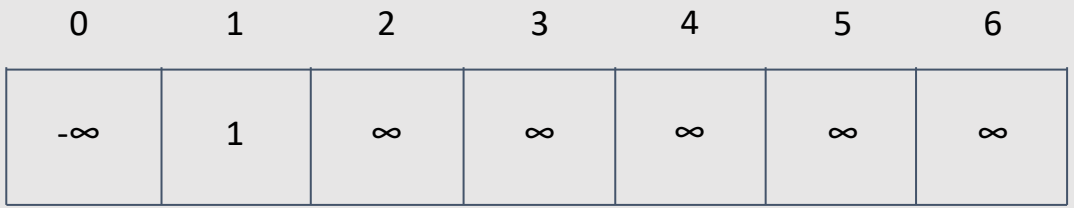
DP3(GIS)

- dp[i] = minimal element for GIS(.., element) of length i
- dp[0] = -inf, dp[i] = inf for all i > 0
- from i = 0 to n - 1 : a[i]
- dp[j] = min(dp[j], a[i]), for all j : dp[j - 1] < a[i]
- max(dp[i]) for all i : dp[i] != inf - answer


1
3
-1
2
3
0
4


0
1
2
3
4
5
6
-∞
∞
∞
∞
∞
∞
∞

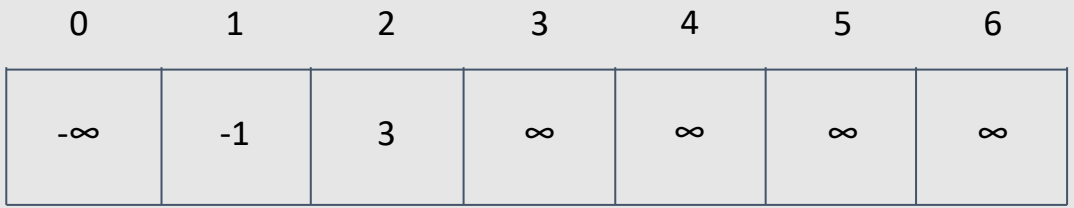

1
3
-1
2
3
0
4


0
1
2
3
4
5
6
-∞
1
∞
∞
∞
∞
∞

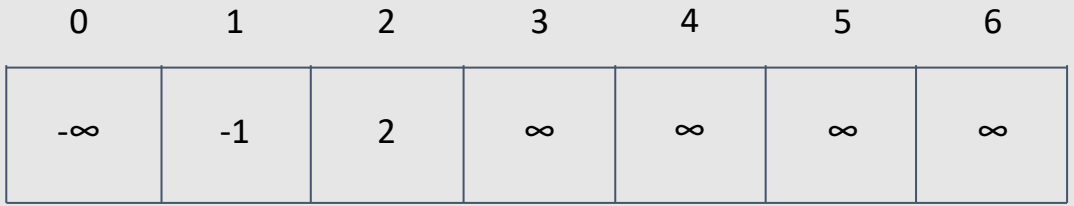
1	3	-1	2	3	0	4

0	1	2	3	4	5	6
-∞	1	3	∞	∞	∞	∞

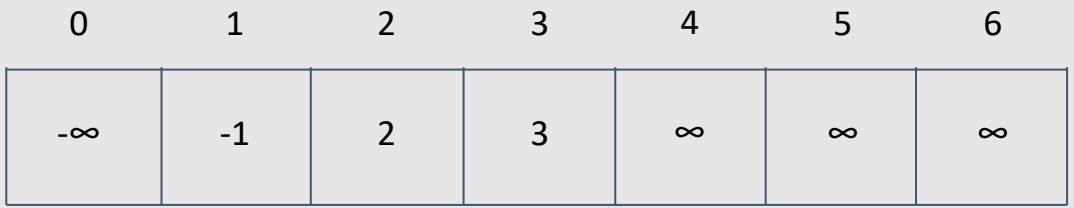

1
3
-1
2
3
0
4


0
1
2
3
4
5
6
-∞
-1
3
∞
∞
∞
∞

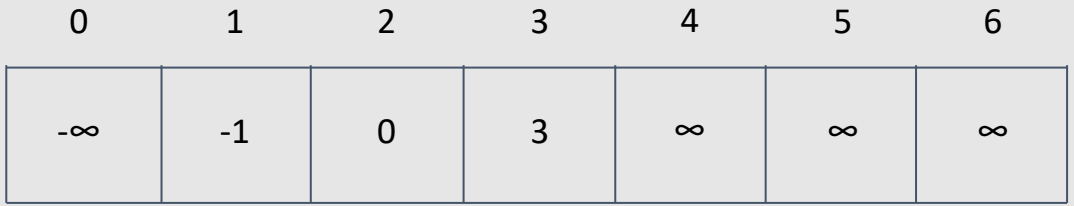

1
3
-1
2
3
0
4


0
1
2
3
4
5
6
-∞
-1
2
∞
∞
∞
∞

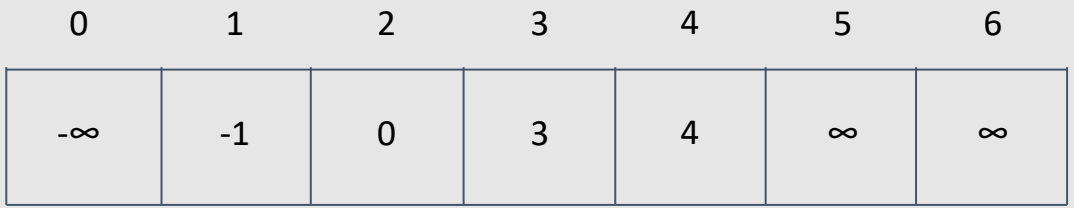

1
3
-1
2
3
0
4


0
1
2
3
4
5
6
-∞
-1
2
3
∞
∞
∞


1
3
-1
2
3
0
4


0
1
2
3
4
5
6
-∞
-1
0
3
∞
∞
∞


1
3
-1
2
3
0
4


0
1
2
3
4
5
6
-∞
-1
0
3
4
∞
∞

```
    int dp[n];
    dp[0] = -INF;

    for (int i = 1; i <= n; i++) {
        dp[i] = INF;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= n; j++) {
            if (dp[j - 1] < a[i] && a[i] < dp[j]) {
                dp[j] = a[i];
            }
        }
    }

    for (int i = 0; i <= n; i++) {
        cout << i << " " << dp[i] << "\n";
    }
}
```

