

ЛЕКЦИЯ 13 - ПИРАМИДАЛЬНАЯ СОРТИРОВКА

До сих пор в наших примерах выбора из дерева в той или иной мере предполагалось, что N есть степень 2 (см. напр. рис. 1 - это рис. 4 предыдущей лекции о древовидной сортировке); в действительности можно работать с произвольным значением N , так как полное бинарное дерево с N концевыми узлами нетрудно построить для любого N .

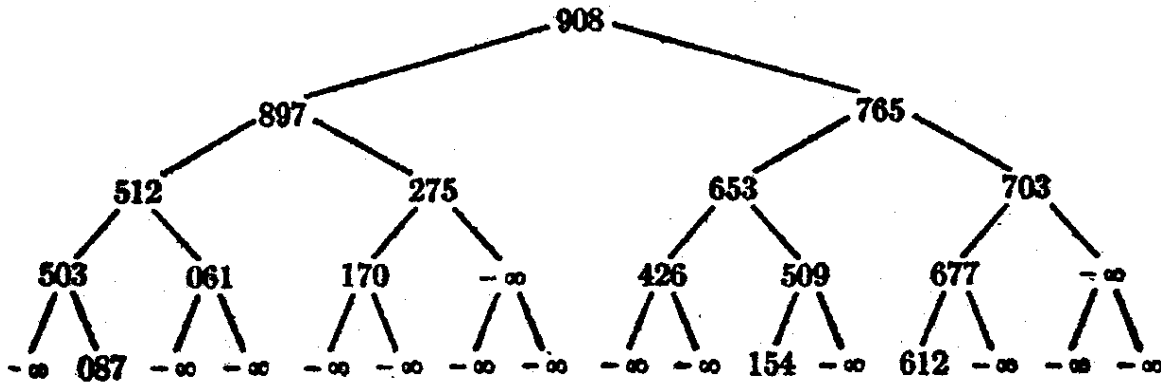


Рис 1. Применение корпоративной системы выдвижений к сортировке. Каждый поднимается на свой уровень некомпетентности в иерархии.

Мы подошли теперь к основному вопросу: нельзя ли в нисходящем методе обойтись совсем без " $-\infty$ "? Не правда ли, было бы чудесно, если бы всю существенную информацию, имеющуюся на рис. 4, удалось расположить в ячейках 1—16 полного бинарного дерева без всяких бесполезных "дыр", содержащих $-\infty$? Поразмыслив, можно прийти к выводу, что эта цель в действительности достижима, причем не только исключается $-\infty$, но и появляется возможность сортировать N записей на том же месте без вспомогательной области вывода. Это приводит к еще одному важному алгоритму сортировки. Его автор Дж. У. Дж. Уильямс [*SACM*, 7 (1964), 347—348] окрестил свой алгоритм "пирамидальной сортировкой" ("heap-sort").

Пирамидальная сортировка. Будем называть файл ключей K_1, K_2, \dots, K_N "пирамидой", если

$$K_{\lfloor j/2 \rfloor} \geq K_j \quad \text{при} \quad 1 \leq \lfloor j/2 \rfloor < j \leq N \quad (1)$$

В этом случае $K_1 \geq K_2, K_1 \geq K_3, K_2 \geq K_4$ и т. д. Именно это условие выполняется на рис. 1. Из него следует, в частности, что наибольший ключ оказывается "на вершине пирамиды":

$$K_1 = \max(K_1, K_2, \dots, K_N) \quad (2)$$

Если бы мы сумели как-нибудь преобразовать произвольный исходный файл в пирамиду, то для получения эффективного алгоритма сортировки можно было бы воспользоваться "нисходящей" процедурой выбора, подобной той, которая описана выше.

