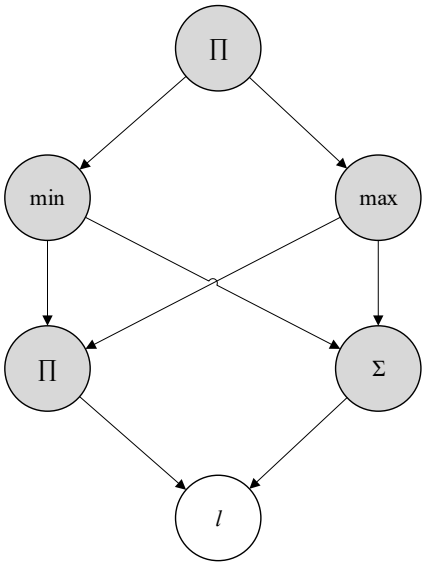
Варианты заданий повышенной сложности по курсу «Параллельное программирование»

Содержание

Задача 1. Декомпозиция и диспетчеризация задач в среде параллельных вычислений

Вычислительная задача представляется направленным ациклическим графом (в частном случае, деревом), в котором каждый узел представляет собой элементарную задачу, а связи между узлами означают передачу результатова решения одной задачи на вход другой. Например: пусть l – некоторый список чисел, тогда приведенный ниже граф изображает структуру следующей задачи: $\prod(\min(\sum l, \prod l), \max(\sum l, \prod l))$:



Направление стрелок показывает функциональную зависимость. По мере того, как вычисляются значения узлов, такие узлы исключаются из графа. Например, в приведенной схеме узел, отмеченный l, считается исключенным, т.к. его значение известно – это список l.

Это значение затем используют два различных узла – они отмечены $\prod$ и Σ; эти узлы считаются минимальными; поскольку они не сравнимы (если принять, что изображенная схема является отношением частичного порядка, а такое допущение приемлемо, т.к. выполнены, по крайней мере, свойства транзитивности и антисимметричности), – т.е. независимы – соответствующие вычисления можно выполнять параллельно, в различных потоках. Те же соображения справедливы по отношению к узлам, отмеченным min и max. Следует обратить внимание, что вычисление min и max (как и вообще любого узла) не может начаться, пока не вычислены значения для всех узлов, от которых они зависят (т.е. для меньших элементов). Таким образом, распараллеливаются всегда минимальные (в указанном смысле) элементы.

Задача на программирование.

1. Написать процедуру декомпозиции вычислительных задач. Входом для нее может быть, например, абстрактное синтаксическое дерево, подобное деревьям, возникающим в задаче 5. Выходом – направленный ациклический граф, упорядочивающий выполнение отдельных операций.
2. Реализовать диспетчер, которых доступные задачи (т.е. такие, которые являются на данный момент минимальными элементами) распределяет между исполняющими потоками. Возможны два варианта реализации:
    a. число потоков заранее фиксировано,
    b. количество потоков подстраивается под количество доступных задач.

Методические указания.

$$X = \{a, b, c, \dots\}$$

Частично упорядоченное:

$$\forall x, y \in X \left((x \le y \vee y \le x) \oplus \left(\neg (x \le y \vee y \le x) \right) \right)$$

$$\exists x, y \in X \left(x \ne y \wedge \neg (x \le y \vee y \le x) \right)$$

Топологическое упорядочение элементов X: $x_1, \dots, x_n$: $\forall i \in \overline{1, n-1} \left(\forall j \in \overline{i+1, n} \left(\neg x_j \le x_i \right) \right)$

$$\text{reduce}(f, l, c) \mapsto f \left(a_n, \left(f \left(a_{n-1}, \left(\dots f \left(a_2, \left(f(a_1 c) \right) \right) \right) \right) \right) \right)$$

$$f : T \to T \to T$$

$$f a b \Leftrightarrow a \text{ `f` } b$$

$$a_n \text{ `f` } a_{n-1} \text{ `f` } \dots \text{ `f` } a_1 \text{ `f` } c$$

$$(a \text{ `f` } b) \text{ `f` } c = a \text{ `f` } (b \text{ `f` } c)$$

$$\underbrace{a_n \text{ `f` } a_{n-1} \text{ `f`}}_{e_1} \text{ ` } \dots \text{ ` } \underbrace{f \text{ `} a_1 \text{ `f` } c}_{e_m} = e_1 \text{ `f` } \dots \text{ `f` } e_m$$