Success #stdin #stdout 0.01s 5312KB

stdin

```
7
1 3 -1 2 3 0 4
```

stdout

```
0 -1000000000
1 -1
2 0
3 3
4 4
5 1000000000
6 1000000000
7 1000000000
```

```python
INF = 1e9

n = int(input())
a = list(map(int, input().split(' ')))

dp = [-INF] + [INF] * (n - 1);

for i in range(n):
    for j in range(1, n + 1):
        if dp[j - 1] < a[i] and a[i] < dp[j]:
            dp[j] = a[i]
for i in range(n):
    print(i, dp[i])
```

Success #stdin #stdout 0.03s 9612KB

stdin

```
7
1 3 -1 2 3 0 4
```

stdout

```
0 -1000000000.0
1 -1
2 0
3 3
4 4
5 1000000000.0
6 1000000000.0
```

Idea

- We need to insert in such pos, that dp[pos] > a[i] > dp[pos - 1] > dp[pos - 2] …
- binary search

```
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    int dp[n];
    dp[0] = -INF;

    for (int i = 1; i <= n; i++) {
        dp[i] = INF;
    }

    for (int i = 0; i < n; i++) {
        int j = int(upper_bound(dp, dp + n + 1, a[i]) - dp);
        if (dp[j - 1] < a[i] && a[i] < dp[j]) {
            dp[j] = a[i];
        }
    }

    for (int i = 0; i <= n; i++) {
        cout << i << " " << dp[i] << "\n";
    }
}
```

Success #stdin #stdout 0.01s 5356KB com

stdin

```
7
1 3 -1 2 3 0 4
```

stdout

```
0 -1000000000
1 -1
2 0
3 3
4 4
5 1000000000
6 1000000000
7 1000000000
```

```python
import bisect

INF = 1e9

n = int(input())
a = list(map(int, input().split()))

dp = [-INF] + [INF] * n

for i in range(n):
    j = bisect.bisect_left(dp, a[i], 1)
    if dp[j - 1] < a[i] and a[i] < dp[j]:
        dp[j] = a[i]

for i in range(n + 1):
    print(i, dp[i])

```

Success #stdin #stdout 0.03s 9672KB

stdin

```
7
1 3 -1 2 3 0 4
```

stdout

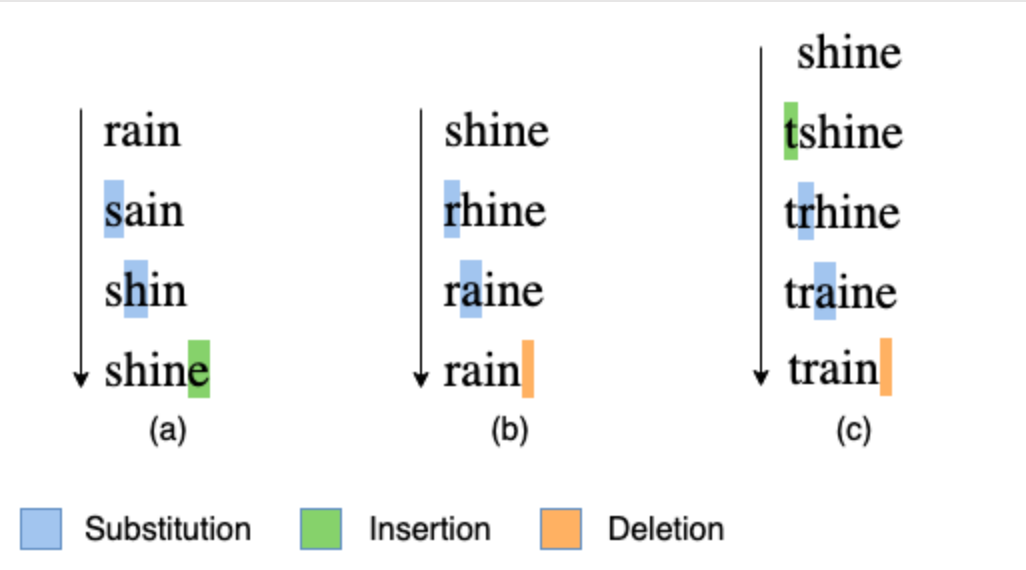
```
0 -1000000000.0
1 -1
2 0
3 3
4 4
5 1000000000.0
6 1000000000.0
7 1000000000.0
```

Damerau–Levenshtein distance

- Damerau–Levenshtein distance between two words is the minimum number of operations required to change one word into the other.

Operations

- insertion (s = “aba”, i = 1, letter = ‘d’, res = “adba”)
- deletion (s = “aba”, i = 1, res = “aa”)
- substitutions (s = “aba”, i = 1, letter = ‘d’, res = “ada”)
- transposition (s = “aba”, i = (0, 1), s = “baa”) - optional !!!


rain
sain
shin
shine
(a)
shine
rhine
raine
rain
(b)
shine
tshine
trhine
traine
train
(c)
Substitution
Insertion
Deletion

Solution

- Given |s| = n, |t| = m
- dp[i][j] - Distance for making equal s(0..i) and t(0..j)
- dp[0][0] = 0
- any order
- next slide
- dp[n][m]

Formulas

- dp[i][j] min= dp[i - 1][j] + 1 (remove symbol from s)
- dp[i][j] min= dp[i][j - 1] + 1 (remove symbol from t)
- dp[i][j] min= dp[i - 1][j - 1] (if s[i] = t[j])
- dp[i][j] min= dp[i - 1][j - 1] + 1 (if s[i] != t[j])
- dp[i][j] min= dp[i - 2][j - 2] + 1 if (s[i] = j[i - 1] and s[i - 1] = j)

```
    string s, t;
    cin >> s >> t;
    int n = s.length(), m = t.length();

    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            dp[i][j] = 1e9;
        }
    }

    dp[0][0] = 0;
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= m; j++)
        {
            if (i) {
                dp[i][j] = min(dp[i][j], dp[i - 1][j] + 1);
            }
            if (j) {
                dp[i][j] = min(dp[i][j], dp[i][j - 1] + 1);
            }
            if (i && j) {
                int substitutionCost = (s[i - 1] != t[j - 1]);
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + substitutionCost);
            }
        }
    }
    cout << dp[n][m];
}
```

Success #stdin #stdout 0.01s 5512KB

comments ()

stdin

copy

```
rain
shine
```

stdout

copy

```
3
```

```python
def levenshtein_distance(s, t):
    n = len(s)
    m = len(t)
    d = [[0] * (m + 1) for i in range(n + 1)]

    for i in range(1, n + 1):
        d[i][0] = i

    for j in range(1, m + 1):
        d[0][j] = j

    for j in range(1, m + 1):
        for i in range(1, n + 1):
            if s[i - 1] == t[j - 1]:
                cost = 0
            else:
                cost = 1
            d[i][j] = min(d[i - 1][j] + 1,       # deletion
                          d[i][j - 1] + 1,       # insertion
                          d[i - 1][j - 1] + cost) # substitution

    return d[n][m]

print(levenshtein_distance("rain", "shine"))
print(levenshtein_distance("shine", "rain"))
print(levenshtein_distance("shine", "train"))
print(levenshtein_distance("train", "shine"))
```

Success #stdin #stdout 0.03s 9420KB

stdin

```
Standard input is empty
```

stdout

```
3
3
4
4
```

Check yourself

- A template is defined as a string composed of English letters (a, ..., z, A, ..., Z) and the characters ? and *.
- Each ? character can be replaced with any single letter, while each * character can be replaced with any sequence of letters (possibly empty).
- Any string made up of letters that can be derived from the template by these replacements is said to satisfy the template.
- The task is to check if a string of length ltl = m matches the template.

