Greedy Algorithms

US Change Problem

United States Change Problem:
Convert some amount of money into the fewest number of coins.

Input: An amount of money, M, in cents.

Output: The smallest number of quarters q, dimes d, nickels n, and pennies p whose values add to M (i.e., $25q + 10d + 5n + p = M$ and $q + d + n + p$ is as small as possible).

```
BetterChange(M, c, d)
  r ← M
  for k ← 1 to d
      i_k ← r/c_k
      r ← r − c_k · i_k
  return (i_1, i_2, ..., i_d)
```

It is a **greedy algorithm:**

At every step of iteration, a greedy algorithm tries to find the best optimal solution (e.g., used the most the coin with the biggest value)

US Change Problem

United States Change Problem:
Convert some amount of money into the fewest number of coins.

Input: An amount of money, M, in cents.

Output: The smallest number of quarters q, dimes d, nickels n, and pennies p whose values add to M (i.e., $25q + 10d + 5n + p = M$ and $q + d + n + p$ is as small as possible).

```
BETTERCHANGE(M, c, d)
  r ← M
  for k ← 1 to d
        i_k ← r/c_k
        r ← r − c_k · i_k
  return (i_1, i_2, . . . , i_d)
```

Why **greedy?**

"greedy" means having excessive desire for something without considering the effect or damage done.

US Change Problem

$\text{BETTERCHANGE}(M, \mathbf{c}, d)$

```
r ← M
for k ← 1 to d
    i_k ← r/c_k
    r ← r − c_k · i_k
return (i_1, i_2, ..., i_d)
```

Does it always find a correct solution?

When $c_1 = 25$, $c_2 = 20$, $c_3 = 10$, $c_4 = 5$, $c_5 = 1$,

if $M = 40$, BetterChange returns $i_1 = 1$, $i_3 = 1$, $i_4 = 1$

We would solve the problem with $i_2 = 2$…

US Change Problem

BETTERCHANGE$(M, \mathbf{c}, d)$

$r \leftarrow M$

for $k \leftarrow 1$ **to** d

  $i_k \leftarrow r / c_k$

  $r \leftarrow r - c_k \cdot i_k$

return $(i_1, i_2, \ldots, i_d)$

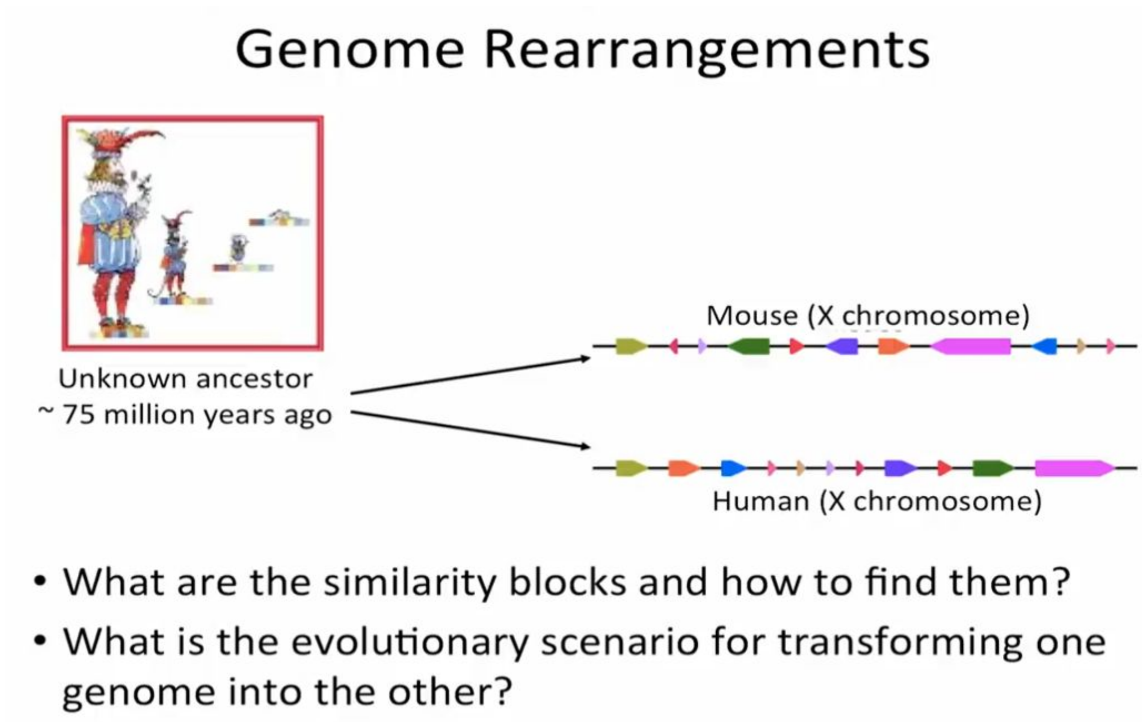
We can ask ourselves: how close are we from the optimal solution?