Задача 2. Построение решетки для множества точек в n-мерном пространстве

Пусть имеется некоторое n-мерное (например, вещественное) пространство $\boldsymbol{O}$. Пусть далее на этом множестве задано отношение частичного порядка: $x \leq y \leftrightarrow (x_1 \leq y_1) \wedge \ldots \wedge (x_n \leq y_n)$.

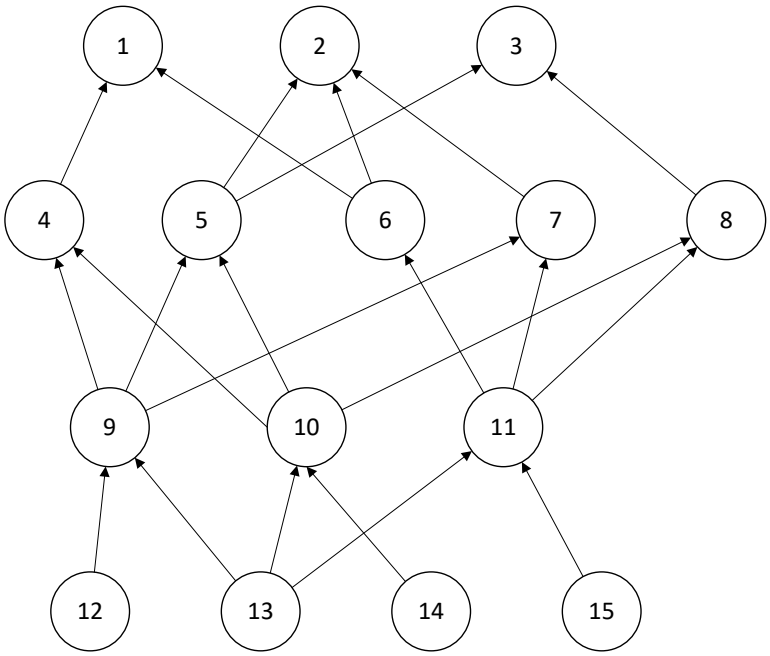
Примечание. Множество $\boldsymbol{O}$ является декартовым произведением некоторых $D_1 \times \ldots \times D_n$, $n \geq 1$. Единственное ограничение, которое накладывается на D_i – это чтобы D_i было полностью упорядоченным множеством, т.е. $\forall x, y \in D_i (x \leq y \vee y \leq x)$.

Пусть есть подмножество $X \subseteq \boldsymbol{O}$. Элемент $x \in X$ называется *максимальным* элементом этого множества ($x \in \max(X)$), если $\neg \exists y \in X (x \leq y)$. Аналогично определяется *минимальный* элемент множества. Минимальных и максимальных элементов у множества может быть несколько. Пусть $\downarrow x = \{y | y \leq x\}$ – множество элементов, расположенных ниже x, а $\uparrow x = \{y | x \leq y\}$ – множество элементов, расположенных выше x. Тогда $\max(\downarrow x)$ – это множество (точных) нижних оценок x, а $\min(\uparrow x)$ – множество (точных) верхних оценок x. Будем писать $y \ll x$, если $y \in \max(\downarrow x)$ и $x \ll y$, если $y \in \min(\uparrow x)$.

