Ещё о выравниваниях

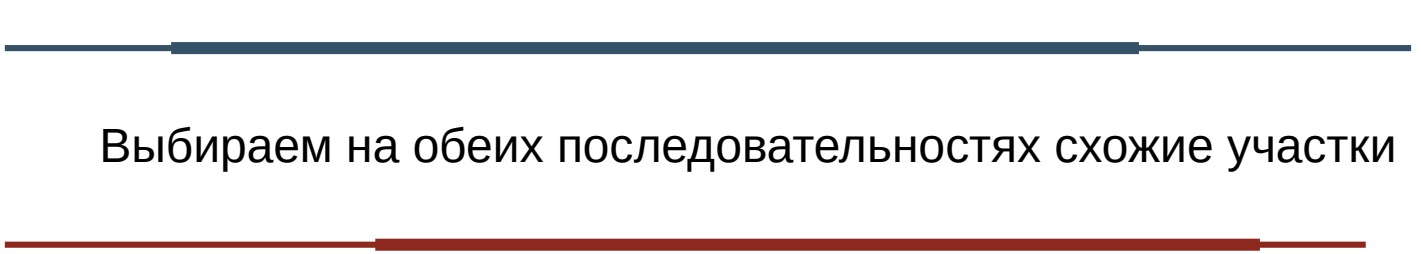
Дополнительная лекция
С.А. Спирин, 4 мая 2021

Локальное выравнивание

Seq1
(длины L_1)

Seq2
(длины L_2)

Локальное выравнивание

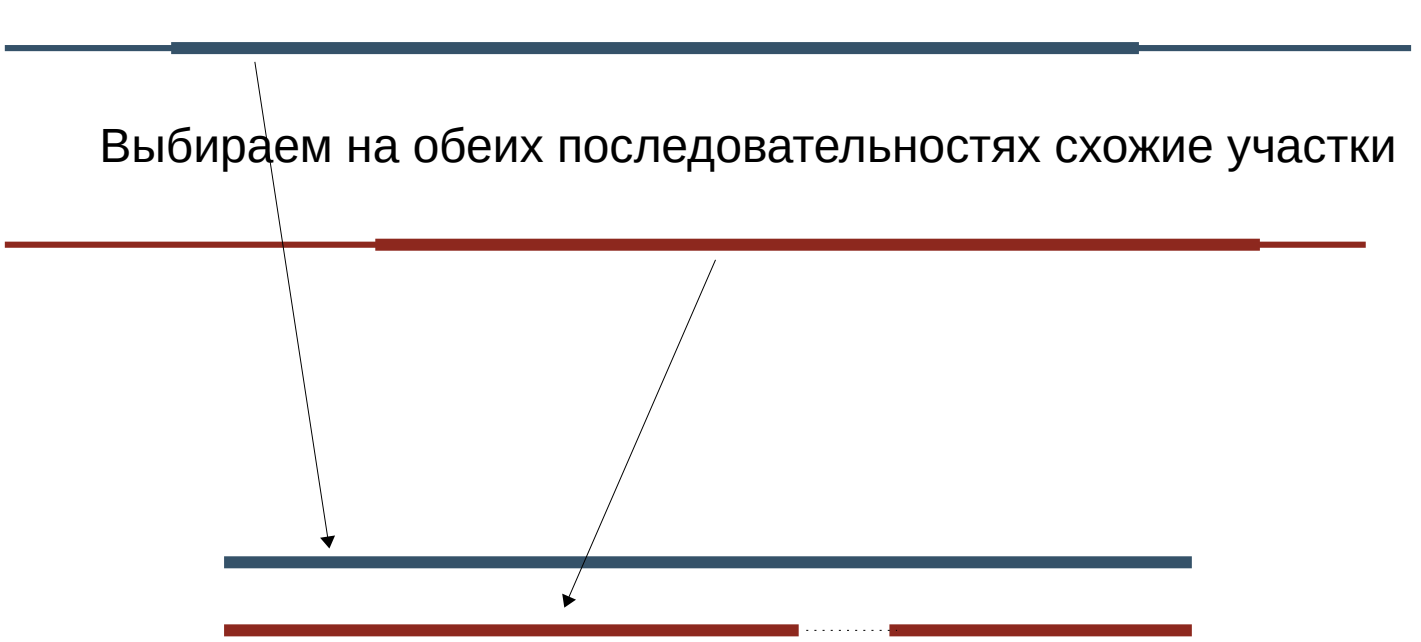


Seq1
(длины L_1)

Выбираем на обеих последовательностях схожие участки

Seq2
(длины L_2)

Локальное выравнивание



Локальное выравнивание

Покрытие 1 = l_1/L_1

Покрытие 2 = l_2/L_2

Алгоритмы Нидельмана – Вунша и Смита – Уотермена

На входе:
- Пара последовательностей
- Матрица замен
- Штрафы за открытие и удлинение гэпа