Examples

- s = a*b?
- t = aaabc - match
- t = abc - match
- t = ab - no match
- t = acccccccccc - no match

Plan

- What do we consider a state of dynamics (e.g. dp[i])?
- What values of dp do we know?
- What is the order in which dp is counted?
- Which formula is used to calculate dp?
- What's the answer to our problem?

Solution

- dp[i][j] - if t[0..j] matches for s[0..i]
- dp[0][0] = true, because empty string matches for pattern
- Any
- next slide
- dp[n][m]

Solution

- dp[i][j] |= dp[i - 1][j - 1], if s[i] = t[j]
- dp[i][j] |= dp[i][j - 1], if s[i] = *
- dp[i][j] |= dp[i - 1][j], if s[i] = *
- dp[i][j] |= dp[i - 1][j - 1], if s[i] = ?

ATM

- In the country of "COINLAND", there are n denominations of coins, each denomination is denoted as a_i.
- The task is to check if it's possible to collect a sum of exactly X using the coins?
- You can take any number of coins of denomination a_i.

Example

- a = [2, 5]
- X = 6 - Yes, 2 + 2 + 2
- X = 25 - Yes, 5 + 5 + 5 + 5 + 5
- X = 11 - Yes, 2 + 2 + 5
- X = 3 - No

Idea

- If we can collect exactly i euro cents and we have coin j -> we can collect i + j, i + j * 2 and so on

DP(ATM)

- dp[i][j] - can we collect exactly i euro cents, using only j first coins
- dp[0][0] = true
- any
- dp[i][j] = dp[i - a[j]][j] or dp[i][j - 1]
- dp[X][n]

Formulas

- dp[i][j] = dp[i - a[j]][j], if we took one more coin with denomination a[j]
- dp[i][j] = dp[i][j - 1], if we skip denomination a[j]

```
    int X;
    cin >> X;

    for (int i = 0; i <= X; i++) {
        for (int j = 0; j <= n; j++) {
            dp[i][j] = false;
        }
    }

    dp[0][0] = true;

    for (int i = 0; i <= X; i++) {
        for (int j = 0; j < n; j++) {
            if (j > 0) {
                dp[i][j] |= dp[i][j - 1];
            }
            if (i >= a[j]) {
                dp[i][j] |= dp[i - a[j]][j];
            }
        }
    }
    for (int i = 0; i <= X; i++) {
        cout << i << " " << dp[i][n - 1] << "\n";
    }
}
```

stdin

```
2
2 5
12
```

stdout

```
0 1
1 0
2 1
3 0
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
```

```
dp = []
dp.append([True] + [False] * (n - 1))
for i in range(1, X + 1):
    dp.append([False] * n)

for i in range(X + 1):
    for j in range(n):
        if j > 0:
            dp[i][j] = dp[i][j] or dp[i][j - 1]
        if i >= a[j]:
            dp[i][j] = dp[i][j] or dp[i - a[j]][j];

for i in range(X + 1):
    print(i, dp[i][n - 1])
```

Success #stdin #stdout 0.03s 9708KB

stdin

```
2
2 5
12
```

stdout

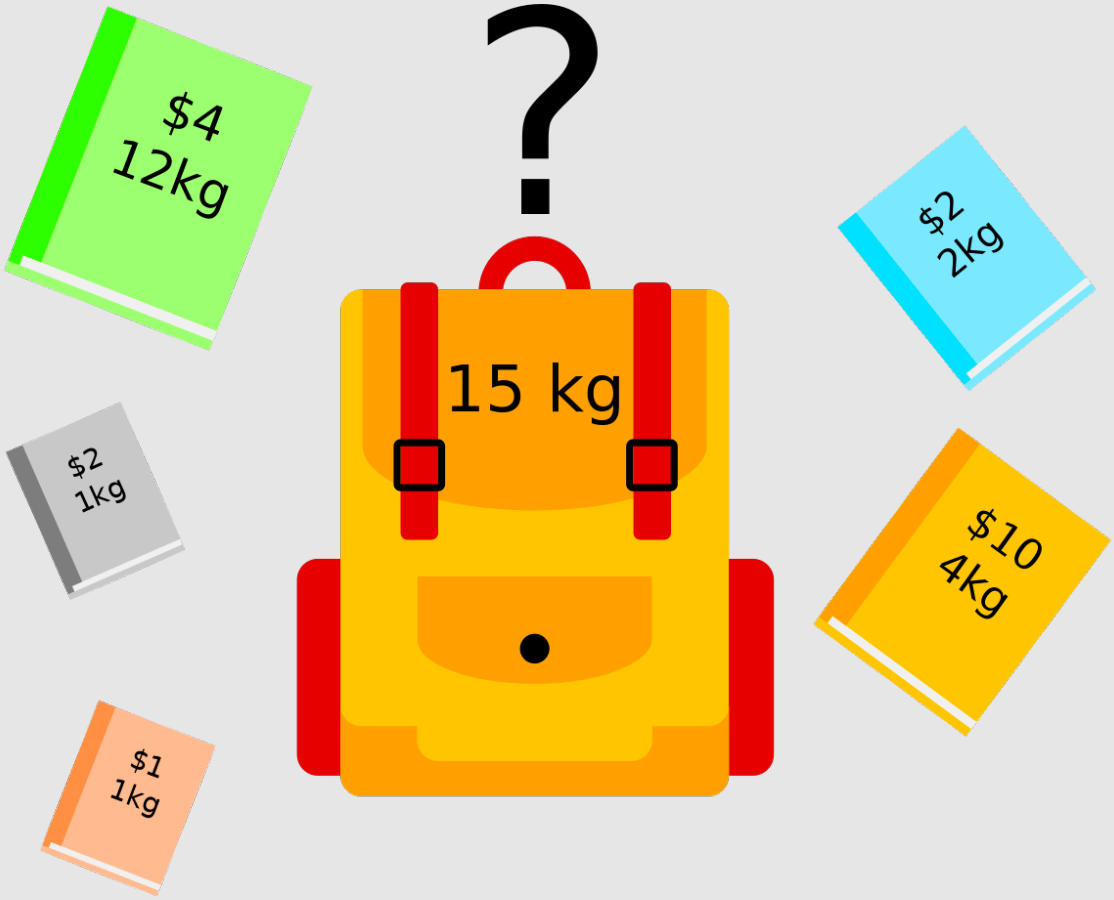
```
0 True
1 False
2 True
3 False
4 True
5 True
6 True
7 True
8 True
9 True
10 True
11 True
12 True
```

Knapsack

- You have n items, each with its own weight w_i and price c_i.
- You have a backpack with a carrying capacity of W kilograms.
- What is the maximum value of items you can obtain?

