

# AtCoder Regular Contest 005

## 解説



AvtCoder株式会社 代表取締役  
高橋 直大

- 競技プログラミングをやったことがない人へ
  - まずはこっちのスライドを見よう！
  - <http://www.slideshare.net/chokudai/abc004>

# A問題 おいしいたこ焼きの作り方

---

1. 問題概要
2. アルゴリズム

- Xグラムの小麦粉がある。
- Yグラムの小麦粉につき、1個たこ焼きを作ることが可能である
- 最大いくつのたこ焼きを作ることが出来るか？

- 基本的なプログラムの流れ
  - 標準入力から、必要な入力を受け取る
    - 今回の場合は、 $x, y$ の2つの整数
  - 問題で与えられた処理を行う
    - 今回は、 $x$ グラムの小麦粉でたこ焼きが何個作れるか算出する
  - 標準出力へ、答えを出力する

- 入力

- 2つの数字を、標準入力から受け取る

- Cであれば、`scanf("%d %d", &x, &y);` など

- C++であれば、`cin >> x >> y;`

- 入力の受け取り方は、下記の練習問題に記載があります。

- [http://practice.contest.atcoder.jp/tasks/practice\\_1](http://practice.contest.atcoder.jp/tasks/practice_1)

- Xグラムの小麦粉から、Yグラムで作れるたこ焼きを、いくつ作れるか考える。
  - これは、単純な算数の問題
- $X \div Y$ で求めることが出来る。
  - 整数同士の演算なので、殆どの言語で勝手に小数点以下は切り捨てられる。
  - 小数で計算して、整数で出力、などをしてしまうと、四捨五入された結果などが出力されてしまうことがあるのに注意

- 具体的な記述
  - `int ret = X / Y;` のような感じ
  - 答えを格納する変数に、計算結果を入れておく



- 出力

- 求めた答えを、標準出力より出力する。
- 言語によって違います。
  - `printf("%d\n", ret);` (C)
  - `cout << ret << endl;` (C++)
  - `System.out.println(ret);` (Java)
  - 各言語の標準出力は、下記の練習問題に記載があります。
    - [http://practice.contest.atcoder.jp/tasks/practice\\_1](http://practice.contest.atcoder.jp/tasks/practice_1)

## B問題 おいしいたこ焼きの食べ方

---

1. 問題概要
2. アルゴリズム

- たこ焼きがN個作られている
- それぞれのたこ焼きが、何秒前に作られているかが与えられる
- 一番できたてのたこ焼きが、何秒前に出来ているかを出力しなさい

- 入力

- まずたこ焼きの数を表す整数 $N$ を受け取る
- 次に、長さ $N$ の配列を確保する
  - リストなどの動的な配列でも問題ない
- $N$ 回のループを回す
  - それぞれのたこ焼きが、何秒後に出来ているかを配列に格納する

- 処理

- 解法1: 最小値を順番に探す

- `int ret = 9999999;`など、大きい値を答えに入れておく。
    - ループを回し、1個ずつ、`ret`より小さいかどうか調べる
      - `If(ret > T[i]) ret = T[i];`