На выходе у алгоритма Нидельмана – Вунша (NW, Saul B. Needleman and Christian D. Wunsch, 1970):
- Оптимальное глобальное выравнивание
- Вес этого выравнивания

На выходе у алгоритма Смита – Уотермена (SW, Temple F. Smith and Michael S. Waterman, 1981):
- Оптимальное локальное выравнивание
- Вес этого выравнивания

Оптимальное — значит с наибольшим весом среди всех возможных выравниваний этих последовательностей

Число возможных выравниваний

```
TGGAGTAACCAT-
TGGGATAACCTTG
```

```
TGGAGTAACCAT-------------
------------TGGGATAACCTTG
```

```
-TGGAGTAACCAT
TGGGATAACCTTG
```

```
TGGA--GTAACCAT--
TGGGATAA---CCTTG
```

Теорема: даны последовательности длины *m* и *n*. Если выравнивания, в которых все сопоставления букв одинаковы, считать эквивалентными, то существует $C^{m}_{m+n} = C^{n}_{m+n}$ различных (неэквивалентных другу другу) выравниваний.

Например, вот эти два выравнивания эквивалентны:

```
--TGGAGTAACCAT     =     T--GGAGTAACCAT
TG-GGATAACCTTG           -TGGGATAACCTTG
```

Число возможных выравниваний

```
TGGAGTAACCAT-
TGGGATAACCTTG
```

```
TGGAGTAACCAT----------
----------TGGGATAACCTTG
```

```
-TGGAGTAACCAT
TGGGATAACCTTG
```

```
TGGA--GTAACCAT--
TGGGATAA---CCTTG
```

Теорема: даны последовательности длины *m* и *n*. Если выравнивания, в которых все сопоставления букв одинаковы, считать эквивалентными, то существует $C^{m}_{m+n} = C^{n}_{m+n}$ различных (неэквивалентных другу другу) выравниваний.

Идея доказательства:
В сумме в двух последовательностях *m* + *n* букв.
Выберем изо всех *m* + *n* какие-нибудь *m* букв.
Теперь сопоставим **выбранные** буквы второй последовательности (пусть их оказалось *k* штук) **невыбранным** буквам первой последовательности (в первой выбрано *m* – *k*, значит не выбрано *k*). Это задаёт выравнивание, взаимно однозначно.

За какое время можно найти оптимальное выравнивание?

C^{m}_{m+n} — очень большое число, если *m* и *n* — типичные длины белков. При равных длинах $C^{m}_{m+n} = C^{n}_{2n}$ примерно равно 2^{2n}.
Для *n* = 100 это примерно 10^{60}.

Поэтому перебрать все возможные выравнивания, дабы найти оптимальное, нереально.

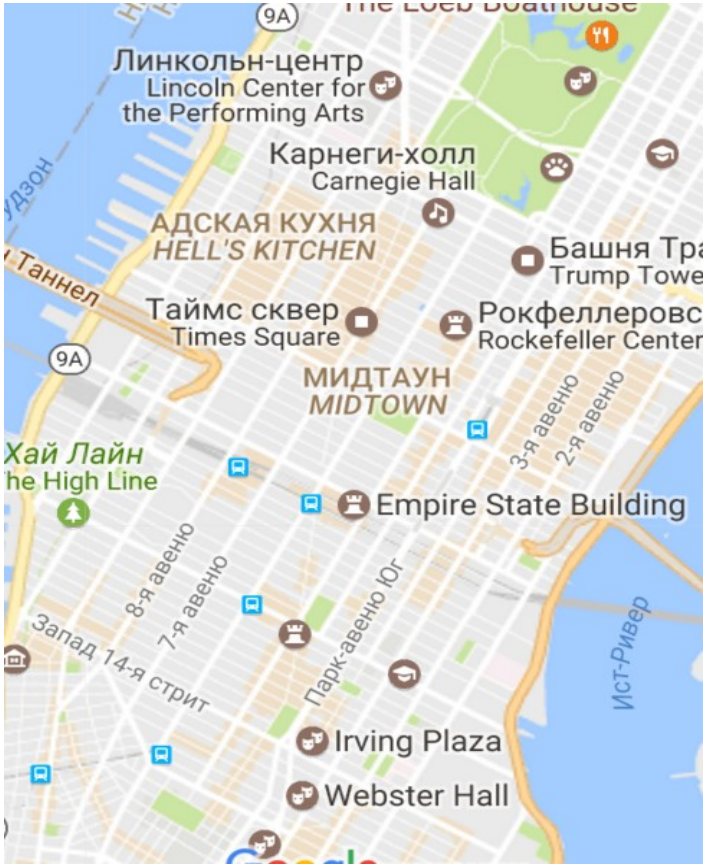
Тем не менее, алгоритмы NW и SW находят их за сравнительно короткое время.
Время работы обоих алгоритмов пропорционально произведению *mn* длин последовательностей (эта величина растёт с ростом *m* и *n* намного медленнее, чем C^{m}_{m+n})

Секрет — в правильном применении принципа **динамического программирования**.