Example

- W = 15
- w = [1, 1, 2, 4, 12]
- c = [2, 1, 2, 10, 4]
- Answer - 2 + 1 + 2 + 10 = 15


$4
12kg
?
$2
2kg
15 kg
$2
1kg
$10
4kg
$1
1kg

DP(Knapsack)

- dp[i][j] - can we collect exactly weight i, using only j first items
- dp[0][0] = true
- any
- next slide
- max(dp[i][n]) for all i <= W

Formulas

- dp[i][j] = dp[i - w[j - 1]][j - 1] + c[j - 1], if we took j-th item with weight w[j - 1] and cost c[j - 1]
- dp[i][j] = dp[i][j - 1], if we skip item j - 1

```
23.
24.     for (int i = 0; i <= X; i++) {
25.         for (int j = 0; j <= n; j++) {
26.             dp[i][j] = -INF;
27.         }
28.     }
29.
30.     dp[0][0] = 0;
31.
32.     for (int i = 0; i <= X; i++) {
33.         for (int j = 0; j <= n; j++) {
34.             if (j > 0) {
35.                 dp[i][j] = max(dp[i][j], dp[i][j - 1]);
36.                 if (i >= w[j - 1]) {
37.                     dp[i][j] = max(dp[i - w[j - 1]][j - 1] + c[j - 1], dp[i][j]);
38.                 }
39.             }
40.         }
41.     }
42.     int maxx = 0;
43.     for (int i = 0; i <= X; i++) {
44.         maxx = max(dp[i][n], maxx);
45.     }
46.     cout << maxx;
47. }
```

stdin

```
5
1 1 2 4 12
2 1 2 10 4
15
```

stdout

```
15
```

```python
dp = []
dp.append([0] + [-1e9] * n)
for i in range(1, X + 1):
    dp.append([-1e9] * (n + 1))

for i in range(X + 1):
    for j in range(n + 1):
        if j > 0:
            dp[i][j] = max(dp[i][j], dp[i][j - 1])
            if i >= w[j - 1]:
                dp[i][j] = max(dp[i][j], dp[i - w[j - 1]][j - 1] + c[j - 1])

maxx = 0
for i in range(X + 1):
    maxx = max(maxx, dp[i][n])

print(maxx)
```

Success #stdin #stdout 0.03s 9784KB

comment

stdin

```
5
1 1 2 4 12
2 1 2 10 4
15
```

stdout

```
15
```

Spoiler

- solution works for O(n * W)
- polynomial - O(n^x)
- Exists solution for knapsack for something like O(n^x)?

DP on segments

- idea - dp[l][r] - answer for segment [l..r]
- ~~order - l = 1..n, r = 1..n~~
- order - len = 1..n, l = 1..n

Palindrome

- A palindrome is a word, number, phrase, or other sequence of symbols that reads the same backwards as forwards
- aba - palindrome
- asd - no

Task

- We have string, we need to find amount of palindromic substrings
- for “abac” answer - [“a”, “b”, “a”, “c”, “aba”] - 5

abac

abac

Plan

- dp[l][r] - is substring [l .. r] palindromic?
- dp[l][l] = true, dp[l][l + 1] = (s[l] == s[l + 1])
- len = 3..n, l = 0..n
- dp[l][r] = (dp[l + 1][r - 1] and s[l] == s[r])
- sum(dp[l][r]) for all l and r

```cpp
int main() {
    string s;
    cin >> s;
    int n = s.length();

    for (int i = 0; i < n; i++) {
        f[i][i] = true;
        f[i][i + 1] = (s[i] == s[i + 1]);
    }

    for (int d = 3; d <= n; d++) {
        for (int l = 0, r = l + d - 1; r < n; l++, r++) {
            f[l][r] = (s[l] == s[r]) && f[l + 1][r - 1];
        }
    }

    int ans = 0;
    for (int l = 0; l < n; l++) {
        for (int r = l; r < n; r++) {
            ans += f[l][r];
        }
    }
    cout << ans;
}
```

Success #stdin #stdout 0.01s 5512KB

comn

stdin

abac

stdout

5

```python
f = []
for i in range(n + 1):
    f.append([False] * (n + 1))

for i in range(n):
    f[i][i] = True

for i in range(n - 1):
    f[i][i + 1] = (s[i] == s[i + 1])

for d in range(3, n + 1):
    l = 0
    r = l + d - 1
    while r < n:
        f[l][r] = (s[l] == s[r]) and f[l + 1][r - 1]
        l += 1
        r += 1

ans = 0
for l in range(0, n):
    for r in range(l, n):
        ans += f[l][r]

print(ans)
```

Success #stdin #stdout 0.03s 9576KB

stdin

abac

stdout

5

Task

- We have string, we need to find length of maximum palindromic subsequence
- for “adabaca” answer - “aabaa”, length - 5

adabaca

Plan

- dp[l][r] - the length of max palindromic subsequence s[l .. r]
- dp[l][l] = 1, dp[l][l + 1] = (s[l] == s[l + 1]) * 2
- len = 3..n, l = 0..n
- next slide
- max(dp[l][r]) for all l and r

Idea

- “a” + s[l+1..r-1] + “a” - dp[l+1..r-1] + 2
- “a” + s[l+1..r] - dp[l+1..r]
- dp[l..r-1] + “b” - dp[l..r-1]

Formula

- dp[l][r] = max(dp[l + 1][r], dp[l][r - 1], dp[l + 1][r - 1] + 2 * (s[l] == s[r]))

DP on numbers

- idea - dp[len][conditions] - answer for number of length - len and some conditions(for example sum) OR
- idea - dp[len][digit][conditions] - answer for number of length - len with last digit - digit and some conditions(for example sum)

Simple task

- Let’s count amount of numbers with length n and even sum of digits.

