Институт проблем передачи информации

Российской академии наук

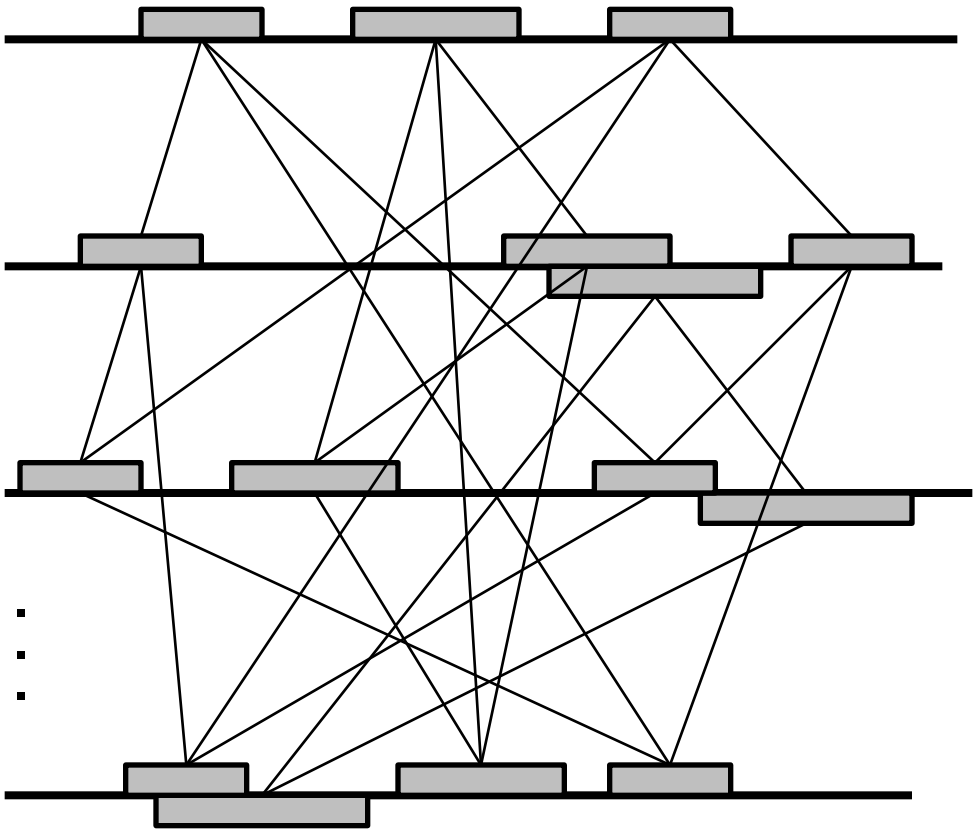
Лаборатория «**Математических методов и моделей в биоинформатике**»

сайт лаборатории: **http://lab6.iitp.ru/**

адрес: **lyubetsk@iitp.ru**

В. Любецкий, Л. Рубанов

Эффективный параллельный алгоритм для поиска приблизительно совпадающих слов в большом наборе длинных символьных последовательностей



4-клика

3-клика

4-плотный подграф (без 4- и 3-клик)

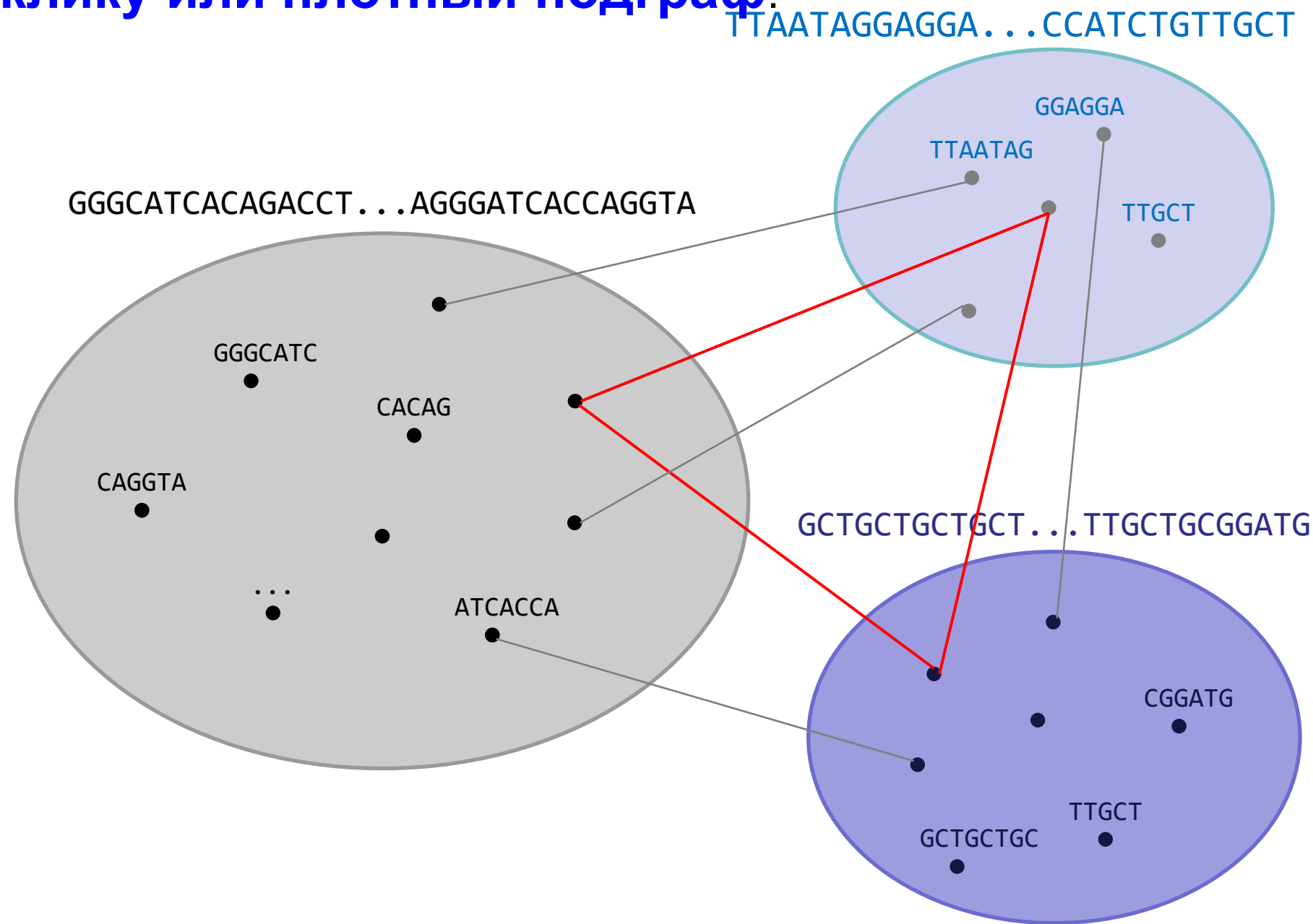
Одна последовательность соответствует одной доле графа. Поэтому долей столько, сколько последовательностей. Вершин в доле столько, сколько слов в последовательности.

Ребра проводятся между похожими словами из разных последовательностей (долей).

Задача - найти возможно большую клику в таком большом многодольном графе.

Менее трудная задача: найти большие плотные подграфы в большом многодольном графе.

