Строительные блоки алгоритмов

Формат представления алгоритма

- Название и краткое описание
- Входные и выходные данные алгоритма
- Контекст применения алгоритма
- Реализация алгоритма
- Анализ алгоритма
- Вариации алгоритма

Строительные блоки

В языке C++ программы построены на нескольких базовых типах данных:

› Целые числа (int)

› Числа с плавающей точкой (float)

› Символы (char)

Тип данных – *это множество значений и набор операций с ними.*

Операции связаны с типами, а не наоборот.

Структуры данных

В зависимости от реализации (посредством массивов или указателей) структуры данных разбиваются на два типа

- *Смежные структуры данных* реализованы в виде непрерывных блоков памяти. К ним относятся массивы, матрицы, кучи и хэш-таблицы
- *Связанные структуры данных* реализованы в отдельных блоках памяти, связанных вместе с помощью *указателей*. К этому виду структур данных относятся списки, деревья и списки смежных вершин графов

Массивы

Достоинства массивов:

- *Постоянное время доступа при условии наличия индекса.*
- *Эффективное использование памяти*
- *Локальность в памяти*

Недостаток:

- *Размер нельзя изменять в процессе исполнения программы*

Решение – *динамическое выделение памяти*

Указатели и связанные структуры данных

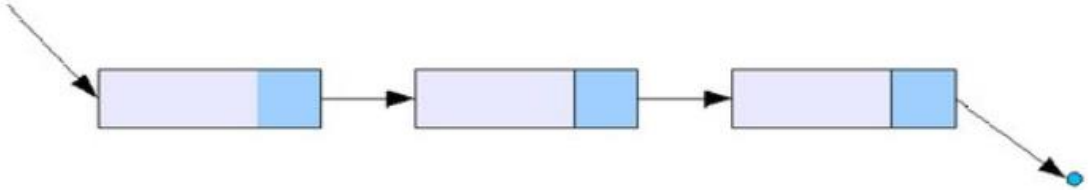
Указатель – это адрес ячейки памяти

Связный список — это набор элементов, причем каждый из них является частью *узла (node),* который также содержит *ссылку (link)* на узел.

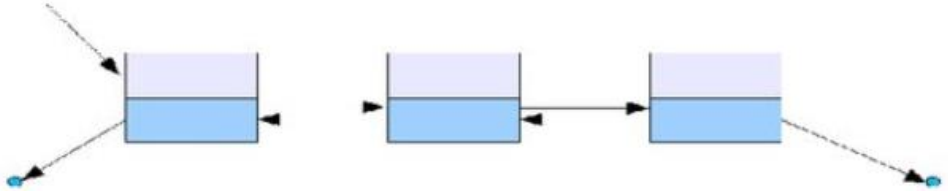
Основное преимущество ***связных списков*** перед ***массивами*** заключается в возможности эффективного изменения расположения элементов. За эту гибкость приходится жертвовать скоростью доступа к произвольному элементу списка, поскольку единственный способ получения элемента состоит в отслеживании связей от начала списка.

Связные списки

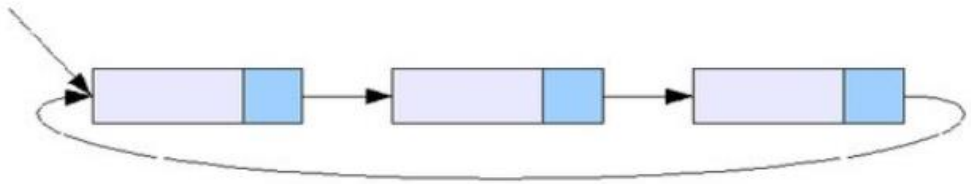
Односвязный список



Двусвязный список



Кольцевой связный список



Операции в связных списках

- *Поиск* (search)
- *Вставка* (insert)
- *Удаление* (delete)

Сравнение связных списков и статических массивов

Преимущества

- Переполнение невозможно, если только не переполнена сама память
- Операции вставки и удаления элементов проще соответствующих операций над массивами
- При работе с большими записями перемещение указателей происходит легче и быстрее, чем перемещение самих записей

Сравнение связных списков и статических массивов

Недостатки

- Необходимо дополнительное место для хранения указателей
- Нет эффективного произвольного доступа к элементам
- Массивы обладают лучшей локальностью в памяти и более эффективны в использовании кэш-памяти, чем связные списки

Динамическое выделение памяти обеспечивает гибкость в выборе способа и момента использования этого ограниченного ресурса

Примеры рекурсивных объектов

- *Списки.* После удаления первого элемента связного списка мы имеем такой же связный список, только меньшего размера. То же самое справедливо для строк, поскольку в результате удаления символов из строки получается более короткая строка
- *Массивы.* Отделение первых *k* элементов из массива из *n* элементов нам дает два массива меньших размеров: (*k*) + (*n*-*k*)

Абстрактные типы данных

Абстрактный тип данных (АТД) — это тип данных (набор значений и совокупность операций для этих значений), доступ к которому осуществляется только через ***интерфейс***. Программу, которая использует АТД, будем называть ***клиентом***, а программу, в которой содержится спецификация этого типа данных — ***реализацией***.