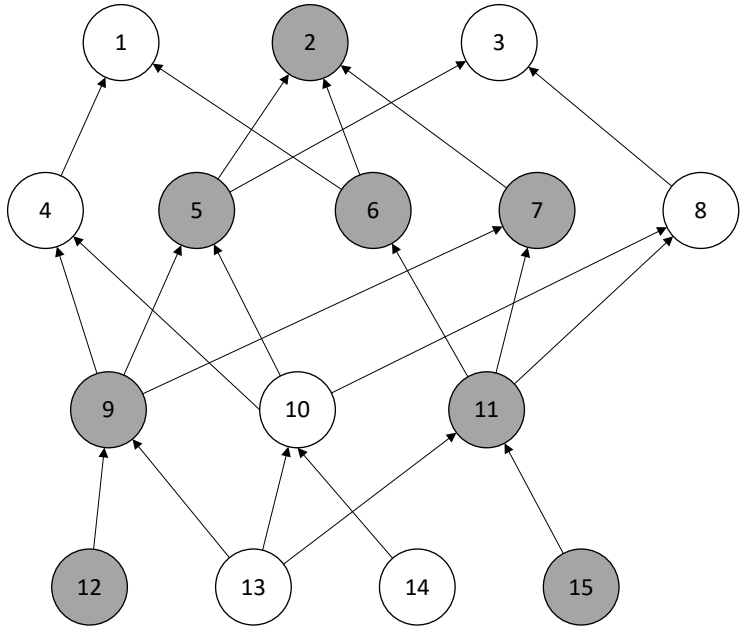
Пусть $\downarrow X = \{y | x \in X \rightarrow y \leq x\}$ – множество элементов, расположенных ниже множества X; пусть $\uparrow X = \{y | x \in X \rightarrow y \geq x\}$ – множество элементов, расположенных выше множества X. Тогда если $L, U \subset X$, то $\langle L, U \rangle = \{\downarrow U \cap \uparrow L\} \subset X$ – множество элементов между L и U.

Задача на программирование. Множество $X \subseteq \boldsymbol{O}$ порождается процедурно: например, это различные пути между двумя данными вершинами графа, сравниваемые по множеству критериев (расстояние, время в пути, денежная стоимость, трудозатраты и др.). Для такого множества требуется построить направленный ациклический граф, в котором вершинами являются элементы X, а дуга между элементами x и y имеется тогда и только тогда, когда $x \ll y$. По имеющемуся графу, найти для заданных L и U множество $\langle L, U \rangle$. Построенную решетку необходимо отобразить графически. При реализации полезно пользоваться абстракциями «полностью упорядоченное множество» и «частично упорядоченное множество».

Пример. Рассмотрим решетку, изображенную на рисунке ниже:



Пусть $L = \{12,15\}$, а $U = \{2\}$. Тогда $\langle L, U\rangle = \{2,5,6,7,9,11,12,15\}$:



Задача 3. Вычисление граничных поверхностей тел в 2-х и 3-мерном пространствах

Постановка задачи. Пусть имеется некоторое конечное множество $\boldsymbol{X^O}$ точек в -мерном (например, вещественном) пространстве $\boldsymbol{O}$ с введенной метрикой ρ, причем эти точки, с некоторой степенью условности, можно разделить на группы-кластеры произвольной конфигурации (в т. ч. эти кластеры не обязаны быть выпуклыми множествами, но, для простоты, будем считать, что это, по крайней мере, связные множества). Требуется для каждого кластера найти множество точек, образующих его границу.

Предположение. Две точки $x, y \in \boldsymbol{X^O}$ назовем *близлежащими*, если $\rho(x,y) < R$, где R – некоторый заранее фиксированный параметр. Отметим, что строгое неравенство технически предпочтительнее по сравнению с нестрогим. Множество точек $K \subseteq \boldsymbol{X^O}$ называется *кластером*, если $\forall x \in V\ \exists y \in V(\rho(x,y) < R)$, т.е. каждая точка, входящая в кластер, окружена некоторым количеством близлежащих точек, у каждой из которых, в свою очередь, тоже есть близлежащие. Пусть $K_1 \subset \boldsymbol{X^O}$ и $K_2 \subset \boldsymbol{X^O}$ $(K_1 \neq K_2)$ – два различных кластера; тогда они отделены друг от друга пространством ширины R или более: $\forall x_1 \in K_1 \forall x_2 \in K_2(\rho(x_1,x_2) \geq R)$. Последующие определения дают основание для алгоритма для выявления кластеров.

