Основы распараллеливания и оптимизации алгоритмов

Д. В. Луцив

Кафедра системого программирования СПбГУ

CS103

Содержание

Назначние

- Никакие алгоритмы не помогут ускорить процесс при помощи нескольких процессоров с такой же суммарной вычислительной мощностью. Только замедлят.
- Разбиение данных с параллельной обработкой фрагментов и разбиение алгоритмов с конвейеризацией — наиболее универсальные подходы.

Основная проблема в том, что поднять мощность одного процессора в n раз очень непросто.

Параллелизм по задачам

- *Уровень заданий*.
  Несколько независимых заданий одновременно выполняются на разных процессорах, практически не взаимодействуя друг с другом. Реализуется на ВС с множеством процессоров в многозадачном режиме.
- *Уровень программ*.
  Части одной задачи выполняются на множестве процессоров. Достигается на параллельных ВС.
- *Уровень команд*.
  Выполнение команды разделяется на фазы, а фазы нескольких последовательных команд м.б. перекрыты за счет конвейеризации. Достижим на ВС с одним процессором.
- *Уровень битов (арифметический уровень)*.
  Биты слова обрабатываются одновременно. Реализуется в обычных и суперскалярных процессорах

По архитектурам

- Суперскалярные процессоры
- VLIW
- Векторные и матричные процессоры
- Cluster
- Grid

Языки программирования

- Обычные
- С использованием библиотек параллельной обработки данных
- С использованием параллельных расширений
- Специализированные параллельные языки
- Обычные языки с естественной поддержкой параллельности

Топологии

- Шина
- Кольцо
- Плоскость
- Цилиндр
- Тор
- M динамических линий между N процессорами
- Гиперкуб
- Топологии, имитирующие физику задач

Показатели I

- $O(n), O(1)$ — количество операций на системе с n или 1 процессорами
- $T(n), T(1)$ — время выполнения
- $S(n) = \frac{T(1)}{T(n)}$ — ускорение

Берём $T(1) = O(1)$

Параллельное выполнение д.б., по крайней мере, не медленнее, чем послдовательное, так что:

- $T(n) \leq O(n) \leq nO(1) = nT(1)$
- $1 \leq S(n) \leq n$ — ускорение
- $1/n \leq E(n) = S(n)/n = \frac{T(1)}{nT(n)} \leq 1$ — эффективность

Показатели I

- $O(n), O(1)$ — количество операций на системе с n или 1 процессорами
- $T(n), T(1)$ — время выполнения
- $S(n) = \frac{T(1)}{T(n)}$ — ускорение

Берём $T(1) = O(1)$
Параллельное выполнение д.б., по крайней мере, не медленнее, чем послдовательное, так что:

- $T(n) \leq O(n) \leq nO(1) = nT(1)$
- $1 \leq S(n) \leq n$ — ускорение
- $1/n \leq E(n) = S(n)/n = \frac{T(1)}{nT(n)} \leq 1$ — эффективность

Показатели I

- $O(n), O(1)$ — количество операций на системе с n или 1 процессорами
- $T(n), T(1)$ — время выполнения
- $S(n) = \frac{T(1)}{T(n)}$ — ускорение

Берём $T(1) = O(1)$
Параллельное выполнение д.б., по крайней мере, не медленнее, чем послдовательное, так что:

- $T(n) \leq O(n) \leq nO(1) = nT(1)$
- $1 \leq S(n) \leq n$ — ускорение

- $1/n \leq E(n) = S(n)/n = \frac{T(1)}{nT(n)} \leq 1$ — эффективность

Показатели II

- $1 \leq R(n) = \frac{O(n)}{O(1)} \leq n$ —
  коэффициент избыточности
- $C(n) = nT(n)$ —
  цена вычислений
- $U(n) = \frac{O(n)}{C(n)} = \frac{O(n)}{nT(n)} = \frac{T(1)O(n)}{nT(n)O(1)} = E(n)\frac{O(n)}{O(1)} = R(n)E(n)$ —
  утилизация — насколько много работы за единицу цены
- $Q(n) = \frac{S(n)E(n)}{R(n)} = \frac{T^3(1)}{nT^2(n)O(n)}$
  — эмпирический показатель качества