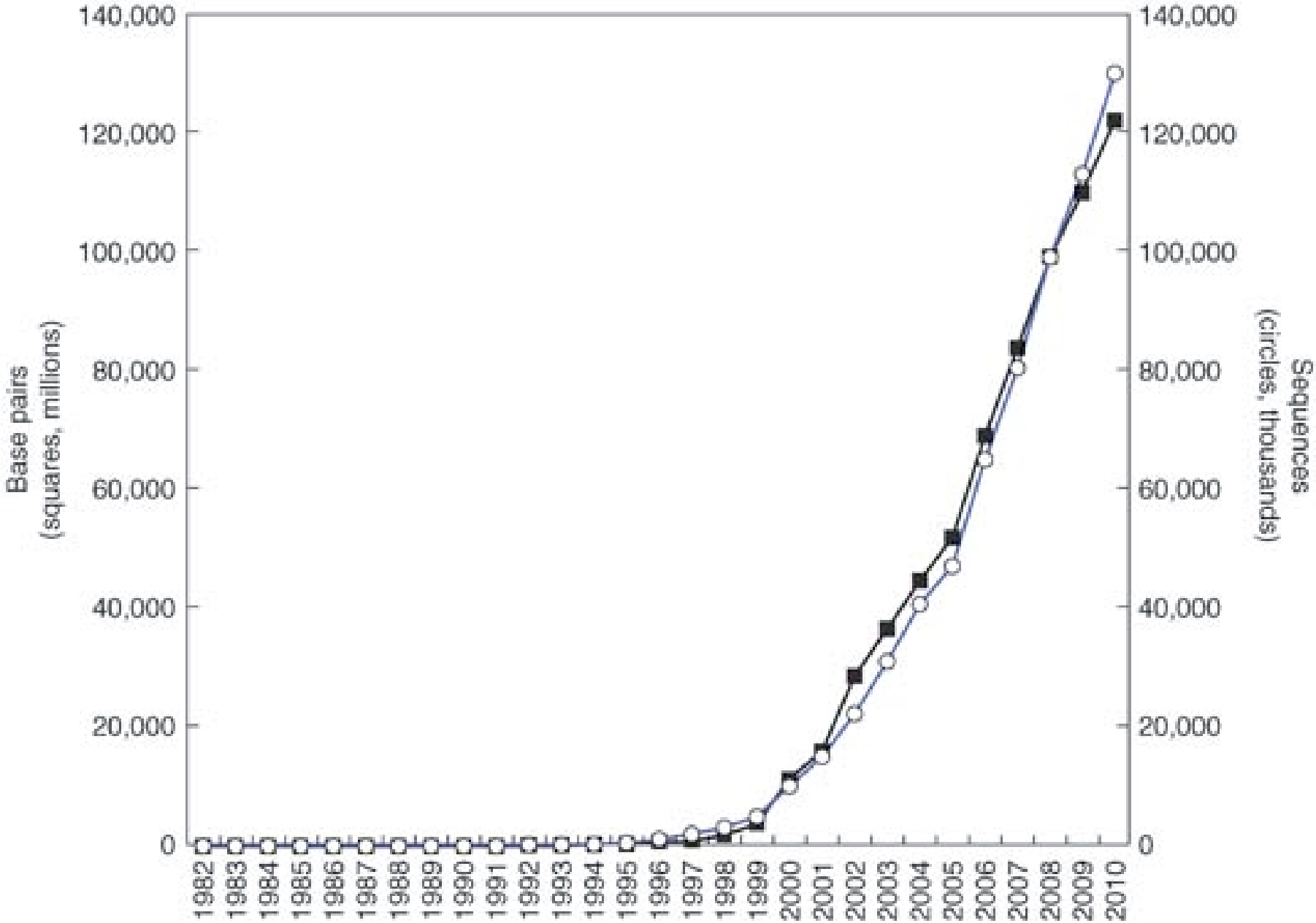
Пример **многодольного графа**, в котором ищем **клику или плотный подграф**:



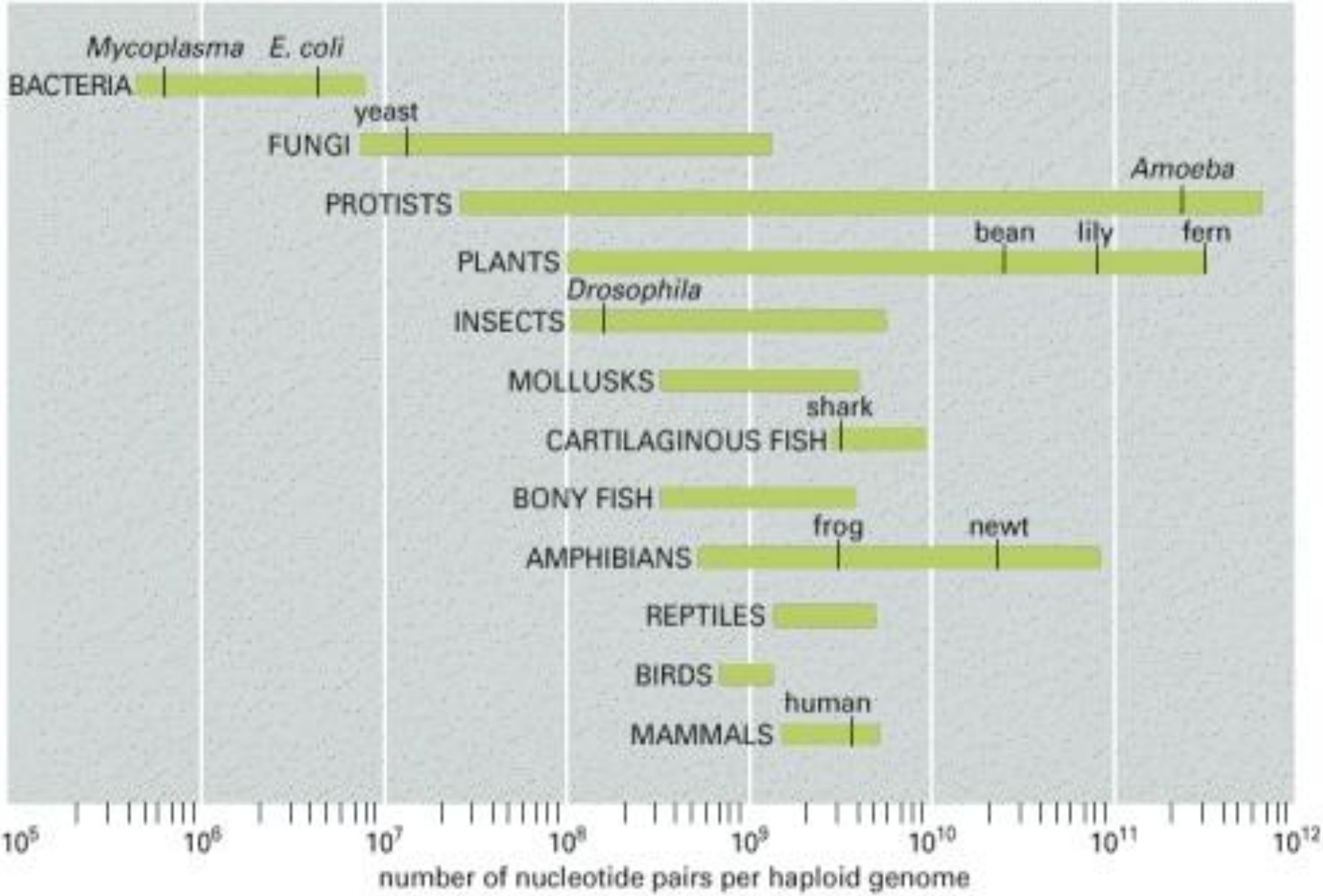
Какого размера графы реально возникают в прикладных задачах?

Как должна быть организована память и обмен между процессами у суперкомпьютера, решающего такие задачи?

Рост числа последовательностей в базе данных



Длины последовательностей



Итак, **о размере графа**. Может потребоваться: число долей 10^8, длина последовательности 10^9-10^{12}, берутся короткие слова, т.е. **вершин 10^{19}** и средняя степень вершины имеет порядок числа долей.