Зафиксируем некоторую «абсолютную» систему координатных осей $\boldsymbol{O^0_{XYZ}}$ (для простоты, в 3-мерном пространстве; однако все рассуждения несложно обобщить для случая произвольного n), в которой измеряются координаты всех точек данного рассматриваемого множества. Далее, через каждую точку $x \in \boldsymbol{X^O}$ можно провести систему координатных осей $\boldsymbol{O^x_{XYZ}}$, параллельную $\mathbf{O^0_{XYZ}}$ и разбивающую пространство на 8 $(= 2^n, n = 3)$ квадрантов $Q^x_1, \dots, Q^x_8$. Рассмотрим бинарный вектор $\vec{v}^x_y = (\mathrm{sg}(y_1 - x_1), \mathrm{sg}(y_2 - x_2), \mathrm{sg}(y_3 - x_3))$, где $\mathrm{sg}(x) = \begin{cases} 1, \text{если } x \geq 0 \\ 0, \text{иначе} \end{cases}$. Такой вектор позволяет однозначно определить, в каком квадранте относительно x лежит точка y. Например, можно положить, что $y \in Q^x_i$, если $i = 8 - \sum_{k=0}^{2} \vec{v}^x_{y,k} \cdot 2^k$, где $\vec{v}^x_{y,k} -$ k-я координата вектора $\vec{v}^x_y$.

Множество $\varepsilon^i_R(x) = \left\{\overline{y} \middle| \rho(x,\overline{y}) = \min_{y \in Q^x_i} \rho(x,y) \wedge \rho(x,\overline{y}) < R\right\}$ (т.е. множество точек, ближайших к x в -м квадранте и лежащих на расстоянии не больше R) – называется *множеством точек, соседних с x в i-м квадранте*. Точка y называется *соседней* с x, если $y \in \bigcup_{i=\overline{1..8}} \varepsilon^i_R(x)$. Множество $\varepsilon_R(x) = \bigcup_{i=\overline{1..8}} \varepsilon^i_R(x)$ называется множеством точек, соседних с x, или просто *множеством соседей x*. Отметим, что в заданном квадранте у данной точки может быть несколько соседей (в том случае, если они равноудалены).

Отношение «быть соседом» является симметричным и рефлексивным; транзитивное замыкание этого отношения образует разбиение $\boldsymbol{X^O}$ на множество кластеров.

Граничной точкой кластера называется точка, у которой хотя бы в одном из квадрантов нет ни одного соседа. *Границей* кластера называется множество граничных точек. У любого кластера, состоящего из конечного числа точек, есть хотя бы одна граница. Границ может быть 2 или более, если кластер содержит «полости».

Алгоритмическая схема (кластеризация). Исходные данные – множество точек $\boldsymbol{X^O}$, точки заданы набором своих координат. Выходные данные: разбиение $\boldsymbol{X^O}$ на кластеры $K_1, \dots, K_n$, $n \geq 1$.

Шаг 1. Для каждой точки $x_i \in \boldsymbol{X^O}$ построить множество ее соседей $\varepsilon_R(x_i)$, с разбиением по квадрантам. Положить $i = 1$.

Шаг 2. Выбрать произвольную точку $x \in \boldsymbol{X^O}$ и положить $K_i = \{x\}$. Положить $\varepsilon_R(K_i) = \bigcup_{x \in K_i} \varepsilon_R(x)$.

Шаг 3. Положить $K_i = \varepsilon_R(K_i)$[1]. Положить $\boldsymbol{X^O} = \boldsymbol{X^O} \backslash K_i$.

Шаг 4. Выполнять Шаг 3, пока $K_i \neq \varepsilon_R(K_i)$.

Шаг 5. Положить $i = i + 1$. Повторять Шаг 2 - Шаг 4, пока $\boldsymbol{X^O} \neq \emptyset$.

Алгоритмическая схема (определение границ кластера). Исходные данные – кластер K_i. Выходные данные: семейство попарно непересекающихся множеств $S_1^i, \dots, S_p^i,\ p \geq 1,$ – границ кластера K_i.

Шаг 1. Построить множество $S = S(K_i)$ – граничных точек кластера K_i (у которых хотя бы в одном квадранте нет соседей). Положить $j = 1$.

Шаг 2. Выбрать произвольную точку $x \in S(K_i)$ и положить $S_j = \{x\}$.