- 解法2: ソートしてしまう

- 配列に入っている値をソートする
      - `Sort(T);` みたいな。大抵の言語に標準で入っている。
    - 最初の値を出力

- 出力
  - A問題と同じく、答えを出力するだけ

- おまけ
  - 入力を配列に格納しなくても解ける
    - 解法1であれば、配列に格納していなくても、順番に数字を読んでいけば良い。

## C問題 おいしいたこ焼きの売り方

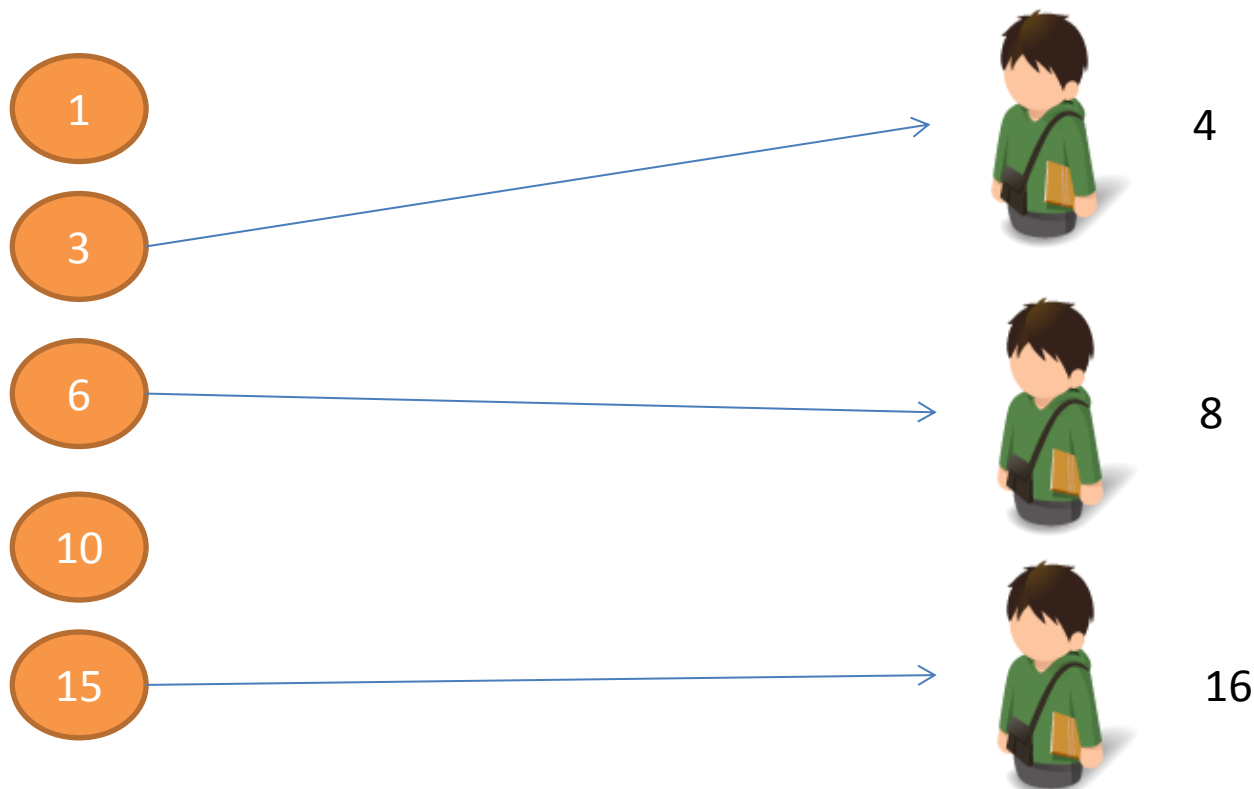
---

1. 問題概要
2. アルゴリズム

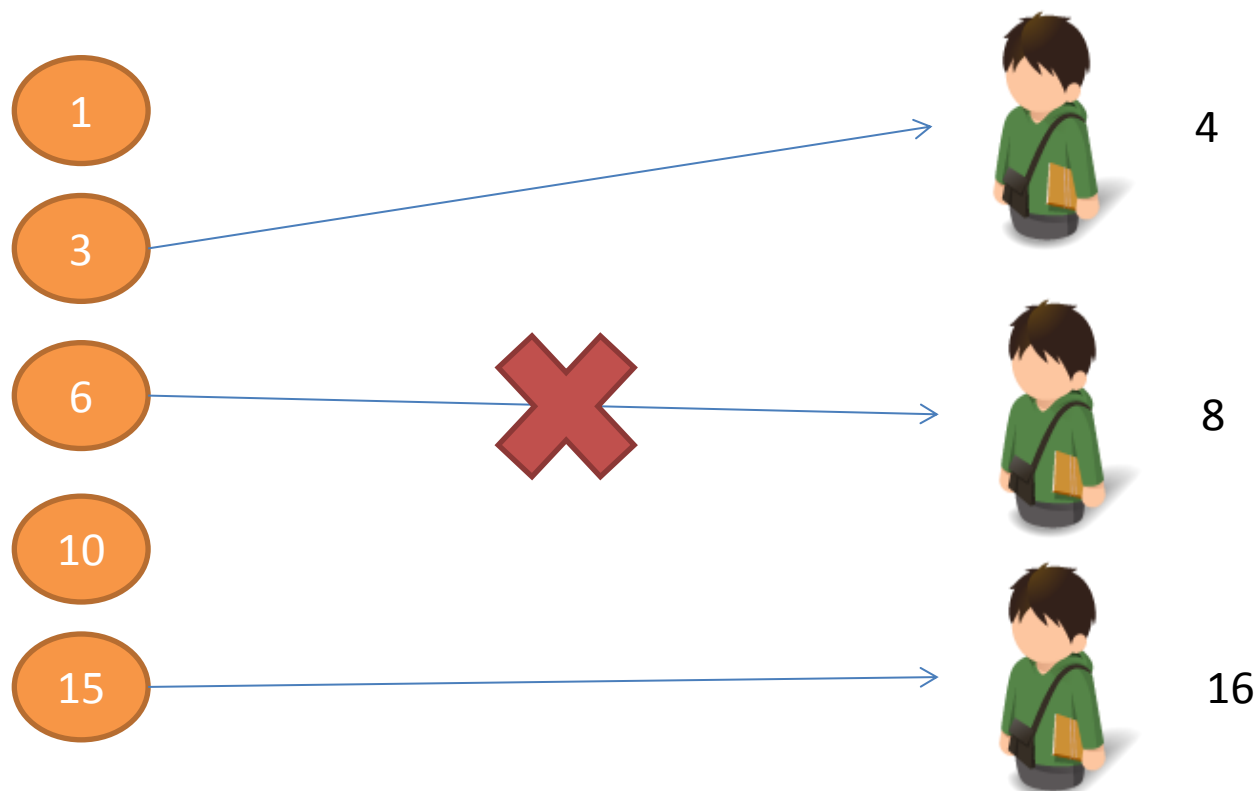


- たこ焼きは、出来てから $T$ 秒以内に売らなければならない。
- たこ焼きを $N$ 個作る
  - 各たこ焼きは $A_i$ 秒後に作成される
- お客さんは $M$ 人くる
  - 各お客さんは $B_j$ 秒後に訪れる
- 全てのお客さんに対し、作られてから $T$ 秒以内のたこ焼きを提供できるかどうかを出力しなさい。

- Sample5  $T = 2$ 
  - 以下のように、客に対してたこ焼きを割り当てる



- $T = 1$ になった場合
  - 割り当てることが出来なかったら失敗



- 各お客さんに対し、どのたこ焼きを売るかを決める
  - M人のお客さんに、N個のたこ焼きを1つずつ割り当てる？
  - 組み合わせの数は、 $N! / (N-M)!$ 通り。
    - N,Mともに上限は100
    - $N=M=100$ の時、 $100!$ 通り存在する
- 2秒の制限時間で処理可能なのは、高速な言語でも1億ループ程度まで。
  - 全通りを試すことが出来ない！

- 実行時間を早くするためには？
  - 現在、明らかに無駄な組み合わせを大量に考えてしまっている
  - 無駄な組み合わせを取り除こう！

- 明らかに無駄なパターン1

- $A < B$ の時、A番目のお客さんのたこ焼きより、B番目のお客さんのたこ焼きの方が新しいケースは、考えなくて良い
  - 後からきたお客さんに、先に来たお客さんよりも古いたこ焼きを売る理由はない
- これだと、パターン数は大幅に減る！
  - $N$ 個のたこ焼きから、 $M$ 個のお客さんを選ぶ
    - これは  $N! / (M! * (N-M)!)$ 通り
    - $N=100, M=50$ が最大だが、これも非常に大きな数になってしまう
- これでも間に合わないので、もう少し無駄なパターンを削る