Maybe this algorithm works almost always correctly

Outline of the lecture

We are going to see a **bioinformatic problem** that we try to solve with **different greedy algorithms** and we are going to **evaluate their goodness** in finding the **best solution**

Let us see a problem in Biology



https://www.youtube.com/watch?v=lCoUp2Bq8OA&list=PLQ-85lQlPqFOcGz6A3g2ZArRL09Ffpp_N (until 8:52)

Reversal Distance Problem

- Goal: Given two permutations, find the shortest series of reversals that transforms one into another

- Input: Permutations $\boldsymbol{\pi}$ and $\boldsymbol{\sigma}$

- Output: A series of reversals $\rho_1,\ldots\rho_t$ transforming $\boldsymbol{\pi}$ into $\boldsymbol{\sigma}$, such that t is minimum

- ***t*** - reversal distance between π and σ
- ***d***(π, σ) - smallest possible value of t, given π and σ

Sorting By Reversals Problem

- <u>Goal</u>: Given a permutation (i.e., a vector in a random order), find a shortest series of reversals that transforms it into the identity permutation (*1 2 … n*)

- <u>Input</u>: Permutation π

- <u>Output</u>: A series of reversals $\rho_1, \ldots \rho_t$ transforming π into the identity permutation such that *t* is minimum

Reversals

- Reversal ρ (*i, j*) reverses (flips) the elements from *i* to *j* in π

$$\pi = \pi_1 \text{------} \pi_{i-1} \underline{\pi_i \pi_{i+1} \text{------} \pi_{j-1} \pi_j} \pi_{j+1} \text{-----} \pi_n$$

$$\downarrow \rho(i,j)$$

$$\pi_1 \text{------} \pi_{i-1} \pi_j \pi_{j-1} \text{------} \pi_{i+1} \pi_i \pi_{j+1} \text{-----} \pi_n$$

Reversals: Example

$\pi = 1\ 2\ \underline{3\ 4\ 5}\ 6\ 7\ 8$

index of the array

$\rho(2,4)$

$1\ 2\ 5\ 4\ 3\ 6\ 7\ 8$

Reversals: Example

π = 1 2 3 4 5 6 7 8

index of the array

$\rho(2,4)$

1 2 5 4 3 6 7 8

$\rho(4,5)$

1 2 5 4 6 3 7 8

Sorting By Reversals: Example

- $t = d(\pi)$ - reversal distance of π
- Example :

$$\pi = \underline{3\ 4}\ 2\ 1\ 5\ 6\ 7\ 10\ 9\ 8$$
$$4\ 3\ 2\ 1\ 5\ 6\ 7\ \underline{10\ 9\ 8}$$
$$\underline{4\ 3\ 2\ 1}\ 5\ 6\ 7\ 8\ 9\ 10$$
$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$

So $d(\pi) = 3$

Sorting by reversals: 5 steps

Step 0: π	2	-4	-3	5	-8	-7	-6	1
Step 1:	2	3	4	5	-8	-7	-6	1
Step 2:	2	3	4	5	6	7	8	1
Step 3:	2	3	4	5	6	7	8	-1
Step 4:	-8	-7	-6	-5	-4	-3	-2	-1
Step 5: γ	1	2	3	4	5	6	7	8

Sorting by reversals: 4 steps

Step 0: π	2	-4	-3	5	-8	-7	-6	1
Step 1:	2	3	4	5	-8	-7	-6	1
Step 2:	-5	-4	-3	-2	-8	-7	-6	1
Step 3:	-5	-4	-3	-2	-1	6	7	8
Step 4: γ	1	2	3	4	5	6	7	8

Sorting by reversals: 4 steps

Step 0: π	2	-4	-3	5	-8	-7	-6	1
Step 1:	2	3	4	5	-8	-7	-6	1
Step 2:	-5	-4	-3	-2	-8	-7	-6	1
Step 3:	-5	-4	-3	-2	-1	6	7	8
Step 4: γ	1	2	3	4	5	6	7	8

What is the reversal distance for this permutation? Can it be sorted in 3 steps?