Закон Амдала

f — доля последовательного кода, $(1-f)$ — параллельного.

$$T_{par} = fT_{seq} + \frac{(1-f)T_{seq}}{n}$$

$\Rightarrow$

$$S = T_{seq}/T_{par} = \frac{n}{1+(n-1)f} \overset{n\to\infty}{\longrightarrow} 1/f$$

Алгоритм Штрассена

$$C = AB \qquad A, B, C \in \mathbb{R}^{2^n \times 2^n}$$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}.$$

$$A_{i,j}, B_{i,j}, C_{i,j} \in \mathbb{R}^{2^{n-1} \times 2^{n-1}}$$

Это 8 умножений:

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}; C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2};$$

$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}; C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}.$$

▸ А можно за 7!

Алгоритм Штрассена

$$C = AB \qquad A, B, C \in \mathbb{R}^{2^n \times 2^n}$$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}.$$

$$A_{i,j}, B_{i,j}, C_{i,j} \in \mathbb{R}^{2^{n-1} \times 2^{n-1}}$$

Это 8 умножений:

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}; C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2};$$

$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}; C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}.$$

▸ А можно за 7!

Алгоритм Штрассена

$$C = AB \qquad A, B, C \in \mathbb{R}^{2^n \times 2^n}$$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}.$$

$$A_{i,j}, B_{i,j}, C_{i,j} \in \mathbb{R}^{2^{n-1} \times 2^{n-1}}$$

Это 8 умножений:

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}; C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2};$$

$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}; C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}.$$

▸ А можно за 7!

Поразрядная сортировка

Пример для двоичной системы.

- LSD – сперва младшие, затем консервативно (стабильно) относительно порядка младших старшие:
  - Сперва фильтруются те, у которых старший бит 0,
  - затем те, у кого 1,
  - затем они сливаются.
  - Это частный случай блочной (корзинной, карманной) сортировки.
- MSD – сперва старшие, затем рекурсивно младшие:
  - Фактически, это QuickSort, но разделение по значению очередного разряда,
  - в отличие от «оригинального» QuickSort, при фиксированной разрядности трудоёмкость ограничена $N \ln N$.

Поразрядная сортировка

Пример для двоичной системы.

- LSD – сперва младшие, затем консервативно (стабильно) относительно порядка младших старшие:
  - Сперва фильтруются те, у которых старший бит 0,
  - затем те, у кого 1,
  - затем они сливаются.
  - Это частный случай блочной (корзинной, карманной) сортировки.
- MSD – сперва старшие, затем рекурсивно младшие:
  - Фактически, это QuickSort, но разделение по значению очередного разряда,
  - в отличие от «оригинального» QuickSort, при фиксированной разрядности трудоёмкость ограничена $N \ln N$.

Поразрядная сортировка

Пример для двоичной системы.

- LSD – сперва младшие, затем консервативно (стабильно) относительно порядка младших старшие:
  - Сперва фильтруются те, у которых старший бит 0,
  - затем те, у кого 1,
  - затем они сливаются.
  - Это частный случай блочной (корзинной, карманной) сортировки.
- MSD – сперва старшие, затем рекурсивно младшие:
  - Фактически, это QuickSort, но разделение по значению очередного разряда,
  - в отличие от «оригинального» QuickSort, при фиксированной разрядности трудоёмкость ограничена $N \ln N$.

Поразрядная сортировка

Пример для двоичной системы.

- LSD – сперва младшие, затем консервативно (стабильно) относительно порядка младших старшие:
  - Сперва фильтруются те, у которых старший бит 0,
  - затем те, у кого 1,
  - затем они сливаются.
  - Это частный случай блочной (корзинной, карманной) сортировки.
- MSD – сперва старшие, затем рекурсивно младшие:
  - Фактически, это QuickSort, но разделение по значению очередного разряда,
  - в отличие от «оригинального» QuickSort, при фиксированной разрядности трудоёмкость ограничена $N \ln N$.

Сортировальная машина

Табулятор

Распараллеливание QuickSort

При эффективном делении пополам левая и правая половина могут сортироваться параллельно. Если процессоров много, то четверти и т.д.

Это не очень эффективно из-за высокой нагрузки на кэш.