Plan

- dp[len][digit][0/1] - amount of numbers with length len, last digit = digit and sum of digits % 2 = 0/1
- dp[1][0/2/4/6/8][0] = 1, dp[1][1/3/5/7/9][1] = 1
- len = 1..n
- next slide
- sum(dp[n][1..9][0])

Formula

- dp[i][digit][0] = sum(dp[i - 1][digit_last][(10 - digit)%2])
- dp[i][digit][1] = sum(dp[i - 1][digit_last][(11 - digit)%2])

Example

- dp[2][1][0] = dp[1][0][1] + dp[1][1][1] + … + dp[1][9][1]

why?

- dp[1][1][1] = 1(only “1”)
- dp[2][1][0] += 1(because we take “1” and add another “1” and it’s “11”)

```cpp
#include <bits/stdc++.h>

using namespace std;
const int N = 10;

int dp[10][10][2];

int main() {
    int ans = 0;
    for (int i = 100; i < 1000; i++) {
        int sum = (i % 10) + (i / 100) + (i / 10 % 10);
        if (sum % 2) {
            ans++;
        }
    }
    cout << ans;
}
```

Success #stdin #stdout 0.01s 5384KB

stdin

```
3
```

stdout

```
450
```

```cpp
int dp[10][10][2];

int main() {
    int n;
    cin >> n;

    for (int i = 0; i <= 9; i++) {
        dp[1][i][i % 2] = 1;
    }

    for (int i = 2; i <= n; i++) {
        for (int digit = 0; digit <= 9; digit++) {
            for (int digit_last = 0; digit_last <= 9; digit_last++) {
                dp[i][digit][0] += dp[i - 1][digit_last][(10 - digit) % 2];
                dp[i][digit][1] += dp[i - 1][digit_last][(11 - digit) % 2];
            }
        }
    }
    int answer = 0;
    for (int i = 1; i <= 9; i++) {
        answer += dp[n][i][0];
    }
    cout << answer;
}
```

Success #stdin #stdout 0.01s 5516KB

comments ()

stdin copy

```
3
```

stdout copy

```
450
```

Fibonacci again

- What if everything is math?
- Sometimes we can transform dp to matrix multiplication.

Fibonacci again

- Let A = [1, 1] = [Fib_0, Fib_1]
- Let B = [[0, 1], [1, 1]]
- A * B = ?
- A * B * B = ?
- A * B * B * B = ?

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 3 \end{pmatrix}$$

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 5 \end{pmatrix}$$

Fibonacci again

- A = [Fib_0, Fib_1]
- A * B = [Fib_1, Fib_2]
- A * B * B = [Fib_2, Fib_3]
- A * B * B * B = [Fib_3, Fib_4]
- A * B^n = [Fib_n, Fib_(n + 1)]

$$3 \cdot 0 + 1 \cdot 2 + 0 \cdot 0 = 2$$

$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 4 \\ 2 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & \\ & \end{bmatrix}$$

2×3 $\quad$ 3×2 $\quad$ 2×2

Why? Formally

- $C_{(i, j)} = (A * B)_{(i, j)} = A_i * B_{,j} = \sum(A_{ik} * B_{kj})$
- $C_{(0, 0)} = A_{(0, 0)} * B_{(0, 0)} + A_{(0, 1)} * B_{(1, 0)} = 0 * Fib_n + 1 * Fib_{(n + 1)} = Fib_{(n + 1)}$
- $C_{(0, 1)} = A_{(0, 0)} * B_{(0, 1)} + A_{(0, 1)} * B_{(1, 1)} = 1 * Fib_n + 1 * Fib_{(n + 1)} = Fib_{(n + 2)}$

Why? Not formally

- In this case, matrix A at each step was the matrix of the last two Fibonacci numbers.
- C_(0, 0) is the sum of the element-wise products of the first row of A and the first column of B.
- Informally, matrix B can be interpreted as a transition matrix, that is, which elements will be included in the answer and which will not.

Examples

- For example, if matrix B = [[1, 0], [0, 1]], it would be a matrix that doesn't change anything;
- the first element goes to the first, the second to the second.
- If matrix B = [[0, 1], [1, 0]], it would be a matrix that swaps elements; the first element goes to the second, and the second to the first.

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 2 \end{pmatrix}$$

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 3 \end{pmatrix}$$

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 2 \end{pmatrix}$$

```cpp
vector<vector<int> > e({{1, 0}, {0, 1}});
auto E = Matrix(e);

Matrix binary_pow(Matrix a, int n) {
    if (n == 0) {
        return E;
    }
    if (n == 1) {
        return a;
    }
    auto res = binary_pow(a, n / 2);
    return res * res * binary_pow(a, n % 2);
}

int main() {
    vector<vector<int> > fib({{1, 1}}), step({{0, 1}, {1, 1}});
    auto Fib = Matrix(fib), Step = Matrix(step);
    int n;
    cin >> n;
    auto Res = Fib * binary_pow(Step, n);
    Res.write();
    return 0;
}
```

Success #stdin #stdout 0.01s 5480KB

comments (0)

stdin

copy

```
10
```

stdout

copy

```
89 144
```

Barrels

- We have three types of barrels - A(safe), B(dangerous), C(very dangerous).
- We need to put n barrels in a column
- We can't put a C barrel on another C barrel
- How many ways are there to put the barrels in?

Barrels

- Let's imagine matrix A as the matrix of answers, i.e., [amount_A, amount_B, amount_C] = [1, 1, 1].
- We want to save all transitions except from letter C to letter C, so the matrix B will be [[1, 1, 1], [1, 1, 1], [1, 1, 0]], only the element of transition from C to C is 0.

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 3 & 2 \end{pmatrix}$$

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 3 & 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 8 & 8 & 6 \end{pmatrix}$$

```cpp
vector<vector<int> > e({{1, 0, 0}, {0, 1, 0}, {0, 0, 1}});
auto E = Matrix(e);

Matrix binary_pow(Matrix a, int n) {
    if (n == 0) {
        return E;
    }
    if (n == 1) {
        return a;
    }
    auto res = binary_pow(a, n / 2);
    return res * res * binary_pow(a, n % 2);
}

int main() {
    vector<vector<int> > barrels({{1, 1, 1}}), step({{1, 1, 1}, {1, 1, 1}, {1, 1, 0}});
    auto Barrels = Matrix(barrels), Step = Matrix(step);
    int n;
    cin >> n;
    auto Res = Barrels * binary_pow(Step, n);
    Res.write();
    return 0;
}
```

Success #stdin #stdout 0.01s 5360KB

stdin

```
3
```

stdout

```
22 22 16
```

Bitmask

- 33 = 100001
- 36 = 100100
- 35 = 100011

Bitmask

- mostly int - 32 bits, long long - 64, char - 8
- bit operations - &, |, ^, ~
- bit shifts - <<, >>

bit operations c++

```cpp
#include <iostream>
using namespace std;

int main() {
  unsigned char a = 11, b = 21;
  unsigned char c = ~a;
  cout << (a & b) << " " << (a | b) << " " << (a ^ b)
  << " " << int(c) << " " << (a >> 2) << " " << (a << 2);
  return 0;
}
```

