M1821 - Contest Score
(Original Version by Percy Wong)

Alex Tung
alex20030190@yahoo.com.hk

24 March 2018

Subtask 1: $P = 1, F = 2$

- It is clear that the achievable scores are $0, 1, \ldots, 2K$.

Subtask 2: $K \leq 500$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.

Subtask 2: $K \leq 500$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.
- Generate all scores. There are $O(K^2)$ of them.

Subtask 2: $K \leq 500$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.
- Generate all scores. There are $O(K^2)$ of them.
- Answer each query in $O(log\ K)$. (Use binary search or STL set/map.)

Subtask 2: $K \leq 500$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.
- Generate all scores. There are $O(K^2)$ of them.
- Answer each query in $O(log\ K)$. (Use binary search or STL set/map.)
- Time complexity: $O((K^2 + N) log\ K)$.

Subtask 3: $S[i] \leq 10^7$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.

Subtask 3: $S[i] \leq 10^7$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.
- Declare a Boolean array $ok[0 \ldots 10^7]$.

Subtask 3: $S[i] \leq 10^7$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.
- Declare a Boolean array $ok[0 \ldots 10^7]$.
- Iterate x from 0 to K, but stop when ($x > 0$ and $Px \,\%\, F == 0$). Why? It is because of the following

Crucial Observation

To test whether a person can get score S, assume that the person has solved x problems, where x is the **smallest** nonnegative integer such that $S \equiv Px \pmod{F}$.

Subtask 3: $S[i] \leq 10^7$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.
- Declare a Boolean array $ok[0 \ldots 10^7]$.
- Iterate x from 0 to K, but stop when ($x > 0$ and $Px \,\%\, F == 0$). Why? It is because of the following

Crucial Observation

To test whether a person can get score S, assume that the person has solved x problems, where x is the **smallest** nonnegative integer such that $S \equiv Px \pmod F$.

- Iterate y from 0 to $K - x$. If $Px + Fy > 10^7$, break. Otherwise, set $ok[Px + Fy] = True$.

Subtask 3: $S[i] \leq 10^7$

- All achievable scores are of the form $Px + Fy$ for some nonnegative integers x, y, $x + y \leq K$.
- Declare a Boolean array $ok[0 \ldots 10^7]$.
- Iterate x from 0 to K, but stop when ($x > 0$ and $Px \,\%\, F == 0$). Why? It is because of the following

Crucial Observation

To test whether a person can get score S, assume that the person has solved x problems, where x is the **smallest** nonnegative integer such that $S \equiv Px \pmod{F}$.

- Iterate y from 0 to $K - x$. If $Px + Fy > 10^7$, break. Otherwise, set $ok[Px + Fy] = True$.
- Time complexity is $O(10^7)$ because each cell of $ok[]$ is visited at most once. Queries can be answered via simple lookup.

Subtask 4: $N \leq 50$

- Again we are trying to solve $S = Px + Fy$.

Subtask 4: $N \leq 50$

- Again we are trying to solve $S = Px + Fy$.
- Iterate x from 0 to K. The corresponding y can be found easily.

Subtask 4: $N \leq 50$

- Again we are trying to solve $S = Px + Fy$.
- Iterate x from 0 to K. The corresponding y can be found easily.
- Time complexity: $O(NK)$.

Subtask 4: $N \leq 50$

- Again we are trying to solve $S = Px + Fy$.
- Iterate x from 0 to K. The corresponding y can be found easily.
- Time complexity: $O(NK)$.
- This algorithm can solve subtasks 1, 2, 4!

Full solution 1

- Per observation, declare an array $x_{min}[0...(F-1)]$.

Full solution 1

- Per observation, declare an array $x_{min}[0...(F-1)]$.
- $x_{min}[i]$ should store the smallest nonnegative x such that $xP \equiv i \pmod{F}$, or -1 if there is no solution.

Full solution 1

- Per observation, declare an array $x_{min}[0...(F-1)]$.
- $x_{min}[i]$ should store the smallest nonnegative x such that $xP \equiv i \pmod{F}$, or -1 if there is no solution.
- By iterating x, $x_{min}[]$ can be built in linear time.

Full solution 1

- Per observation, declare an array $x_{min}[0...(F-1)]$.
- $x_{min}[i]$ should store the smallest nonnegative x such that $xP \equiv i \pmod{F}$, or -1 if there is no solution.
- By iterating x, $x_{min}[]$ can be built in linear time.
- With this array, each query can be answered in $O(1)$ time.

Full solution 1

- Per observation, declare an array $x_{min}[0...(F-1)]$.
- $x_{min}[i]$ should store the smallest nonnegative x such that $xP \equiv i \pmod{F}$, or -1 if there is no solution.
- By iterating x, $x_{min}[]$ can be built in linear time.
- With this array, each query can be answered in $O(1)$ time.
- If $x_{min}[S \% F] == -1$ or $x_{min}[S \% F] > K$, of course S is not achievable.

Full solution 1

- Per observation, declare an array $x_{min}[0...(F-1)]$.
- $x_{min}[i]$ should store the smallest nonnegative x such that $xP \equiv i \pmod{F}$, or -1 if there is no solution.
- By iterating x, $x_{min}[]$ can be built in linear time.
- With this array, each query can be answered in $O(1)$ time.
- If $x_{min}[S \% F] == -1$ or $x_{min}[S \% F] > K$, of course S is not achievable.
- Otherwise, find out the corresponding y and check if it is valid.

