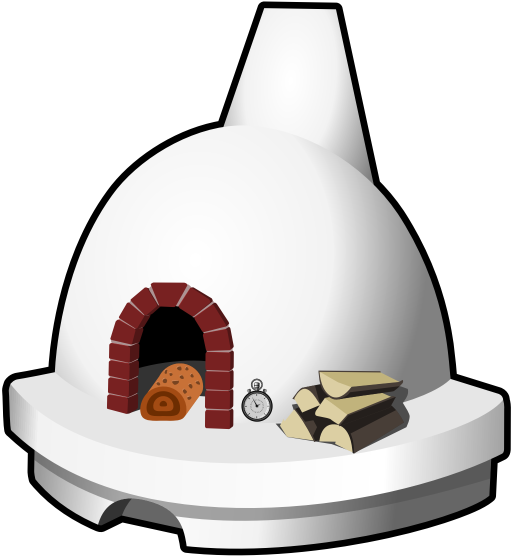
Problem Analysis Session

SWERC judges

November 30, 2017

A - Cakey McCakeFace



A - Cakey McCakeFace

Algorithm

Iterate over the two sets and count the occurences of the differences with a hash map.

Complexity

$O(n^2)$ (time and space)

Python Solution

```
def solve(A, B):  # A, B are list(int).
    C = collections.Counter(b - a for b in B for a in A
                            if b - a >= 0)
    occ, negative_offset = max(((C[k], -k) for k in C),
                               default=(0,0))
    return -negative_offset
```

A - Cakey McCakeFace

Other algorithm in $O(n^2 \log(n))$

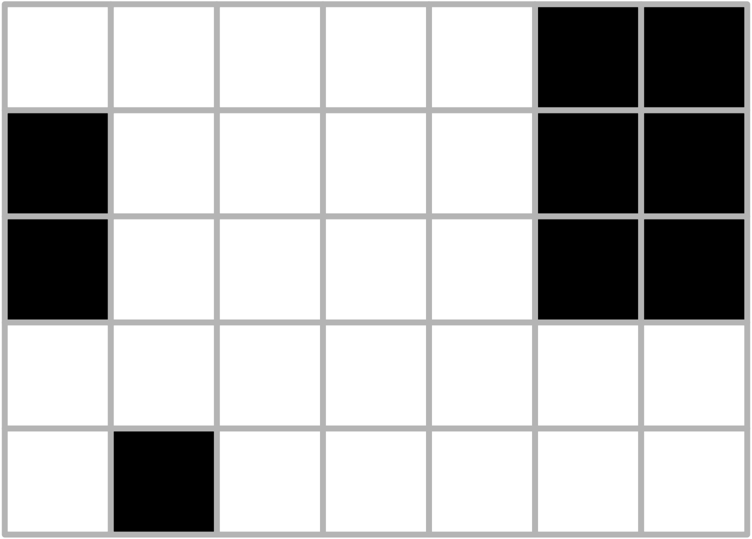
- Maintain a heap containing, for each elemt of the second set, the smallest time shift for matching an element of the first set.
- Iterate over time shifts stored in the heap, update the heap as we go.

$O(n^2 \log(n))$ in time, $O(n)$ in space.

Running time

- In practice, $\sim 5x$ faster than $O(n^2)$ solution naively implemented until memory becomes an issue.
- Cause: CPU stalled on main memory latency (a few tens of ns).

B - Table

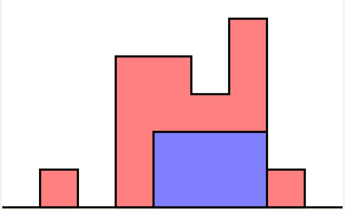


□	□	□	□	□	■	■
■	□	□	□	□	■	■
■	□	□	□	□	■	■
□	□	□	□	□	□	□
□	■	□	□	□	□	□

B - Table

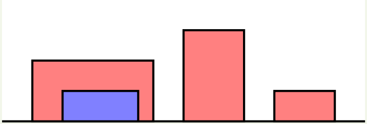
First simplifications

- Lots of ornaments: this is **just a bitmap**.
- Lots of queries ⇒ We **compute all the results**.
- Fix the low y coordinate of counted rectangles, then accumlate.



Simplified version

The **free** area contains only **separated rectangles**:

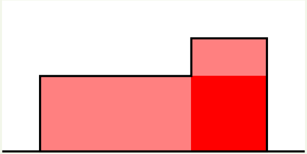


Cumulative array

- $+1$ at the size of every red rectangle
- Sum twice on x, once on y

B - Table

What if rectangles intersect?