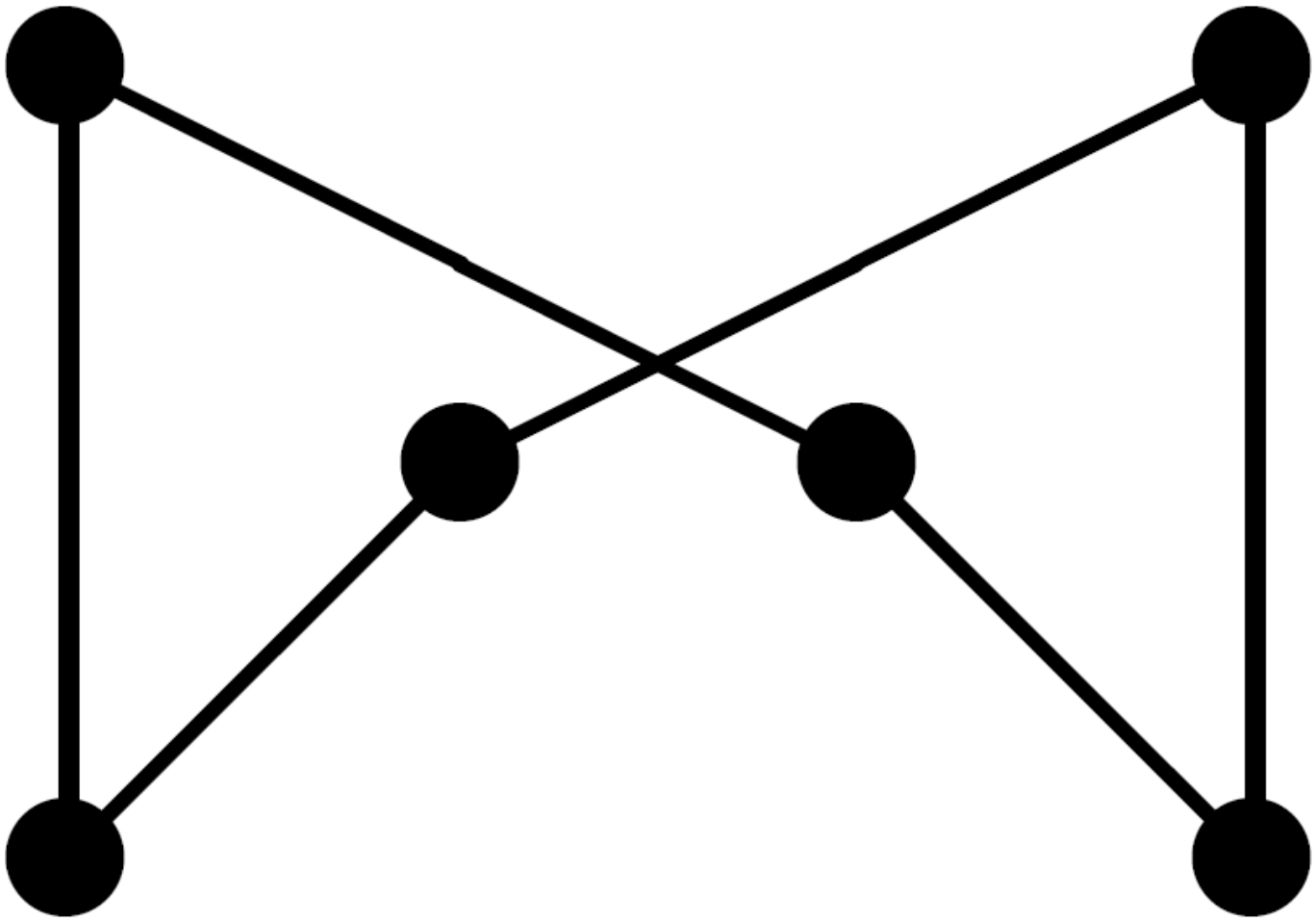
Кроме того, **задача поиска клики «неразрешима»**: поиск *m*-клики в *m*-дольном графе, если каждая доля содержит только по три вершины, – **NP-трудная задача**. Это показано для графа с очень высокой плотностью рёбер. Если плотность рёбер низкая, то сложность этой задачи не известна. **Мы предлагаем эффективный алгоритм решения «такой» задачи**.

Поэтому: вместо клики ищем **плотные подграфы**.

Подграф многодольного графа называется ***m*-плотным**, если он связный и каждая его **вершина соединена** ребром **с вершиной** в не менее чем (m-1)ой доле.

Представим себе, что m равно числу долей в графе.

Пример **3-дольного графа**,
он же – **3-плотный** граф
(в нём нет 3-клик):



Рёбрам приписывается **редакционное расстояние** слов на его концах; или иная мера их близости, сходства, например, **минимум/сумма длин** этих слов. Такое число называется **весом ребра**. Представляет интерес и случай, когда **все веса равны 1**.

КЛАСТЕР – *m*-плотный и массивный (т.е. сумма весов его рёбер велика) подграф в **исходном графе**; часто добавляют ещё требования: подграф не содержит много вершин из одной доли и сами слова высокой сложности (=слабо сжимаемые)

Итак, **наш метод** состоит из:

1) **алгоритма построения исходного графа**

(тут есть проблема, как его строить!)

И

2) **алгоритма поиска плотных подграфов**

(кластеров) в исходном графе

(найден хороший, но нужен его

теоретический анализ!)

Описание метода. **1)** Выделить **пары слов** высокой сложности (например, слабо сжимаемые алгоритмом Лемпеля-Зива), которые соответствуют друг другу в смысле прикладной задачи (типично: **похожие и длинные**).

2) Построить сам **исходный граф**, его долям приписаны последовательности, вершинам доли – слова (или наборы близких слов) из одной последовательности, соответствующей доли; рёбрам – веса.

3) В исходном графе **найти плотные подграфы**.

Алгоритм поиски подграфа. В каждой вершине графа алгоритм независимо и параллельно выполняет:

1) Если вершина соединена рёбрами менее чем с (m–1) долей, то она и инцидентные ей рёбра удаляются. Если вершина соединена с какой-то долей лишь одним ребром, то ребро помечается (оно удаляется только вместе с одним из его концов)

Если граф изменился, то снова выполняется вышесказанное.

2) Если ребро инцидентно вершине, не помечено и его вес строго меньше весов всех других таких её рёбер, то ребро удаляется. (= т.е. слабые ребра удаляются)

Если после выполнения шага 2 в графе произошли изменения, то возвращаемся к шагу 1.

В противном случае алгоритм заканчивает работу. ///

Каждая компонента связности графа – искомый m-плотный подграф (кластер). ///

О программировании (кратко): 1) После каждого шага выполняется синхронизация процессов во всех вершинах графа: следующий шаг начинается после окончания предыдущего шага во всех вершинах.

2) Подразумевается некоторый межпроцессный обмен: если на другом конце ребра, инцидентного данной вершине, это ребро «удалено», то оно «удаляется» и в ней; если помечено, то «помечается» и в ней.

Представьте себе работу клеточного автомата: «клетка»=вершина, «соседние клетки» = смежные вершины.

Наш опыт показал, что параллельный алгоритм, работающий таким образом, завершает работу **после небольшого числа итераций даже для графов огромных размеров**.

Почему? Число итераций не превышает диаметра наибольшей связной компоненты,
по ходу алгоритма компоненты могут только распадаться.

Здесь было бы интересно провести статистически значимое тестирование. Как его организовать?
Конечно, важно и теоретическое исследование, хотя бы асимптотическое!

Вернёмся к **проблеме построения исходного графа**. В зависимости от задачи различают два случая:

слова нефиксированной или фиксированной **длины**.

Первый случай сложнее, и мы расскажем именно о нём.

Для второго случая нами найден другой, чем для первого случая, эффективный алгоритм.

Итак, для **построения исходного графа**:

1. Для каждой пары последовательностей **находим в них пары максимально длинных слов со сходством выше порога** (и соединяем их ребром), **весом** ребра служит **сумма длин найденных слов**. Получается огромное множество рёбер, фактически не образующее граф.

2. В каждой последовательности **склеиваем соседние вершины в новые вершины** (образуя множества старых вершин в качестве новой вершины). Результат – **искомый граф**.

Пример: 14 долей (споровики), 64*(2GHz CPU) сервер.

Этап 1 (1200 cpu·h): построено 43.2 млн рёбер, соединяющих 86.4 млн вершин, степени вершин 1

Этап 2 (85 cpu·h): построен исходный граф: 32.5 млн рёбер, соединяющих 3 млн вершин, средняя степень вершины 21.7

Этап 3 (300 cpu·h): поиск 3-плотных и выше кластеров; декомпозиция графа: 20.5 млн рёбер и 1 млн вершин, средняя степень вершины 39.
После 22 итераций алгоритма найдено 2918 кластеров. Из них для ***m*=14 получен 1 кластер**, для 13 – 3, 12 – 1, 11 – 3, 10 – 6, 9 – 11, 8 – 13, 7 – 29, 6 – 35, 5 – 80, прочие (*m*=4 и 3) – 2734 кластеров. Остаток – кластеры с *m*=13…7 в сумме **содержат всего 619 вершин**.

Как **найти нужные пары слов** – рёбра

(это мы умеем делать)

и

как **склеить старые вершины в новые**

(это в значительной мере

остаётся проблемой)

При достаточно большом ε имеется такое число *k*, 16…24 < *k* < *l*, что в любой пригодной для нас паре слов имеется одно и то же подслово с длиной ≥ *k*. **Это позволяет искать пары слов в любых двух последовательностях, расширяя найденное слово длины *k* («ключ»)**. **Итак**,

1) По первой последовательности составить хэш-таблицу из всех ключей с их позициями. Хэш-таблица занимает порядка 4-16 гигабайт для последовательностей длиной до 120 млн.

2) Во второй последовательности перебрать все ключи и для каждого из них проверить, **совпадает ли он с каким-то ключом** из хэш-таблицы. И т.д.

A. Для каждой последовательности склеиваем **вершины с одним и тем же центром** слова (или с близкими центрами): **новый вес** вершины равен наименьшему из весов склеиваемых вершин.

