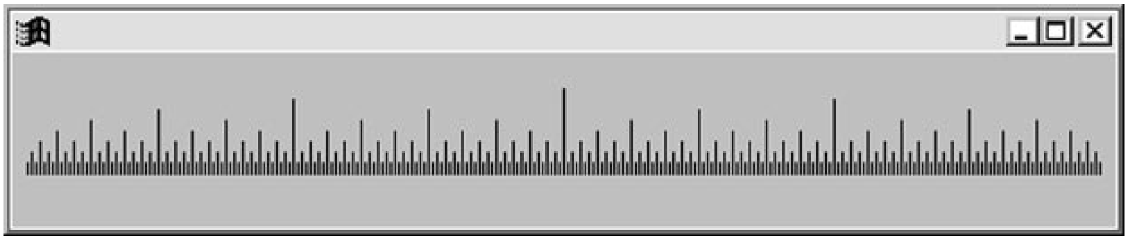
Chapter 7

Recursion

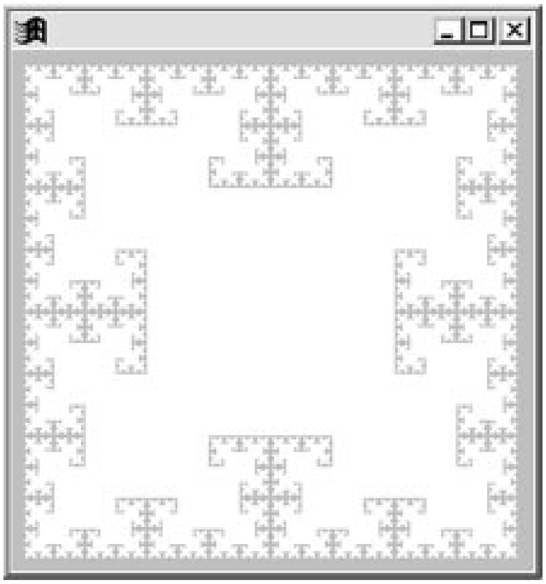
TOP:	`s(2)`
	`s(3)`
	`s(4)`
	`main()`

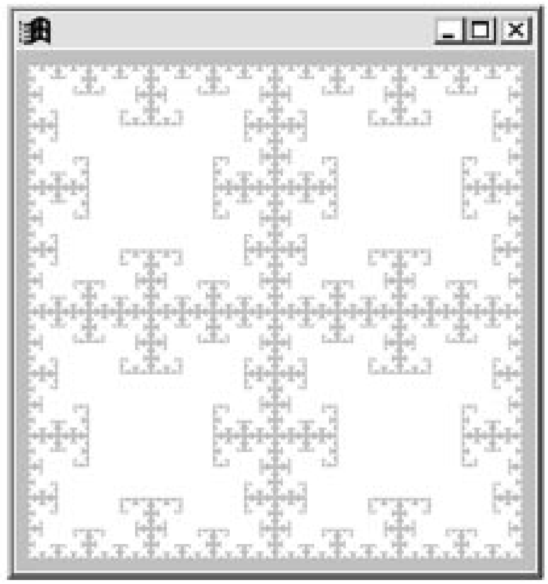
Stack of activation records



```
private void
drawRuler( Graphics g, int left, int right, int level)
{
    if( level < 1 )
        return;

    int mid = ( left + right ) / 2;

    g.drawLine( mid, 80, mid, 80 - level * 5 );

    drawRuler( g, left, mid - 1, level- 1 );
    drawRuler( g, mid + 1, right, level - 1 );
}
```

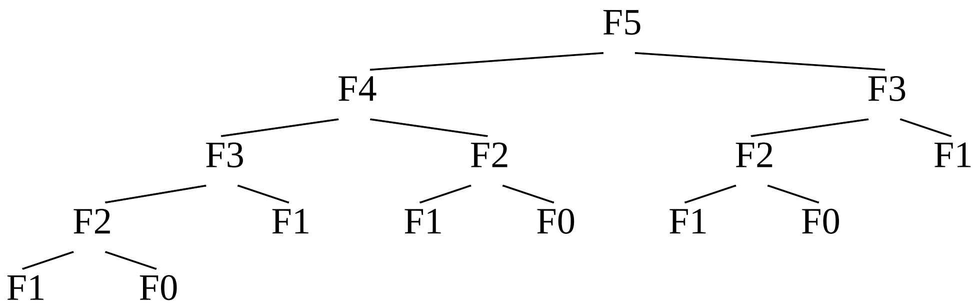
Ruler





```
// Draw picture above (left)
private void drawFractal( Graphics g, int xCenter,
                          int yCenter, int boundingDim )
{
    int side = boundingDim / 2;

    if( side < 1 )
        return;

        // Compute corners
    int left =   xCenter - side / 2;
    int top =    yCenter - side / 2;
    int right =  xCenter + side / 2;
    int bottom = yCenter + side / 2;

        // Recursively draw four quadrants
    drawFractal( g, left, top, boundingDim / 2 );
    drawFractal( g, left, bottom, boundingDim / 2 );
    drawFractal( g, right, top, boundingDim / 2 );
    drawFractal( g, right, bottom, boundingDim / 2 );

        // Draw central square, overlapping quadrants
    g.fillRect( left, top, right - left, bottom - top );
}
```