Count -1 for intersection

Solution of the full problem in $O(X \times Y + D)$

- Enumerate all the maximal free rectangles
  - Use classical algorithm: “largest rectangle of zeros”
- Use a cumulative array
  - Count $+1$ for each maximal rectangle, and intersections negatively

C - Macarons

C - Macarons

The problem

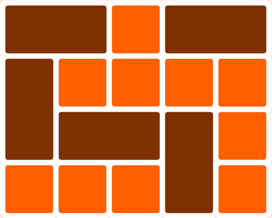
tiling a $N \times M$ grid with monominos and dominos

Homage

to Pierre Hermé, of course

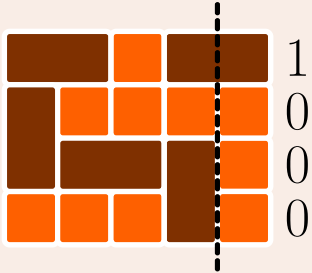
Example

one of the 120,465 solutions for $N = 4$ and $M = 5$



C - Macarons

Transition



Transition matrix

$T[i][j]$ is the number of columns with left mask i and right mask j

Solution

the number of path of length M from 0 to 0, that is

$$T^M[0][0]$$

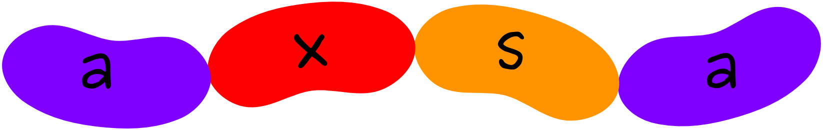
Algorithmic techniques

- Fast exponentiation
- Matrix multiplication
- Modulo arithmetic

Complexity

- matrix has size $2^N \times 2^N$
- one multiplication costs $(2^N)^3$
- overall complexity is $(2^N)^3 \times \log(M)$

D - Candy Chain



D - Candy Chain

Key idea

Dynamic programming: Compute $\mathbf{F(i, j, require\_full\_consumption, p, k)}$, the maximum score of selling the Candy Chain range $[\mathbf{i}, \mathbf{j})$ given:

- Prefix $[\mathbf{0}, \mathbf{k})$ of child's portion $\mathbf{p}$ was already produced from prefix $[\mathbf{0}, \mathbf{i})$ of the Candy Chain.
- Full consumption of range $[\mathbf{i}, \mathbf{j})$ is required depending on **require_full_consumption** (boolean).

D - Candy Chain

Computing **F**

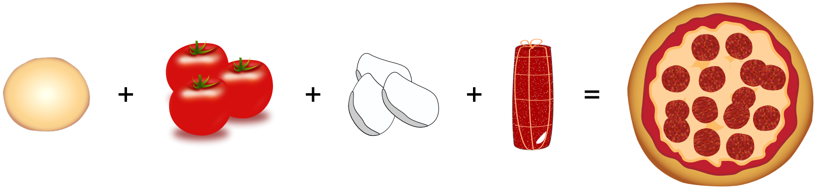
At state $\mathbf{i}, \mathbf{j}, \mathbf{require\_full\_consumption}, \mathbf{p}, \mathbf{k}$ we can:

- Make immediate progress on the current child portion **p** (if $\mathbf{candy\_chain[i]} == \mathbf{portions[p][k]}$) using $\mathbf{F(i+1, j, require\_full\_consumption, p, k+1)}$
- For $\mathbf{m} \in [\mathbf{i+1}, \mathbf{j}]$, try to skip $\mathbf{candy\_chain[i..m)}$ for the current child portion:
  - Maximize score for the skipped range $[\mathbf{i}, \mathbf{m})$ using: $\mathbf{F(i, m, -1, true)}$ (require full consumption of this range, no child portion already consumed)
  - Continue current child portion **p** after the skipped range with: $\mathbf{F(m, j, p, require\_full\_consumption)}$

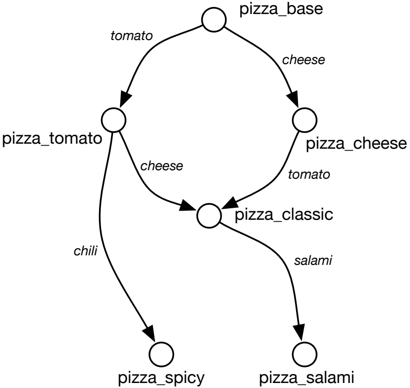
Complexity

$O(N^4 \times W)$ in time, $O(N^3 \times W)$ in space.

E - Ingredients



E – Ingredients

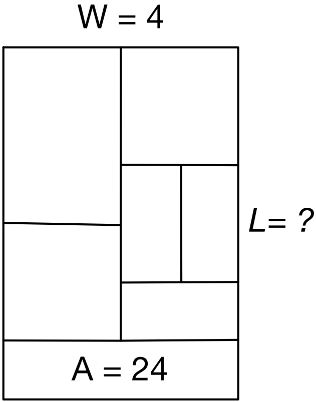


The solution combines *shortest paths* and *0/1 knapsack* algorithms:

1. the recipes form a DAG: compute first the topological sort of the recipe graph, and then compute in $O(N)$ time the dish costs;
2. dynamic program for the knapsack problem in $O(NB)$, using the costs and prestiges.