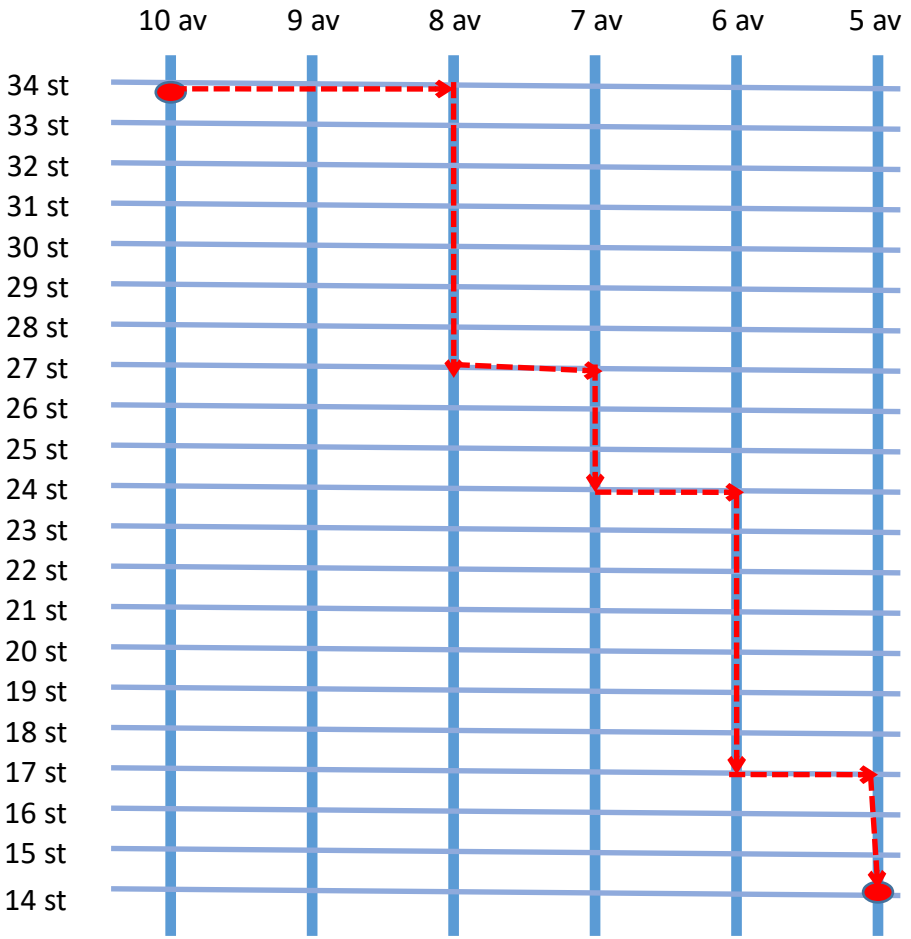
Динамическое программирование

- Идея динамического программирования состоит в сведении задачи к набору подзадач так, чтобы одни подзадачи использовали решения других.
  Ускорение достигается за счёт того, что каждая подзадача решается только один раз, в то время как при простом переборе их каждый раз приходится решать все.
- Простой пример — на следующем слайде

Манхэттен и его «формализация»