B. Присваиваем всем полуребрам вес вершины, из которой они выходят.

C. Для каждого ребра: если два его полуребра имеют разный вес, то больший вес заменяем на меньший.

D. Если вес полуребра меньше веса его вершины, то вес вершины заменяем на вес полуребра.

Повторяем шаги **B…D**, пока что-нибудь меняется.

Работа выполнена совместно с А. Селиверстовым.

Поддержана РНФ
(проект 14-50-00150)

СПАСИБО за внимание

Возникающие при склейке вершин **кратные рёбра заменяем одним ребром** с наименьшим весом среди всех ребер между двумя новыми вершинами.

В результате описанных шагов веса всех вершин и рёбер в пределах связной компоненты одинаковы и приближенно равны удвоенной длине искомого слова в вершинах этой компоненты связности.

Рассматривались случаи $k=32$, $l=150$ и $k=16$, $l=60$ и т.д. с порогом редакционного расстояния от 4 до 17.

Это пока НЕ делается, потому что нарушает результат на предыдущем слайде! Тут надо думать и пробовать.

Для каждой последовательности **соседние по центрам вершины** склеиваются, если расстояние на 1 бук между центрами слов меньше взвешенного среднего весов их рёбер.

Новый центр – центр объединения соответствующих слов

Новый вес вершины – мин весов удвоенная сумма весов без расстояния между центрами. **Новый вес** ребра – вес вершсумма весов на концах.

Делаем это в порядке возрастания центров слов с повторением текущей позиции. Из новой вершины выходят **кратные** рёбра с новыми весами.

Эффективное вычисление редакционного расстояния для слов с заранее не фиксированными длинами:

для этого найден линейной сложности алгоритм (полулокальное выравнивание с одним закреплённым концом).

Алгоритм полулокального выравнивания в одну сторону от ключа:

	0	1	2	3	4	5	6	7	8	9	10
0	0,0	0,1	0,2								
1	1,0	1,1	1,2	1,3							
2	2,0	2,1	2,2	2,3	2,4						
3		3,1	3,2	3,3	3,4	3,5					
4			4,2	4,3	4,4	4,5	4,6				
5				5,3	5,4	5,5	5,6	5,7			
6					6,4	6,5	6,6	6,7	6,8		
7						7,5	7,6	7,7	7,8	7,9	
8							8,6	8,7	8,8	8,9	8,10
9								9,7	9,8	9,9	9,10
10									10,8	10,9	10,10
11										11,9	11,10

DIAG UP
LEFT

F и P

0,2	1,3	2,4	3,5	4,6	5,7	6,8	7,9	8,10		
0,1	1,2	2,3	3,4	4,5	5,6	6,7	7,8	8,9	9,10	
0,0	1,1	2,2	3,3	4,4	5,5	6,6	7,7	8,8	9,9	10,10
1,0	2,1	3,2	4,3	5,4	6,5	7,6	8,7	9,8	10,9	11,10
2,0	3,1	4,2	5,3	6,4	7,5	8,6	9,7	10,8	11,9	

	0	1	2	3	4	5	6	7	8	9	10
0	3	8	13	18	23	28	33	38	43		
1	1	6	11	16	21	26	31	36	41	46	
2	0	5	10	15	20	25	30	35	40	45	49
3	2	7	12	17	22	27	32	37	42	47	50
4	4	9	14	19	24	29	34	39	44	48	

B и Q

ULEFT UP
LEFT
DLEFT DOWN

- закреплён один конец каждого слова, другой конец заранее неизвестен;
- матрица преобразуется (D - максимум делеций *подряд*; 2)
- вычисления проводятся по столбцам преобразованной части матрицы в порядке нумерации на рисунке, поэтому их можно остановить, когда минимальное расстояние в столбце достигает заданного предела.
- имеет сложность $O(DM)$ (**линейный**, если $D << M$);

Задача позволяет ограничиться маленькими долями

Тогда кроме указанного алгоритма нами предложен **альтернативный алгоритм**: **разбиение остовного дерева на поддеревья**. Кластер – множество слов, приписанных листьям в финальном наборе деревьев

Исходный граф строится проще: локальным выравниванием всех «специальных слов» (белков из данной последовательности, методом blast).

После чего, оставляем ребра с весами выше порога.

Стандартный алгоритм глобального выравнивания

Needleman, Wunsch (1970)

a b →		G	C	G	T	T	G
↓	0	2	4	6	8	10	12
A	2	1	3	5	7	9	11
C	4	3	1	3	5	7	9
G	6	4	3	1	3	5	7
C	8	6	4	3	2	4	6
T	10	8	6	5	3	3	5
T	12	10	8	7	5	3	4
G	14	12	10	8	7	5	**3**

Инициализация:

$$F[0][0] = 0;$$

$$F[i][0] = id,\ P[i][0] = \mathrm{UP},\ i = \overline{1, M};$$

$$F[0][j] = jd,\ P[0][j] = \mathrm{LEFT},\ j = \overline{1, N}$$

Заполнение матриц:

$$F[i][j] = \min \begin{cases} F[i-1][j-1] + g(i,j) & (\mathrm{case1}) \\ F[i-1][j] + d & (\mathrm{case2}) \\ F[i][j-1] + d & (\mathrm{case3}) \end{cases}$$

$$\text{где}\ g(i,j) = \begin{cases} 0, a[i] = b[j] \\ 1, a[i] \neq b[j] \end{cases};\ d = 2\ (\text{пример})$$

$$P[i][j] = \begin{cases} \mathrm{DIAG}\ \textbf{if}\ (\mathrm{case1}) \\ \mathrm{UP}\ \textbf{if}\ (\mathrm{case2}) \\ \mathrm{LEFT}\ \textbf{if}\ (\mathrm{case3}) \end{cases}$$

Редакционное расстояние: $F[M][N] = \mathbf{3}$

Обратный ход: **GCGCTTG**

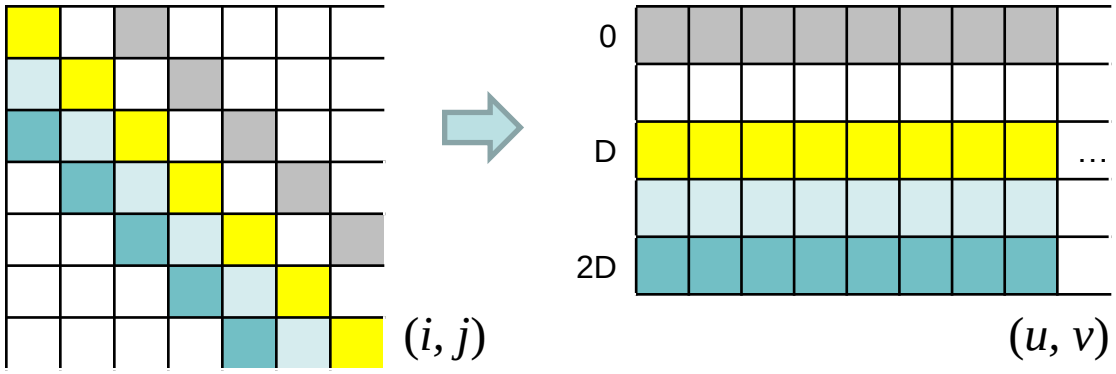
ACG–TTG

Сложность $O(MN)$, память $O(MN)$

Недостатки стандартного алгоритма:

- глобальный (нужно точно знать начало и конец обоих участков),
- трудоемкий (квадратичный),
- нельзя ограничить вычисления заданным пределом расстояния.