F - Shattered Cake

F – Shattered Cake



We know that we have all the pieces of the cake and they cannot be rotated, so we simply have to divide the *total area* by the given width W:

$$L = \frac{\sum_{1 \leqslant i \leqslant N} w_i \cdot l_i}{W}.$$

G - Cordon Bleu

G - Cordon bleu

Fitting a known problem

1. If every courier could handle exactly one bottle, we could solve a maximum bipartite matching problem of minimum weight (*a.k.a* assignment problem).
2. By introducing $N_b - 1$ additional virtual couriers starting from the restaurant, we can represent extra fares by a courier.
3. We can now match every bottle with an exclusive courier.

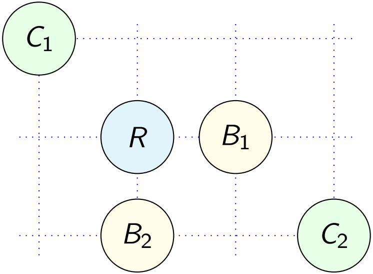
Solving the assignment problem

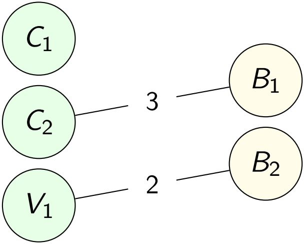
1. The matching can be computed in $O(n^3)$ using the Kuhn-Munkres algorithm (*a.k.a* the Hungarian method).

G - Cordon bleu

Example

One courier (out of two) will take care of delivering both bottles. One virtual courier V_1 is introduced at R.

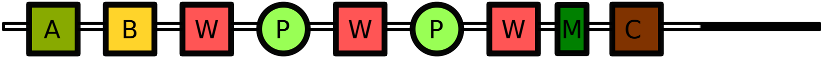




	C_1	C_2	V_1
B_1	4	**3**	2
B_2	4	3	**2**

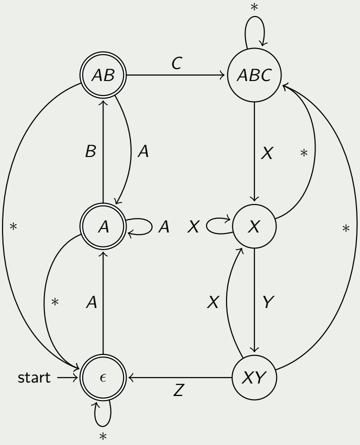
$\Rightarrow$ Total cost is 5

H - Kabobs



H - Kabobs

Automaton for rule $ABC > XYZ$



Algorithmic techniques

- Remove inaccessible states
- Use a default transition
- Counting paths of given size

Complexity

$O(\text{Size of the automaton} \times \#\text{steps})$

too much?

H - Kabobs

Number of states

Each automaton for a rule has *rulesize* states thus:

$$\#states \leq avg(rulesize)^{\#rules}$$

States are determined by pending rules and prefix read:

$$\#states \leq 2^{\#rules} * \#letters$$

Number of transitions

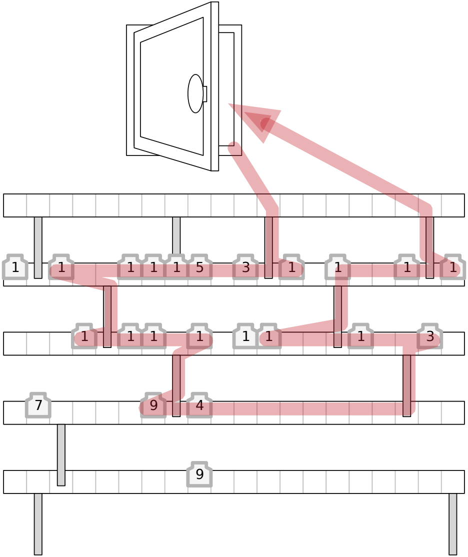
A state of the product automaton $(s_1, \ldots, s_r)$ has a rule named c when at least one the states s_i has a rule named c thus

$$\frac{\#trans}{\#states} \leq 1 + 2 \times r$$

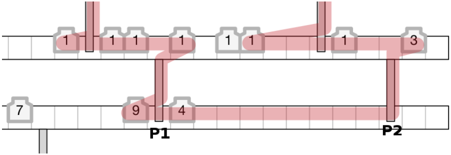
How many exactly?

Combining the above bounds gives us $\#trans < 3 \times 10^6$ and we can even lower this bound and pass easily!