Мидтаун Манхэттен



Задача об оптимальном проезде

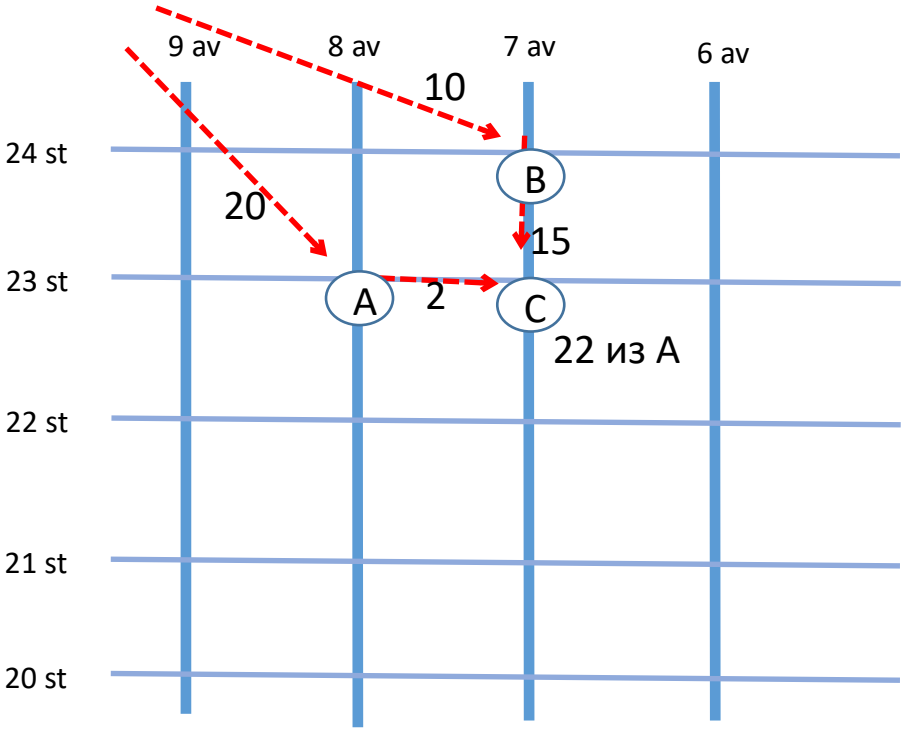
Пример задачи

- Пусть дана решётка размера $(n+1)\times(m+1)$, причём для всех соседних узлов решётки указана цена перехода
  (например, решётка — это улицы Манхэттена, а ценой может быть время с учётом пробок и светофоров)
- Задача: найти оптимальный путь из левого верхнего узла в правый нижний
  В случае времени «оптимальный» — значит с наименьшей суммой времён
  Очень важно, что итоговая «цена» получается как сумма цен переходов! Это обстоятельство позволяет применить динамическое программирование.

Пример с решёткой: решение перебором

- Можно просто перебрать все возможные пути
- Сколько таких путей?
  - Мы должны проехать n+m отрезков, из них n с запада на восток и m с севера на юг
  - В последовательности проезжаемых отрезков любые n могут оказаться горизонтальными
  - Значит, различных путей C^{n}_{n+m} .

Основной шаг алгоритма

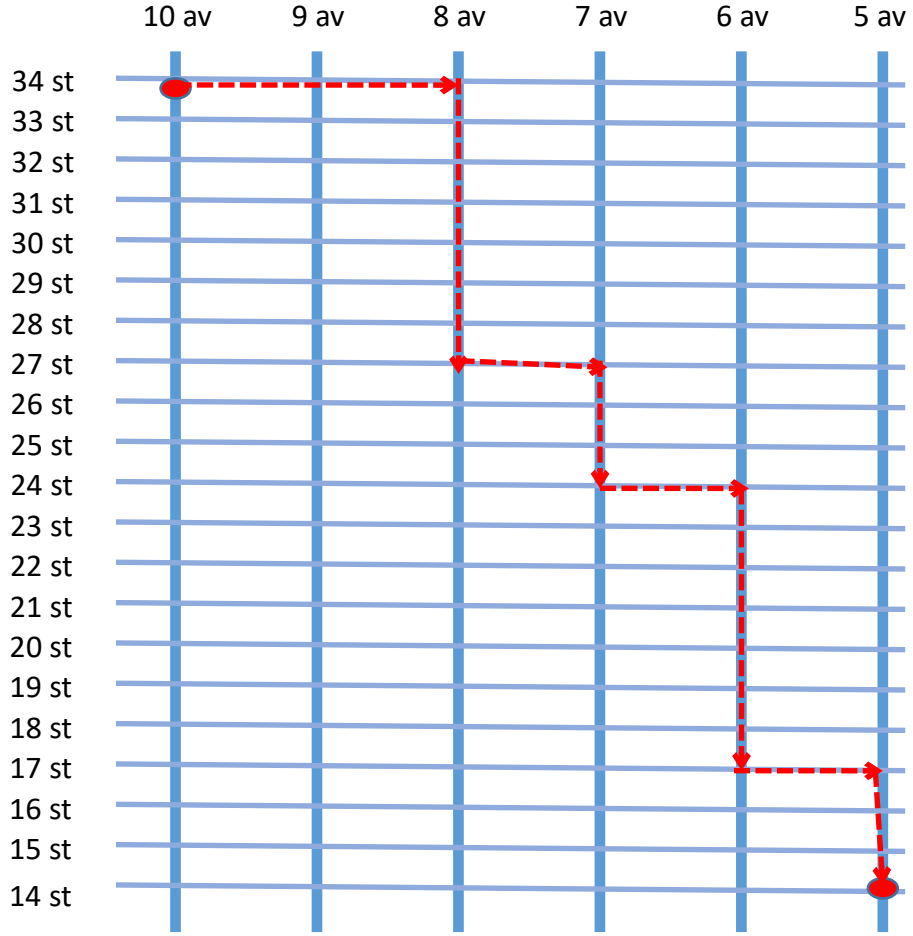


- На перекресток C можно попасть либо из A, либо из B.
- Если оптимальный путь в A занимает 20 мин, путь A-C – 2 мин, то путь в C через A занимает 22 мин
- Если оптимальный путь в B – 10 мин, B-C – 15 мин (пробка), то путь в C через B занимает 25 мин
- Значит, оптимальный путь до C занимает 22 мин, и это путь из A.
- Запомним это в C

Идея в том, чтобы посчитать оптимальное время движения из точки старта **для всех** перекрёстков, которые могут попасться на пути!