Успешно #stdin #stdout 0.01s 5408KB

comments (0)

stdin

copy

```
Standard input is empty
```

stdout

copy

```
1 31 30 244 2 44
```

```
1. a, b = 11, 21
2. c = ~a
3. print(a & b, a | b, a ^ b, c, a >> 2, a << 2)
```

Success #stdin #stdout 0.03s 9512KB

stdin

```
Standard input is empty
```

stdout

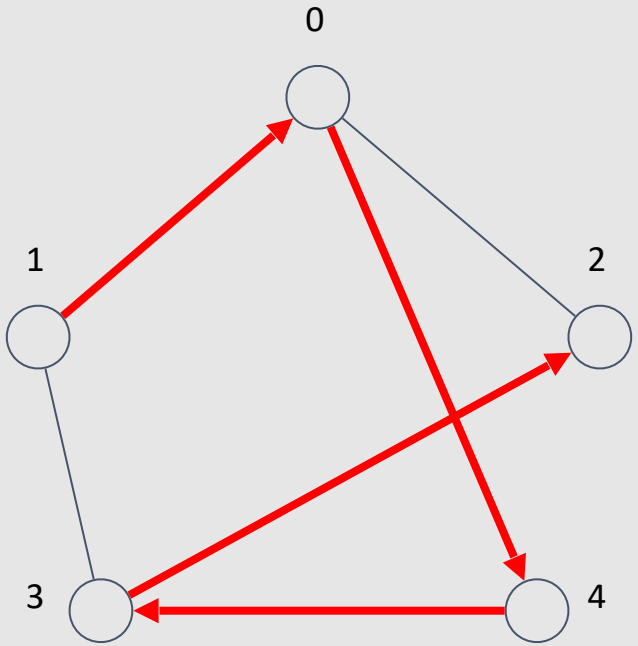
```
1 31 30 -12 2 44
```

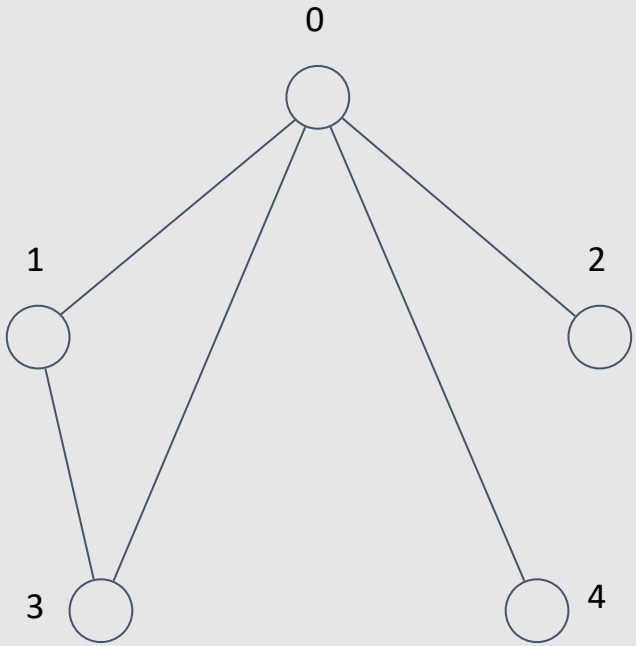
Idea

- dp[mask] - answer for the bitmask of elements(1 - included, 0 - not included)

Task

- You have n cities
- You can visit each city only once
- You have full list of roads
- You need to say - can you visit all cities or not.


0
1
2
3
4


0
1
2
3
4

Shift

- 1 << i - number with i-th digit filled

Plan

- dp[mask][i] - is it possible to visit mask mask and the last city is i
- dp[1<<i][i] = true
- mask = 0..(1 << n) (why - next slides)
- next slides
- ANY(dp[(1 << n) - 1][i]) for i

Order

- mask = 0..(1 << n)
- knights - we need to go from lowest diag to the highest
- subsegment - we need to go from lowest length to the highest one
- masks - we need to go from lowest amount of bits filled to highest

Example

1, 2, 4, 8, 16 … - 1 bit

3, 5, 6, 9 … - 2 bit

1 2 3 4 5

1 bit 1 bit 2 bits (1 bit ????) 2 bits

Order

- We need to visit firstly masks with 1 city, then with 2 and so on
- But it’s hard and we can just visit masks from 0 to (1 << n)
- if a is submask of b, then a < b

Example

- 5(101) is submask of 7(111), $5 < 7$
- 10 (1010) is submask of 15(1111), $10 < 15$

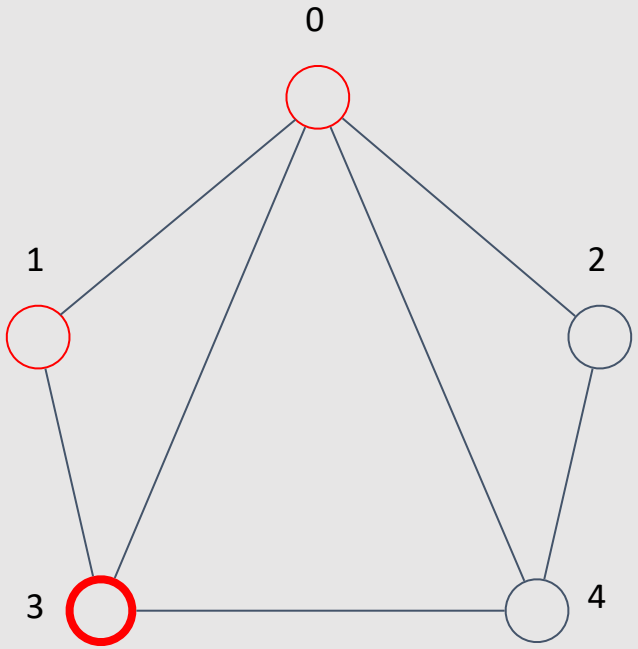
Proof

- if a is submask of b, then all elements of a included in b and b has at least one extra element.
- So at least one new bit in a must be 1, so b is greater.