Выполним преобразование части матриц *F* и *P*:

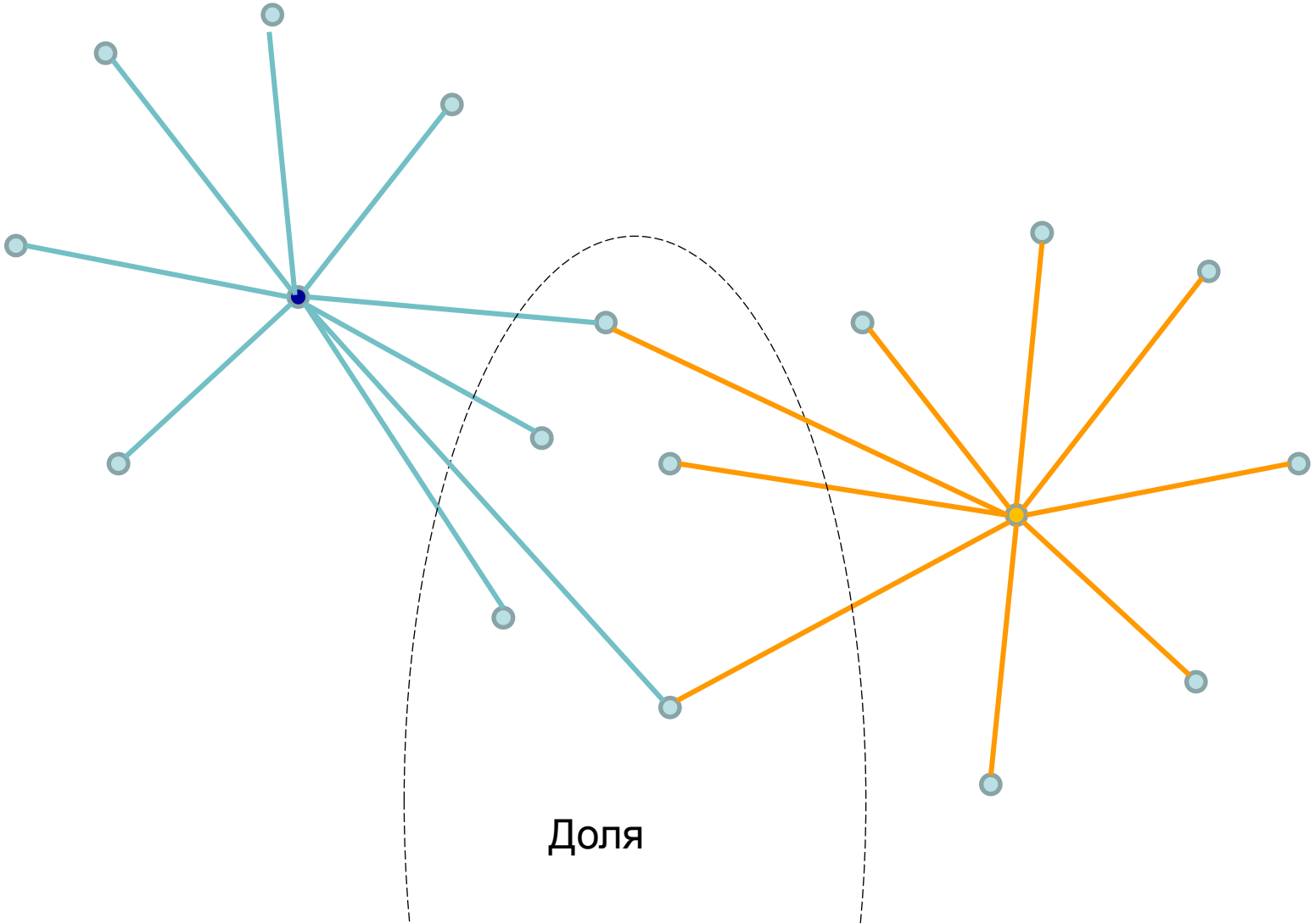


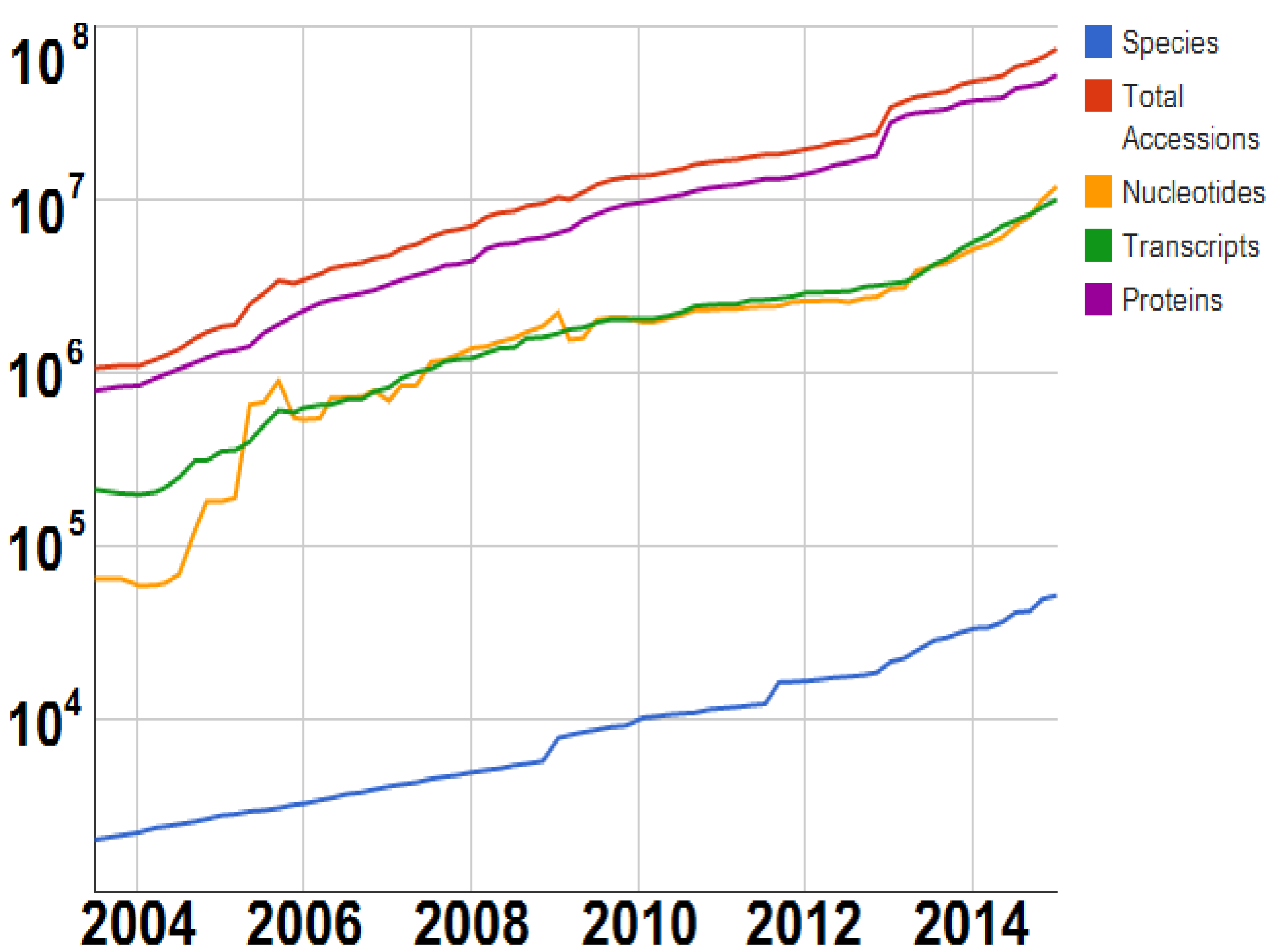
$$i = v + (u - D)^{+}$$

$$j = v + (D - u)^{+}$$

т.е. рассматриваем главную диагональ матрицы и *D* соседних с ней диагоналей с каждой стороны

Вместо отдельных рёбер эффективнее рассматривать звёзды со многими откликами, и склеивать отклики из одной доли: **??**




10^8
10^7
10^6
10^5
10^4
2004
2006
2008
2010
2012
2014
Species
Total Accessions
Nucleotides
Transcripts
Proteins

Нами предложен **алгоритм поиска кластеров**, допускающий **эффективное распараллеливание**.

Однако для применения алгоритма нужен **исходный граф**: его построение – самостоятельная трудная проблема.

Наш алгоритм поиска кластеров универсальный, но **построение исходного графа** зависит от задачи, которая решается.