Это легко сделать для самых северных перекрёстков.
Но и для остальных это занимает совсем немного времени!
В конце получаем оптимальное время для полного пути.

Оптимальный путь



Лучшее время вычислили. А как ехать-то?
По ходу вычислений для каждого перекрестка C мы запоминали не только лучшее время, но и из какого соседнего перекрестка надо приехать: из A или из B (см. предыдущий слайд)
Оптимальный маршрут прокладывается от конечного пункта до начального по запомненным “стрелочкам”.

Сколько операций?

- Если надо проехать n кварталов направо и m – вниз, то имеем $(n+1)\times(m+1)$ перекрестков
- В каждом надо выполнить несколько операций: два сложения, одно сравнение, запомнить два числа. Требуется одно и то же число K операций в каждой вершине
- Значит, всего $K\cdot(n+1)\cdot(m+1)$ операций
- При $n = m = 50$ получаем $5 \cdot 51 \cdot 51 = 13\,005$ операций – пустяки для компьютера!
  (ср. с $C^{50}_{100} \approx 10^{30}$ — ноналлион вариантов, которые нужно было бы просмотреть при переборе)

А при чём тут выравнивание?

- Вместо пути теперь выравнивание
- Вместо времени — вес выравнивания
  - нужен не минимальный вес, а максимальный.
    Но главное осталось: вес выравнивания равен **сумме** весов по всем позициям этого выравнивания
- Вместо перекрёстка у нас будет **выравнивание префиксов**

Префикс

- В языкознании это то же, что приставка:
  (в слове «пересдача» префикс «пере»)
- В комбинаторике (и теории алгоритмов) префиксом заданной последовательности длины n называется подпоследовательность от начала до k-ой буквы включительно (k может быть от нуля до n)
  (а последовательность обычно называется «слово»)
- Например, возьмём последовательность: ATGGCT
  Её префиксы:
  ∅, A, AT, ATG, ATGG, ATGGC, ATGGCT

Выравнивание префиксов

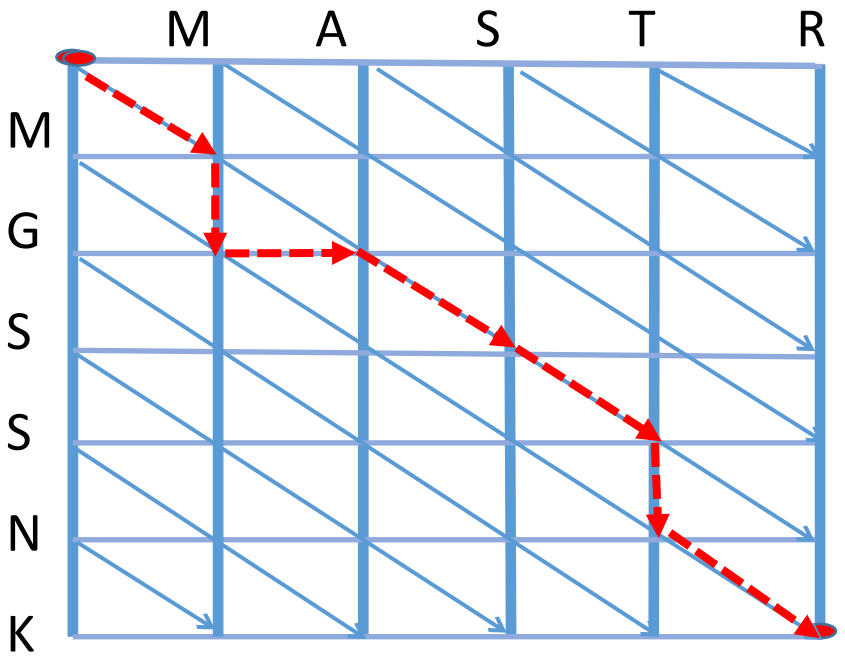
- Мы можем получить выравнивание k-го префикса первой последовательности с l-ым префиксом второй тремя способами:
  - взять **любое** выравнивание (k–1)-го префикса первой с (l–1)-ым префиксом второй и добавить колонку из k-ой буквы первой против l-ой буквы второй
  - взять любое выравнивание (k–1)-го префикса первой с l-ым префиксом второй и добавить колонку из k-ой буквы первой против **гэпа**
  - взять любое выравнивание k-го префикса первой с (l–1)-ым префиксом второй и добавить колонку из l-ой буквы второй против **гэпа**

```
M-AST-R     =   M-AST-  +  R
MG-SSNK         MG-SSN     K

M-AST--R    =   M-AST--  +  R
MG-SSNK-        MG-SSNK     -

M-AST-R-    =   M-AST-R  +  -
MG-SSN-K        MG-SSN-     K
```