Эффективный подход к задаче построения пирамиды предложил Р. У. Флойд [*SACM*, 7 (1964), 701]. Пусть нам удалось расположить файл таким образом, что

$$K_{\lfloor j/2 \rfloor} \geq K_j \quad \text{при} \quad l \leq \lfloor j/2 \rfloor < j \leq N \quad (3)$$

где l —некоторое число ≥ 1 . (В исходном файле это условие выполняется "автоматически" для $l = \lfloor N/2 \rfloor$, поскольку ни один индекс j не удовлетворяет условию $\lfloor N/2 \rfloor < \lfloor j/2 \rfloor < j \leq N$.) Нетрудно понять, как, изменяя лишь поддерево с корнем в узле l , преобразовать файл, чтобы

распространить неравенства (3) и на случай, когда $\lfloor j/2 \rfloor = l$. Следовательно, можно уменьшать l на единицу, пока в конце концов не будет достигнуто условие (1). Эти идеи Уильямса и Флойда приводят к изящному алгоритму, который заслуживает пристального изучения.

Алгоритм Н. (*Пирамидальная сортировка.*) Записи R_1, \dots, R_N переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$. Сначала файл перестраивается в пирамиду, после чего вершина пирамиды многократно исключается и записывается на свое окончательное место. Предполагается, что $N \geq 2$.

- Н1.** [Начальная установка.] Установить $l \leftarrow \lfloor N/2 \rfloor + 1, r \leftarrow N$.
- Н2.** [Уменьшить l или r .] Если $l > 1$, то установить $l \leftarrow l-1, R \leftarrow R_l, K \leftarrow K_l$ (Если $l > 1$, это означает, что происходит процесс преобразования исходного файла в пирамиду; если же $l = 1$, то это значит, что ключи K_1, K_2, \dots, K_r уже образуют пирамиду.) В противном случае установить $R \leftarrow R_r, K \leftarrow K_r, R_r \leftarrow R_1, r \leftarrow r - 1$; если в результате оказалось, что $r = 1$, то установить $R_1 \leftarrow R$ и завершить работу алгоритма.
- Н3.** [Приготовиться к "протаскиванию".] Установить $j \leftarrow l$. (К этому моменту

$$K_{\lfloor j/2 \rfloor} \geq K_j \text{ при } l < \lfloor j/2 \rfloor < j \leq r, \quad (4)$$

а записи $R_k, r < k \leq N$, занимают свои окончательные места. Шаги **Н3** — **Н8** называются алгоритмом "протаскивания"; их действие эквивалентно установке $R_l \leftarrow R$ с последующим перемещением записей R_l, \dots, R_r таким образом, чтобы условие (4) выполнялось и при $\lfloor j/2 \rfloor = l$.)

- Н4.** [Продвинуться вниз.] Установить $i \leftarrow j$ и $j \leftarrow 2j$. (В последующих шагах $i = \lfloor j/2 \rfloor$.) Если $j < r$, то перейти к шагу **Н5**; если $j = r$, то перейти к шагу **Н6**; если же $j > r$, то перейти к шагу **Н8**.
- Н5.** [Найти "большого" сына.] Если $K_j < K_{j+1}$, то установить $j \leftarrow j+1$.
- Н6.** [Больше K ?] Если $K \geq K_j$, то перейти к шагу **Н8**.
- Н7.** [Поднять его вверх.] Установить $R_i \leftarrow R_j$ и возвратиться к шагу **Н4**.
- Н8.** [Занести R .] Установить $R_i \leftarrow R$. (На этом алгоритм "протаскивания", начатый в шаге **Н3**, заканчивается.) Возвратиться к шагу **Н2**.

Пирамидальную сортировку иногда описывают как ∇ -алгоритм, это обозначение указывает на характер изменения переменных l и r . Верхний треугольник соответствует фазе построения пирамиды, когда $r = N$, а l убывает до 1; нижний треугольник представляет фазу выбора, когда $l = 1$, а r убывает до 1. В табл. 1 показан процесс пирамидальной сортировки все тех же шестнадцати чисел. (В каждой строке изображено состояние после шага **Н2**, скобки указывают на значения переменных l и r .) Блок-схема алгоритма пирамидальной сортировки приведена на рис. 2.