Fractal star outline



Trace of the recursive calculation of the Fibonacci numbers

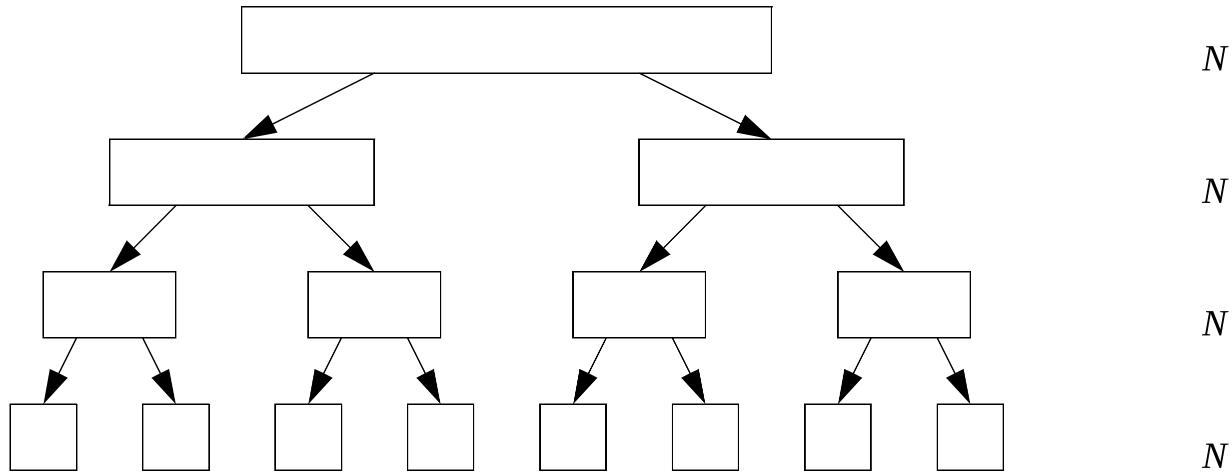
- *Divide*: Smaller problems are solved recursively (except, of course, base cases).
- *Conquer*: The solution to the original problem is then formed from the solutions to the subproblems.

Divide-and-conquer algorithms

First Half				Second Half				
4	-3	5	-2	-1	2	6	-2	Values
4*	0	3	-2	-1	1	7*	5	Running Sums

Running Sum from the Center (*denotes maximum for each half)

Dividing the maximum contiguous subsequence problem into halves



Trace of recursive calls for recursive maximum contiguous subsequence sum algorithm

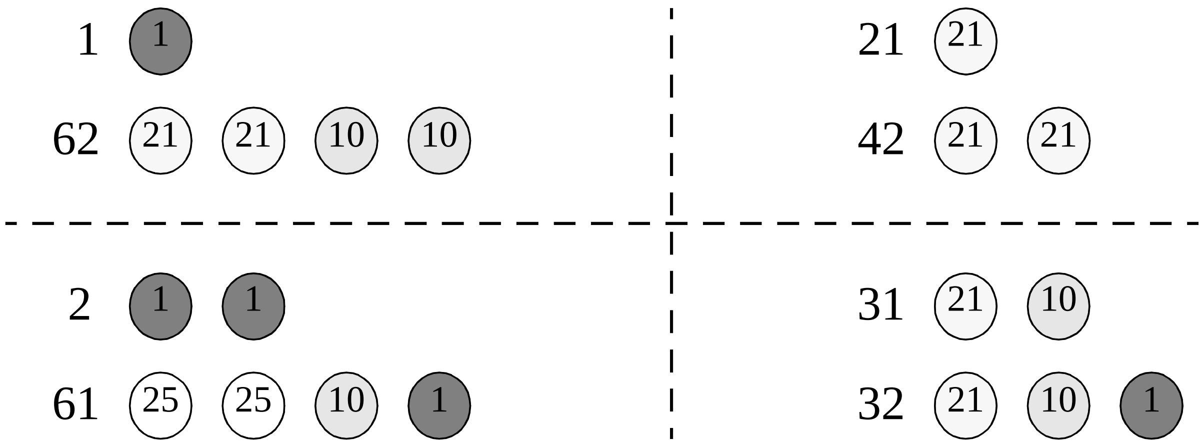
Assuming N is a power of 2, the solution to the equation $T(N) = 2T(N/2) + N$, with initial condition $T(1) = 1$ is $T(N) = N \log N + N$.

Basic divide-and-conquer running time theorem

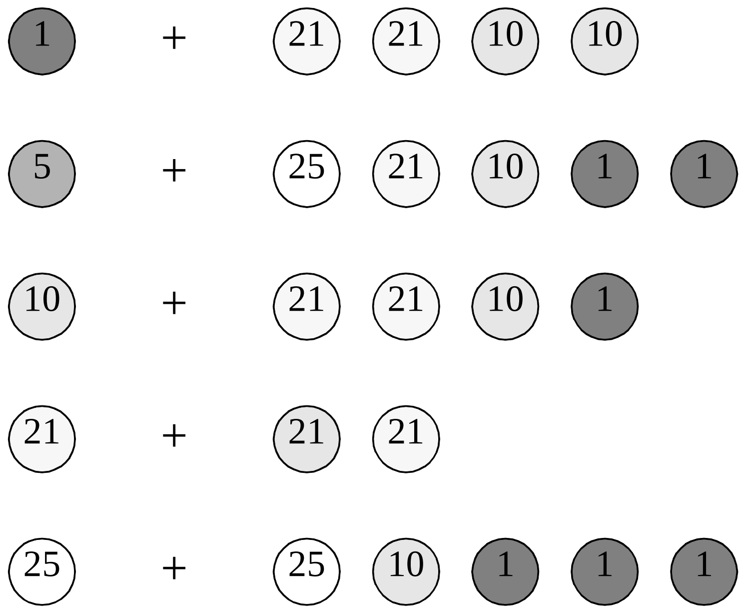
The solution to the equation $T(N) = AT(N/B) + O(N^k)$, where $A \geq 1$ and $B > 1$, is