- 明らかに無駄なパターン2
  - T秒以内に作られたたこ焼きの中で、最も古いたこ焼き以外は考えなくても良い
    - そうした方が、後のお客さんの選択肢が増える
- こうすると、実は売り方は1通りしかなくなってしまう
  - 売れるたこ焼きがあれば、最も古いものを売る
  - 売れるたこ焼きが無ければ、全員に対して、T秒以内に出来たたこ焼きを全て売ることが出来ない。

- 実装の流れ

- M人に対して、たこ焼きが売れるかどうかループを回して判定を行う
  - N個のたこ焼きに対してループを回し、まだ売られていないもので、T秒以内に作られたものがあるかどうかを探す
    - あれば、最も昔に作られたものを売る
    - 外側にフラグか何かを立てておく
    - なければ、noを出力して終了
- 全て売切れればyesを出力する
- ループの最大数は $N * M$ 回くらいなので、100程度であれば十分間に合う

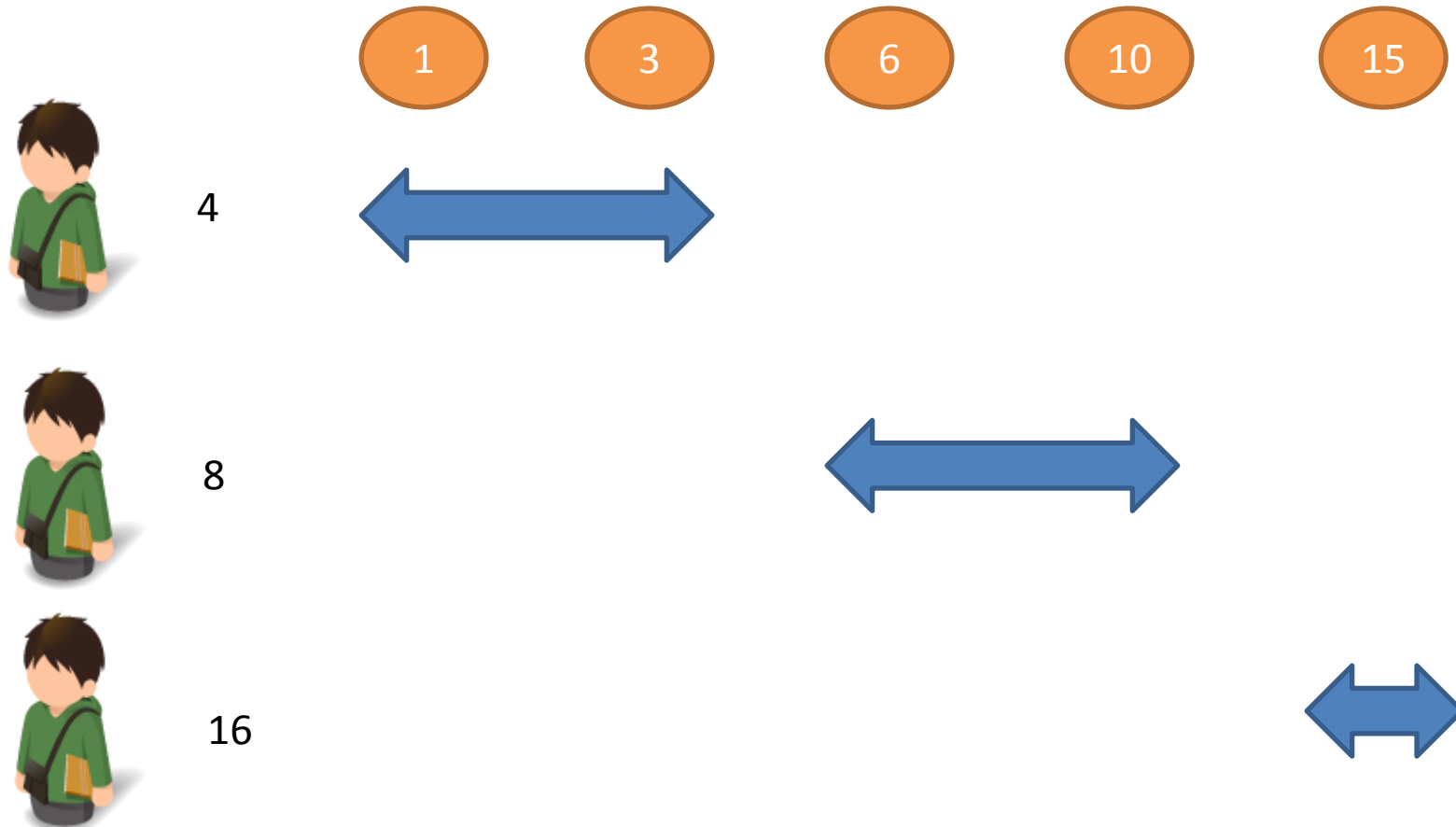


- おまけ1

- 実装を工夫すると、 $1 \leq N, M \leq 100,000$ でも間に合う
- 工夫する点
  - 前の人食べたたこ焼きより、次の人が食べるたこ焼きは絶対に後のものである
  - つまり、前の人食べたたこ焼きの次のたこ焼きから調べて良い
  - さらに、早く作られたたこ焼きから順番に調べていき、売ることが可能なたこ焼きが見つかったら、それより後のたこ焼きについて調べる必要がない
  - こうすると、同じたこ焼きを複数人に対して調べる必要がなくなる
  - たこ焼き1つにつき調べる回数は1回なので、計算量は $O(N)$ になる