Выравнивание префиксов

- Мы можем получить выравнивание k-го префикса первой последовательности с *l*-ым префиксом второй тремя способами:
  - взять **любое** выравнивание (*k*–1)-го префикса первой с (*l*–1)-ым префиксом второй и добавить колонку из *k*-ой буквы первой против *l*-ой буквы второй
  - взять любое выравнивание (*k*–1)-го префикса первой с *l*-ым префиксом второй и добавить колонку из *k*-ой буквы первой против **гэпа**
  - взять любое выравнивание *k*-го префикса первой с (*l*–1)-ым префиксом второй и добавить колонку из *l*-ой буквы второй против **гэпа**
- Вес каждого такого выравнивания получается из веса выравнивания меньших префиксов прибавлением веса сопоставления букв либо вычитанием штрафа за гэп.
  Из трёх вариантов можно выбрать оптимальный.
- Последовательно запомним для **всех** пар префиксов (начиная с пары пустых) оптимальный вес выравнивания и каким способом мы его получили. В конце получим оптимальный вес выравнивания последовательностей в целом.
- Время работы пропорционально произведению длин последовательностей

Выравнивание — это почти путь по Манхэттену



M-AST-R

MG-SSNK

- Каждому перекрёстку соответствует пара префиксов
- Путь превращается в выравнивание согласно примеру
- Вес диагонали берется из матрицы весов
- Вес горизонтальной и вертикальной стрелки равен штрафу за гэп
- Выравнивание восстанавливается обратным проходом из правого нижнего угла в левый верхний.

Аффинные штрафы за гэпы

- Предыдущая картинка — для **линейных** штрафов за гэпы
- Чтобы учесть аффинные штрафы, нужно в каждый перекрёсток поместить не одну сущность (оптимальное выравнивание префиксов), а три:
  - лучшее из выравниваний, кончающихся сопоставлением букв
  - лучшее из выравниваний, кончающихся гэпом в первой последовательности
  - лучшее из выравниваний, кончающихся гэпом во второй последовательности

Алгоритм Смита – Уотермэна

- Отличается от алгоритма Нидлмана – Вунша следующими особенностями:
  - в узел ставится только **неотрицательный** вес
    (вес оптимального локального выравнивания двух префиксов не может быть отрицательным, потому что вес пустого выравнивания равен 0);
  - если оптимальный вес оказался отрицательным, заменяем его на 0, а вместо стрелки запоминаем, что ничто хорошее здесь не кончается
    (лучшее из локальных выравниваний, кончающихся здесь, пустое)
  - когда найдём максимальный вес для всех пар префиксов, начинаем обратный проход не с правого нижнего угла, а с **лучшего** из локальных выравниваний префиксов;
  - заканчиваем обратный проход, когда доходим до узла без стрелки.

Про множественные выравнивания

Для множественных выравниваний (число последовательностей больше двух) алгоритмы, основанные на динамическом программировании, на практике не применяются.

Дело в том, что время работы таких алгоритмов пропорционально произведению длин **всех** входных последовательностей, что, как правило, слишком много.

Поэтому существует множество **эвристических** алгоритмов множественного выравнивания

(Эвристических — значит не решающих никакой точно поставленной формальной задачи, вроде нахождения выравнивания с максимальным весом)

Самые популярные из них вы можете видеть в меню программы

Jalview: Web service → Alignment

На kodomo из командной строки доступны muscle, mafft, prank, edialign, emma.

(edialign — это "EMBOSS-обёртка" программы Dialign, а emma — программы ClustalW)

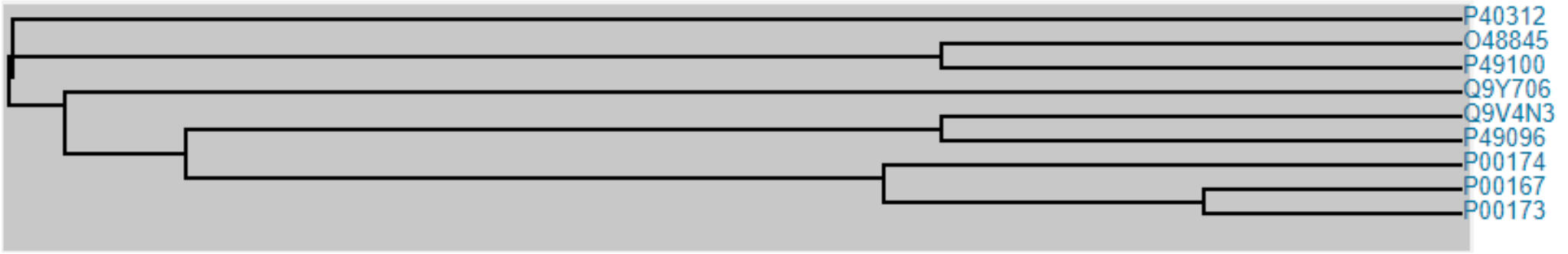
Прогрессивное множественное выравнивание