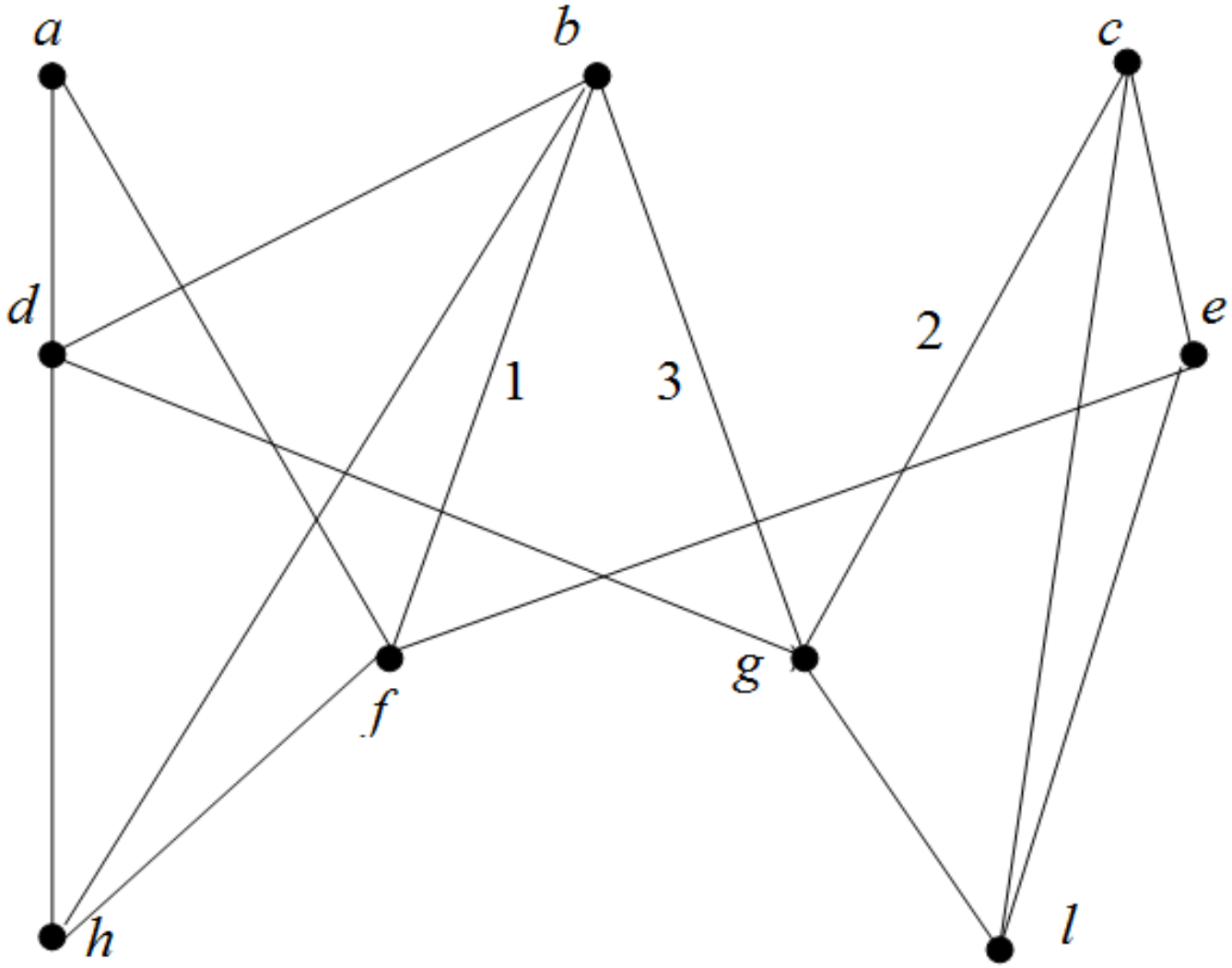
<u>Итак</u>,

сначала нашим методом строится **исходный граф**;

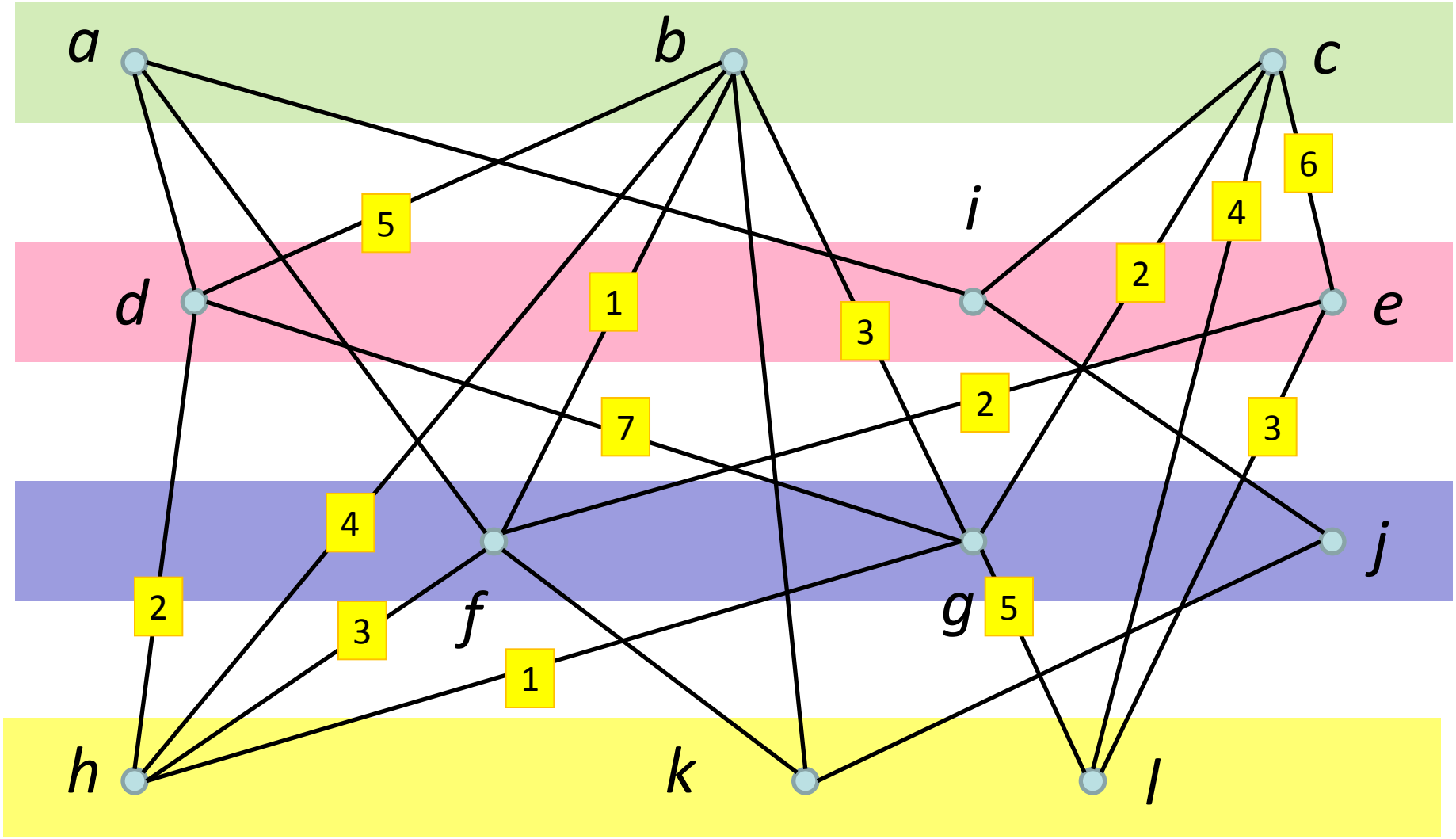
затем применяется наш **универсальный алгоритм поиска кластеров**.

Начнём с **изложения самого алгоритма поиска кластеров**.