Шаг 3. Положить $S_j = \varepsilon_R(S_j)$ и $S = S \backslash S_j$.

Шаг 4. Повторять Шаг 3, пока $S_j \neq \varepsilon_R(S_j)$.

Шаг 5. Положить $j = j + 1$. Повторять Шаг 2 - Шаг 4, пока $S \neq \emptyset$.

Постановка задачи на программирование. Выполнить программную реализацию для случая $n = 2$ или $n = 3$.

1. Реализовать структуры данных для представления точек, множества их соседей, кластеров.
2. Реализовать алгоритм кластеризации.
3. Реализовать алгоритм определения границ кластера.
4. Реализовать средства визуализации результатов работы алгоритмов в n-мерном евклидовом пространстве.

Задача 4. Кластеризация на графах

Задача является вариацией на тему задачи 4 («Вычисление граничных поверхностей тел в 2-х и 3-мерном пространствах»).

Формальная постановка задачи. Пусть имеется связный ориентированный взвешенный граф $\mathcal{G}$. Требуется построить разбиение графа на компоненты V_i таким образом, чтобы:

1. каждая вершина отнесена точно к одной компоненте ($\bigcup_i V_i = \mathcal{G}$ и $V_k \cap V_l = \emptyset$ для всех $k \neq l$);
2. каждая компонента V_i обладает свойством сильной связности (при этом она не обязана быть компонентой сильной связности – скорее всего, будет просто подмножеством одной из таких компонент графа);
3. диаметры $d(V_i)$ всех компонент должны быть примерно одинаковы (например, можно потребовать, чтобы было минимальным отклонение от среднего $\overline{d} = \frac{1}{n}\sum V_i$: $\min_i\left(\left|\overline{d} - d(V_i)\right|\right)$.

Количество компонент, на которые необходимо разбить граф, следует подбирать исходя из оптимальности соотношения между диаметром отдельной компоненты и этим количеством. Например: если веса дуг нормализованы таким образом, что некоторая величина D

[1] Дело в том, что, строго говоря, $K_i \subseteq \varepsilon_R(K_i)$, точно так же, как $x \in \varepsilon_R(x)$. На практике реализацию удобнее выполнять для варианта с «проколотыми окрестностями».

соответствует диаметру графа (например, пусть $D = 10$), то оптимальное количество компонент можно искать, минимизируя сумму $n + \overline{d}$. Например: пусть $n = 1$, тогда $n + \overline{d} = 11$ (случай, когда выделяется всего 1 компонента – весь исходный граф). Если, например, при $n = 4$ удастся найти разбиение, при котором $\overline{d} = 4$, то $n + \overline{d} = 8$, и это лучше, чем при $\mathrm{n} = 1$. Если же, наоборот, количество компонент совпадает с количеством вершин графа, то $\overline{d} = 0$ и $n + \overline{d} = n$. Такой вариант будет лучше предыдущего, если в графе меньше 8 вершин. Естественно, ситуация изменится при ином выборе значения для параметра D.

Алгоритмическая схема (разбиение графа на компоненты).

Шаг 1. .

Алгоритмическая схема (выбор оптимального разбиения). Исходные данные – список вершин и дуг графа.

Шаг 1. Выполнить нормализацию расстояний между вершинами; число D является параметром алгоритма и его следует выбирать, исходя из количества вершин в графе.

Шаг 2. Построить матрицу расстояний (кратчайший путей) между вершинами графа.

Шаг 3. Для $n = \overline{1, D}$ решить задачу разбиения графа на n компонент.

Шаг 4. Для всех полученных вариантов разбиения выполнить оценку $n + \overline{d}$.

Шаг 5. Выбрать разбиение, обладающее наилучшей оценкой.

Иллюстративный пример. Пусть имеется множество городов, которые необходимо снабжать овощами. Города соединены сетью автомобильных дорог, по которым и доставляются овощи. Необходимо спланировать размещение овощехранилищ таким образом, чтобы оптимизировать расходы на развоз овощей. Пояснения:

- города являются вершинами графа, а дороги – дугами;
- каждое хранилище обслуживает один и только один город;
- условия на равенство диаметров выделяемых компонент графа позволяет ограничить максимальный путь, который будут проделывать машины при доставке.

