Д.Кнут. Искусство программирования для ЭВМ. т 3. Москва. Мир. 1996-1998
Т.Кормен, Ч.Лейзерсон, Р.Ривест. Алгоритмы. Построение и анализ. Москва. МЦНМО. 1999.
Препарата Ф., Шеймос М. Вычислительная геометрия. Москва. Мир. 1989
Брайан Керниган, Деннис Ритчи. Язык программирования Си.

Таненбаум Эндрю, Бос Херберт. Современные операционные системы. 4-е изд. Питер. 2022.

http://lectures.stargeo.ru

+7-916-118-78-10

WhatsApp

Позиционная система счисления

$\mathbf{x} \in \mathbb{Z}$, sign$\in \{1,-1\}$, x_i – k-ичная цифра

$\mathbf{x}=\mathrm{sign}\cdot(x_{n-1},\ldots,x_0)_k=\sum_{i=0}^{i<n} x_i\cdot k^i$

k=2

бит = минимальная ячейка памяти ЭВМ (=одна двоичная цифра)

байт = минимальная ячейка памяти в ЭВМ, которая имеет адрес

Целые числа без знака

$\mathbf{x}=(x_{n-1},\ldots,x_0)_2=\sum_{i=0}^{i<n} x_i\cdot 2^i$

байт: **** ****

7654 3210

Liitle Endian – первый байт младший, порядок: 0 1 2 3

Big Endian – первый байт старший, порядок: 3 2 1 0

Целые числа со знаком

1. Прямой код: -1: $(1000\ 0001)_2$
2. Обратный код: -1: $(1111\ 1110)_2$
3. Долнительный код: x>0:
   x→-x:
   1)битовой представление x
   2)инверсия x
   3)x+=1

Теорема. *Дополнительный код = представление числа в кольце вычетов по модулю 2^n, где n – к-во бит в представлении чесла.*

x →
2)~ ⇔ $(2^n-1)-x$

3)+=1: $(2^n-1)-x+1=2^n-x == -x \pmod{2^n}$

Макс.>0: $(01\ldots1)=2^{n-1}-1$ – предст.в доп.коде

Мин.<0: (10…0)=-x – предст.в доп.коде
x>0:
-=1: (011…11)
~: $(100\ldots00)=2^{n-1}$

Число в доп.коде $\in[-2^{n-1}, 2^{n-1}-1]$

-1+(-1)=(1…1)+ (1…1)=(111…1110)=-x

x:

-=1: (111…101)
~: (00…010)=2

$a<<b \Leftrightarrow a\cdot 2^b$

$a>>b \Leftrightarrow a/2^b$

Вещественные числа с плавающей точкой

$\forall\ x\in\mathbb{R} : x=s\cdot m\cdot 2^d$, где $s=\pm 1$, $m\in\mathbb{R}$, $d\in\mathbb{Z}$

$\forall\ x\in\mathbb{R}, x\neq 0 :\exists! : x=s\cdot m\cdot 2^d$, где $s=\pm 1$, $m\in\mathbb{R}$, $m\in[1,2)$, $d\in\mathbb{Z}$

Представление:

s – один бит

d – n_d бит – называется *степенью*

m – n_m бит – называется *мантиссой*

Вместо степени хранится *характеристика*

$c=d-d_{min} =d-(-(2^{nd-1}-1))= d+(2^{nd-1}-1)$

float: $n_d=8$, $n_m=23$

double: $n_d=11$, $n_m=52$

long double: Microsoft == double

gcc: 64/80/128 бит

$m=(1.**********)_2$

---n_m бит---

1.f = (0) (0111 1111) (00…0)

1 127 = 1

---

d>0: 128+65+5=193+d

d<0: ~(128+65+d)=62-d

m: x>0: b+1

x<0: 101-b

$-1.01=-01.01\cdot 100^0$=(62)(100)(100)(102)

-1.100001=-01.010001=(62)(100)(100)(101)(100)(102)

-1.01=-01.01·100^0= (62)(100)(100)(102)

IEEE

Алгоритм

Допустимые операции S={t_k} – мн-во допустимых операций

Алгоритм – формализоацнная процедура в терминах допустимых операций

Алгоритм m: In(m), Out(m)

m: In(m)→ **конечная** последовательность{t_i}

???