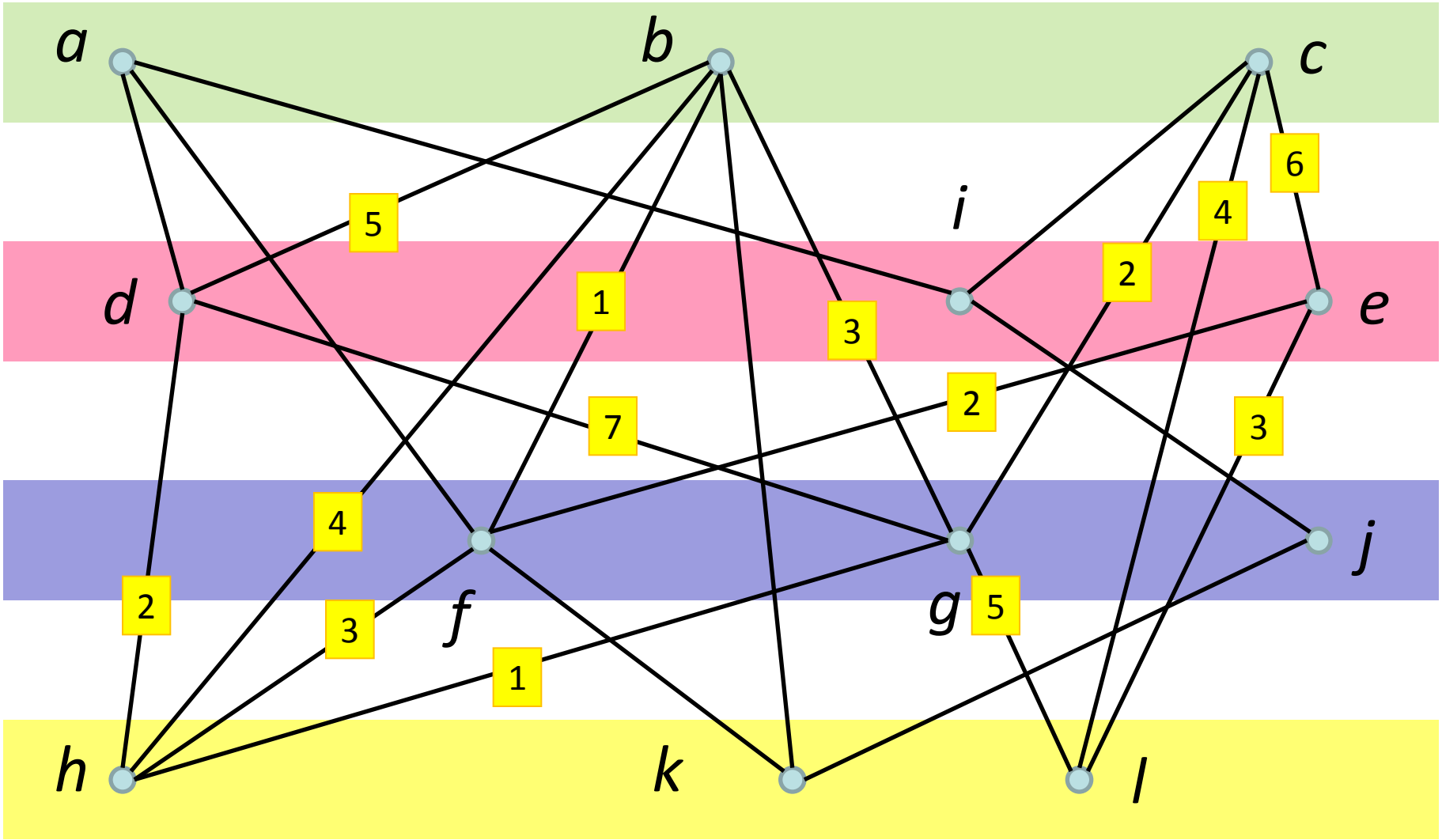
4-дольный граф, содержащий **кластер**: 4-плотный подграф, он состоит из всех вершин, кроме а. Ребра помечены весами (показана часть весов).



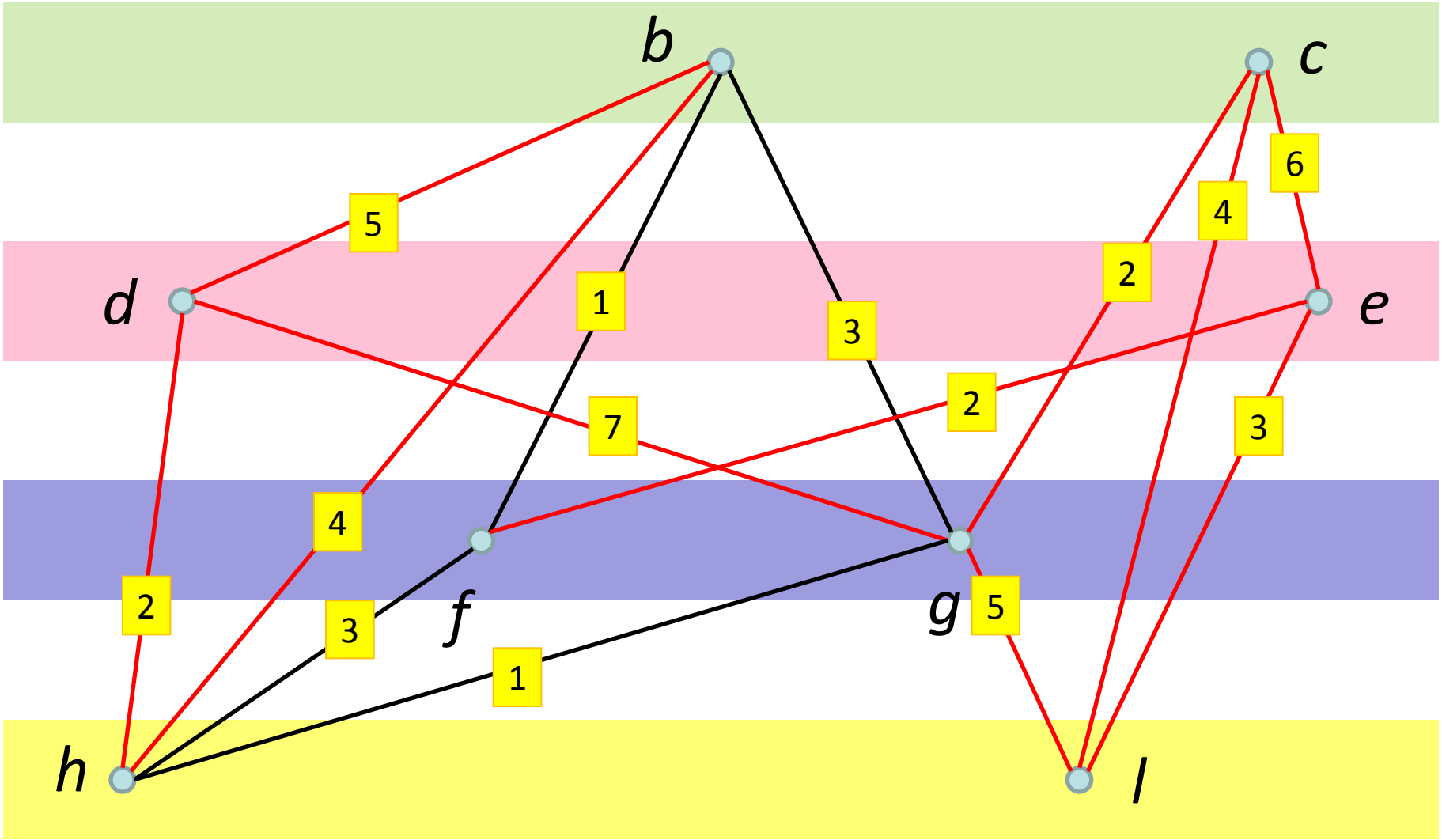
Пример: 4-дольный граф, содержащий 4-плотный подграф. Он состоит из всех вершин, кроме *a, i, j, k*. Некоторые рёбра помечены своими весами.



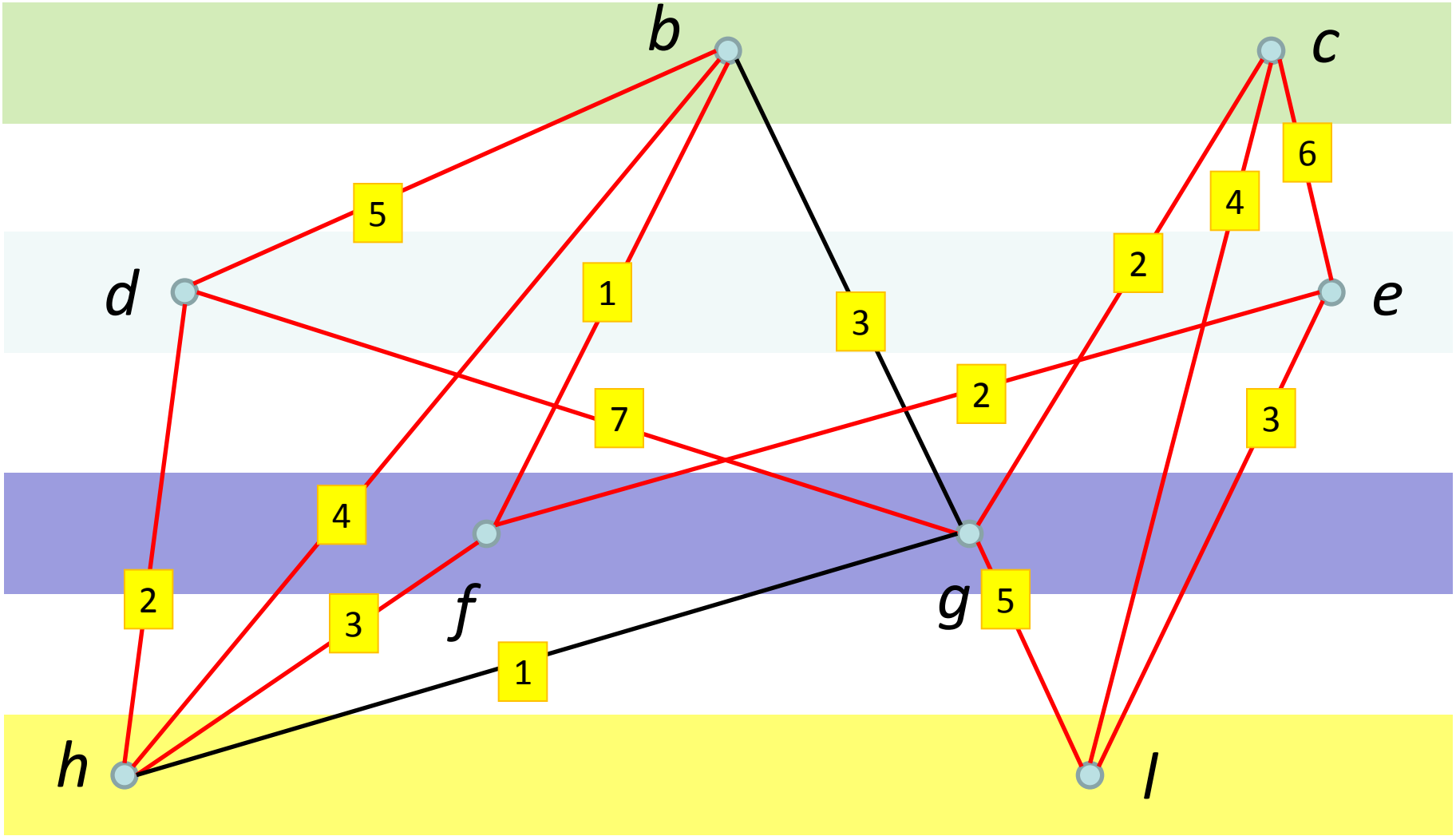
Шаг 1.1: Удаляются вершины *a*, *i*, *j*, *k* со своими рёбрами. Часть других рёбер помечаются как «защищенные» (показаны красным).