Sorting By Reversals: A Greedy Algorithm

- If sorting permutation π = 1 2 3 6 4 5, the first three elements are already in order so it does not make any sense to break them.
- The length of the already sorted prefix of π is denoted *prefix*(π)
  - *prefix*(π) = 3
- This results in an idea for a greedy algorithm: *increase prefix(π) at every step*

Greedy Algorithm: An Example

- Doing so, π can be sorted

$$\color{red}{1\ 2\ 3}\ \underline{6\ 4}\ 5$$

$$\downarrow$$

$$\color{red}{1\ 2\ 3\ 4}\ \underline{6\ 5}$$

$$\downarrow$$

$$\color{red}{1\ 2\ 3\ 4\ 5\ 6}$$

- Number of steps to sort permutation of length *n* is at most *(n – 1)*

Greedy Algorithm: Pseudocode

<u>SimpleReversalSort(π)</u>

for i □ 1 to $n - 1$ # assuming that elements are from 1 to n
    j □ position of element i in π (i.e., $\pi_j = i$)
    if $j \neq i$
        π □ we apply $\rho(i, j)$ on π
    if π is the identity permutation
      return π

Analyzing SimpleReversalSort

- SimpleReversalSort does not guarantee the smallest number of reversals and takes five steps on π = 6 1 2 3 4 5 :

Step 0: 6 1 2 3 4 5

- Step 1: 1 6 2 3 4 5
- Step 2: 1 2 6 3 4 5
- Step 3: 1 2 3 6 4 5
- Step 4: 1 2 3 4 6 5

Analyzing SimpleReversalSort (cont'd)

- But it can be sorted in two steps:

  π = 6 1 2 3 4 5

  - Step 1: 5 4 3 2 1 6
  - Step 2: 1 2 3 4 5 6
- So, SimpleReversalSort(π) is not optimal
- But how good is it?

Analyzing SimpleReversalSort (cont'd)

- But it can be sorted in two steps:

  π = 6 1 2 3 4 5

  - Step 1: 5 4 3 2 1 6
  - Step 2: 1 2 3 4 5 6
- So, SimpleReversalSort(π) is not optimal
- But how good is it?
- Optimal algorithms are **unknown** for many problems; approximation algorithms are used

Approximation Algorithms

- These algorithms find approximate solutions rather than optimal solutions
- The approximation ratio of an algorithm A on the problem with input π is:

  $$A(\pi) / OPT(\pi)$$

  where

  $A(\pi)$ - solution produced by algorithm A

  $OPT(\pi)$ - optimal solution of the problem

+ (in our case, π is an instance of the reversal sorting problem)

Approximation Algorithms

- If an algorithm has an approximation ratio = 1.5, it means that the solution it finds is never more than 150% of the optimal one.
  - For example, if the minimum sorting requires 10 reversals, the approx algorithm will use at most 15.

Approximation Ratio

- For algorithm A that minimizes objective function (minimization algorithm):
  - $\max_{|\pi| = n} A(\pi) / OPT(\pi)$
- For maximization algorithm:
  - $\min_{|\pi| = n} A(\pi) / OPT(\pi)$

Can we do better than SimpleReversalSort(π)?

Yes

Sometimes we need to characterize better the problem to make more sophisticated techniques

DISCLAIMER: DON'T BE AFRAID

You are requested to understand the general idea of greedy not how to elaborate this more sophisticated techniques (right now..)

Adjacencies and Breakpoints

$$\pi = \pi_1 \pi_2 \pi_3 \ldots \pi_{n-1} \pi_n$$

- A pair of elements π_i and π_{i+1} are adjacent if

$$\pi_{i+1} = \pi_i \pm 1$$

- For example:

$$\pi = 1 \;\; 9 \;\; \underline{3 \;\; 4} \;\; \underline{7 \;\; 8} \;\; 2 \;\; \underline{6 \;\; 5}$$

- (3, 4) or (7, 8) and (6,5) are adjacent pairs

Breakpoints: An Example

There is a breakpoint between any adjacent element that are non-consecutive:

$$\pi = 1 \mid 9 \mid 3\ \ 4 \mid 7\ \ 8 \mid 2 \mid 6\ \ 5$$

- Pairs (1,9), (9,3), (4,7), (8,2) and (2,5) form breakpoints of permutation π

Breakpoints: An Example

There is a breakpoint between any adjacent element that are non-consecutive:

$$\pi = 1 \mid 9 \mid 3 \;\; 4 \mid 7 \;\; 8 \mid 2 \mid 6 \;\; 5$$

- Pairs (1,9), (9,3), (4,7), (8,2) and (2,5) form breakpoints of permutation π
- $b(\pi)$ - # breakpoints in permutation π