- Наиболее простая из эвристик для множественного выравнивания — прогрессивное выравнивание
- Самый простой и наивный способ:
  - выровняем первую последовательность со второй
  - подравняем третью последовательность к готовому выравниванию (это можно сделать алгоритмом Нидлмана – Вунша, придумав разумный вес сопоставления колонки выравнивания и буквы)
  - затем четвёртую, пятую и т. д
- Это плохо тем, что ошибки начального этапа уже не исправить
- Можно по крайней мере их минимизировать, для этого первыми надо выравнивать самые близкие последовательности

Прогрессивное множественное выравнивание

- Сначала оцениваем расстояния между последовательностями
  - в старой программе ClustalW это делалось попарными выравниваниями всех со всеми
  - в более новых программах (Muscle, ClustalO, …) это делается отдельными быстрыми алгоритмами
- Строим т. н. «направляющее дерево» — guide tree
- Выравниваем сначала пару самых близких последовательностей, а затем, шаг за шагом — пару самых близких из уже сделанных выравниваний

Направляющее дерево



Улучшение созданного выравнивания

- Делим множество последовательностей готового выравнивание на два подмножества
- Не меняя соответствие букв внутри частей, перевыравниваем между собой эти две части алгоритмом Нидлмана – Вунша
  (в каждую часть может быть вставлен только общий для всей части гэп)
- Повторяем, пока общее качество выравнивания не перестаёт улучшаться

Пример выдачи программы Muscle

```
MUSCLE v3.8.31 by Robert C. Edgar
http://www.drive5.com/muscle
This software is donated to the public domain.
Please cite: Edgar, R.C. Nucleic Acids Res 32(5), 1792-97.

tmp 12 seqs, max length 335, avg  length 312
00:00:00      5 MB(0%)  Iter   1  100.00%  K-mer dist pass 1
00:00:00      5 MB(0%)  Iter   1  100.00%  K-mer dist pass 2
00:00:00      9 MB(0%)  Iter   1  100.00%  Align node
00:00:00      9 MB(0%)  Iter   1  100.00%  Root alignment
00:00:00      9 MB(0%)  Iter   2  100.00%  Root alignment
00:00:00      9 MB(0%)  Iter   3  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   4  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   5  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   5  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   6  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   7  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   8  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   8  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   9  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  10  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  11  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  12  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  13  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  13  100.00%  Refine biparts
```

Матрицы серии BLOSUM

Что мы хотим от матрицы аминокислотных замен?
Чтобы сопоставления букв, говорящие в пользу правильности выравнивания, вносили положительный вклад в вес, а говорящие против правильности выравнивания — отрицательный.

Матрицы серии BLOSUM

Что мы хотим от матрицы аминокислотных замен?

Чтобы сопоставления букв, говорящие в пользу правильности выравнивания, вносили положительный вклад в вес, а говорящие против правильности выравнивания — отрицательный.

Как этого добиться?

Для каждой пары букв посчитаем, как часто они встречаются в **достоверно правильных** выравниваниях в **одной позиции**, обозначим частоту пары через q(a,b) (где a,b — две буквы)

(имеется в виду относительная частота, $0 \le q(a,b) \le 1$)

Теперь для каждой пары букв посчитаем, как часто они могут оказаться друг против друга **по случайным причинам**. Это просто произведение частот букв в банке, p(a)p(b).

Посчитаем **отношение правдоподобия** q(a,b)/p(a)p(b).

Если оно больше 1, то буквы чаще встречаются в гомологичных позициях, это аргумент в пользу их сопоставления. Если меньше 1, то наоборот.

Теперь элемент матрицы — это **логарифм** отношения правдоподобия**:** log(q(a,b)/p(a)p(b))

Матрицы серии BLOSUM

Где взять достоверно правильные выраваниния?

Steven Henikoff & Jorja Henikoff в 1992 г. придумали использовать для этого базу данных BLOCKS, коллекцию множественных **локальных** выравниваний **без гэпов**, созданную из надёжно гомологичных белков.

(Сама база ещё жива, но сейчас редко используется)

Они поняли, что выравнивания из базы BLOCKS **целиком** использовать плохо: в большинстве выравниваний очень много совпадающих и очень близких последовательностей, и их использование приведёт к недооценке веса сопоставлений разных, но близких по свойствам аминокислотных остатков.

Матрицы серии BLOSUM

Henikoff & Henikoff придумали **проредить** каждое выравнивание из BLOCKS, оставив в каждом выравнивании лишь последовательности, **попарно** имеющие менее N% одинаковых букв.