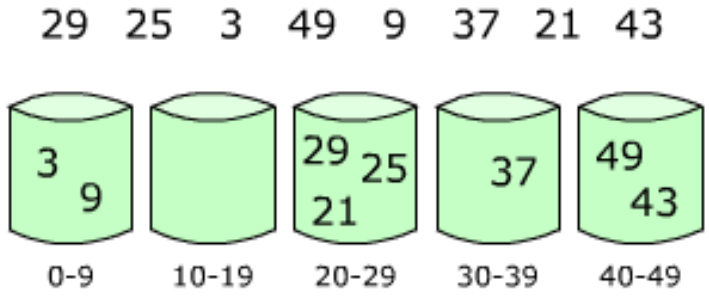
Распараллеливание QuickSort

При эффективном делении пополам левая и правая половина могут сортироваться параллельно. Если процессоров много, то четверти и т.д.
Это не очень эффективно из-за высокой нагрузки на кэш.

BucketSort

Развитие идеи QuickSort

1. Распределяем по блокам



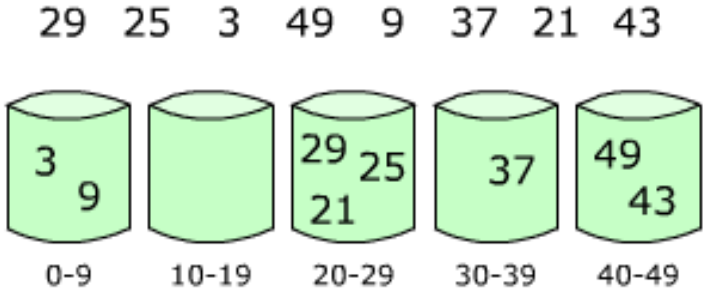
2. Сортируем блоки (можно параллельно и распределённо)
3. Сливаем

Как и QuickSort, можно затормозить специально подобранными данными.

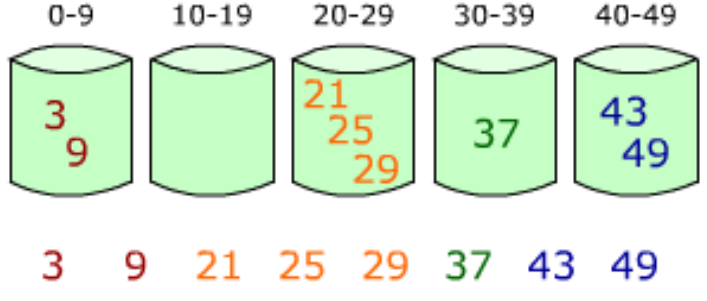
BucketSort

Развитие идеи QuickSort

1. Распределяем по блокам



2. Сортируем блоки (можно параллельно и распределённо)



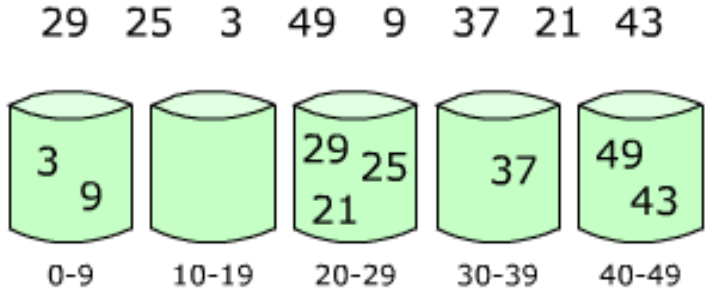
3. Сливаем

Как и QuickSort, можно затормозить специально подобранными данными.

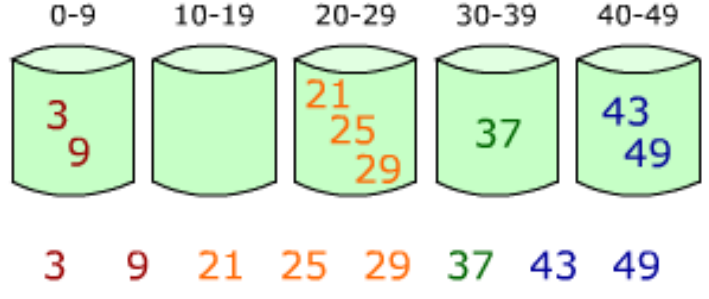
BucketSort

Развитие идеи QuickSort

1. Распределяем по блокам



2. Сортируем блоки (можно параллельно и распределённо)