Каждой допустимой операции $t_k \rightarrow T(t_k) \in \mathbb{R}^+$ - *время выполнения операции t_k*

Время работы алгоритма m на вх.данных In(m)=сумма времен выполнения всех допустимых оепраций из последовтельности операций m(In(m))

Верхняя оценка времени *работы* алгоритма:

$\Phi(n)$: $\mathbb{Z}^+ \rightarrow \mathbb{R}^+$: $\forall$ n$\in\mathbb{Z}^+$ $\forall$ In(m): #In(m)$\leq$n: T(m,In(m))$\leq\Phi$(n)

Нижняя оценка времени *работы* алгоритма:

$\varphi(n)$: $\mathbb{Z}^+ \rightarrow \mathbb{R}^+$: $\forall$ n$\in\mathbb{Z}^+$ $\exists$ In(m): #In(m)=n:

T(m,In(m))$\geq$ φ(n)

Верхняя оценка времени *решения* задачи z:

Φ(n) $\mathbb{Z}^+ \to \mathbb{R}^+$: ∃ m решения z, для которого Φ(n) – верхняя оценка времени работы алгоритма.

Нижняя оценка времени *решения* задачи z:

φ(n): $\mathbb{Z}^+ \to \mathbb{R}^+$:∀ m, решающего z, φ(n) – является нижней оценкой времени работы алгоритма.

Задача z_1 **сводится к задаче** z_2 за время g(n): $\mathbb{Z}^+ \to \mathbb{R}^+$:

∃ алгоритм m: $In(z_1) \to In(z_2)$: # $In(z_1)$= #$In(z_2)$=n +

$Out(z_2) \to Out(z_1)$ – и все это за суммарное время ≤ g(n)

Теорема 1. Пусть задача z_1 **сводится к задаче** z_2 за время g(n). Для z_2 существует верхняя оценка времени решения задачи $\Phi_2(n)$. **То**: $\Phi_1(n)= \Phi_2(n)+g(n)$ - верхняя оценка времени решения задачи z_1 .

Теорема 2. Пусть задача z_1 **сводится к задаче** z_2 за время g(n). Для z_1 существует нижняя оценка времени решения задачи $\Phi_1(n)$. **То**: $\Phi_2(n)= \Phi_1(n)-g(n)$ - нижняя оценка времени решения задачи z_2 .

Stack growth upward/downward

Цикл выполнения команды

1)По регистру с адресом тек.команды команда загр.в процессор

2)Регистр тек.команды увеличивается на размер команды

3)Выполняется команда

---

valgrind
gdb
run
list filename:stringNumber
print var
break filename:stringNumber
set var Name=Value
next – выполнить следующую строку
step – выполнить следующую инструкцию
cont = continue
bt = back trace

Microsoft
_CrtCheckMemoryLeaks
_CrtDumpMemory
/MTd
/RTC1
/D _DEBUG

Сортировки

Перестановка : s: $\{1,\ldots,n\}\to\{1,\ldots,n\}$ – вз-однозначная

X: 1)$\forall$ x,y$\in$ X выполняется ровно одно соотношение:

$x<y$

$x>y$

$x=y$

2)Выполняется транзитивность:

$x,y,z\in\mathbf{X}$: $x<y,\ y<z \Rightarrow x<z$

$x,y,z\in\mathbf{X}$: $x>y,\ y>z \Rightarrow x>z$

$x,y,z\in\mathbf{X}$: $x=y,\ y=z \Rightarrow x=z$

3)Симметричность: $x<y \Leftrightarrow y>x$

Следствия:

1. $x=y \Rightarrow y=x$
2. $x<y\le z \Rightarrow x<z$ д-во:

если $x<y<z \Rightarrow x<z$

если $x<y=z$: от противного:

допустим $x=z \Rightarrow x=z, z=y \Rightarrow x=y$ – противоречие

допустим $x>z \Rightarrow z<x, x<y \Rightarrow z<y$ – противоречие

3. $x\le y< z \Rightarrow x<z$

Сортировкой последовательности $\{x_1,\ldots,x_n\}$, $x_i\in\mathbf{X}$ называется такая перестановка s:

либо $x_{s(i)}< x_{s(i+1)}$, либо $x_{s(i)}= x_{s(i+1)}$ $(1\le i<n)$

Вопрос: $\exists$ сортировка ?

Пример: $\forall$ x,y$\in$ **X**: x>y

Теорема. *При выполнении трех аксиом сортировка существует и единственна с точностью до перестановки равных элементов*. Причем равные элементы после перестановки могут идти только подряд.