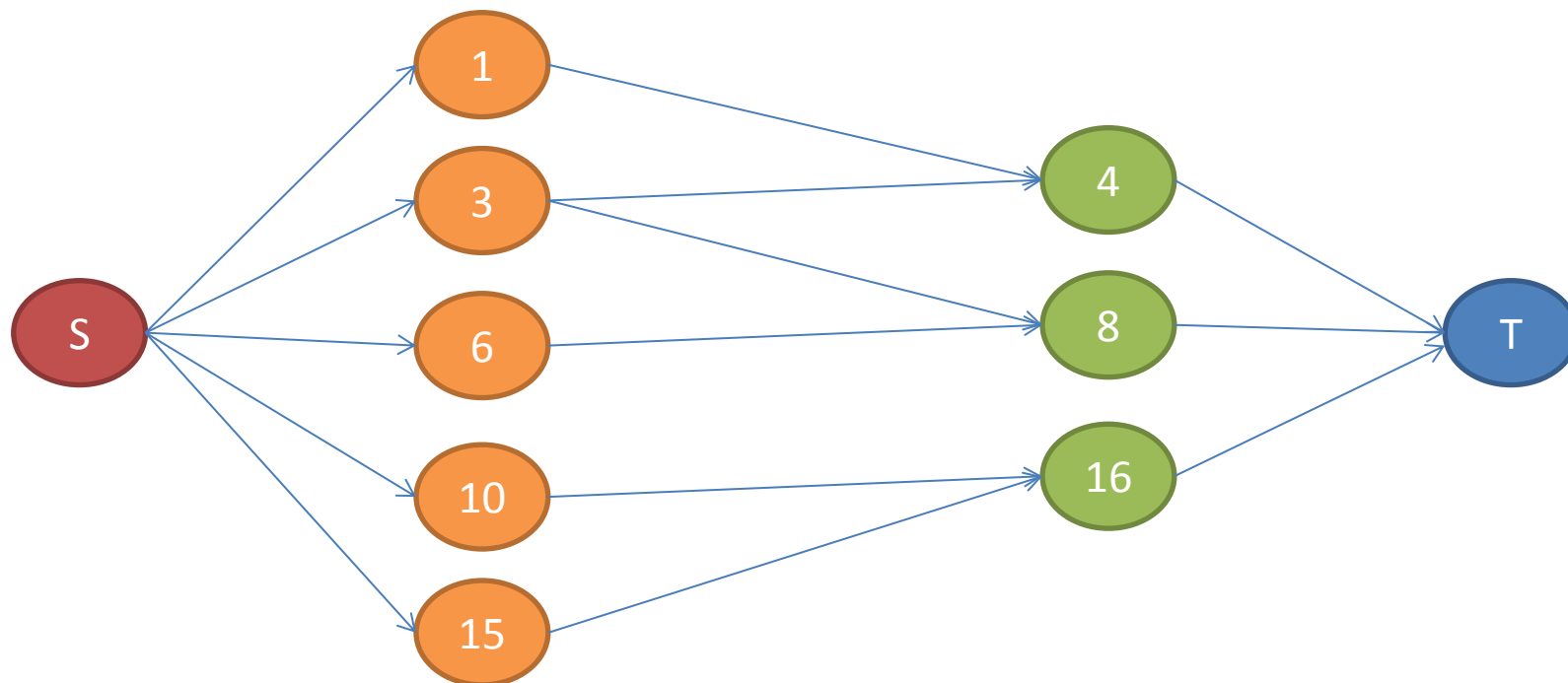
- 計算のイメージ

- こんな感じで、調べる範囲が被らない



- おまけ2
  - 二部グラフの最大マッチングでも解くことが可能
    - ただし、大きなサイズになると解けない
  - 興味のある人は、以下のワードで検索
    - 二部グラフ
    - 二部グラフの最大マッチング

- こんな感じのグラフを作って、フローをM流す
  - $T = 5$  のときのグラフ
  - 流せればyes、流せなければno



## D問題 おいしいたこ焼きの焼き方

---

1. 問題概要
2. アルゴリズム

- $N \times N$ の正方形のたこ焼き器が存在する
- 各マスでは1個のたこ焼きを作ることが可能で、そのマスで作れるたこ焼きの美味しさは整数 $D_{ij}$ である
- 店員が $Q$ 人存在し、各店員は $P_k$ 個のたこ焼きを一度に焼くことが可能である。
- 一度に焼くたこ焼きの範囲は、 $x$ 軸 $y$ 軸に平行な長方形になってなければならない
- 各店員に対して、美味しいたこ焼きが焼けるかどうか答えなさい