Extending Permutations

- We want to ensure that also the first and the last element are in the right positions. To do that…
- We put two elements $\pi_0 = 0$ and $\pi_{n+1} = n+1$ at the ends of π

Example:

$$\pi = 1 \mid 9 \mid 3 \;\; 4 \mid 7 \;\; 8 \;\; 2 \mid 6 \mid 5$$

↓ Extending with *0* and *10*

$$\pi = 0 \;\; 1 \mid 9 \mid 3 \;\; 4 \mid 7 \;\; 8 \mid 2 \mid 6 \;\; 5 \mid 10$$

Note: A new breakpoint was created after extending

Sum up: Adjacency & Breakpoints

- An adjacency - a pair of adjacent elements that are consecutive
- A breakpoint - a pair of adjacent elements that are not consecutive

---

$\pi = 5\ \ 6\ \ 2\ \ 1\ \ 3\ \ 4$ ⟶ Extend π with $\pi_0 = 0$ and $\pi_7 = 7$

adjacencies

0 5 6 2 1 3 4 7

breakpoints

Reversal Distance and Breakpoints

- Observation: each reversal eliminates at most 2 breakpoints.

π = | 2 3 | 1 | 4 | 6 5 |

0 2 | 3 1 | 4 | 6 5 | 7	$b(\pi) = 5$
0 1 3 2 4 | 6 5 | 7	$b(\pi) = 4$
0 1 2 3 4 6 5 7	$b(\pi) = 2$
0 1 2 3 4 5 6 7	$b(\pi) = 0$

Reversal Distance and Breakpoints

- Observation: each reversal eliminates at most 2 breakpoints.
- This implies:

  reversal distance ≥ #breakpoints / 2

π = | 2 3 | 1 | 4 | 6 5 |

0 2 | 3 1 | 4 | 6 5 | *7* $b(\pi) = 5$

0 1 3 2 4 | 6 5 | *7* $b(\pi) = 4$

0 1 2 3 4 6 5 *7* $b(\pi) = 2$

0 1 2 3 4 5 6 *7* $b(\pi) = 0$

Sorting By Reversals: A Better Greedy Algorithm

BreakPointReversalSort(π)

while $b(\pi) > 0$

Among all possible reversals, choose reversal ρ minimizing $b(\pi)$ after its application

π □ *apply* $\rho(i, j)$ *on* π

return π

Sorting By Reversals: A Better Greedy Algorithm

<u>BreakPointReversalSort(π)</u>

```
while b(π) > 0
      Among all possible reversals, choose reversal ρ minimizing b(π) after its application
      π □ apply ρ(i, j) on π
return π
```

Problem: this algorithm may work forever (we cannot reduce the number of breakpoints anymore)

Can we do better than BreakPointReversalSort(π)?

Yes

We need to characterize even better the problem to exploit further aspects

Strips

- Strip: an interval between two consecutive breakpoints in a permutation
  - Decreasing strip: *strip* of elements in decreasing order (e.g. 6 5 and 3 2).
  - Increasing strip: *strip* of elements in increasing order (e.g. 7 8)

0 1 9 4 3 7 8 2 5 6 *10*

  - A single-element strip can be declared either increasing or decreasing. We will choose to declare them as decreasing with exception of the strips with *0* and *n+1*

Reducing the Number of Breakpoints

Observation 1:

If permutation π contains at least one decreasing strip, then there exists a reversal ρ which decreases the number of breakpoints (i.e. $b(\pi)$ after $\rho < b(\pi)$)

Things To Consider

- For π = 1 4 6 5 7 8 3 2

  $0 \;\; 1 \,|\, 4 \,|\, 6 \;\; 5 \,|\, 7 \;\; 8 \,|\, 3 \;\; 2 \,|\, 9$ $\qquad b(\pi) = 5$

  - Choose decreasing strip with the smallest element k in π (k = 2 in this case)

Things To Consider (cont'd)

- For π = 1 4 6 5 7 8 3 2

  $0 \;\; 1 \,|\, 4 \,|\, 6 \;\; 5 \,|\, 7 \;\; 8 \,|\, 3 \;\; 2 \,|\, 9$ $\qquad b(\pi) = 5$

  - Choose decreasing strip with the smallest element k in π (k = 2 in this case)

Things To Consider (cont'd)

- For π = 1 4 6 5 7 8 3 2

  $0\ \ 1 \mid 4 \mid 6\ \ 5 \mid 7\ \ 8 \mid 3\ \ 2 \mid 9$ $\qquad b(\pi) = 5$

  - Choose decreasing strip with the smallest element k in π (k = 2 in this case)
  - Find $k - 1$ in the permutation