Задача на программирование.

1. Выполнить параллельную реализацию вычислительных алгоритмов. Например, это можно сделать, выполнив параллельную реализацию операций map-reduce.
2. Реализовать пользовательский интерфейс, способный отображать граф. Однако необязательно, чтобы граф задавался с помощью графических средств – это можно делать, например, в файле.

Задача 5. Разработка системы описания и исполнения клеточных автоматов

Рассмотрим квадратную матрицу A порядка n, элементы которой принадлежат некоторому фиксированному множеству T, которое будет называть множеством цветов (а элементы этого множества – цветами). Этому множеству T, в частности, принадлежит элемент $\perp$ - белый цвет.

Пусть $f: T \to T$ – некоторое отображение. Тогда работу клеточного автомата можно описать как вычисление на каждом шаге новой матрицы $A^{(i)} = A^{(i-1)} \times (f \cdot E_n)$, где E_n – единичная матрица порядка n. Т.е. отображение f применяется к каждому элементу $a_{ij}^{(i-1)}$ матрицы $A^{(i-1)}$, и строится новая матрица $A^{(i)} = \left(f\left(a_{ij}^{(i-1)}\right)\right)$.

Отображение f может быть произвольным, однако интерес представляют отображения, которые учитывают предыдущее состояние соседних с a_{ij} элементов – расположенных по горизонтали, вертикали и диагоналям на расстоянии 1 или более. Одним из самых известных примеров клеточных автоматов является игра «Жизнь».

Задача на программирование. Требуется реализовать визуализацию игрового поля – матрицы $A^{(i)}$ – и предоставить средства для автоматического и ручного управления выполнением машины. Требует реализовать средства для задания отображения f. Интерес представляют две возможности:

- задание с помощью аналитических выражений и стандартных функций (типа min, max, sin, cos и т.д.);
- задание с помощью правил.

Правила должны быть вида $S_{ij} \to v$, где S_{ij} – это «старое» состояние клетки (ij), а $v \in T$ «новое» значение для (ij). Состояние – это список конъюнкция выражений вида $(i_k, i_k)\theta v_k$. Здесь, (i_k, i_k) – координаты -го соседа элемента (ij), $v_k \in T$ – его значение, а $\theta \in \{=, \neq, <, \leq, \geq, >\}$ – оператор сравнения.

Задача 6. Разработка API для композиционного описания вычислений

Пусть имеется базовый набор комбинационных схем, приведенный в таблице ниже:

№	Функция	Тип	Пример
1.	`$\cdot$	$T \to T$	$`(21 - 3 * 7) \mapsto 0: int$
2.	$if(\cdot)then \cdot else \cdot$	$Bool \times T \times T \to T$	
3.	$(\cdot, \dots, \cdot)$	$(T_1, \dots, T_n)$	(1,2,3,4)
4.	$[\cdot, \dots, \cdot]$	$[T]$	[1,2,3,4]
5.	$\cdot \circ \cdot$	$(T_2 \to T_3) \to (T_1 \to T_2) \to (T_1 \to T_3)$	$(f \circ g \circ h)x \mapsto f\left(g(hx)\right)$
6.	$<\cdot, \dots, \cdot>$	$(T \to T_1) \times \dots \times (T \to T_n) \to T \to T_1 \times \dots \times T_n$	$< +1, toStr > 5 \mapsto (6, "5"): Int \times String$
7.	map	$(T_1 \to T_2) \to [T_1] \to [T_2]$	
8.	$flatMap$	$(T_1 \to [T_2]) \to [T_1] \to [T_2]$	
9.	$fold$	$(T \to T \to T) \to [T] \to T$	
10.	Y		$Ya = a(Ya)$
11.	zip_n	$(T_1 \to \dots \to T_n \to T_{n+1}) \to [T_1] \to \dots \to [T_n] \to T_{n+1}$	
12.	$unzip$	$[(T_1, \dots, T_n)] \to$	