$$T(N) = \begin{cases} O(N^{\log_B A}) & \textit{if } A > B^k \\ O(N^k \log N) & \textit{if } A = B^k \\ O(N^k) & \textit{if } A < B^k \end{cases}$$

General divide-and-conquer running time theorem



Some of the subproblems that are solved recursively in Figure 7.15



Alternative recursive algorithm for coin-changing problem

Chapter 8

Sorting Algorithms

- Words in a dictionary are sorted (and case distinctions are ignored).
- Files in a directory are often listed in sorted order.
- The index of a book is sorted (and case distinctions are ignored).
- The card catalog in a library is sorted by both author and title.
- A listing of course offerings at a university is sorted, first by department and then by course number.
- Many banks provide statements that list checks in increasing order (by check number).
- In a newspaper, the calendar of events in a schedule is generally sorted by date.
- Musical compact disks in a record store are generally sorted by recording artist.
- In the programs that are printed for graduation ceremonies, departments are listed in sorted order, and then students in those departments are listed in sorted order.

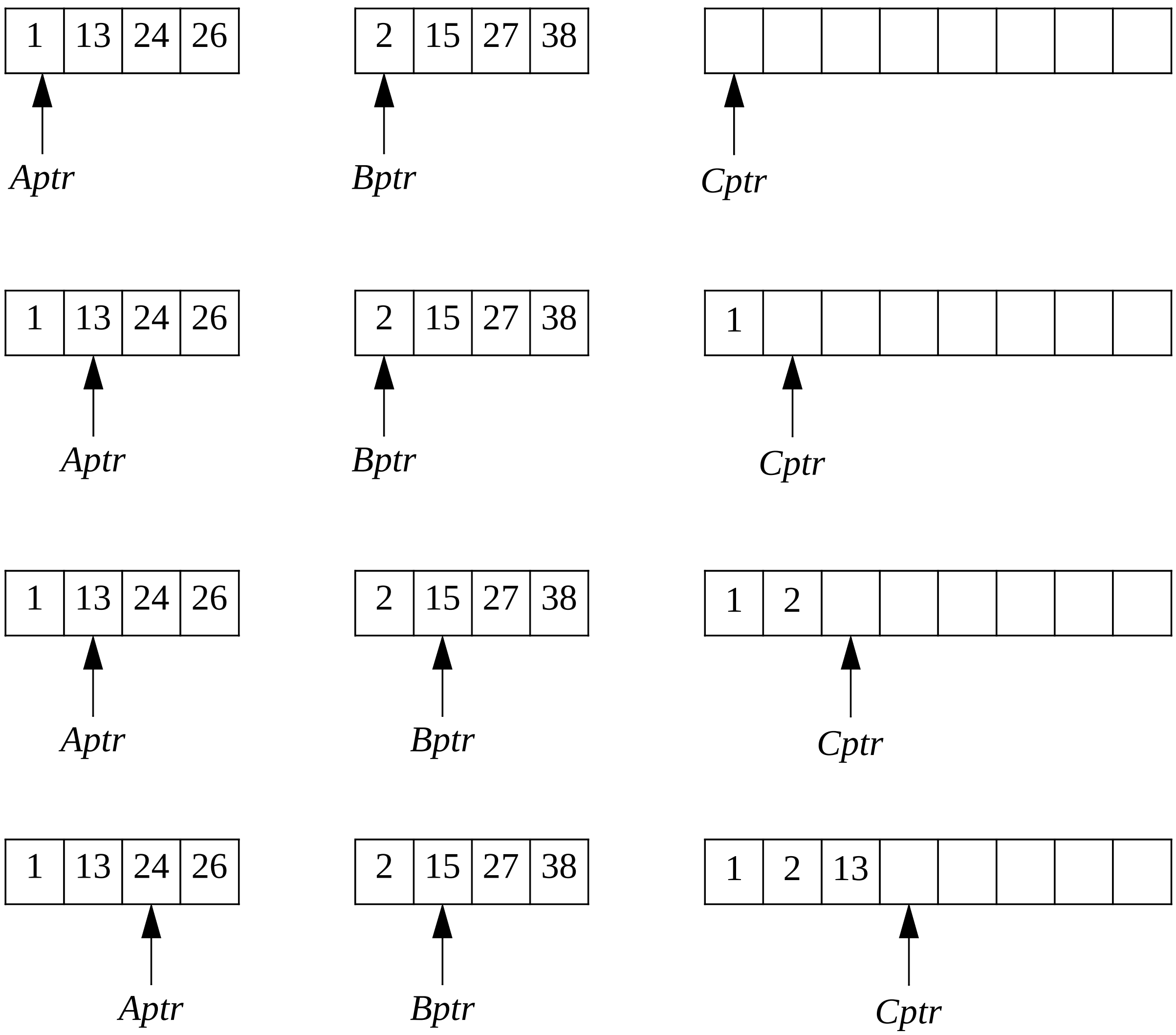
Examples of sorting

Original	**81**	**94**	**11**	**96**	**12**	**35**	**17**	**95**	**28**	**58**	**41**	**75**	**15**
After 5-sort	35	17	11	28	12	41	75	15	96	58	81	94	95
After 3-sort	28	12	11	35	15	41	58	17	94	75	81	96	95
After 1-sort	11	12	15	17	28	35	41	58	75	81	94	95	96