Full solution 1

- Per observation, declare an array $x_{min}[0...(F-1)]$.
- $x_{min}[i]$ should store the smallest nonnegative x such that $xP \equiv i \pmod{F}$, or -1 if there is no solution.
- By iterating x, $x_{min}[]$ can be built in linear time.
- With this array, each query can be answered in $O(1)$ time.
- If $x_{min}[S \% F] == -1$ or $x_{min}[S \% F] > K$, of course S is not achievable.
- Otherwise, find out the corresponding y and check if it is valid.
- Time complexity: $O(N+F)$.

Full solution 2

- Use *extended Euclidean algorithm* (see Maths (I)) to find **one** pair of solution (x, y) to $Px + Fy = S$.

Full solution 2

- Use *extended Euclidean algorithm* (see Maths (I)) to find **one** pair of solution (x, y) to $Px + Fy = S$.
- Let $P' := \frac{P}{gcd(P,F)}$ and $F' := \frac{F}{gcd(P,F)}$.

Full solution 2

- Use *extended Euclidean algorithm* (see Maths (I)) to find **one** pair of solution (x, y) to $Px + Fy = S$.
- Let $P' := \frac{P}{gcd(P,F)}$ and $F' := \frac{F}{gcd(P,F)}$.
- Shift solution (x, y) to $(x + cF', y - cP')$. We need a suitable c such that $x + cF' \geq 0$ and takes minimum value.

Full solution 2

- Use *extended Euclidean algorithm* (see Maths (I)) to find **one** pair of solution (x, y) to $Px + Fy = S$.
- Let $P' := \frac{P}{gcd(P,F)}$ and $F' := \frac{F}{gcd(P,F)}$.
- Shift solution (x, y) to $(x + cF', y - cP')$. We need a suitable c such that $x + cF' \geq 0$ and takes minimum value.
- Time complexity: $O(N\ log\ F)$.

M1822 Power Tower

2018-3-24

Problem statement

- Given a sequence $a_1, a_2, \ldots, a_n$.
- You need to find ${a_1}^{{a_2}^{\cdots^{a_n}}} \; mod \; m$

- $1 \leq m \leq 10^9$
- $1 \leq n \leq 10^5$
- $1 \leq a_i \leq 10^9$ for all i

Subtask 1

- n = 2

- It is just the same as 20374 Big Mod.

- One can use divide and conquer to solve in O(log a_2)

Subtask 1

- Possible solution:

```
long long big_mod(long long x, long long y, long long m) {
        if (y == 0) return 1 % m;
        if (y % 2 == 0) return big_mod(x * x % m, y / 2, m);
        else return x * big_mod(x, y – 1, m) % m;
}
```

Subtask 2

- m = 10

- Observation 1:
  - for a_1, we only need to consider its ones digit

Subtask 2

x % 10	x^1 % 10	x^2 % 10	x^3 % 10	x^4 % 10	x^… % 10
0	0	0	0	0	0,0,0,0…
1	1	1	1	1	1,1,1,1…
2	2	4	8	6	2,4,8,6…
3	3	9	7	1	3,9,7,1…
4	4	6	4	6	4,6,4,6…
5	5	5	5	5	5,5,5,5…
6	6	6	6	6	6,6,6,6…
7	7	9	3	1	7,9,3,1…
8	8	4	2	6	8,4,2,6…
9	9	1	9	1	9,1,9,1…

Subtask 2

- From the previous table, we can see that we only need to know $a_2^{a_3^{\cdots^{a_n}}} \bmod 4$

- Observation 2:
  - for a_2, we only need to consider its remainder when divided 4

Subtask 2

x % 4	x^1 % 4	x^2 % 4	x^... % 4
0	0	0	0,0...
1	1	1	1,1...
2	2	0	0,0...
3	3	1	3,1...

Subtask 2

- From the previous table, we can see that we only need to know $a_3^{a_4^{\cdots^{a_n}}} mod\ 2$

- Observation 3:
  - $a_3^{a_4^{\cdots^{a_n}}}$ is odd if a_3 is odd
  - $a_3^{a_4^{\cdots^{a_n}}}$ is even if a_3 is even

Subtask 2

- So if we found $b_1 = a_1 \% 10$, $b_2 = a_2 \% 4$ and $b_3 = a_3 \% 2$
- Then ${a_1}^{{a_2}^{\cdots^{a_n}}} mod\ 10 = {b_1}^{{b_2}^{b_3}} mod\ 10$ which can be evaluated easily
  - Special case: $b_2 = 2$ then check if $a_3 \geq 2$
    - True -> answer = b_1
    - False -> answer = ${b_1}^2 \% 10$

Subtask 3, 4

- Subtask 3: $m \leq 32768$
  - If you forget to use long long or cannot calculate the prime factorization fast, then you can still pass this subtask
- Subtask 4:$a_i \geq 3$ for all i
  - If you for get to handle some special case, you can still pass this subtask

Subtask 5

- No additional constraints

- From subtask 2, we found that
  - the value of x^1 % 10, x^2 % 10, … repeats
  - The value of x^1 % 4, x^2 % 4, … repeats
  - The value of x^1 % 2, x^2 % 2, … repeats
- Is this true for every m?

Subtask 5

- Euler's totient theorem
- If $\gcd(a_i, m) = 1$
  - Then $a_i^{\varphi(m)} \equiv 1 (mod\ m)$

- In other words, the value of a_i^1 mod m… will repeat after $\varphi(m)$ times if $\gcd(a_i, m) = 1$

- How if $\gcd(a_i, m) \neq 1$?