Д-во

∃ : по индукции

Расставим по сортировке n элементов и справа на лево ищем место n+1-го элемента

! : д-жем, что равные элементы идут подряд. Это следствие из следствий 2,3.

Ч.Т.Д.

Следствие: для работы сортировки достаточно одной операции <

$a>b \Leftrightarrow b<a$

$a=b \Leftrightarrow (!(a<b))\ \&\&\ !(b<a)$

Модель: элементарные операции: сравнение за пост.время, SWAP(a_i,a_j) за время $\Theta(|i-j|)$

Теорема. *Для данной модели нижняя оценка решения задачи сортировки* $= \Theta(n^2)$, *где* n – *размер сортируемого массива.*

Д-во. Возьмем элементы в обратном порядке: n,n-1,…,1 – их можно переставить не быстрее, чем за время $\Theta(n^2)$.

Рассмотрим сортировку пузырьком – он работает в рамках данной модели за время $\Theta(n^2)$

Теорема. *Алгоритм сортировки пузырьком оптимален в рамках данной модели*.

Теорема. Время работы алгоритма Merge = $\Theta(n1+n2)$, где n1, n2 – длины сливаемых массивов.

Теорема. Время работы алгоритма Sort1 = $\Theta(n \cdot \log_2(n))$, где n – длина сортируемого массива ⇔ У алгоритма Sort1 существует верхняя оценка времени работы = $\Theta(n \cdot \log_2(n))$.

Алгоритмы, основанные на сравнениях.

Алгоритм, основанный на сравнениях = алгоритм, представимый в виде *дерева решения*.

Бинарное дерево = состоит из вершин и ребер, из каждой вершины выходит не более двух ребер, в каждую вершину, кроме одного входит одно ребро. В *корень* не входит ребер.

Дерево решения = бинарное дерево, где:

Каждой **вершине** соотв.своя ф-ция f(v), ее сравннение с нулем (какой-то определенной операцией) и в зависимости от рез-та переход на правую или левую дочернюю ветвь.

Каждой **ветви** соот. определенная перестановка.

Длина ветви = количество вершин в ветви
Высота дерева = макс.длина ветвей в дереве

Будм считать, что вычисление ф-ций и их сравнение с 0 происходит за нулевое время.
Произведение перестановки работает за время $\Theta(1)$.
В этих условиях:
Утверждение. Нижняя оценка времени работы алгоритма = Θ(H(дерева решения)).

В деревьях решения отбросим все заврешающие вершины, до которых нельзя добраться.

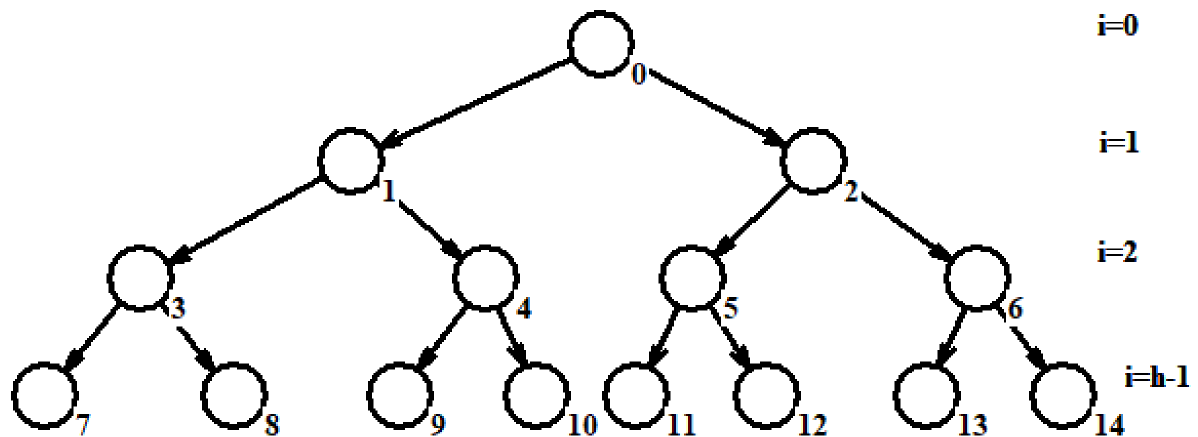
Рассмотрим задачу сортировки на наборе из n различных значений.
Количество завершающих вершин = количеству перестановок упорядоченного массива из n элементов = n!
Пусть есть дерево высоты h
К-во завершающих вершин
$m \leq 2^{h-1}$
$\log_2(m) \leq h-1$
$h \geq \log_2(m)+1$
$h > \log_2(m)$
$h > \log_2(n!)$
$n! = (\sqrt{(2\pi n)}) \cdot n^n \cdot e^{-n} \cdot (1+o(1))$
$h > \log_2((\sqrt{(2\pi n)}) \cdot n^n \cdot e^{-n} \cdot (1+o(1)))=$
$= \log_2(\sqrt{(2\pi n)}) + \log_2(n^n) + \log_2(e^{-n}) + \log_2((1+o(1)))=$
$= \log_2(n^n) \cdot (1+o(1)) = \Theta(n \cdot \log_2(n))$