- 各店員に対し、使うべき長方形が変わってくる

2	2	1
2	2	1
1	1	1



9個

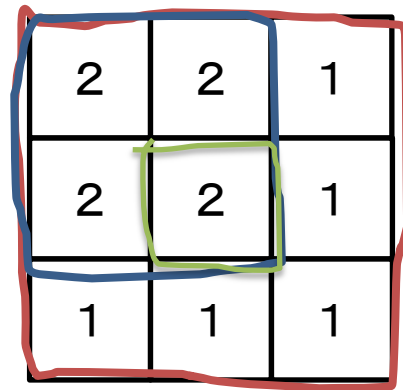


4個



1個

- 各店員に対し、使うべき長方形が変わってくる
  - 美味しさの合計が最大になるときの合計値を出力



2	2	1
2	2	1
1	1	1



9個



4個



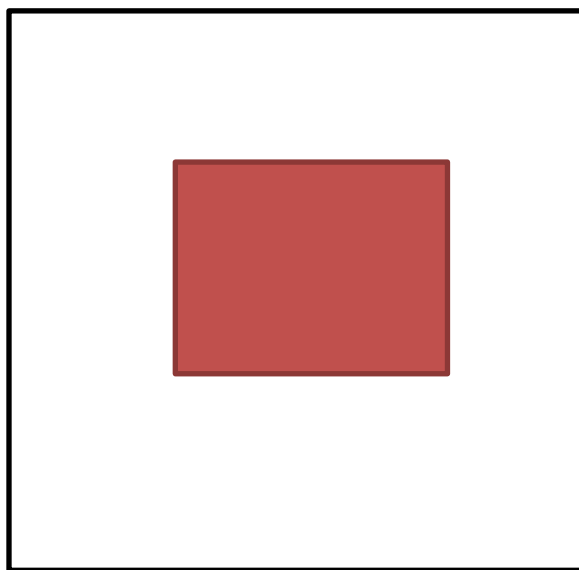
1個



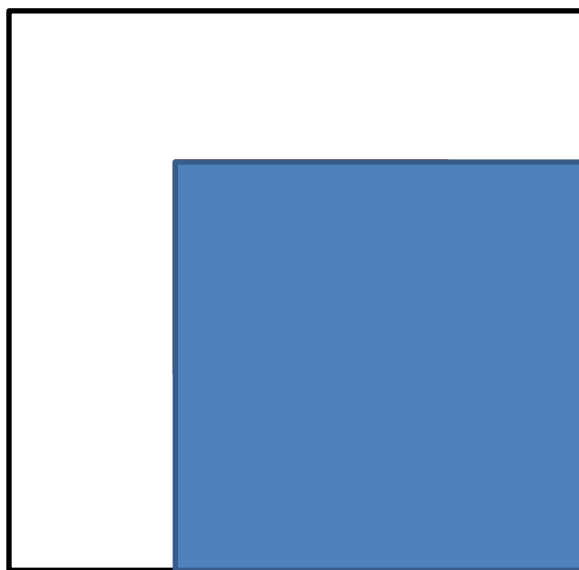
- 普通に判定すると・・・？
- 長方形の作り方は、縦の選び方が $N * (N-1)$ 通り、横の選び方が $N * (N-1)$ 通りあるので、大凡 $N^4$ 通り存在する
- これらに対して、おいしさの合計の計算をすると、調べる必要のあるマス数は、最小で1マス、最大で $N * N$ マス
- さらに、店員の人数が $N * N$ 人
- 計算量は、 $O(N^8)$ になってしまう。
  - $N \leq 50$ なので、到底間に合わない！

- 長方形の中の数字の和を、高速に求めなくてはならない
  - 求め方はたくさんある！
- 高速な求め方を考えよう！

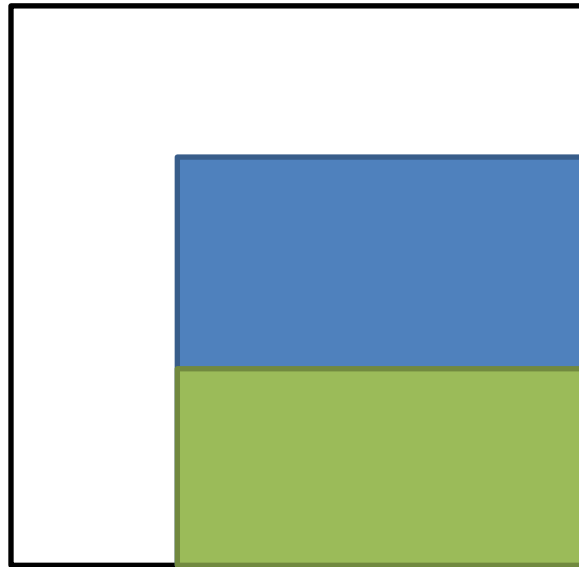
- 解法1: 全ての長方形を一瞬で計算できるようにする
  - 赤い部分を求めたい



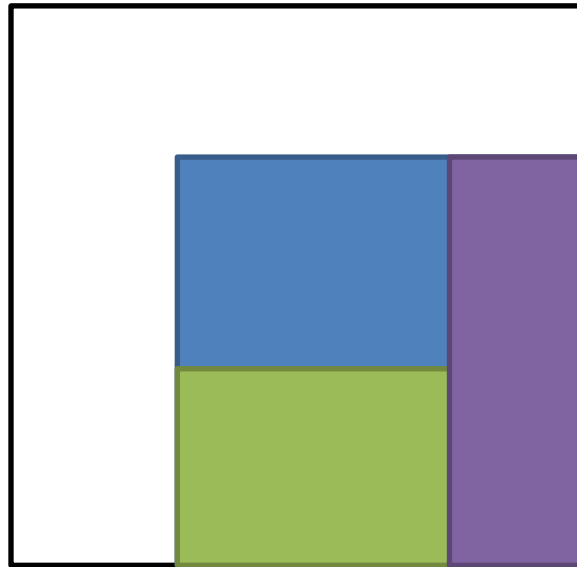
- 解法1: 全ての長方形を一瞬で計算できるようにする
  - 赤い部分を求めたい
    - まずはこの右下の部分までの青い部分を求め



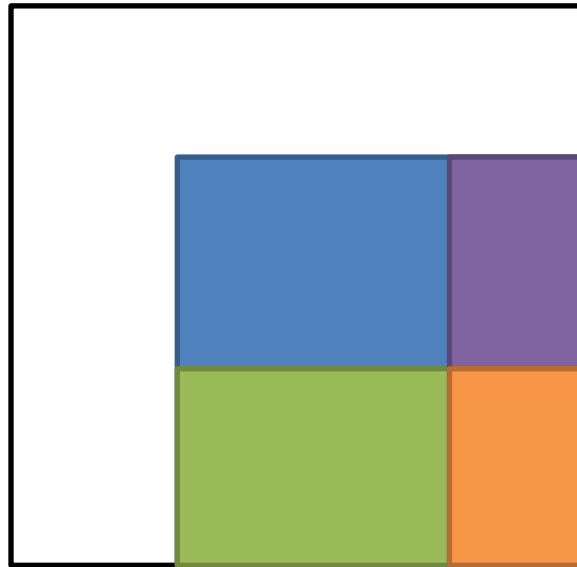
- 解法1: 全ての長方形を一瞬で計算できるようにする
  - 赤い部分を求めたい
    - まずはこの右下の部分までの青い部分を求め、緑の部分と



- 解法1: 全ての長方形を一瞬で計算できるようにする
  - 赤い部分を求めたい
    - まずはこの右下の部分までの青い部分を求め、緑の部分と紫の部分を引き、