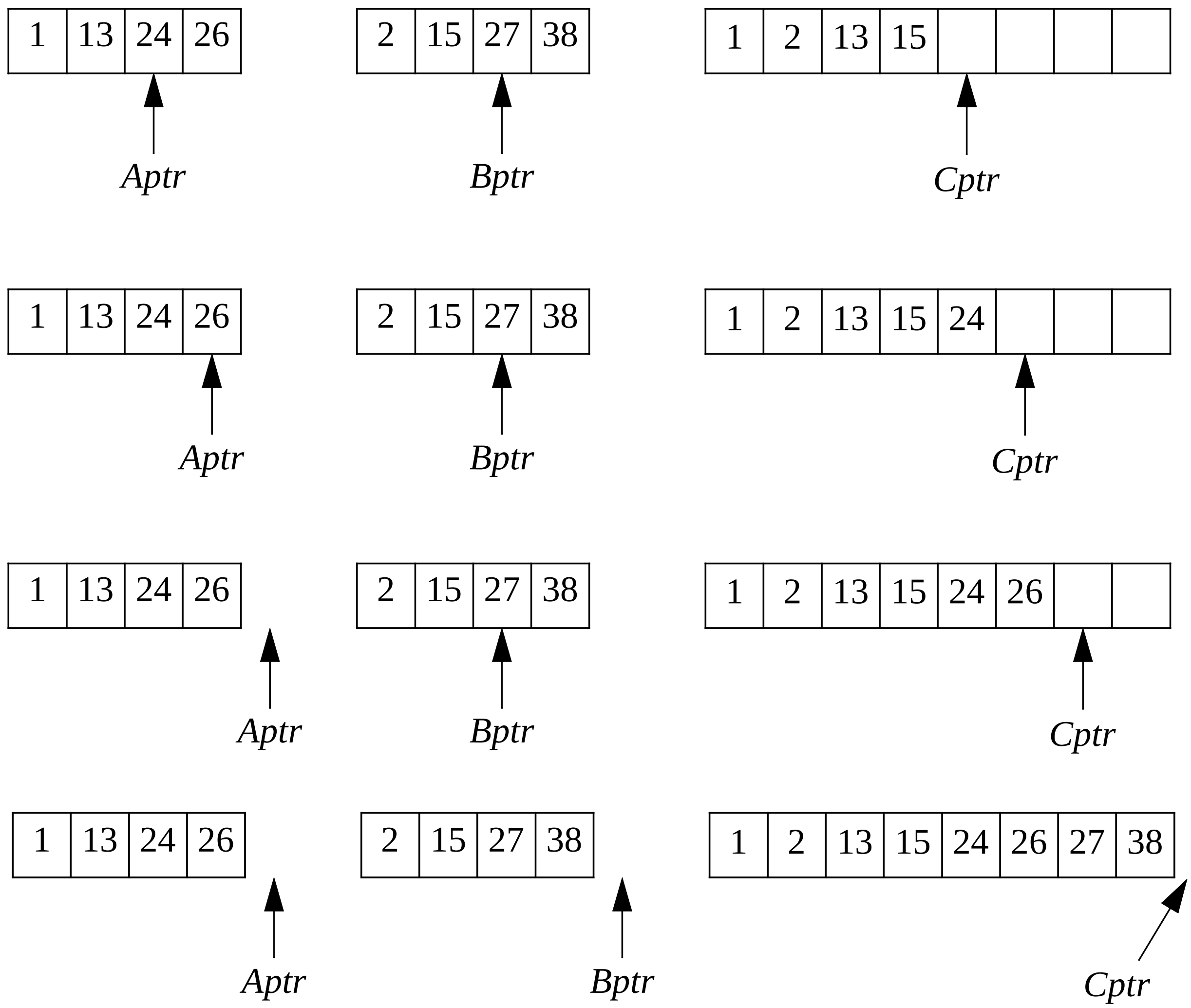
Shellsort after each pass, if increment sequence is {1, 3, 5}

N	Insertion sort	Shellsort		
		Shell’s	Odd gaps only	Dividing by 2.2
1,000	122	11	11	9
2,000	483	26	21	23
4,000	1,936	61	59	54
8,000	7,950	153	141	114
16,000	32,560	358	322	269
32,000	131,911	869	752	575
64,000	520,000	2,091	1,705	1,249

Running time (milliseconds) of the insertion sort and Shellsort with various increment sequences



Linear-time merging of sorted arrays (first four steps)