I - Burglary



I - Burglary

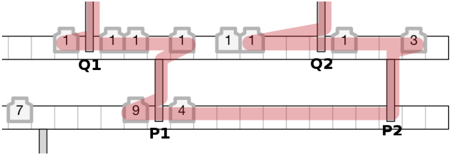


Solution sketch

Shelves: 0 (topmost) to N (floor). Slots: 0 to $M-1$. L = max ladders.
Max(T) = max candy grabbed for a trip with lowest reached shelf = T.
Result $=$ $\max_{\mathbf{1<=T<=N}} \mathbf{Max(T)}$
With P_1 and P_2 "up" ladder endpoints:
$\mathbf{Max(T)} = \max_{\mathbf{P_1,P_2}}(\mathbf{MaxUp(T,P_1,P_2)} + \mathbf{Grabbed(T,P1,P2)})$

- $MaxUp(T, P_1, P_2)$ = max candy grabbed on $0, \dots, T-1$ when reaching ("downwards") T by P_1 and leaving ("upwards") T by P_2
- $Grabbed(T, P_1, P_2)$ = all candy from P_1 to P_2 + potential "safely reachable" side candy (left and/or right).

I - Burglary



Key idea / dynamic programming

Shelves: 0 (topmost) to N (floor). Slots: 0 to $M-1$. $L=$ max ladders.
Idea: Compute $\mathbf{MaxUp(T, P_1, P_2)}$ reccursively based on $\mathbf{MaxUp(T-1, Q_1, Q_2)}$, $\mathbf{Grabbed(T-1, P_1, Q_1)}$, $\mathbf{Grabbed(T-1, P_2, Q_2)}$.

- Consider all Q_1, $Q_2 =$ "up" ladder endpoints for $T-1$
- Discard configs with jars in the intersection (not safe); avoid counting "middle" side candy twice.

Time complexity for all shelves: $\mathbf{O(N * L^4 * Compl\_Grabbed)}$

I - Burglary

Essential observation

$\mathbf{Grabbed(T, P_1, P_2)}$ **can be computed in constant time** for any (T, P_1, P_2) if one precomputes for all slots on all shelves:

- closest jar position left and right
- partial sums $SumLeft[T, P]$=sum of all candy on T left to P.

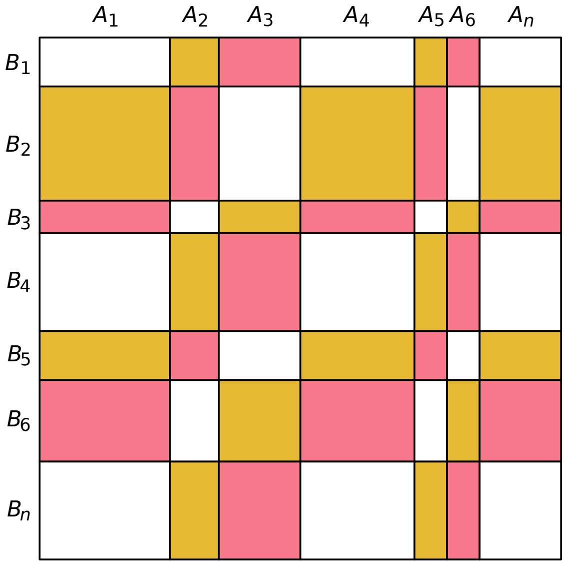
Precomputation: $\mathbf{O(N * M)}$

And so...

Overall complexity $\mathbf{= O(N * M + N * L^4)}$.

- Intersection tests + side candy = slight headache
- "Smaller" optims possible such as exploiting symmmetry, keeping only two rows for *MaxUp*...
- Tests may not be exhaustive but the Bandit is happy!

J - Frosting on the Cake



J - Frosting on the Cake

Key observation

Permuting columns or rows preserve the total area of each color. Hence we can reduce to a 3 by 3 grid, the dimensions are given by the sum of the entry lengths of same base 3 modulo.

Python Solution

```
def read_ints(): return [int(x) for x in input().split()]

def cat(l): return tuple(sum(l[n::3]) for n in [1, 2, 0])

input() # n
A = cat(read_ints())
B = cat(read_ints())
print("{} {} {}".format(B[2]*A[0]+B[0]*A[2]+B[1]*A[1],
                        B[2]*A[1]+B[0]*A[0]+B[1]*A[2],
                        B[2]*A[2]+B[0]*A[1]+B[1]*A[0]))
```

K - Blowing Candles

K - Blowing Candles

Key observation

The narrowest strip touches 3 points of the convex hull, 2 of them being consecutive on the hull

Algorithm

- Compute and restrict to convex hull in $O(n \log n)$
- Loop over all consecutive point pairs (a, b)
- Maintain a point c being furthest from (a, b) in $O(n)$ amortized time.