- Стеки
- Очереди
- Словари и т.д.

Процесс численного решения

- Вычисления с плавающей точкой
- Вычисления неограниченной точности

Проблемы:

- Представление чисел конечным количеством значений
- Ошибки округления
- Ошибки дискретизации
- Сравнения
- Преобразование типов

Вычисления с плавающей точкой

- Невозможность точного представления: π, e
- Невозможность точного численного представления некоторых результатов (частное от деления или даже умножение)

4-х разрядная десятичная арифметика

$0.8132 \cdot 0.6135 = 0.49889820$

0.4988 или 0.4989

Ошибки округления

Представление чисел с плавающей точкой

Тип	Знак, бит	Экспонента, бит	Мантисса, бит
float	1	8	23
double	1	11	52

Пример бинарного представления 3.88f (float)

01000000 01111000 01010001 11101100 (32 бита)

0 – знак положительный (1 бит)

10000000 – экспонента (8 бит) ($128 = 2^7$)

Ошибки округления (2)

Представление чисел с плавающей точкой

Тип	Знак, бит	Экспонента, бит	Мантисса, бит
float	1	8	23
double	1	11	52

Пример бинарного представления 3.88f (float)

[1]11110000101000111101100 = [1/2]+1/4+1/8+1/16+1/32+

1/1024+1/4096+1/65536+1/131072+1/262144+1/524288+

1/20974152+1/4194304 = $+1 \cdot 0.97000002861022949218750 \cdot 2^2$ =

3.880000114440917968750

Ошибки округления (3)

› *Накопление ошибки*

$0.8132 \cdot 0.6135 \cdot 0.2103 = 0.10491829$ (до 8 значащих цифр)

$0.8132 \cdot 0.6135 = 0.4988$ (ошибка $0.9820 \cdot 10^{-4}$)

$0.4988 \cdot 0.2103 = 0.1048$ (ошибка $0.9764 \cdot 10^{-4}$)