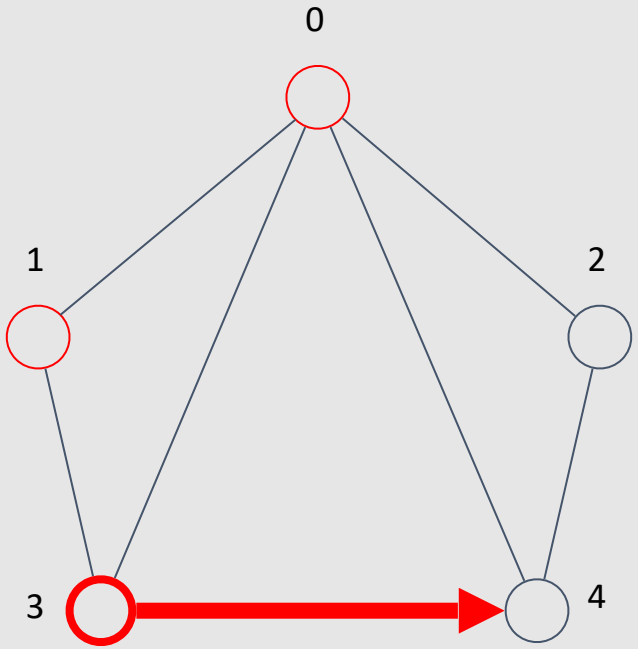
Order

- submask - i-th in order, mask - j-th, $i < j$

dp[01011][3]



dp[11011][4] |= dp[01011][3]



Formulas

- dp[mask][v] = ANY(dp[mask - v][vertex] and road between v and vertex) for all vertex in mask
- dp[mask][v] = ANY(dp[mask ^ (1 << v)][i] and road between i and v) for all i in mask

Formulas

- we had edge(road) from vertex 3 to vertex 4, so we took dp[11011][4] |= dp[01011][3]
- dp[mask][v] |= dp[mask ^ (1 << v)][i] = dp[11011 ^ (1 << 4)][3] = dp[01011][3]

Formulas

- cycle from 0 to (1 << n)

2^n

1 + 2 .. + 2^(n - 1) = 2^n - 1

```
18.     for (int i = 0; i < n; i++) {
19.         dp[(1 << i)][i] = true;
20.     }
21.
22.     for (int mask = 0; mask < (1 << n); mask++) {
23.         for (int j = 0; j < n; j++) {
24.             if (mask & (1 << j)) { // j in mask
25.                 for (int k = 0; k < n; k++) {
26.
27.                     if (mask & (1 << k) // k also in mask
28.                         && adj[k][j] // k and j has road
29.                         && j != k) {
30.
31.                         dp[mask][j] |= dp[mask ^ (1 << j)][k];
32.                     }
33.                 }
34.             }
35.         }
36.     }
37.
```

```
38.         bool answer = false;
39.
40.         for (int i = 0; i < n; i++) {
41.             cout << i << " " << dp[(1 << n) - 1][i] << "\n";
42.             answer |= dp[(1 << n) - 1][i];
43.         }
44.         cout << answer;
45.     }
```

stdin

```
5
0 1 1 1 1
1 0 0 1 0
1 0 0 0 1
1 1 0 0 1
1 0 1 1 0
```

stdout

```
0 1
1 1
2 1
3 1
4 1
1
```

stdin

```
5
0 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
```

stdout

```
0 0
1 0
2 0
3 0
4 0
0
```

How to find an answer

- memorization
- honest backtracking

Grasshopper

- The grasshopper is at point 0 and can jump forward 1 or 2 positions.
- We are given an array of integers where each number indicates the cost of landing on that position.
- It is necessary to determine the minimum cost to reach the end of the array.
- Moreover we need to recover this way.

Plan

- What do we consider a state of dynamics (e.g. dp[i])?
- What values of dp do we know?
- What is the order in which dp is counted?
- Which formula is used to calculate dp?
- What's the answer to our problem?

Honest backtracking

- We reconstruct the path from where the answer lies.
- Suppose at the current step we are in the state dp[i..j].
- On each step, we check all possible states from which we could have arrived here and take the best one.
- We continue this until we reach the base.

```
6.      vector<int> cost = {1, 2, 9, 4, 5, 0, 4, 11, 6};
7.      int n = cost.size();
8.      vector<int> dp(n + 1, INT_MAX);
9.
10.     dp[0] = 0;
11.     for (int i = 0; i < n; i++) {
12.         if (i + 1 < n) dp[i + 1] = min(dp[i + 1], dp[i] + cost[i]);
13.         if (i + 2 < n) dp[i + 2] = min(dp[i + 2], dp[i] + cost[i]);
14.     }
15.
16.     cout << "Minimum cost: " << dp[n - 1] << endl;
17.
```

```
    int i = n - 1;
    vector<int> path;
    while (i > 0) {
        path.push_back(i);
        if (i - 2 >= 0 && dp[i - 2] + cost[i - 2] <= dp[i - 1] + cost[i - 1]) i -= 2;
        else i -= 1;
    }
    path.push_back(0);
    reverse(path.begin(), path.end());

    for (int p : path) {
        cout << p << " -> ";
    }
    cout << "end" << endl;

    return 0;
}

```

Success #stdin #stdout 0.01s 5444KB

comments (0)

stdin

copy

```
Standard input is empty
```

stdout

copy

```
Minimum cost: 11
0 -> 1 -> 3 -> 5 -> 6 -> 8 -> end
```

```python
cost = [1, 2, 9, 4, 5, 0, 4, 11, 6]
n = len(cost)
dp = [float('inf')] * (n + 1)

dp[0] = 0
for i in range(n):
    if i + 1 < n:
        dp[i + 1] = min(dp[i + 1], dp[i] + cost[i])
    if i + 2 < n:
        dp[i + 2] = min(dp[i + 2], dp[i] + cost[i])

print("Minimum cost:", dp[n - 1])

i = n - 1
path = []
while i > 0:
    path.append(i)
    if i - 2 >= 0 and dp[i - 2] + cost[i - 2] <= dp[i - 1] + cost[i - 1]:
        i -= 2
    else:
        i -= 1
path.append(0)
path = path[::-1]

print(" -> ".join(map(str, path)) + " -> end")

```

Success #stdin #stdout 0.05s 9532KB

stdin

```
Standard input is empty
```

stdout

```
Minimum cost: 11
0 -> 1 -> 3 -> 5 -> 6 -> 8 -> end
```

Memorization

Now, we won't reconstruct the path using a loop, but instead, we'll remember where we came from directly within the dynamic programming.

```
    vector<int> cost = {1, 2, 9, 4, 5, 0, 4, 11, 6};
    int n = cost.size();
    vector<int> dp(n + 1, INT_MAX);
    vector<int> from(n, -1);

    dp[0] = 0;
    for (int i = 0; i < n; i++) {
        if (i + 1 < n && dp[i] + cost[i] < dp[i + 1]) {
            dp[i + 1] = dp[i] + cost[i];
            from[i + 1] = i;
        }
        if (i + 2 < n && dp[i] + cost[i] < dp[i + 2]) {
            dp[i + 2] = dp[i] + cost[i];
            from[i + 2] = i;
        }
    }

    cout << "Minimum cost: " << dp[n - 1] << endl;