Subtask 5

- Suppose the power is large enough (actually, 30 is large enough)
- Then we can do the following:

```
n = m;
while(gcd(a[i], n) > 1)
    n /= gcd(a[i], n);
```

- Then a[i]^1 % n,… will repeat after $\varphi(n)$ times
- So a[i]^a[i+1]^… % n = a[i]^(a[i+1]^… % $\varphi(n)$) % n
  - We can run the big mod algorithm once we know a[i+1]^… % $\varphi(n)$
- Also a[i]^a[i+1]^… % (n/m) = 0
- How can we calculate a[i]^a[i+1]^… % m?

Subtask 5

- Chinese Remainder Theorem
- If $x \equiv x_1 (mod\ m_1)$ and $x \equiv x_2 (mod\ m_2)$ and $gcd(m_1, m_2) = 1$
- Then $x \equiv x_1 m_2 n_2 + x_2 m_1 n_1 (mod\ m_1 m_2)$
  - Where $n_1 \equiv {m_1}^{-1} (mod\ m_2)$, $n_2 \equiv {m_2}^{-1} (mod\ m_1)$

- So we can evaluate a[i]^a[i+1]^… % m using the above formula

- Done?

Subtask 5

- NO!!!!
- **Suppose the power is large enough**
  - Why do we need this?
  - a[i]^a[i+1]^… % (n/m) = 0 <= not necessarily true if a[i+1]^… is small
  - Eg: 2^2^2 mod 32768 = 16 != 0

- So we need to compress the power tower so that it contains no ones and the power of last two element >30 or n = 1
  - Eg: 2^2^2 -> 16, 2^2^2^2 -> 2^16
- Why 30 is large enough?
  - Left as an exercise

Subtask 5

- Is this fast enough?
- Big mod part:
  - O(log a_i) every time
  - O(n) times
  - O(n log a_i) overall
- Chinese remainder part:
  - O(log m) every time (calculate inverse using big mod)
  - O(n) times
  - O(n log m) overall
- Calculating $\varphi(n)$ part:
  - O(sqrt m) every time
  - O(n) times ???

Subtask 5

- Calculating $\varphi(n)$ part:
  - O($\sqrt{m}$) every time
  - The number m will be halved after at most two iterations
    - If m is a odd prime then
      - m $\rightarrow \varphi(m) = m - 1 \rightarrow \varphi(m-1) \leq \frac{m-1}{2}$
  - So overall = O($2\left(\sqrt{m} + \sqrt{\frac{m}{2}} + \sqrt{\frac{m}{4}} + \cdots\right)$) = O($\sqrt{m}$)

- So total time complexity is O($\sqrt{m} + n\log a_i + n\log m$)

Wet Corridor

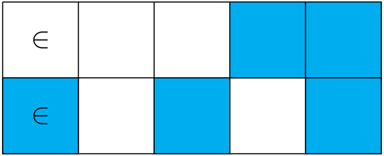
solution

Lau Chi Yung

2018/03/24

Problem

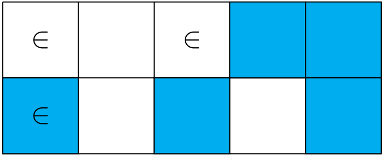
"∈" is bird foot print



- #steps $= 0$
- left wetness $= 0$
- right wetness $= 1$
- total time $= 0 + \max(0, 1) = 1$ second
- Goal: minimize total time

Problem

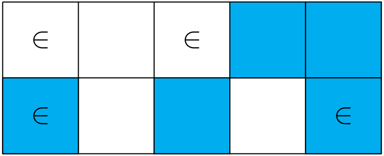
"$\in$" is bird foot print



- #steps $= 1$
- left wetness $= 0$
- right wetness $= 1$
- total time $= 1 + \max(0, 1) = 2$ seconds
- Goal: minimize total time

Problem

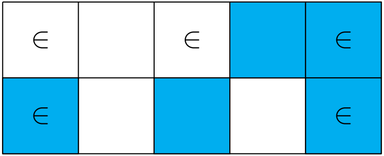
"$\in$" is bird foot print



- #steps $= 2$
- left wetness $= 0$
- right wetness $= 2$
- total time $= 2 + \max(0, 2) = 4$ seconds
- Goal: minimize total time

Problem

"∈" is bird foot print



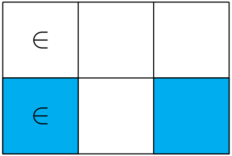
- #steps = 3
- left wetness = 1
- right wetness = 2
- total time = 3 + max(1, 2) = 5 seconds
- Goal: minimize total time

Subtask 1

Denotation:

$$wet_{r,c} = \begin{cases} 1 & \text{if tile } (r, c) \text{ is wet} \\ 0 & \text{otherwise} \end{cases}$$

Consider when $N \leq 3$.



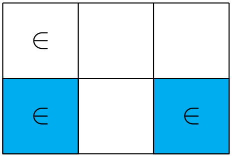
- #steps = 0
- right wetness = 1
- total time = 1 second

Subtask 1

Denotation:

$$wet_{r,c} = \begin{cases} 1 & \text{if tile } (r, c) \text{ is wet} \\ 0 & \text{otherwise} \end{cases}$$

Consider when $N \leq 3$.



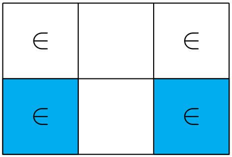
- #steps $= 1$
- right wetness $= 2$
- total time $= 3$ seconds

Subtask 1

Denotation:

$$wet_{r,c} = \begin{cases} 1 & \text{if tile } (r, c) \text{ is wet} \\ 0 & \text{otherwise} \end{cases}$$

Consider when $N \leq 3$.