Программа, основанная на рассмотренном алгоритме, будет приблизительно лишь вдвое длиннее программы, построенной по алгоритму S , но при больших N она гораздо более эффективна. Ее время работы зависит от:

$P = N + \lfloor N/2 \rfloor - 2 =$ число "протаскиваний";

$A =$ число протаскиваний, при которых ключ K в конце попадает во внутренний узел пирамиды;

$B =$ суммарное число ключей, просмотренных во время протаскиваний;

$C =$ число присваиваний $j \leftarrow j + 1$ в шаге **Н5**;

$D =$ число случаев, когда в шаге **Н4** $j = r$.

Пример пирамидальной сортировки

K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}	l	r	K
503	087	512	061	908	170	897	[275	653	426	154	509	612	677	765	703]	8	16	275
503	087	512	061	908	170	[897	703	653	426	154	509	612	677	765	275]	7	16	897
503	087	512	061	908	[170	897	703	653	426	154	509	612	677	765	275]	6	16	170
503	087	512	061	[908	612	897	703	653	426	154	509	170	677	765	275]	5	16	908
503	087	512	[061	908	612	897	703	653	426	154	509	170	677	765	275]	4	16	061
503	087	[512	703	908	612	897	275	653	426	154	509	170	677	765	061]	3	16	512
503	[087	897	703	908	612	765	275	653	426	154	509	170	677	512	061]	2	16	087
[503	908	897	703	426	612	765	275	653	087	154	509	170	677	512	061]	1	16	503
[908	703	897	653	426	612	765	275	503	087	154	509	170	677	512]	908	1	15	061
[897	703	765	653	426	612	677	275	503	087	154	509	170	061]	897	908	1	14	512
[765	703	677	653	426	612	512	275	503	087	154	509	170]	765	897	908	1	13	061
[703	653	677	503	426	612	512	275	061	087	154	509]	703	765	897	908	1	12	170
[677	653	612	503	426	509	512	275	061	087	154]	677	703	765	897	908	1	11	170
[653	503	612	275	426	509	512	170	061	087]	653	677	703	765	897	908	1	10	154
[612	503	512	275	426	509	154	170	061]	612	653	677	703	765	897	908	1	9	087
[512	503	509	275	426	087	154	170]	512	612	653	677	703	765	897	908	1	8	061
[509	503	154	275	426	087	061]	509	512	612	653	677	703	765	897	908	1	7	170
[503	426	154	275	170	087]	503	509	512	612	653	677	703	765	897	908	1	6	061
[426	275	154	061	170]	426	503	509	512	612	653	677	703	765	897	908	1	5	087
[275	170	154	061]	275	426	503	509	512	612	653	677	703	765	897	908	1	4	087
[170	087	154]	170	275	426	503	509	512	612	653	677	703	765	897	908	1	3	061
[154	087]	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	2	061
[061]	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	1	061