В результаты получилась серия матриц BLOSUM (BLOcks SUbstitution Matrix): BLOSUM30, BLOSUM35, …, BLOSUM100.

Число в конце названия матрицы означает порог на процент совпадений между парами последовательностей в «образцовых» выравниваниях.

В результате испытаний наиболее подходящей для выравнивания и поиска по сходству была признана матрица BLOSUM62.

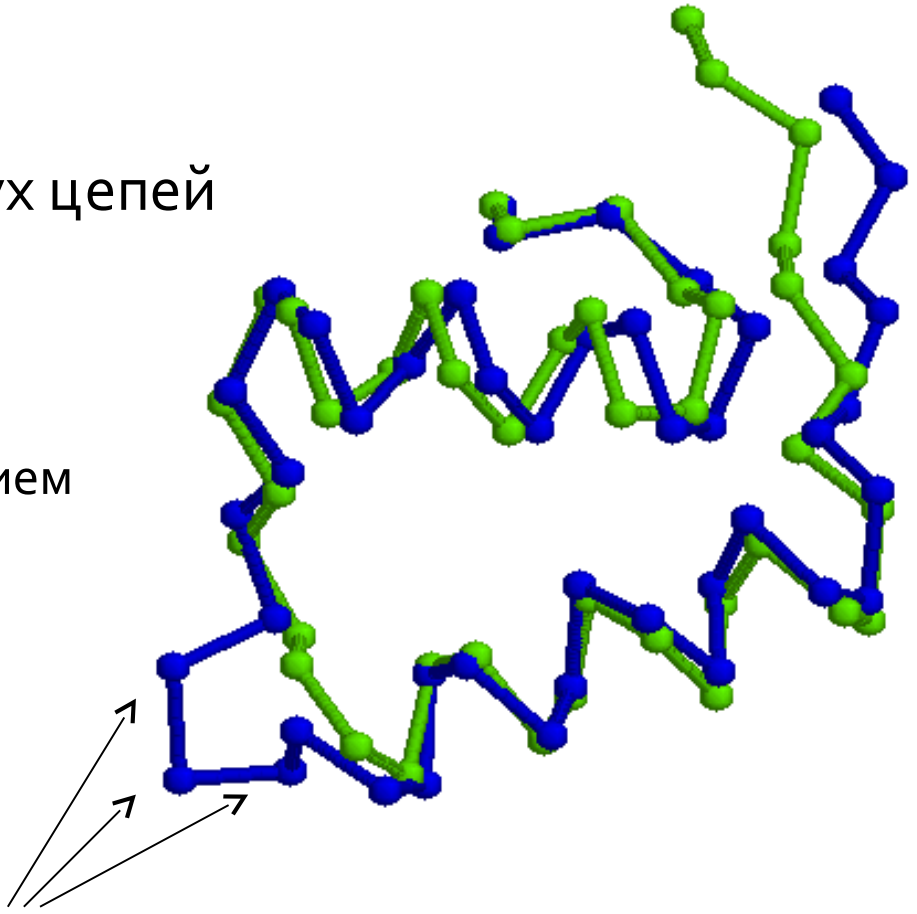
Все матрицы можно посмотреть здесь: ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/

В программах needle, water, stretcher, matcher можно задать матрицу, отличную от BLOSUM62, опцией -datafile, прибавив к названию матрицы спереди "E", например -datafile EBLOSUM50
В программе blastp это опция -matrix, E не нужно, например -matrix BLOSUM45

Выравнивание на основе сопоставления пространственных структур

Большая часть остатков двух цепей соответствуют друг другу.

Соответствие в данном случае определяется хорошим наложением структур в пространстве.



Этим Cα-атомам в «синей» структуре ничего не соответствует в «зелёной»

Сравнение выравниваний

Пусть есть набор последовательностей и пусть есть **два** выравнивания этого набора последовательностей.

Например, для двух последовательностей: оптимальное глобальное и оптимальное локальное.
Или оба оптимальные глобальные, но при разных параметрах вычисления веса.
Или оптимальное и неоптимальное:

- выданное BLAST“ом;
- полученное из множественного выравнивания путём ограничения на эти две последовательности;
- полученное путём анализа пространственных структур.

Для многих последовательностей: полученные разными программами множественного выравнивания (Muscle, MAFFT, Prank, Dialign, …).

Сравнение выравниваний

Пусть есть набор последовательностей и пусть есть **два** выравнивания этого набора последовательностей.

Сравнить выравнивания — значит найти сопоставления аминокислотных остатков (нахождение их в одной колонке), которые имеются в одном выравнивании, но отсутствуют в другом.

```
AFTGAHAYL          AFTGAHAYL
AYS---AYM          AY---SAYM
```

Например, в правом выравнивании есть сопоставление 6-ого H первой последовательности 3-му S второй, а в левом нет.
Вместо этого в левом выравнивании 3-й S второй последовательности сопоставлен 3-му T первой.