Things To Consider (cont'd)

- For π = 1 4 6 5 7 8 3 2

  0 1 | 4 | 6 5 | 7 8 | 3 2 | 9 $b(\pi) = 5$

  - Choose decreasing strip with the smallest element k in π (k = 2 in this case)
  - Find $k - 1$ in the permutation
  - Reverse the segment between k and k-1:
  - 0 1 | 4 | 6 5 | 7 8 | 3 2 | 9 $b(\pi) = 5$
  - 0 1 2 3 | 8 7 | 5 6 | 4 | 9 $b(\pi) = 4$

Reducing the Number of Breakpoints Again

- If there is no decreasing strip, there may be no reversal ρ that reduces the number of breakpoints (i.e. $b(\pi)$ after $\rho \geq b(\pi)$ for any reversal ρ).
- By reversing an increasing strip (# of breakpoints stay unchanged), we will create a decreasing strip at the next step. Then the number of breakpoints will be reduced in the next step (observation 1).

Things To Consider (cont'd)

- There are no decreasing strips in $\boldsymbol{\pi}$, for:

$$\pi = 0\ 1\ 2\ |\ 5\ 6\ 7\ |\ 3\ 4\ |\ 8 \qquad b(\pi) = 3$$

$$\rho(6,7) \text{ on } \pi = 0\ 1\ 2\ |\ 5\ 6\ 7\ |\ 4\ 3\ |\ 8 \qquad b(\pi) = 3$$

✔ ρ(6,7) does not change the # of breakpoints

✔ ρ(6,7) creates a decreasing strip thus guaranteeing that the next step will decrease the # of breakpoints by doing ρ(6,2).

ImprovedBreakpointReversalSort

<u>ImprovedBreakpointReversalSort(π)</u>

while $b(\pi) > 0$
    if π has a decreasing strip
        Among all possible reversals, choose reversal ρ
            that minimizes $b(\pi)$ *after* ρ
    else
        Choose a reversal ρ that flips an increasing strip in π
    $\pi \leftarrow$ *apply* ρ *on* π
return π

ImprovedBreakpointReversalSort: performance?

- *ImprovedBreakPointReversalSort is the optimal solution?*

ImprovedBreakpointReversalSort: performance?

- *ImprovedBreakPointReversalSort is the optimal solution? Unfortunately, no..*
- *ImprovedBreakPointReversalSort* is an approximation algorithm
  - Optimal algorithm eliminates at most 2 breakpoints in every step: $d(\pi) \geq b(\pi) / 2$
  - It eliminates at least one breakpoint in every two steps; at most $2b(\pi)$ steps
  - Approximation ratio: $2b(\pi) / d(\pi)$

ImprovedBreakpointReversalSort: performance?

- *ImprovedBreakPointReversalSort is the optimal solution? Unfortunately, no..*
- *ImprovedBreakPointReversalSort* is an approximation algorithm
  - Approximation ratio: $2b(\pi) / d(\pi)$

- NOTE: we can compute $d(\pi)$ for a specific instance of π but not for all the instance of sorting by reversal problem, in this way we can compute the approximation ratio

Take home messages

When should we use Greedy Algorithms?

When should we use Greedy Algorithms?

- **Simple and easy to understand**

  we follow a simple idea: for every subproblem, a greedy algorithm tries to find the best optimal solution (e.g., in ImprovedBreakpointReversalSort, choose reversal ρ that minimizes $b(\pi)$ after ρ)

When should we use Greedy Algorithms?

- **Simple and easy to understand**

  we follow a simple idea: for every subproblem, a greedy algorithm tries to find the best optimal solution (e.g., in ImprovedBreakpointReversalSort, choose reversal ρ that minimizes b(π) after ρ)

This is also the limit: following a simple rule (or rules) for **all** the subproblems might lead to **naive solutions** (i.e., solutions that consider the characteristics of every subproblem missing details)

When should we use Greedy Algorithms?

- **Can be used as a building block for other algorithms:** it can be used as a starting point for developing more complex algorithms.

We started with SimpleReversalSort
We improve it with BreakPointReversalSort
We improve it with ImprovedBreakpointReversalSort

When should we use Greedy Algorithms?

- **Fast and efficient** (compared to other techniques)

example: SimpleReversalSort runs in O(n) (n size of the vector π)

When should we use Greedy Algorithms?

- **Provides a good enough solution** (we have seen how good today for one problem)

ImprovedBreakPointReversalSort still finding acceptable **approximate** solutions (not the best ones)

We still don't know an **efficient algorithm** computing the **optimal solution** for sorting by reversal