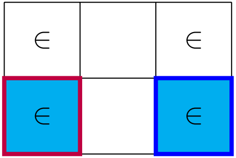
- #steps = 2
- right wetness = 2
- total time = 4 seconds

Subtask 1

Denotation:

$$wet_{r,c} = \begin{cases} 1 & \text{if tile } (r, c) \text{ is wet} \\ 0 & \text{otherwise} \end{cases}$$

Consider when $N \leq 3$.



- #steps is always 2
- right wetness $= wet_{2,1} + wet_{2,N}$
- total time $= 2 + wet_{2,1} + wet_{2,N}$

Subtask 1

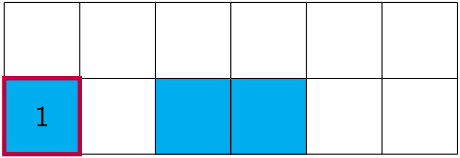
Consider when $N \geq 4$.

dp$[i]$ $=$ minimum sum of $\#$steps and right wetness to step right foot on tile $(2, i)$

Subtask 1

Base cases:

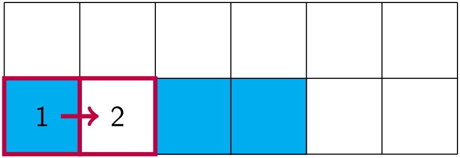
- dp$[1] = wet_{2,1}$



Subtask 1

Base cases:

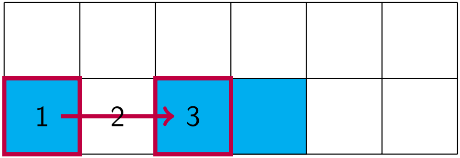
- dp$[1] = wet_{2,1}$
- dp$[2] = wet_{2,1} + wet_{2,2} + 1$



Subtask 1

Base cases:

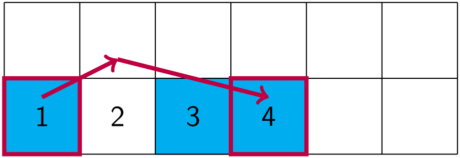
- $\mathsf{dp}[1] = wet_{2,1}$
- $\mathsf{dp}[2] = wet_{2,1} + wet_{2,2} + 1$
- $\mathsf{dp}[3] = wet_{2,1} + wet_{2,3} + 1$



Subtask 1

Base cases:

- dp$[1] = wet_{2,1}$
- dp$[2] = wet_{2,1} + wet_{2,2} + 1$
- dp$[3] = wet_{2,1} + wet_{2,3} + 1$
- dp$[4] = wet_{2,1} + wet_{2,4} + 2$



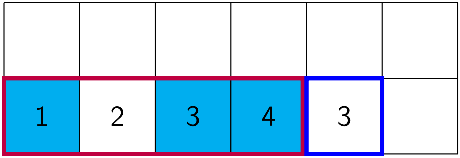
Subtask 1

Base cases:

- $\text{dp}[1] = wet_{2,1}$
- $\text{dp}[2] = wet_{2,1} + wet_{2,2} + 1$
- $\text{dp}[3] = wet_{2,1} + wet_{2,3} + 1$
- $\text{dp}[4] = wet_{2,1} + wet_{2,4} + 2$

Recurrence:

- $\text{dp}[i] = \min_{j \in [i-4, i-1]} \{\text{dp}[j]\} + wet_{2,i} + 2$



Subtask 1

Base cases:

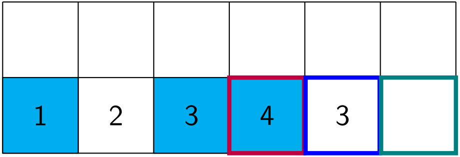
- dp$[1] = wet_{2,1}$
- dp$[2] = wet_{2,1} + wet_{2,2} + 1$
- dp$[3] = wet_{2,1} + wet_{2,3} + 1$
- dp$[4] = wet_{2,1} + wet_{2,4} + 2$

Recurrence:

- dp$[i] = \min_{j \in [i-4, i-1]} \{\text{dp}[j]\} + wet_{2,i} + 2$

Answer:

- $\min\left(\min(\text{dp}[N-2], \text{dp}[N-1]) + wet_{2,N} + 2, \text{dp}[N] + 1\right)$



Subtask 1

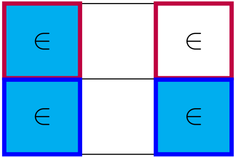
Time complexity:

- $O(N)$

Subtask 2

Consider when $N \leq 3$.

Answer: $\max(wet_{1,1} + wet_{1,N}, wet_{2,1} + wet_{2,N}) + 2$



Subtask 2

Consider when $N \geq 4$.

dp$[1][i][wl][wr]$ = minimum #steps to step left foot on $(1, i)$ with left and right wetness being wl and wr

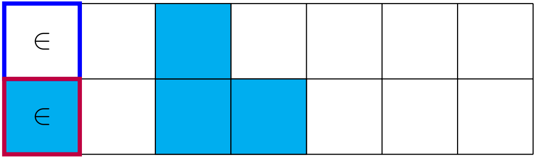
dp$[2][i][wl][wr]$ = minimum #steps to step right foot on $(2, i)$ with left and right wetness being wl and wr

Derivation of dp$[1][i][wl][wr]$ will be omitted because it is similar to dp$[2][i][wl][wr]$

Subtask 2

Base cases:

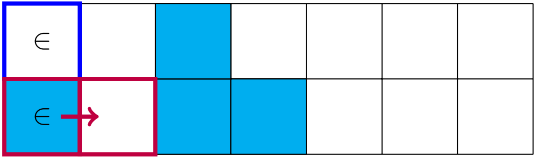
- dp$[2][1][wet_{1,1}][wet_{2,1}] = 0$



Subtask 2

Base cases:

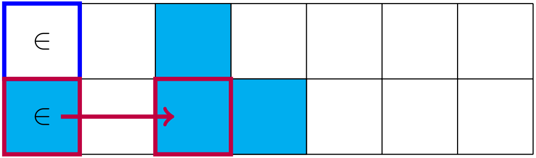
- dp$[2][1][wet_{1,1}][wet_{2,1}] = 0$
- dp$[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$



Subtask 2

Base cases:

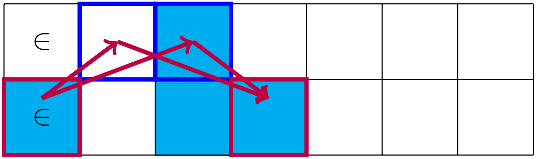
- dp$[2][1][wet_{1,1}][wet_{2,1}] = 0$
- dp$[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$
- dp$[2][3][wet_{1,1}][wet_{2,1} + wet_{2,3}] = 1$



Subtask 2

Base cases:

- dp$[2][1][wet_{1,1}][wet_{2,1}] = 0$
- dp$[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$
- dp$[2][3][wet_{1,1}][wet_{2,1} + wet_{2,3}] = 1$
- dp$[2][4][wet_{1,1} + \min(wet_{1,2}, wet_{1,3})][wet_{2,1} + wet_{2,4}] = 2$



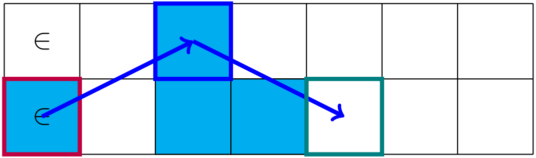
Subtask 2

Base cases:

- $\mathsf{dp}[2][1][wet_{1,1}][wet_{2,1}] = 0$
- $\mathsf{dp}[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$
- $\mathsf{dp}[2][3][wet_{1,1}][wet_{2,1} + wet_{2,3}] = 1$
- $\mathsf{dp}[2][4][wet_{1,1} + \min(wet_{1,2}, wet_{1,3})][wet_{2,1} + wet_{2,4}] = 2$

Recurrence:

- $\mathsf{dp}[2][i][wl][wr]$
  $$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \mathsf{dp}[2][j][wl - wet_{1,k}][wr - wet_{2,i}] \right\} + 2$$



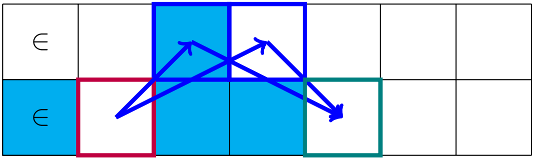
Subtask 2

Base cases:

- $\mathsf{dp}[2][1][wet_{1,1}][wet_{2,1}] = 0$
- $\mathsf{dp}[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$
- $\mathsf{dp}[2][3][wet_{1,1}][wet_{2,1} + wet_{2,3}] = 1$
- $\mathsf{dp}[2][4][wet_{1,1} + \min(wet_{1,2}, wet_{1,3})][wet_{2,1} + wet_{2,4}] = 2$

Recurrence:

- $\mathsf{dp}[2][i][wl][wr]$
  $= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \mathsf{dp}[2][j][wl - wet_{1,k}][wr - wet_{2,i}] \right\} + 2$



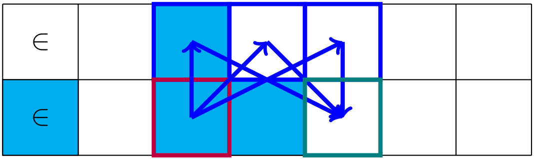
Subtask 2

Base cases:

- dp$[2][1][wet_{1,1}][wet_{2,1}] = 0$
- dp$[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$
- dp$[2][3][wet_{1,1}][wet_{2,1} + wet_{2,3}] = 1$
- dp$[2][4][wet_{1,1} + \min(wet_{1,2}, wet_{1,3})][wet_{2,1} + wet_{2,4}] = 2$

Recurrence:

- dp$[2][i][wl][wr]$
  $$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \text{dp}[2][j][wl - wet_{1,k}][wr - wet_{2,i}] \right\} + 2$$



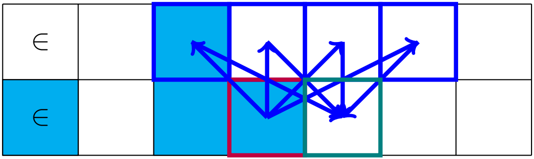
Subtask 2

Base cases:

- $\mathsf{dp}[2][1][wet_{1,1}][wet_{2,1}] = 0$
- $\mathsf{dp}[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$
- $\mathsf{dp}[2][3][wet_{1,1}][wet_{2,1} + wet_{2,3}] = 1$
- $\mathsf{dp}[2][4][wet_{1,1} + \min(wet_{1,2}, wet_{1,3})][wet_{2,1} + wet_{2,4}] = 2$

Recurrence:

- $\mathsf{dp}[2][i][wl][wr]$
  $$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \mathsf{dp}[2][j][wl - wet_{1,k}][wr - wet_{2,i}] \right\} + 2$$