```

```
    int i = n - 1;
    vector<int> path;
    while (i >= 0) {
        path.push_back(i);
        i = from[i];
    }
    reverse(path.begin(), path.end());

    for (int p : path) {
        cout << p << " -> ";
    }
    cout << "end" << endl;

    return 0;
}
```

Success #stdin #stdout 0.01s 5564KB

stdin

```
Standard input is empty
```

stdout

```
Minimum cost: 11
0 -> 1 -> 3 -> 5 -> 6 -> 8 -> end
```

```python
cost = [1, 2, 9, 4, 5, 0, 4, 11, 6]
n = len(cost)
dp = [float('inf')] * (n + 1)
from_ = [-1] * n

dp[0] = 0
for i in range(n):
    if i + 1 < n and dp[i] + cost[i] < dp[i + 1]:
        dp[i + 1] = dp[i] + cost[i]
        from_[i + 1] = i
    if i + 2 < n and dp[i] + cost[i] < dp[i + 2]:
        dp[i + 2] = dp[i] + cost[i]
        from_[i + 2] = i

print("Minimum cost:", dp[n - 1])

i = n - 1
path = []
while i >= 0:
    path.append(i)
    i = from_[i]
path = path[::-1]

print(" -> ".join(map(str, path)) + " -> end")

```

Success #stdin #stdout 0.03s 9608KB

stdin

```
Standard input is empty
```

stdout

```
Minimum cost: 11
0 -> 1 -> 3 -> 5 -> 6 -> 8 -> end
```

Solve together

There are n dishes, each with a tastiness of a_i. There's also a pleasure derived from consuming some pairs of dishes (i, j) - t_i_j. Your task is to determine which dishes should be eaten and in what order to achieve maximum sum of tastiness and pleasure.

Solve together

a = [1, 1, 2, 3]

t = [(1, 2, 3), (1, 3, 2), (2, 4, 2)]

answer = [1 2 4 3]

pleasure = 1 + 1 + 2 + 3 + 3 + 2 = 12

Plan

- dp[mask][dish] - max sum of tastiness with combination mask and last dish - dish.
- dp[0][any dish] = 0
- for 0 to (1 << n)
- dp[mask][i] = max(dp[mask ^ (1 << i)][j] + a[i] + t[i][j])
- max(dp[i][j]), for all i and j

Solve together

Given a set of n weights with masses m_1, …, m_n. Is it possible to distribute them onto two scales in such a way that they balance out?

Example

m = [1, 2, 3, 1, 2] - not possible

m = [1, 3, 2, 2] - possible [1, 3] and [2, 2]

Idea

m = [...]

sum = S

we need to check if we can get weight (S / 2) using things from m

Plan

- dp[i][j] - is it possible to get exactly weight i using only j first elements
- dp[0][0] = true
- from 0 to S / 2
- dp[i][j] = dp[i - a[j]][j - 1] | dp[i][j - 1]
- dp[S / 2][n]

Contrexample

1, 2, 3, 4, 5, 13

1, 13

2, 3, 4, 5

Solve together

Given a string composed of round, square, and curly brackets. Determine the minimum number of characters that need to be removed from this string so that the remaining characters form a valid bracket sequence.

Example

([)] - []

{([(]{)})] - [({})]

([[)]] - [[]], not ()

Plan

- dp[l][r] - answer on segment l, r
- dp[l][l] = 0
- len = 2..n, l = 1..n
- dp[l][r] = max(dp[l + 1][r], dp[l][r - 1], dp[l + 1][r - 1] + good_pair(s[l], s[r]) * 2)
- dp[0][n-1] - our answer

Solve together

https://projecteuler.net/problem=845

Let $D(n)$ be the n-th positive integer that has the sum of its digits a prime.
For example, $D(61) = 157$ and $D(10^8) = 403539364$.

Find $D(10^{16})$.

Plan

- dp[len][digit][sum] - number of ways to make a number with len length, last digit - digit and sum = sum
- dp[1][digit][digit] = 1
- regular one
- dp[i][digit][sum] = SUM(dp[i - 1][digit_2][sum - digit_2]) for digit_2
- not so easy

Assumptions

- 10^16 number with prime sum
- this number < 10^100
- prime - 2, length n - at least n^2 ways to put 1

  prime - 50, length - 100, at least 100^50 ways
- Let’s calculate all prime numbers to 100
- sum_dp[n] = sum(dp[n][digit][prime_i]) for all digit and prime

All codes

fib-rec(python) - https://ideone.com/3LzBco

fib-rec-opt(python) - https://ideone.com/3Pcx2M

fib(python) - https://ideone.com/3uns8s

fib-rec(c++) - https://ideone.com/Gn4O6N

fib-rec-opt(c++) - https://ideone.com/y6NHcJ

fib(c++) - https://ideone.com/E4hhIf

stairs(c++) - https://ideone.com/EHfJWx

stairs(python) - https://ideone.com/yV14E4

barrels(python) - https://ideone.com/CBEYa2

barrels(c++) - https://ideone.com/sEn9Jr

turtle(c++) - https://ideone.com/QWzEYh

turtle(python) - https://ideone.com/MYs2uS

knight(c++) - https://ideone.com/za8Hrz

knight(python) - https://ideone.com/h10Jh9

leven(python) - https://ideone.com/V4eS4O

leven(c++) - https://ideone.com/OHtuqC

fib_matrix(c++) - https://ideone.com/xZd8UK

barrels_matrix(c++) - https://ideone.com/zCYXnl

gcs(python) - https://ideone.com/brry2Z

All codes

gcs(python) - https://ideone.com/brry2Z

gcd(c++) - https://ideone.com/kt1Wqv

gis1(c++) - https://ideone.com/4rAjkW

gis1(python) - https://ideone.com/8cdNxn

gis3(python) - https://ideone.com/QL2oys

gis3(c++) - https://ideone.com/tEKBAS

gis nlogn(c++) - https://ideone.com/dVI2zL

gis nlogn(python) - https://ideone.com/cQ6j4M

atm(c++) - https://ideone.com/MjdWUN

atm(python) - https://ideone.com/RFVDj3

knapsack(c++) - https://ideone.com/Cx4aYL

knapsack(python) - https://ideone.com/lkZNng

dp - substrings(c++) - https://ideone.com/93Krqx

segments(python) - https://ideone.com/MntVrf

numbers(c++) - https://ideone.com/DUAmtZ

lazy recovery(c++) - https://ideone.com/RlcoGR

lazy recovery(python) - https://ideone.com/YSjORW

memorization(c++) - https://ideone.com/sxw5Mm

memorization(python) - https://ideone.com/6PXtpQ

That's All Folks!