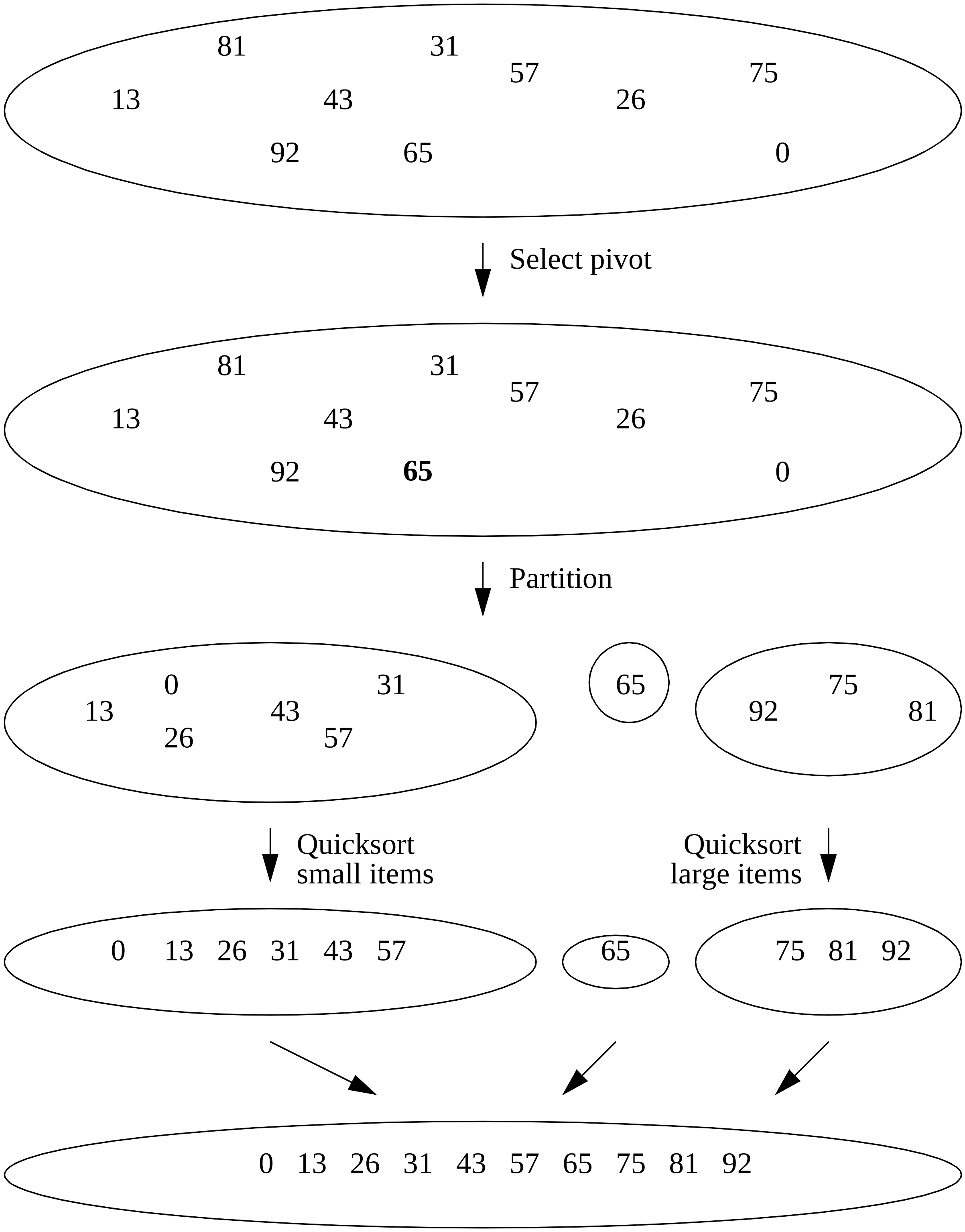


Linear-time merging of sorted arrays (last four steps)

The basic algorithm *Quicksort*(S) consists of the following four steps:

1. If the number of elements in S is 0 or 1, then return.
2. Pick *any* element v in S. This is called the *pivot*.
3. *Partition* $S - \{v\}$ (the remaining elements in S) into two disjoint groups: $L = \{x \in S - \{v\} | x \le v\}$ and $R = \{x \in S - \{v\} | x \ge v\}$.
4. Return the result of *Quicksort*(L) followed by v followed by *Quicksort*(R).

Basic quicksort algorithm



The steps of quicksort

Because recursion allows us to take the giant leap of faith, the correctness of the algorithm is guaranteed as follows:

- The group of small elements is sorted, by virtue of the recursion.
- The largest element in the group of small elements is not larger than the pivot, by virtue of the partition.
- The pivot is not larger than the smallest element in the group of large elements, by virtue of the partition.
- The group of large elements is sorted, by virtue of the recursion.

Correctness of quicksort

8	1	4	9	0	3	5	2	7	6

Partitioning algorithm: pivot element 6 is placed at the end

8	1	4	9	0	3	5	2	7	6

Partitioning algorithm: `i` stops at large element 8; `j` stops at small element 2

2	1	4	9	0	3	5	8	7	6

Partitioning algorithm: out-of-order elements 8 and 2 are swapped

2	1	4	9	0	3	5	8	7	6

Partitioning algorithm: `i` stops at large element 9; `j` stops at small element 5

2	1	4	5	0	3	9	8	7	6

Partitioning algorithm: out-of-order elements 9 and 5 are swapped

2	1	4	5	0	3	9	8	7	6

Partitioning algorithm: `i` stops at large element 9; `j` stops at small element 3

2	1	4	5	0	3	6	8	7	9

Partitioning algorithm: swap pivot and element in position `i`

8	1	4	9	6	3	5	2	7	0

Original array

0	1	4	9	6	3	5	2	7	8

Result of sorting three elements (first, middle, and last)

0	1	4	9	7	3	5	2	6	8

Result of swapping the pivot with next-to-last element

- We should not swap the pivot with the element in the last position. Instead, we should swap it with the element in the next to last position.
- We can start `i` at `low+1` and `j` at `high-2`.
- We are guaranteed that, whenever `i` searches for a large element, it will stop because in the worst case it will encounter the pivot (and we stop on equality).
- We are guaranteed that, whenever `j` searches for a small element, it will stop because in the worst case it will encounter the first element (and we stop on equality).