На основе этих схем реализуются следующие способы композиции

№	Функция	Пример	Комментарий
1.	`.Value`	`.Value(12)`	`` ` · `` $(\cdot,\dots,\cdot)$ $[\cdot,\dots,\cdot]$
2.	`.If` `.Then` `.Else`	`.If(x=>…).Then(x=>…)` `.Else(x=>…)`	$if(\cdot)then\cdot else\cdot$
3.	`.Switch` `.Case` `.Default`	`.Switch(x=>…).Case(1,x=>…)` `.Case(2,x=>…).Case(3,x=>)` `.Default(x=>…)`	$if(\cdot)then\cdot else\,(\dots(if(\cdot)then\cdot else\cdot)\dots)$
4.	`.Then`	`.Then(x=>…)`	$f\circ F$
5.	`.Every`	`.Every(x=>,…,x=>…)` `.Then((x1,…,xn)=>…)`	$<\cdot,\dots,\cdot>$
6.	`.Project`	`.Value((1,2,3,4))` `.Project(v1=>Some(…), v2=>None, v3=>Some(…), v4=>None)` `.Then((x,y)=>…)//(1,3)=>…`	
7.	`.JoinValues`	`.Value(1,2,3)` `.JoinValues(4,5)` `.Then((x1,x2,x3,x4,x5)=>…)`	$\cdot\circ\cdot$
8.	`.Map`	`.Value([a1,…,an])` `.Map(x=>…)`	map
9.	`.All`	`.All([x=>…,…,x=>…])` `.Then([x1,…,xn]=>…)`	$map\ (\lambda f.fx)\ [f_1,\dots,f_n]$
10.	`.Any`	`.Any([x=>…,…,x=>…])` `.Then(x=>…)`	$(filter\dots)\circ(map\ [f_1,\dots,f_n])$
11.	`.FlatMap`	`.Value([a1,…,an])` `.FlatMap(x=>…)`	$flatMap$
12.	`.Fold`	`.Value([a1,…,an])` `.Fold(x y =>…)`	$fold$
13.	`.Rec` `.Recurse`	`.Rec<int>()` `.Recurse(x=>x+1)`	$Y\big(\lambda fx.f(x+1)\big)$
14.	`.Zip`	`.Value([1,3],[2,4,6])` `.Zip() //=[(1,2),(3,4)]`	zip_n $List\langle T_1\rangle\times List\langle T_2\rangle\to List\langle T_1\times T_2\rangle$
15.	`.Unzip`	`.Value([(1,2),(3,4)])` `.Unzip() //=([1,3],[2,4])`	$unzip$

Пример вычислений $\left(\left(\frac{a+b}{2}\right)^2+\left(\frac{a-b}{2}\right)^2\right)\cdot\frac{\min(a,b)}{\max(a,b)}$, если $a=2,b=3$:

```
var computation = Expression
   .Value(2,3)      // либо .Variable<int,int>()
   .Every(
         (a,b)=> ((a+b)/2)**2,
         (a,b)=> ((a-b)/2)**2,
         (a,b)=> max(a,b),
```

```
        (a,b)= min(a,b)>)
    .Then((halfSum, halfDiff, mx, mn) => (halfSum+halfDiff)*mx/mn)
var result = computation.Eval()    // соответственно .Eval(1,2)
computation.EvalAsync(result => …) //соответственно .EvalAsync(1,2,… )
```

Задача на программирование. Требуется реализовать описанный API. Исполнение кода, подобного тому, который приведен в примере выше, приводит к конструированию выражения, которое затем может быть выполнено. Выполнение должно запускаться методом Eval.

Базовая задача: параллельность вычислений достигается за счет многопоточной реализации отдельных операций (таких, как map, flatMap и частного случая reduce/fold, когда функция-аргумент является ассоциативной операцией).

Расширенная задача:

1. распараллеливание вычислений на уровне обработки дерева выражений (например, за счет реализации диспетчера задач)
2. выполнение выражений в асинхронном режиме

Задача 7. Восстановление поврежденных сообщений

Пусть два абонента, скажем, А и Б, общаются посредством обмена текстовыми сообщениями. Канал связи, которым они пользуются, подвержен влиянию помех, в результате чего сообщения подвергаются более или менее существенным искажениям: части некоторых слов, или даже отдельные слова, невозможно прочитать.