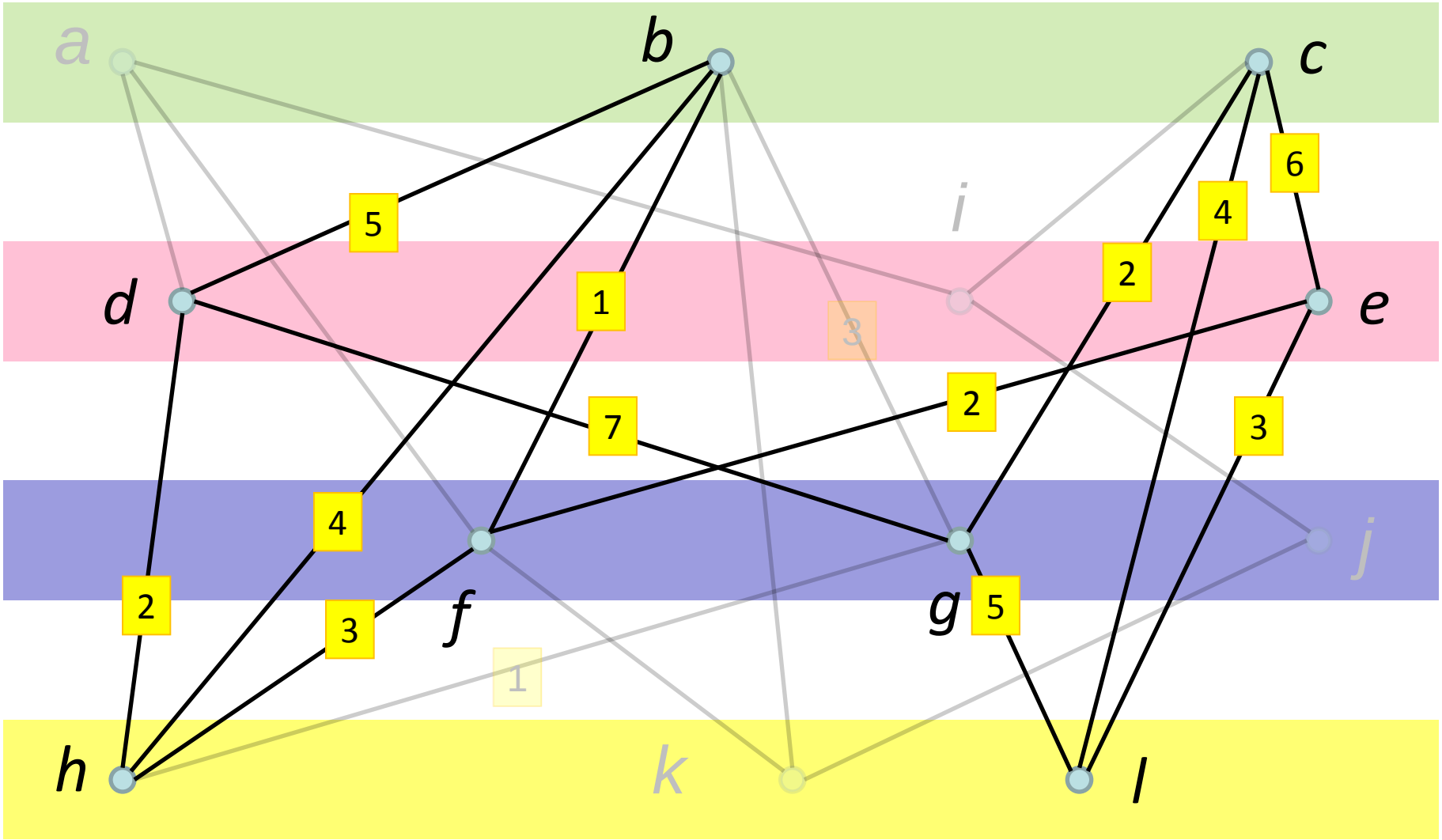
Шаг 1.2: Дополнительно рёбра *fb* и *fh* помечаются как «защищенные» (а *bg* и *gh* – нет). После этого шаг 1 больше не применим.



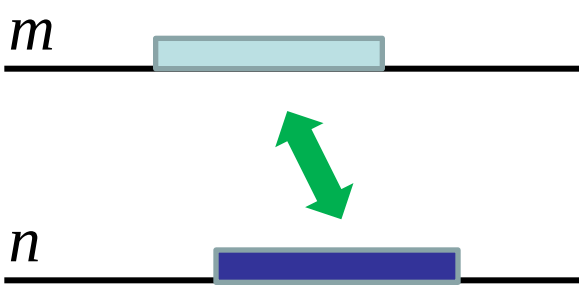
Шаг 2.1: Удаляется незащищенное ребро *hg* при вершине *h* и ребро *bg* при вершине *b* (но только *gh* при вершине *g*). После этого шаги 1 и 2 больше не применимы. Конец.



Кластер: 4-плотный подграф, построенный алгоритмом:



Этап 1. Даны две последовательности длиной *m* и *n*. Найти все **пары слов** максимальной длины (более *l*), имеющих сходство выше заданного ε и высокую сложность (например, сжимаемые алгоритмом Лемпеля-Зива с коэффициентом менее *r*).



Сложность наивного алгоритма $O(mnl^2)$ – неприемлема для поиска таких пар слов, тем более, сразу во многих парах последовательностей (тогда нужно ещё умножить на N^2).

Необходим очень быстрый алгоритм. **Приведём его**.

3) Расширить (вправо и влево) каждую найденную пару «ключ – ключ», чтобы найти **максимальные по длине слова**, для которых сходство больше порога ε, а сложность выше соответствующего порога.

Эти слова (= максимальные расширения) соединяются ребром с весом равным сумме длин слов и так **образуется искомое ребро**.

4) Продолжаем так с каждой следующей последовательностью, <u>используя построенную таблицу для 1й последовательности (одну и ту же)</u>.

5) То же самое делаем для 2й последовательности против списка всех оставшихся последов-тей; и т.д.

Получаем **<u>исходный набор рёбер</u>**. **<u>Конец этапа 1</u>**.

Вычислительная сложность шагов 1-3: $O((m+n)\cdot l^2)$ <u>и даже линейная</u>, т.к. совпадение ключей длины 24 и более в двух последовательностях встречается достаточно редко, типично < 10% ключей.

Память: $O(m\cdot k+n)$. Расширение на другие последовательности <u>линейно</u>.

Этап 2. Построение исходного графа

В каждой **вершине** (в ней слово) запоминаем:

центр вершины = **позицию** середины слова

И

вес каждого ребра из этой вершины =

сумма длин слов на концах ребра

И

вес вершины = **минимальный** вес ребра,

выходящего из неё.