Median-of-three partitioning optimizations

1. If the number of elements in S is 1, then presumably k is also 1, and we can return the single element in S.
2. Pick any element v in S. This is the pivot.
3. *Partition* $S - \{v\}$ into L and R, exactly as was done for quicksort.
4. If k is less than or equal to the number of elements in L, then the item we are searching for must be in L. Call *Quickselect(L, k)* recursively. Otherwise, if k is exactly equal to one more than the number of items in L, then the pivot is the kth smallest element, and we can return it as the answer. Otherwise, the *kth* smallest element lies in R, and it is the $(k - |L| - 1)$th smallest element in R. Again, we can make a recursive call and return the result.

Quickselect algorithm

Chapter 9

Randomization

Winning Tickets	0	1	2	3	4	5
Frequency	0.135	0.271	0.271	0.180	0.090	0.036

Distribution of lottery winners if expected number of winners is 2

An important nonuniform distribution that occurs in simulations is the *Poisson distribution*. Occurrences that happen under the following circumstances satisfy the Poisson distribution:

- The probability of one occurrence in a small region is proportional to the size of the region.
- The probability of two occurrences in a small region is proportional to the square of the size of the region and is usually small enough to be ignored.
- The event of getting k occurrences in one region and the event of getting j occurrences in another region disjoint from the first region are independent. (Technically this statement means that you can get the probability of both events simultaneously occurring by multiplying the probability of individual events.)
- The mean number of occurrences in a region of some size is known.

Then if the mean number of occurrences is the constant a, then the probability of exactly k occurrences is $a^k e^{-a} / k!$.

Poisson distribution

Chapter 10

Fun and Games

	0	**1**	**2**	**3**
0	t	h	i	s
1	w	a	t	s
2	o	a	h	g
3	f	g	d	t

Sample word search grid

```
for each word W in the word list
    for each row R
        for each column C
            for each direction D
                check if W exists at row R, column C
                in direction D
```

Brute-force algorithm for word search puzzle

```
for each row R
    for each column C
        for each direction D
            for each word length L
                check if L chars starting at row R column C
                            in direction D form a word
```

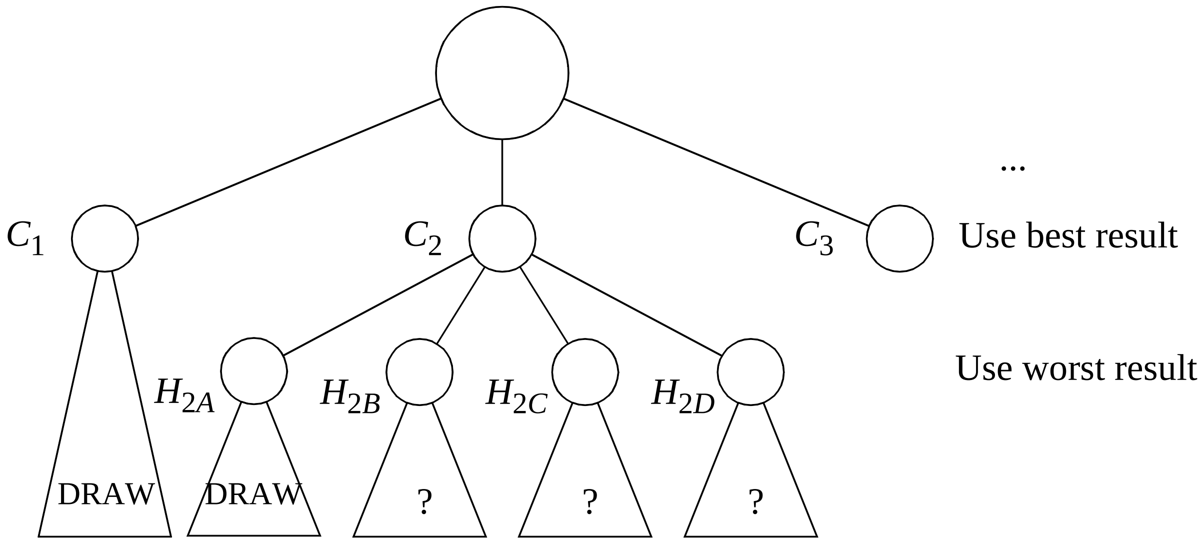
Alternate algorithm for word search puzzle

```
for each row R
    for each column C
        for each direction D
            for each word length L
                check if L chars starting at row R column
                            C in direction D form a word
                if they do not form a prefix,
                    break;   // the innermost loop
```