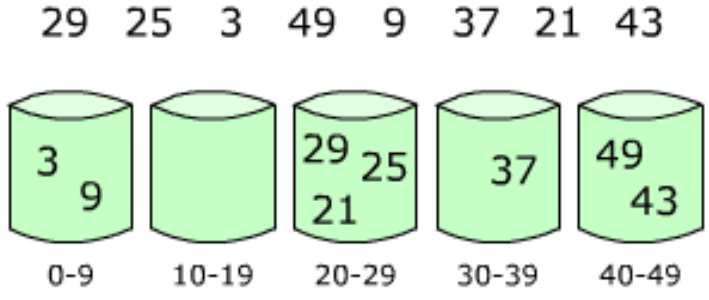
3. Сливаем

Как и QuickSort, можно затормозить специально подобранными данными.

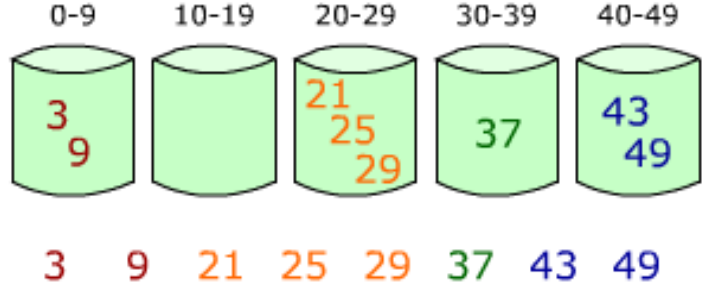
BucketSort

Развитие идеи QuickSort

1. Распределяем по блокам



2. Сортируем блоки (можно параллельно и распределённо)



3. Сливаем

Как и QuickSort, можно затормозить специально подобранными данными.

Map-Reduce: модель

Парадигма и паттерн. Но не «технология».

V – пространство значений.
$m : V \to C$ – функция разбиения на классы эквивалентности.
$r : W \in 2^V \to R \mid \forall v \in W \; f(v) = c \in C$ – функция редукции.

Редукция может быть иерархической.

Map-Reduce: модель

Парадигма и паттерн. Но не «технология».

V – пространство значений.
$m : V \to C$ – функция разбиения на классы эквивалентности.
$r : W \in 2^V \to R \mid \forall v \in W \; f(v) = c \in C$ – функция редукции.

Редукция может быть иерархической.