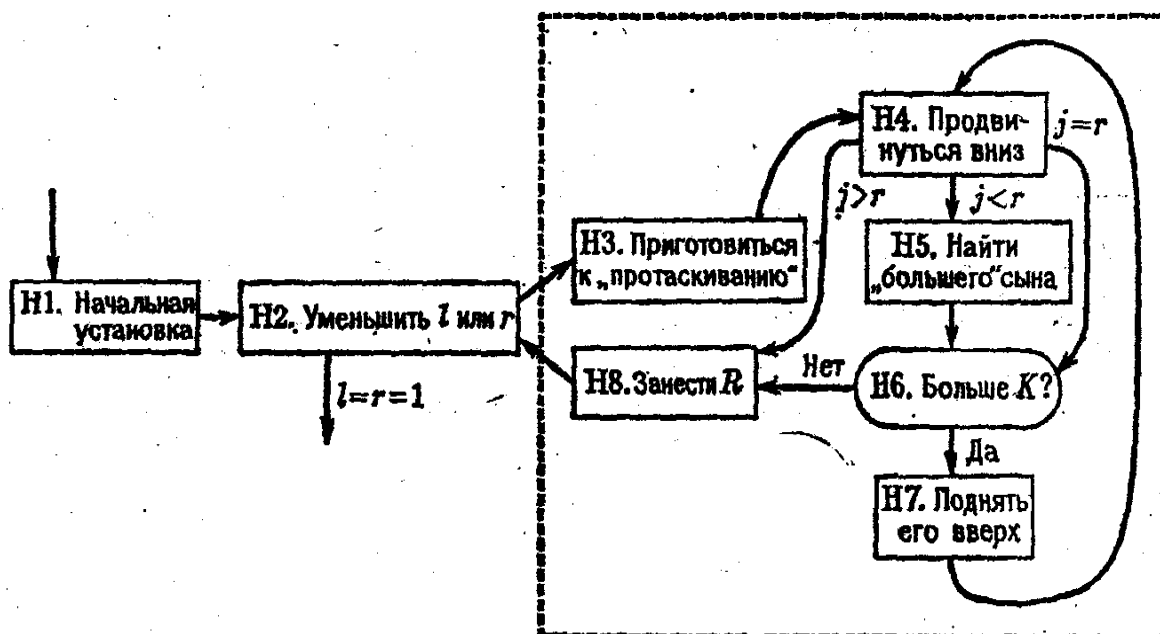


Рис. 2. Блок - схема алгоритма пирамидальной сортировки

Эти величины проанализированы ниже. Как показывает практика, они сравнительно мало отклоняются от своих средних значений:

$$A \approx 0.349N; \quad B \approx N \log_2 N - 1.87N; \quad C \approx \frac{1}{2} N \log_2 N - 0.9N; \quad D \approx \ln N. \quad (5)$$

При $N = 1000$, например, четыре эксперимента со случайными исходными данными показали соответственно результаты $(A, B, C, D) = (371, 8055, 4056, 12)$, $(351, 8072, 4087, 14)$, $(341, 8094, 4017, 8)$, $(340, 8108, 4083, 13)$.

Общее время работы $7A+14B+4C+20N-2D+15 \lfloor N/2 \rfloor - 28$ равно, таким образом, в среднем примерно $16N \log_2 N + 0.2N$ единиц.

Глядя на табл. 1, трудно поверить в то, что пирамидальная сортировка так уж эффективна: большие ключи перемещаются влево прежде, чем мы успеваем отложить их вправо! Это и в самом деле странный способ сортировки при малых N . Время сортировки 16 ключей из табл. 1 равно 1068 ед., тогда как обычный метод простых вставок требует всего 514 ед. При сортировке простым выбором требуется 853 ед.

При больших N алгоритм **Н** более эффективен. Напрашивается сравнение с сортировкой методом Шелла с убывающим шагом и быстрой сортировкой Хоара, так как во всех трех алгоритмах сортировка производится путем сравнения ключей, причем вспомогательной памяти используется мало или она не используется вовсе. При $N=1000$ средние времена работы равны приблизительно

160000 ед. для пирамидальной сортировки;

130000 ед. для сортировки методом Шелла;

80000 ед. для быстрой сортировки.

С ростом N пирамидальная сортировка превзойдет по скорости метод Шелла, но асимптотическая формула $16N \log_2 N \approx 23.08N \ln N$ никогда не станет лучше выражения для быстрой сортировки $12.67N \ln N$. С другой стороны, быстрая сортировка эффективна лишь в среднем; в наихудшем случае ее время работы пропорционально N^2 . Пирамидальная же сортировка обладает тем интересным свойством, что для нее наихудший случай не намного хуже среднего. Всегда выполняются неравенства

$$A \leq 1.5N, \quad B \leq N \lfloor \log_2 N \rfloor, \quad C \leq N \lfloor \log_2 N \rfloor; \quad (6)$$

таким образом, независимо от распределения исходных данных выполнение алгоритма **Н** не займет более $18N \lfloor \log_2 N \rfloor + 38N$ единиц времени. Пирамидальная сортировка — первый из рассмотренных нами до сих пор методов сортировки, время работы которого *заведомо* имеет порядок $N \log N$. Сортировка посредством слияний, которую мы обсудим позднее, тоже обладает этим свойством, но она требует больше памяти.