Теорема. Для алгоритмов, основанных на сравнениях, существует нижняя оценка времени решения задачи = $\Theta(n \cdot \log_2(n))$.

Теорема. Время работы алгоритма Sort1 = $\Theta(n \cdot \log_2(n))$, где n – длина сортируемого массива ⬄ У алгоритма Sort1 существует верхняя оценка времени работы = $\Theta(n \cdot \log_2(n))$. Алгоритм Sort1 требует $\Theta(n)$ дополнительнй памяти в куче + $\Theta(\log_2(n))$ памяти в стеке.

Теорема. Время работы алгоритма Sort2 = $\Theta(n \cdot \log_2(n))$, где n – длина сортируемого массива ⬄ У алгоритма Sort2 существует верхняя оценка времени работы = $\Theta(n \cdot \log_2(n))$. Алгоритм Sort2 требует $\Theta(n)$ дополнительнй памяти в куче + $\Theta(1)$ памяти в стеке.

HeapSort



Опр. Массив m[] длины n называется

**-упорядоченным* если для всех корректных i

$(2i+1<n, 2i+2<n)$:

$m[i] \geq m[2i+1]$ и $m[i] \geq m[2i+2]$ = св-во *

```
1)for(int i=n-1;i>=0;i--)Heapify(m,n,i);
2)Heapify(m,n, I)
```

Пусть св-во * выполняется для всех i>I

⇨ св-во * выполняется i=I

```
for(int i=n-1;i>0;i--)
{swap(m[i],m[0]); Heapify(m,i,0);}
```

Теорема. Алгоритм HeapSort для массива из n элементов требует $\Theta(1)$ дополнительной памяти и работает за время $\Theta(n \cdot \log n)$.

Теорема. Создание пирамиды в алгоритме HeapSort для массива из n элементов требует $\Theta(1)$ дополнительной памяти и работает за время $\Theta(n)$.

Д-во.

$T(In(m),n) \le \Theta(\sum_{i=0}^{i<h} 2^i(h-i)) = [i=h-i] = \Theta(\sum_{i=1}^{i\le h} 2^{h-i}\, i) =$
$\Theta(2^h \cdot \sum_{i=1}^{i\le h} 0.5^i\, i)$ (=)

$2^{h-1} \le n \le 2^h-1$

$h=\Theta(\log_2(n))$

$2^h=\Theta(n)$

$2^{h-1} \le n$

$h-1 \le \log_2(n)$

$h \le \log_2(n)+1$

$\mathbf{2^h \le 2n}$

(=) $\Theta(n \cdot \sum_{i=1}^{i\le h} 0.5^i\, i)$ (=)

$\sum_{i=1}^{i\le h} 0.5^i\, i = [x=0.5] = \sum_{i=1}^{i\le h} x^i\, i = f(x) = x \cdot (\sum_{i=1}^{i\le h} x^i)' =$
$x \cdot (\sum_{i=0}^{i\le h} x^i)' = x \cdot ((1-x^{h+1})/(1-x))' =$

$= x \cdot (((-(h+1)x^h(1-x) + (1-x^{h+1})))/(1-x)^2) = x \cdot 1/(1-x)^2 \cdot$
$(1+o_{h\to\infty}(1)) = 2 \cdot (1+o_{h\to\infty}(1))$

(=)$\Theta(n)$.

Ч.Т.Д

QSort

QSort(m,0,n-1)

```
QSort_(m,p,q)
if(p≥q)return;
i=p;j=q; M=m[p]
while(1)
  while(m[i]<M) i++
```

while(M<m[j])j—
Если встретились
QSort_(*для первой половины*)
QSort_(*для второй половины*)
return
swap(m[i],m[j]); i++;j--;

Теорема. Алгоритм QSort_ требует O(n) дополнительной памяти в стеке и работает за время $O(n^2)$