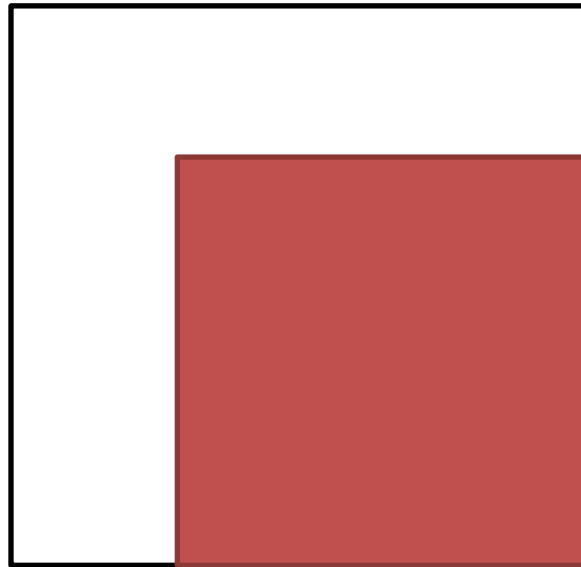
- 解法1: 全ての長方形を一瞬で計算できるようにする
  - 赤い部分を求めたい
    - まずはこの右下の部分までの青い部分を求め、緑の部分と紫の部分を引き、オレンジの部分を足せば、赤い部分が求まる



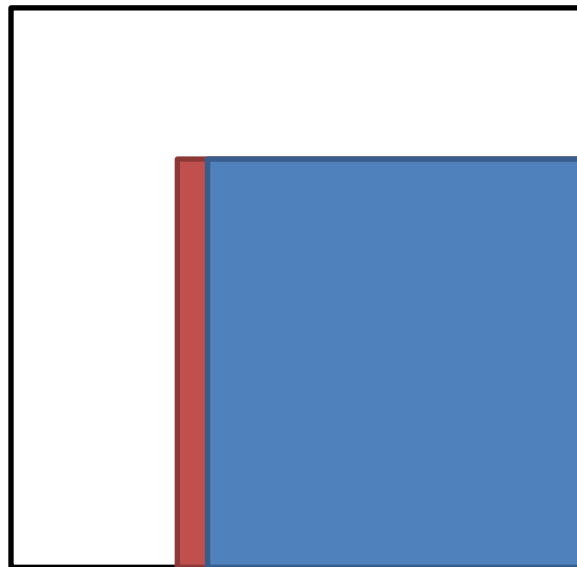
- つまり、右下までの面積さえ予め求められれば、全ての長方形の面積は、一瞬で求められる
  - じゃあ、どうやって予め計算するの？
  - パターンは $N^2$ 通りしかないので、 $O(N^4)$ で愚直に計算できる
    - 上手くやると $O(N^2)$ でも計算できる！



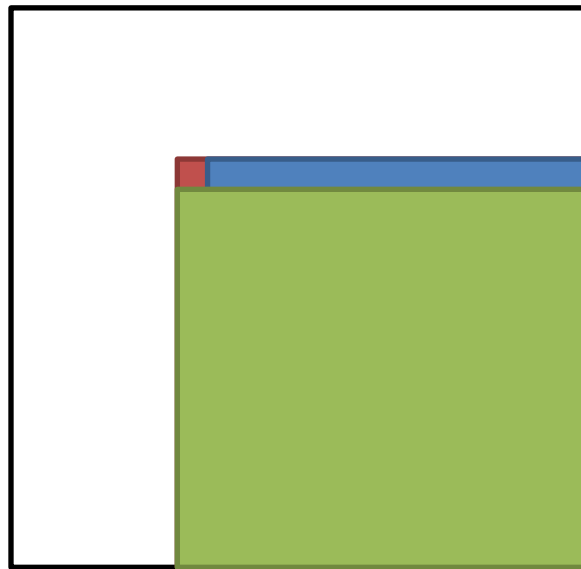
- $O(n^2)$ で上手くやる方法
  - 赤い部分を求めたい



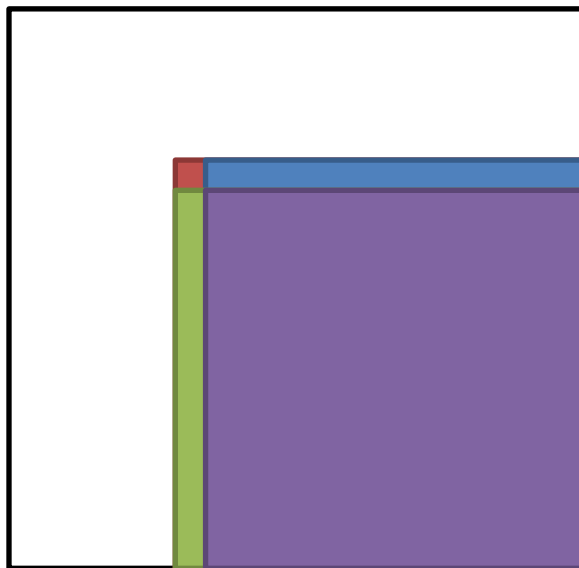
- $O(n^2)$ で上手くやる方法
  - 赤い部分を求めたい
    - 青い部分と



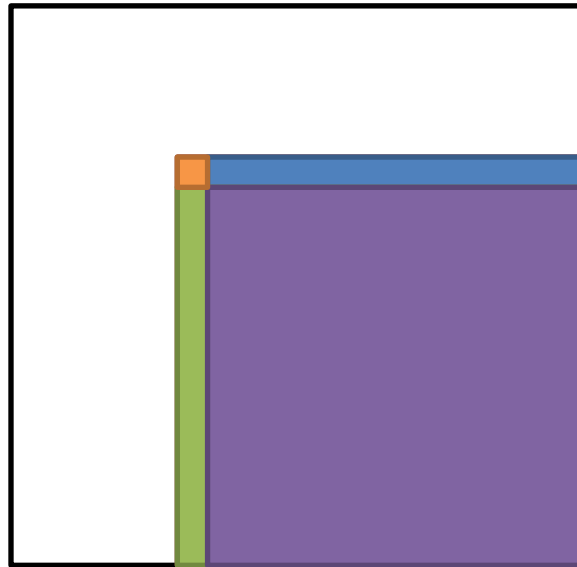
- $O(n^2)$ で上手くやる方法
  - 赤い部分を求めたい
    - 青い部分と緑の部分足して、