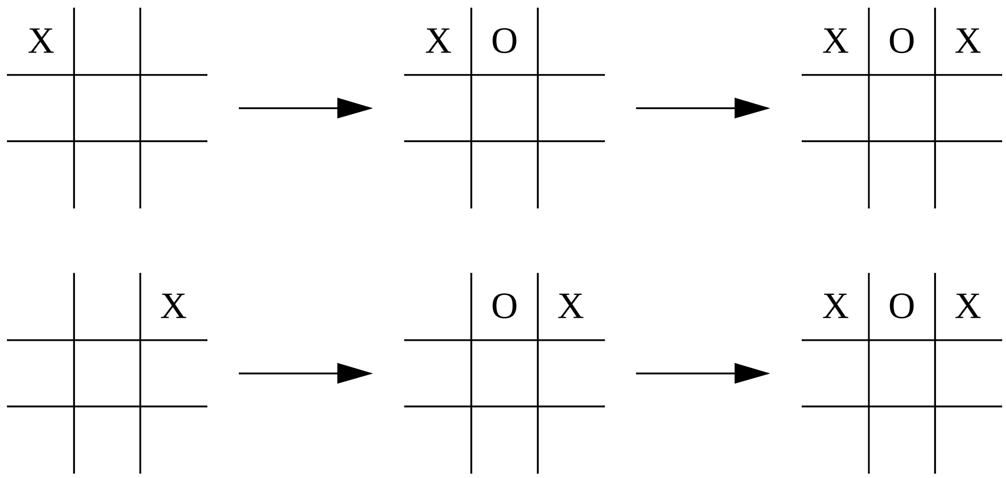
Improved algorithm for word search puzzle; incorporates a prefix test

1. If the position is *terminal* (that is, can immediately be evaluated), return its value.
2. Otherwise, if it is the computer's turn to move, return the maximum value of all positions reachable by making one move. The reachable values are calculated recursively.
3. Otherwise, it is the human's turn to move. Return the minimum value of all positions reachable by making one move. The reachable values are calculated recursively.

Basic minimax algorithm



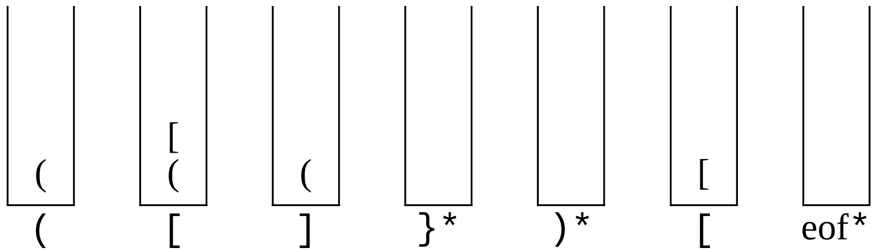
Alpha-beta pruning: After H_{2A} is evaluated, C_2, which is the minimum of the H_2's, is at best a draw. Consequently, it cannot be an improvement over C_1. We therefore do not need to evaluate H_{2B}, H_{2C}, and H_{2D}, and can proceed directly to C_3



Two searches that arrive at identical positions

Chapter 11

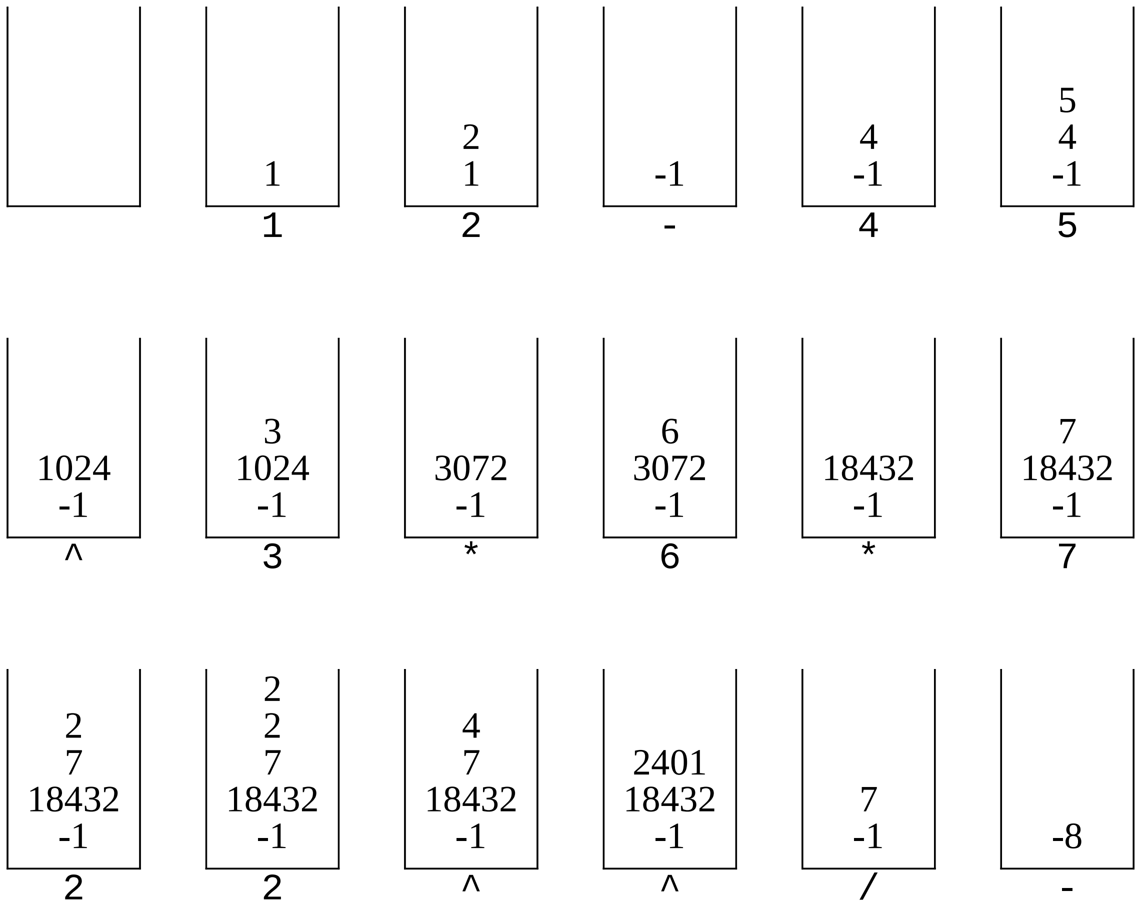
Stacks and Compilers




```
Errors (indicated by *):
    } when expecting )
    ) with no matching opening symbol
    [ unmatched at end of input
```