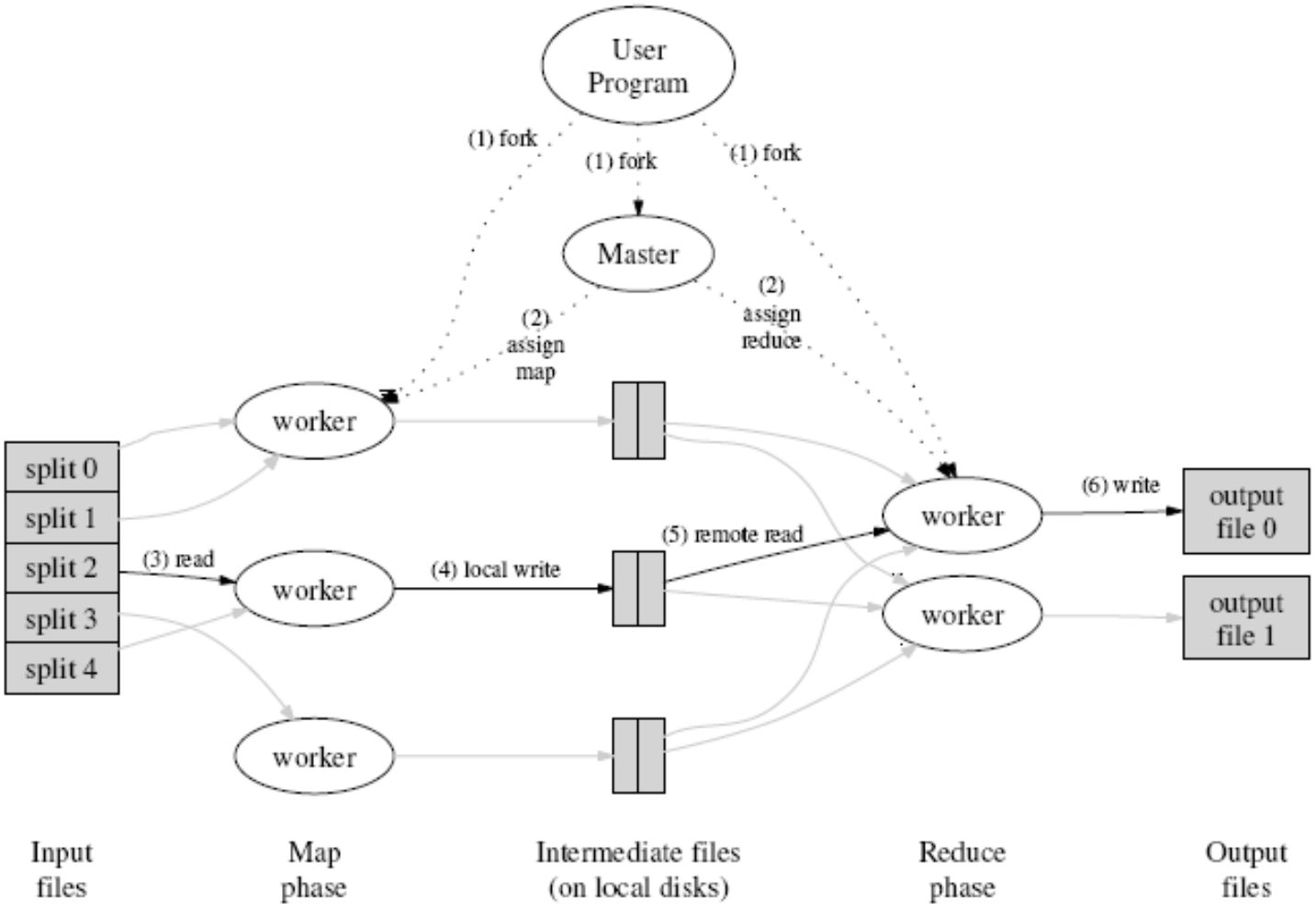
Map-Reduce: модель

Парадигма и паттерн. Но не «технология».

V – пространство значений.
$m : V \to C$ – функция разбиения на классы эквивалентности.
$r : W \in 2^V \to R \mid \forall v \in W \ f(v) = c \in C$ – функция редукции.

Редукция может быть иерархической.

Map-Reduce: топология



Map-Reduce: пример

1. Два человека берут стаканы с деньгами, выбирают монетки и сортируют на кучки с одним достоинством (map).
2. Считается количество в каждой кучке и, таким образом, её достоинство (reduce1).
3. Суммируются достоинства кучек монет: сперва для одного человека (reduce2), потом общее (reduce3).

Задача о складе цветного металла

1. На складе мелкий лом.
2. Можем унести ограничено.
3. Берём самый дорогой, сколько есть и сколько влезет.
4. Если осталось место, то повторяем (3) для следующего по цене.