- $O(n^2)$ で上手くやる方法
  - 赤い部分を求めたい
    - 青い部分と緑の部分足して、紫の部分引いた後、



- $O(n^2)$ で上手くやる方法
  - 赤い部分を求めたい
    - 青い部分と緑の部分足して、紫の部分引いた後、オレンジの1マス部分だけ足してあげれば良い
  - よって、右下から順番に動的計画法で埋めていけば良い



- 解き方1 解き方と計算量まとめ
  - 右下まで必ず使う長方形を全通り列挙する  $O(n^2)$ 
    - その長方形の美味しさの合計を列挙する  $O(1)$
  - 各長方形を全通り列挙する  $O(n^4)$ 
    - その長方形の美味しさの合計を列挙する  $O(1)$
  - 全てのたこ焼きの数に対して、最大値が求められる

- 店員について
  - 1回の判定で、1から $N*N$ までの全パターンについて考えてしまえば、全ての店員について、計算し終わった後に出力すれば良い。
  - よって、一人一人の店員に対して、全ての長方形を考えてあげる必要はない

- 以下のようなケースの場合、3個のたこ焼きを焼いた方が、4個のたこ焼きを焼くより良いパターンが存在する

1	1	1
1	1	1
9	9	9

たこ焼き数	1	2	3	4	5	6	7	8	9
美味しさ	9	18	27	20	0	30	0	0	33



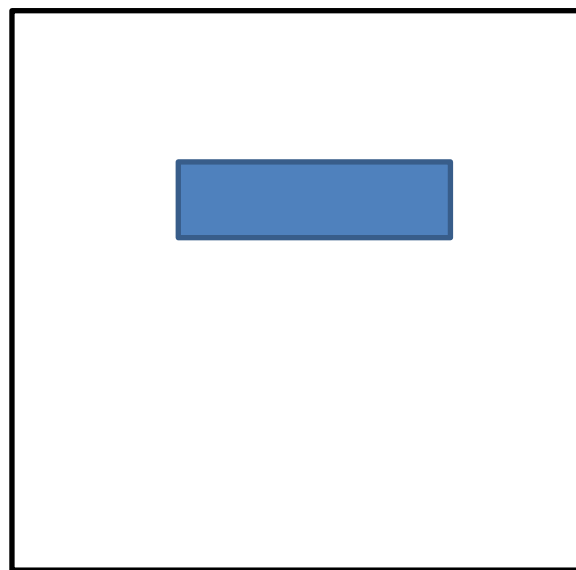
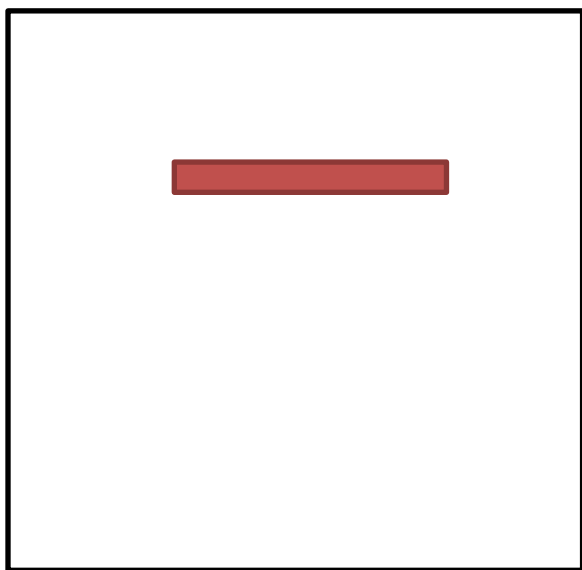
- 以下のようなケースの場合、3個のたこ焼きを焼いた方が、4個のたこ焼きを焼くより良いパターンが存在する
  - 小さい方から順番に、これまでの最大値を塗りつぶしていけば良い

1	1	1
1	1	1
9	9	9

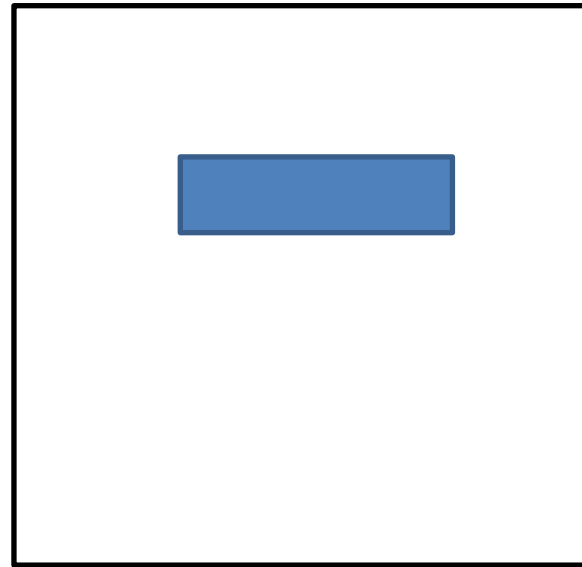
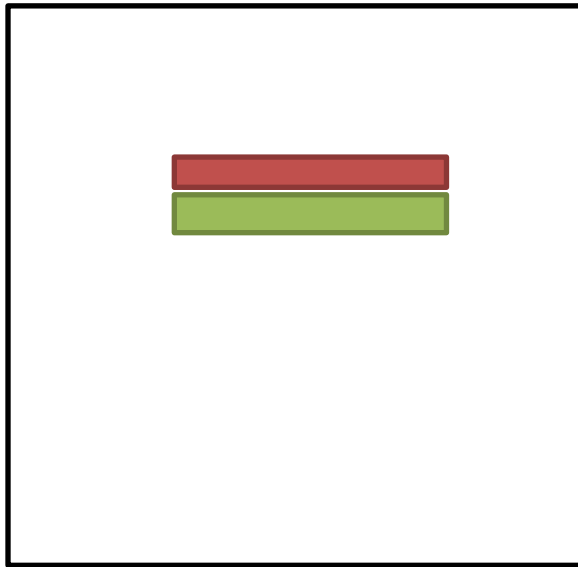
たこ焼き数	1	2	3	4	5	6	7	8	9
美味しさ	9	18	27	27	27	30	30	30	33