Subtask 2

Base cases:

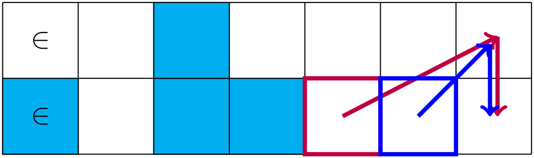
- dp$[2][1][wet_{1,1}][wet_{2,1}] = 0$
- dp$[2][2][wet_{1,1}][wet_{2,1} + wet_{2,2}] = 1$
- dp$[2][3][wet_{1,1}][wet_{2,1} + wet_{2,3}] = 1$
- dp$[2][4][wet_{1,1} + \min(wet_{1,2}, wet_{1,3})][wet_{2,1} + wet_{2,4}] = 2$

Recurrence:

- dp$[2][i][wl][wr]$
  $$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \text{dp}[2][j][wl - wet_{1,k}][wr - wet_{2,i}] \right\} + 2$$

Answer: minimum of

- dp$[2][N-2][wl][wr] + \max(wl + wet_{1,N}, wr + wet_{2,N}) + 2$
- dp$[2][N-1][wl][wr] + \max(wl + wet_{1,N}, wr + wet_{2,N}) + 2$



Subtask 2

Time complexity:

- $1 \leq i \leq N$
- $0 \leq wl \leq N$
- $0 \leq wr \leq N$
- $O(2 \times N \times N \times N) = O(N^3)$

Subtask 3

Just tune the limits in Subtask 2

- $1 \leq i \leq N$
- $0 \leq wl, wr \leq 40$

Subtask 4

Consider when $N \geq 4$.

- In subtask 2, we considered all possible combinations of wl and wr
- Observe that given $m = \min(wl, wr)$ and $d = wl - wr$, we can reconstruct wl and wr
- Observe that m will never decrease as Alice walks

Subtask 4

Consider when $N \geq 4$.

- In subtask 2, we considered all possible combinations of wl and wr
- Observe that given $m = \min(wl, wr)$ and $d = wl - wr$, we can reconstruct wl and wr
- Observe that m will never decrease as Alice walks

- Now for each DP state, rather than storing $\#steps$, we store $\#steps + m$
- Rather than considering all combinations of wl and wr, we only consider all possible values of d

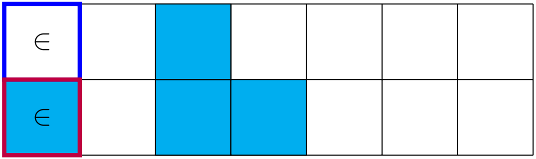
dp$[2][i][d]$ = minimum $\#steps + m$ to step right foot on $(2, i)$

Derivation dp$[1][i][d]$ will be omitted because it is similar to dp$[2][i][d]$

Subtask 4

Base cases:

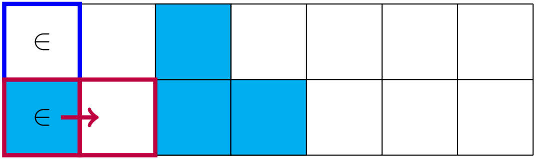
- dp$[2][1][wet_{1,1} - wet_{2,1}]$
  $= \min(wet_{1,1}, wet_{2,1})$



Subtask 4

Base cases:

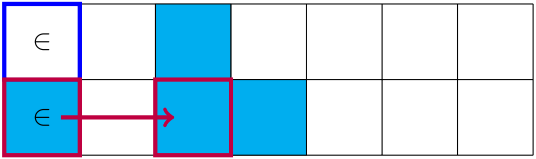
- dp$[2][1][wet_{1,1} - wet_{2,1}]$
  $= \min(wet_{1,1}, wet_{2,1})$
- dp$[2][2][wet_{1,1} - wet_{2,1} - wet_{2,2}]$
  $= \min(wet_{1,1}, wet_{2,1} + wet_{2,2}) + 1$



Subtask 4

Base cases:

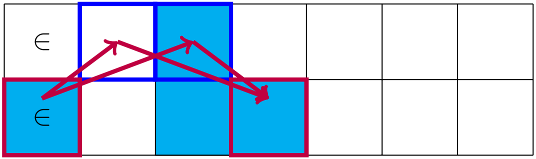
- dp[2][1][$wet_{1,1} - wet_{2,1}$]
  $= \min(wet_{1,1}, wet_{2,1})$
- dp[2][2][$wet_{1,1} - wet_{2,1} - wet_{2,2}$]
  $= \min(wet_{1,1}, wet_{2,1} + wet_{2,2}) + 1$
- dp[2][3][$wet_{1,1} - wet_{2,1} - wet_{2,3}$]
  $= \min(wet_{1,1}, wet_{2,1} + wet_{2,3}) + 1$



Subtask 4

Base cases:

- dp[2][1][$wet_{1,1} - wet_{2,1}$]
  $= \min(wet_{1,1}, wet_{2,1})$
- dp[2][2][$wet_{1,1} - wet_{2,1} - wet_{2,2}$]
  $= \min(wet_{1,1}, wet_{2,1} + wet_{2,2}) + 1$
- dp[2][3][$wet_{1,1} - wet_{2,1} - wet_{2,3}$]
  $= \min(wet_{1,1}, wet_{2,1} + wet_{2,3}) + 1$
- dp[2][4][$wet_{1,1} + \min(wet_{1,2}, wet_{1,3}) - wet_{2,1} - wet_{2,4}$]
  $= \min(wet_{1,1} + \min(wet_{1,2}, wet_{1,3}), wet_{2,1} + wet_{2,4}) + 2$