Пример. Рассмотрим набор слов $\{aab, baccb, bbba, aacb, baaac, ccb\}$ в алфавите $\{a, b, c\}$. Пример сообщения, отправленного одним из абонентов: $baccb\ aab\ baccb\ aacb\ baaac\ bbba$. Пример сообщения, полученного вторым абонентом: $baccb\ a\ ******\ cb\ aacb\ baaac\ bb\ **$. Исходное полученное сообщение имеют одинаковую длину – 31 символ, считая пробелы. В процессе передачи сообщение было искажено так, что в итоге 8 символов искажены и не поддаются прочтению. Конец сообщения восстановить легко: лишь одно слово в словаре имеет префикс bb. Восстановить другие 6 символов однозначно не удается: одинаково подходят варианты $aab\ aacb$ и $aacb\ ccb$.

Задача на программирование. Требуется написать программу, которая сможет восстанавливать искаженные части сообщений. Оба абонента пользуются одним и тем же фиксированным словарем. Вариации задачи:

1. Искажение может состоять не только в том, что некоторые символы становятся не читаемыми, но и в замене отдельных символов на другие из того же алфавита. Искаженного слова в словаре, скорее всего, не окажется, но это не обязательно.
2. Из-за того, что сообщение может быть искажено в нескольких местах и в каждом месте может оказаться много подходящих вариантов, итоговое количество вариантов восстановленного сообщения может быть большим. Его можно сократить, если учитывать «семантику» сообщений. Для этого все или некоторые слова в словаре следует снабдить одной или несколькими тематическими метками. В таком случае,

если большинство неиспорченных слов относятся к определенной тематике, то и искаженные слова, скорее всего, относятся к той же тематике.

Указание. Алфавит и словарь (а также тематическую разметку, если она применяется) следует хранить в виде настроек, например, в конфигурационном файле.

Указание. Для решения этой задачи удобно использовать подход, описанный в задаче 2. В роли числовой оценки выступает мера похожести текстов.

Приложение 1. Инструментальные средства

Рисование графов в Web:

1. http://sigmajs.org/
2. http://visjs.org/

3D графика в Web:

1. https://threejs.org/

Библиотеки для создания парсеров и компиляторов:

1. для F# - FsYACC (http://fsprojects.github.io/FsLexYacc/)
2. для javascript (https://habrahabr.ru/post/183400/, https://github.com/peter-leonov/bison-lalr1.js)
3. java – небольшой зоопарк (https://java-source.net/open-source/parser-generators)

```
namespace ConsoleApplication14
{
   class Program
   {
      static void Main(string[] args)
      {
         var expr = Expression.Map<int, int>(x => x - 1).Map<int, int>(x => x).Fold<int, int>((x, y) => x + y);
         Expression.Eval<List<int>,int>(expr, new List<int> {1,2,3,4,5 });
      }
   }

   public static class Expression
   {
```

```
    public static Expression<T, R> Map<T, R>(Func<T, R> f) { throw new NotImplementedException(); }
    public static Expression<T, R> Fold<T, R>(Func<T, R, R> f) { throw new NotImplementedException(); }
    public static R Eval<T, R>(Expression<T, R> expr, T arg)
    {
      switch (expr)
      {
        case MapExpression<T, R> mexpr:
          break;
      }
    }
  }

  public abstract class Expression<T, R>
  {
    public Type ArgumentType { get; protected set; }
    public Type ReturnType { get; protected set; }
    public Expression Argument { get; set; }
    public Expression<T, R> Map<T, R>(Func<T, R> f) { throw new NotImplementedException(); }
    public Expression<T, R> Fold<T, R>(Func<T, R, R> f) { throw new NotImplementedException(); }
  }

  public sealed class MapExpression<T, R> : Expression<T, R>
  {
    public Func<T, R> Computation { get; private set; }
  }

  public sealed class FoldExpression<T, R, R> : Expression<T, R>
  {
    public Func<T, R, R> Computation { get; private set; }
```

```
    }

    public sealed class ValueExpression<T> : Expression<T, T>
    {
        public T Value { get; private set; }
    }
}
```