- 解法2:計算順序を変えるだけで多少早くなる
  - 例えば、赤い部分を求めた後、青い部分を求めたい時

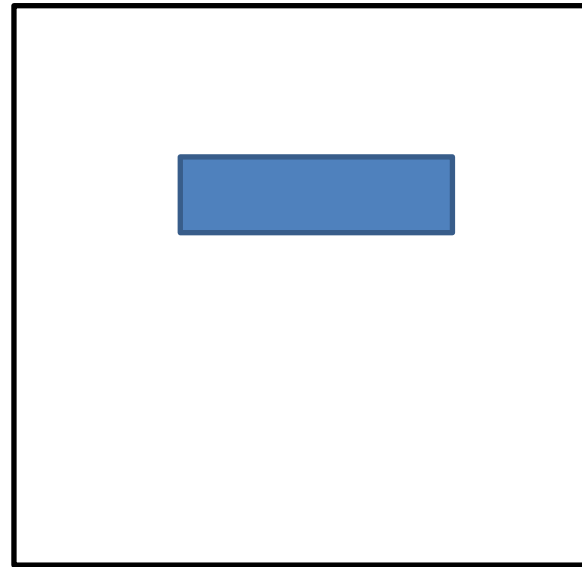
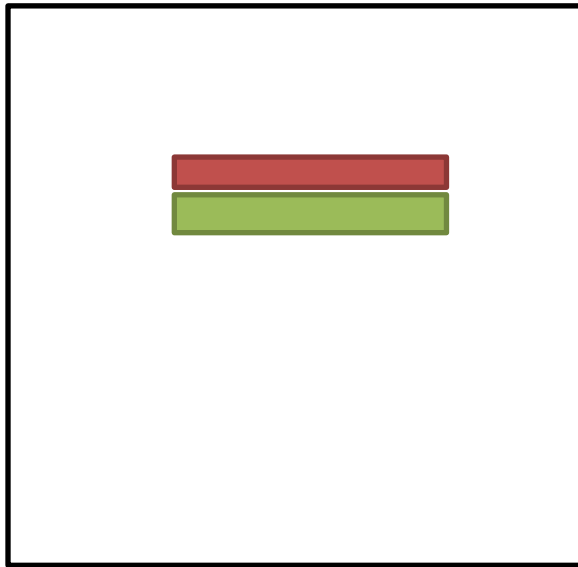


- 解法2:計算順序を変えるだけで多少早くなる
  - 例えば、赤い部分を求めた後、青い部分を求めたい時、予め計算した赤い部分に、新しく増えた緑の部分足を足せば良い

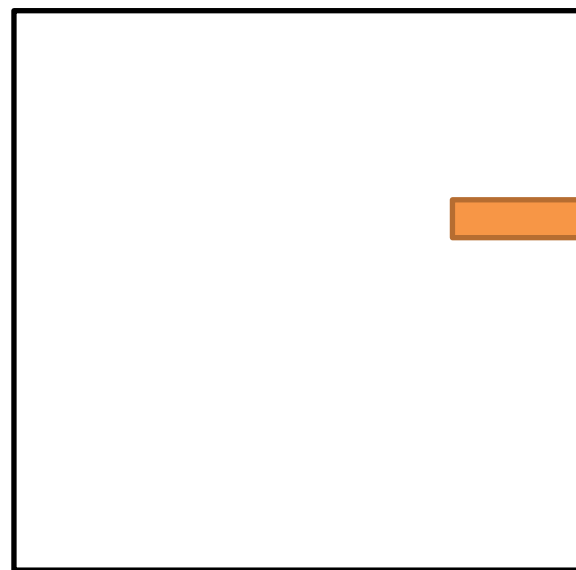
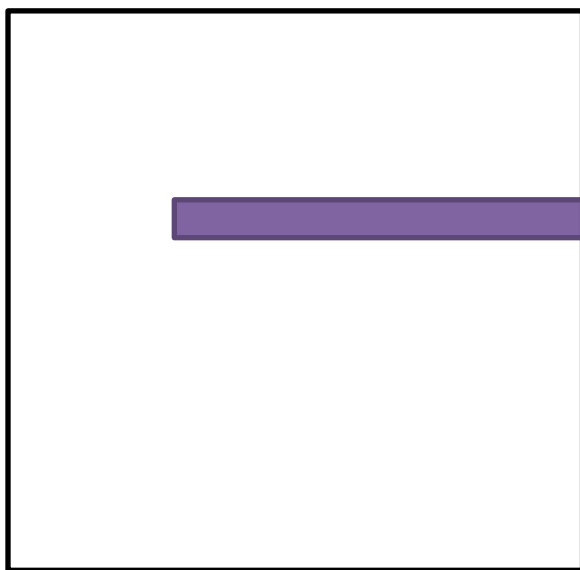


- 計算量の変化は？
  - 全部の長方形が $O(N^4)$ 
    - 和は、一回り小さい長方形から、1行足すだけで求められる。
    - 1行足すコストは、最大 $N$ マスなので、 $O(N)$
  - これをさらに早くすることが出来る！

- この緑の部分を一瞬で求められるように、事前計算をしておきたい。
  - 1行の全列挙はパターン数も $O(n^3)$ と少ないので、簡単に計算可能
    - 愚直で $O(n^5)$  解法2と同じ同じ計算方法で $O(n^4)$



- おまけ 事前計算の高速な計算方法
  - 先ほどと同じで、紫の部分からオレンジの部分を引くだけ
    - これは普通に後ろからループするだけで事前計算出来る。
      - これが $O(n^2)$



- 
- 高速に計算するためには、計算するパーツの事前計算が大切！

- $O(n^4)$ のアルゴリズムを紹介したが、 $O(n^5)$ でも間に合う
  - 回答例2の、横のメモを使わないパターン
  - 愚直な計算で、横のメモだけ使うパターン