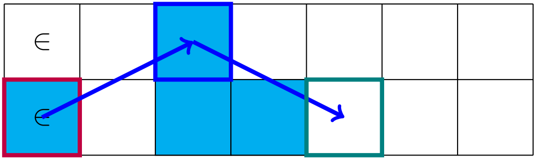


Subtask 4

Recurrence:

- dp$[2][i][d]$

$$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \begin{array}{l} \text{dp}[2][j][d + wet_{2,i} - wet_{1,k}] \\ + \max(0, |d + wet_{2,i} - wet_{1,k}| - |d + wet_{2,i}|) \\ + \max(0, |d + wet_{2,i}| - |d|) + 2 \end{array} \right\}$$

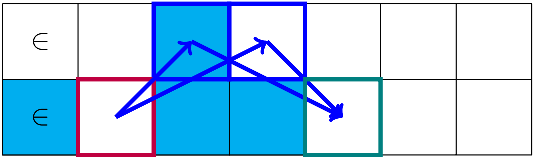


Subtask 4

Recurrence:

- $\mathsf{dp}[2][i][d]$

$$= \min_{\substack{j\in[i-4,i-1] \\ k\in[i-2,j+2]}} \left\{ \begin{array}{l} \mathsf{dp}[2][j][d + wet_{2,i} - wet_{1,k}] \\ +\max(0, |d + wet_{2,i} - wet_{1,k}| - |d + wet_{2,i}|) \\ +\max(0, |d + wet_{2,i}| - |d|) + 2 \end{array} \right\}$$

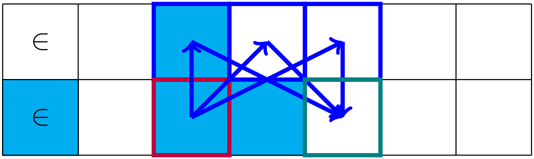


Subtask 4

Recurrence:

- $\text{dp}[2][i][d]$

$$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \begin{array}{l} \text{dp}[2][j][d + wet_{2,i} - wet_{1,k}] \\ + \max(0, |d + wet_{2,i} - wet_{1,k}| - |d + wet_{2,i}|) \\ + \max(0, |d + wet_{2,i}| - |d|) + 2 \end{array} \right\}$$

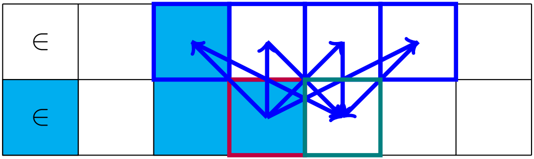


Subtask 4

Recurrence:

- dp$[2][i][d]$

$$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \begin{array}{l} \text{dp}[2][j][d + wet_{2,i} - wet_{1,k}] \\ + \max(0, |d + wet_{2,i} - wet_{1,k}| - |d + wet_{2,i}|) \\ + \max(0, |d + wet_{2,i}| - |d|) + 2 \end{array} \right\}$$



Subtask 4

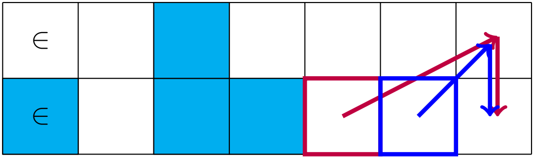
Recurrence:

- $\mathrm{dp}[2][i][d]$

$$= \min_{\substack{j \in [i-4, i-1] \\ k \in [i-2, j+2]}} \left\{ \begin{array}{l} \mathrm{dp}[2][j][d + wet_{2,i} - wet_{1,k}] \\ + \max(0, |d + wet_{2,i} - wet_{1,k}| - |d + wet_{2,i}|) \\ + \max(0, |d + wet_{2,i}| - |d|) + 2 \end{array} \right\}$$

Answer:

- $\min(\mathrm{dp}[2][N-2][d], \mathrm{dp}[2][N-1][d])$
  $+ \max(0, |d| - |d + wet_{1,N}|)$
  $+ \max(0, |d + wet_{1,N}| - |d + wet_{1,N} - wet_{2,N}|)$
  $+ |d + wet_{1,N} - wet_{2,N}| + 2$

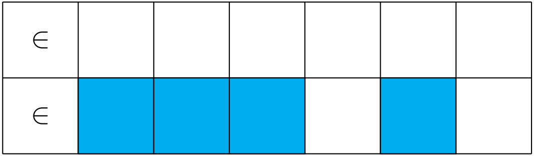


Subtask 4

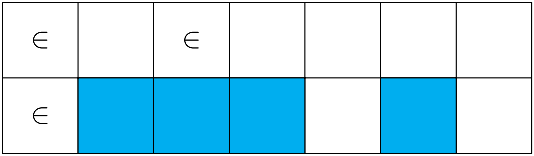
Time complexity:

- $1 \leq i \leq N$
- $0 \leq |d| \leq N$
- $O(2 \times N \times N) = O(N^2)$

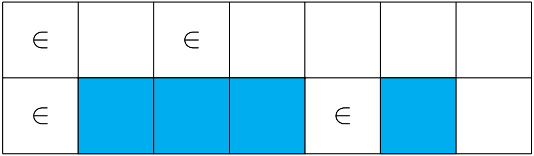
Tricky cases



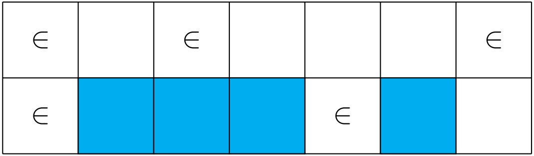
Tricky cases



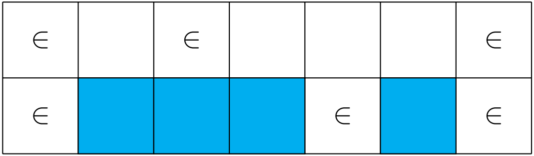
Tricky cases



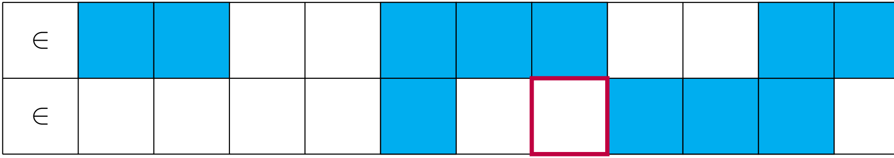
Tricky cases



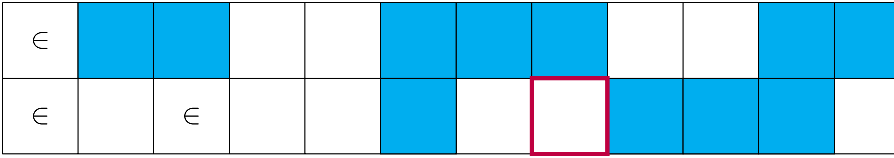
Tricky cases



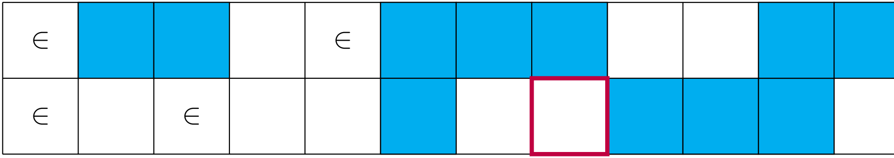
Tricky cases



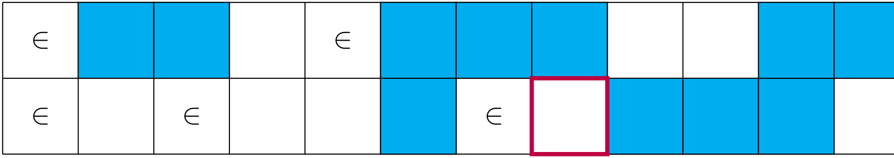
Tricky cases



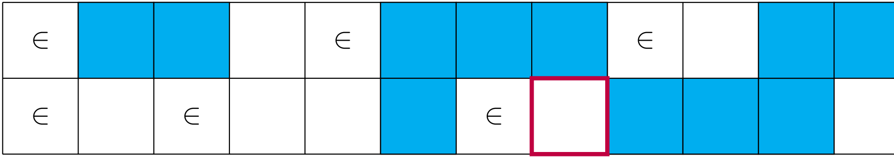
Tricky cases



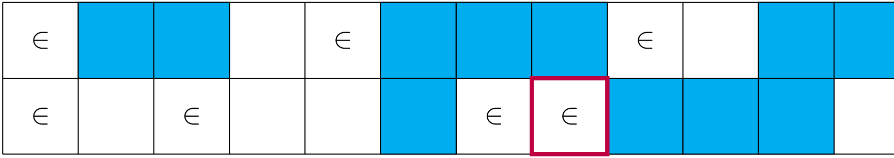
Tricky cases



Tricky cases



Tricky cases



M1824 - Internal Network

Percy Wong {percywtc}

Background

Problem Idea By - kctung

Prepared By - percywtc

Subtask 1 - Exhaustion

Exhaust which node to be upgraded (**2^N** combinations)

For remaining nodes run any MST algorithm,
and find shortest edge to connect any of the upgraded node

Be careful situation that with remaining nodes cannot be connected

Time Complexity: $O(2^N M + M \log M)$
Expected Score: 22/100

Subtask 2

Note that $c_i=10^9$, very large compared to $w_i \le 10^4$

We should perform least number of "upgrades"

Perform any fast enough MST algorithm for each connected component

If it forms a tree, that's the answer
Otherwise, it forms a forest of K trees, we have to "upgrade" a node from each component to make them connected

Time Complexity: O(MlogM + N)
Expected Score: 26/100

Subtask 3

If $N = 2$ with one edge connecting them, compare $c_1 + c_2$ and w_1
Otherwise, we must perform upgrades

For each node without edges connecting them, simply add c_i to the answer

For each edge (u, v, w), compare $c_u + c_v$ and $w + \min(c_u, c_v)$

- $c_u + c_v$ smaller means upgrading both to connect to network
- $w + \min(c_u, c_v)$ smaller means connect that cable and upgrade one office

Time Complexity: $O(N + M)$
Expected Score: 20/100

Full Solution

Let's first assume no upgrades required,

the answer is simply MST (if the whole graph can be connected)

Otherwise, we can consider "upgrades" as building portals,

which can teleport to any other "upgraded office"

We can imagine the portals will first get you to some "unknown space",

and then back to another office

To build a portal to the "unknown space" costs c_i

Full Solution

Therefore we can transform the graph by:

adding extra edges (0, i) with cost c_i (node 0 is the “unknown space”)

Then we can simply run any fast enough MST algorithm

Time Complexity: O((M+N)log(M+N))
Expected Score: 100/100

or slow MST algorithm with time complexity $O(VE+E^2)$...

Time Complexity: $O((M+N)^2)$
Expected Score: 37/100