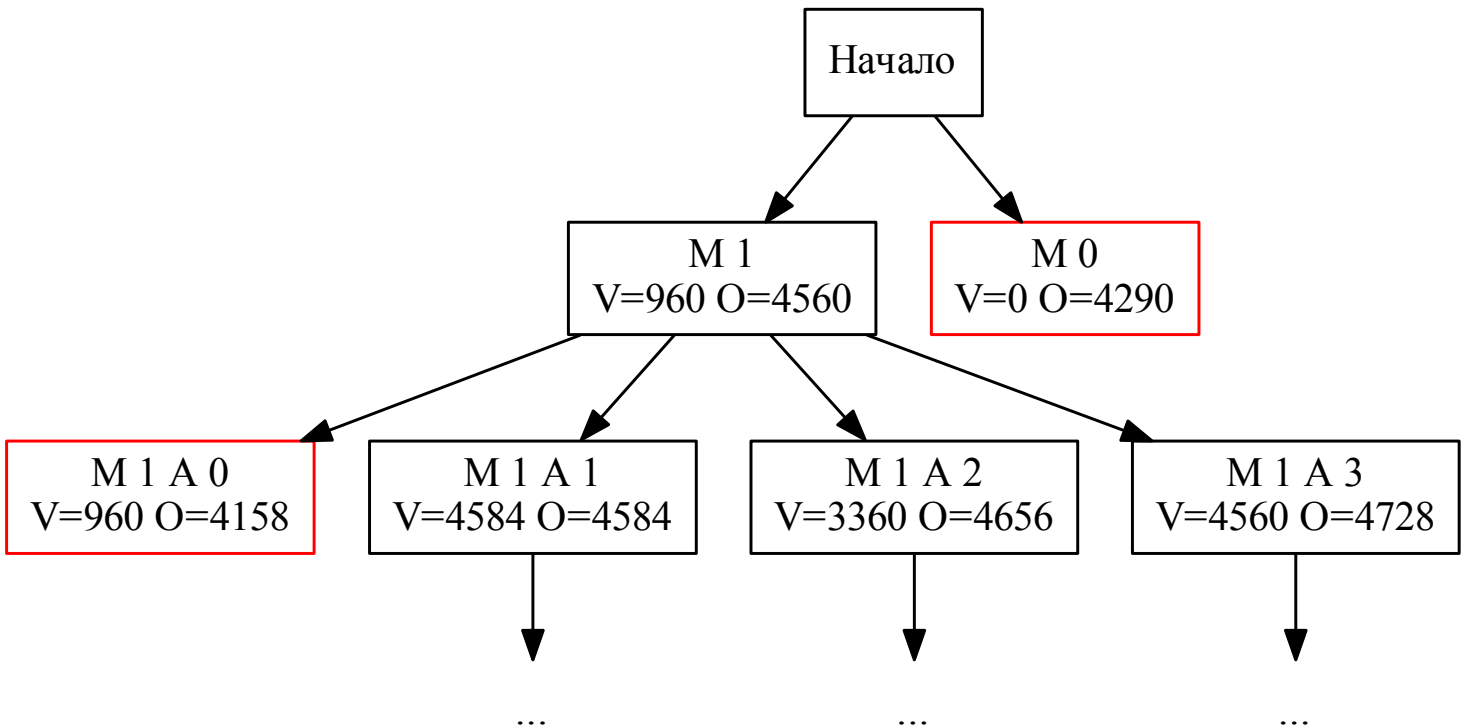
Основная идея

- Решаем задачу в несколько шагов, «ветвясь в ширину».
- На следующем шаге пользуемся частичными данными со всех ветвей предыдущих шагов.

Метод ветвей и границ

На складе не лом, а изделия (стоят дороже).

▸ Пример



Кратчайший путь

Кратчайший путь в графе между двумя точками.

1. Идём «фронтом» от конечной.
2. В каждой доступной вершине перезаписываем расстояние, если оно станет меньше, и помечаем, откуда пришли.
3. Пока не дойдём до начальной.

Каждый раз используется весь фронт (и только он).
«Расстояние» в самом общем смысле. Например для ракеты взвешенная сумма расхода топлива, времени полёта и риска быть сбитой в данной точке.

Кратчайший путь

Кратчайший путь в графе между двумя точками.

1. Идём «фронтом» от конечной.
2. В каждой доступной вершине перезаписываем расстояние, если оно станет меньше, и помечаем, откуда пришли.
3. Пока не дойдём до начальной.

Каждый раз используется весь фронт (и только он).

«Расстояние» в самом общем смысле. Например для ракеты взвешенная сумма расхода топлива, времени полёта и риска быть сбитой в данной точке.

Кратчайший путь

Кратчайший путь в графе между двумя точками.

1. Идём «фронтом» от конечной.
2. В каждой доступной вершине перезаписываем расстояние, если оно станет меньше, и помечаем, откуда пришли.
3. Пока не дойдём до начальной.

Каждый раз используется весь фронт (и только он).
«Расстояние» в самом общем смысле. Например для ракеты взвешенная сумма расхода топлива, времени полёта и риска быть сбитой в данной точке.

Вопросы

EDU.DLUCIV.NAME