Накопленная ошибка – $0.1183 \cdot 10^{-3}$

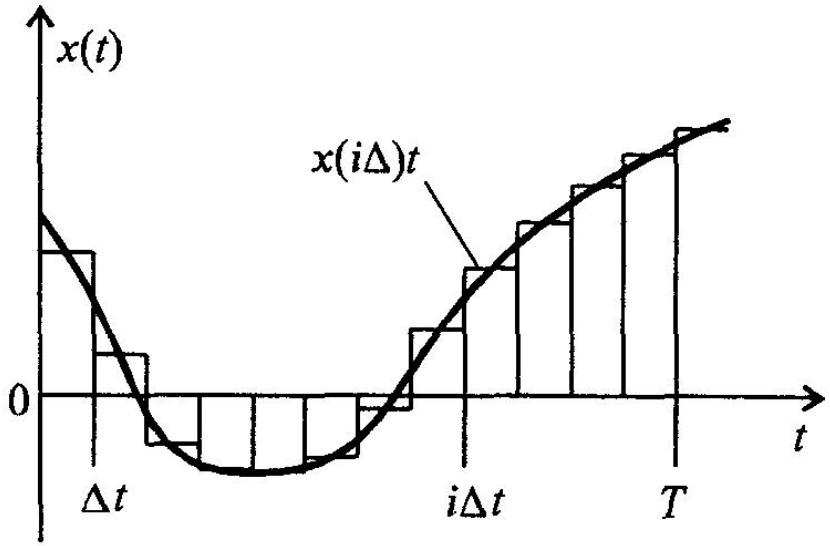
› *Катастрофическая потеря знака*

$$a-b$$

Ошибки численных решений

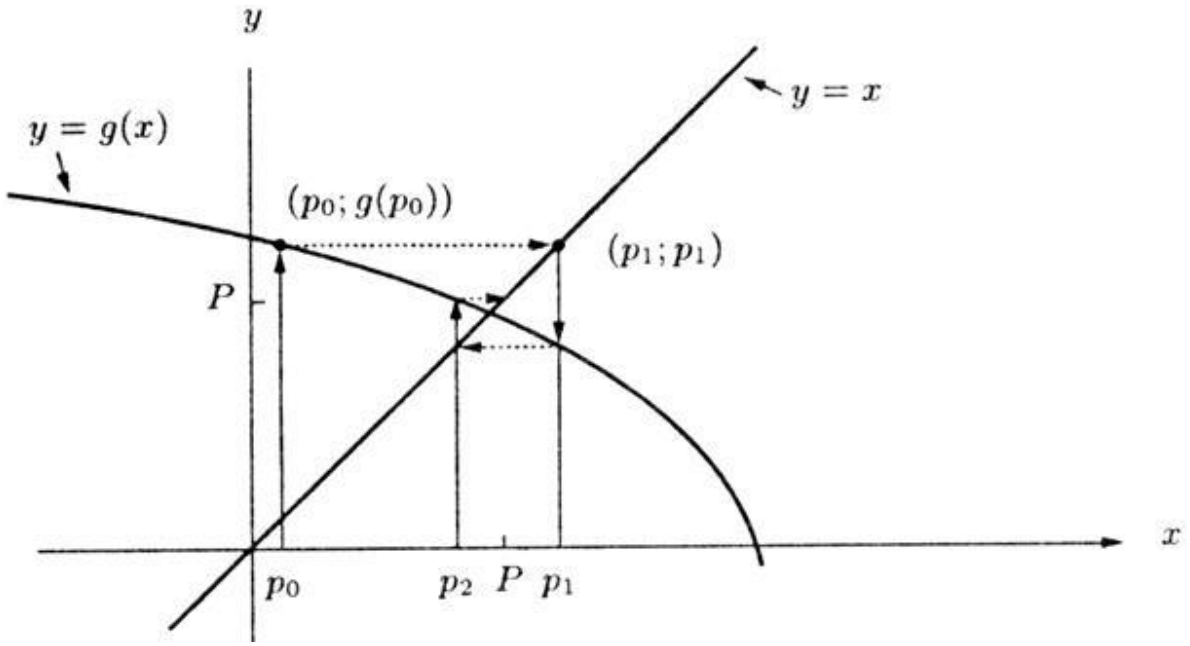
› *Ошибки дискретизации*

Замена непрерывных задач дискретными



› *Ошибки сходимости*

Возможность выполнить конечное число приближений



Сравнение значений с плавающей точкой

```
if (x == y) {…}
```

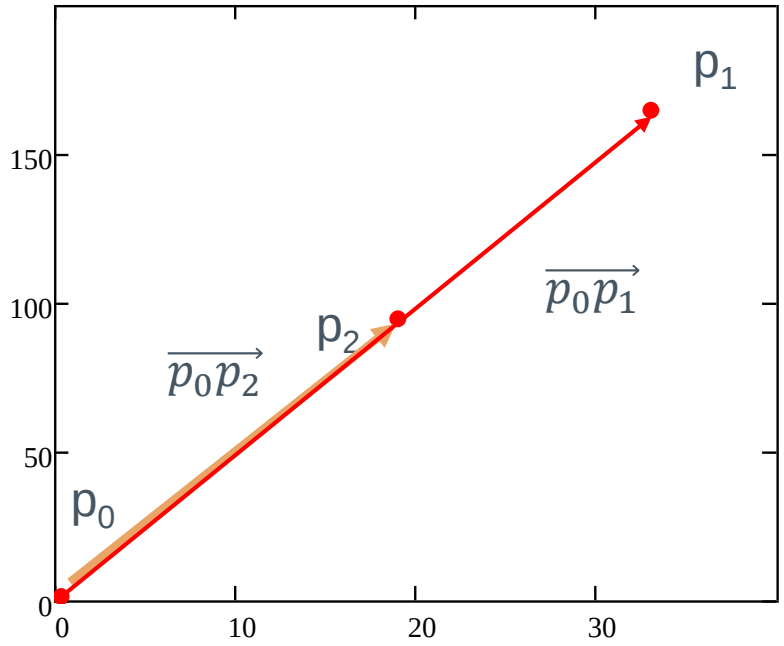
Можно ли сравнивать числа с плавающей точкой?

Практический пример

$p_0=(a,b)$; $p_1=(c,d)$; $p_2=(e,f)$

(c-a)(f-b)-(d-b)(e-a)

- =0, отрезки коллинеарны;
- < 0, отрезки повернуты влево
- > 0, отрезки повернуты вправо



Арифметические ошибки с плавающей точкой

	32-битное значение (float)	64-битное значение (double)
a=1/3	0.33333334	0.3333333333333333
b=5/3	1.6666666	1.6666666666666667
c=33	33.0	33.0
d=165	165.0	165.0
e=19	19.0	19.0
f=95	95.0	95.0
(c-a)(f-b)-(d-b)(e-a)	4.8828125E-4	-4.5474735088641E-13

y = 5x

$|x - y| < \delta$

$x \approx y, y \approx z, \quad x \not\approx z$

Специальные значения

Специальное значение	64-битное представление IEEE 754
Положительная бесконечность	`0x7ff0000000000000L`
Отрицательная бесконечность	`0xfff0000000000000L`
Не число (NaN)	`от 0x7ff0000000000001L`
	`до 0x7fffffffffffffffL`
	`и от 0xfff0000000000001L`
	`до 0xffffffffffffffffL`
Отрицательный нуль	`0x8000000000000000`
Положительный нуль	`0x0000000000000000`