Stack operations in balanced symbol algorithm

Postfix Expression: `1 2 - 4 5 ^ 3 * 6 * 7 2 2 ^ ^ / -`



Steps in evaluation of a postfix expression

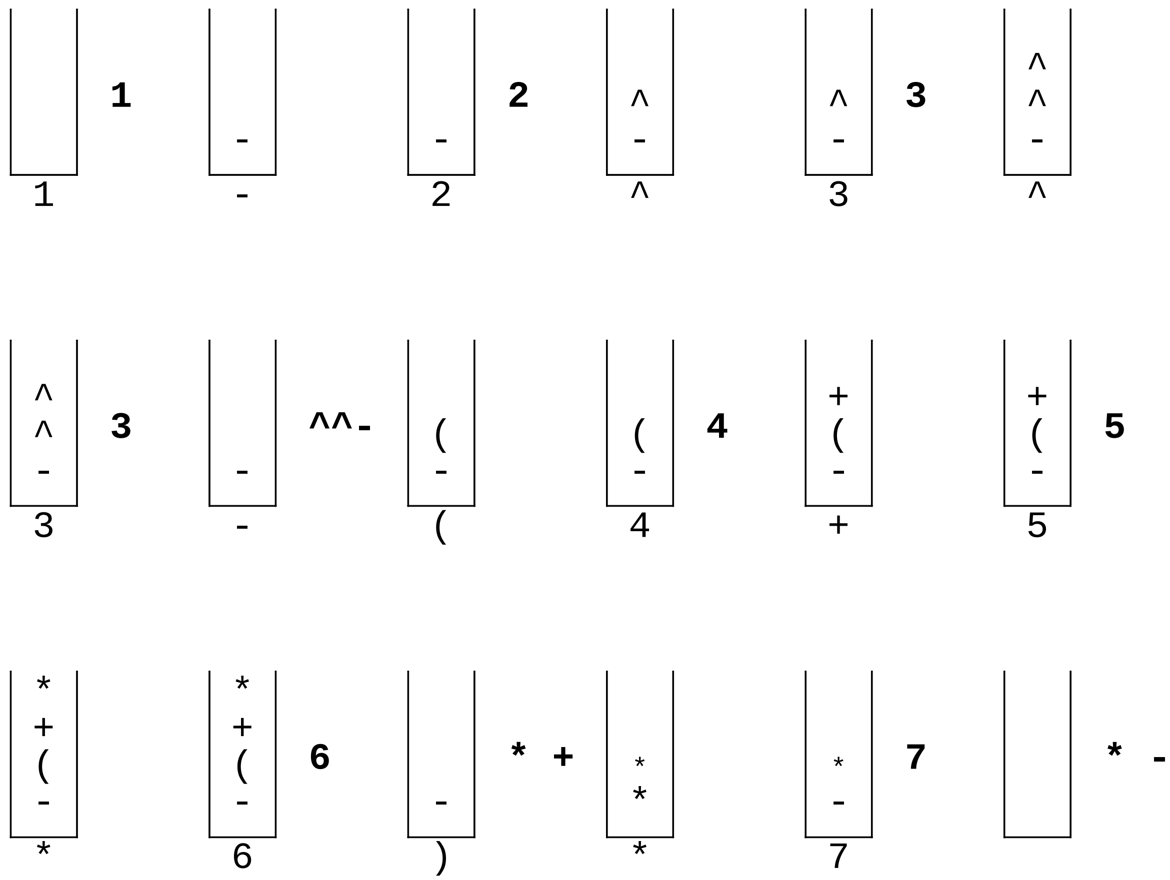
Infix expression	Postfix expression	Associativity
2 + 3 + 4	2 3 + 4 +	Left associative: Input + is lower than stack +
2 ^ 3 ^ 4	2 3 4 ^ ^	Right associative: Input ^ is higher than stack ^

Associativity rules

- *Operands*: Immediately output.
- *Close parenthesis*: Pop stack symbols until an open parenthesis is seen.
- *Operator*: Pop all stack symbols until we see a symbol of lower precedence or a right associative symbol of equal precedence. Then push the operator.
- *End of input*: Pop all remaining stack symbols.

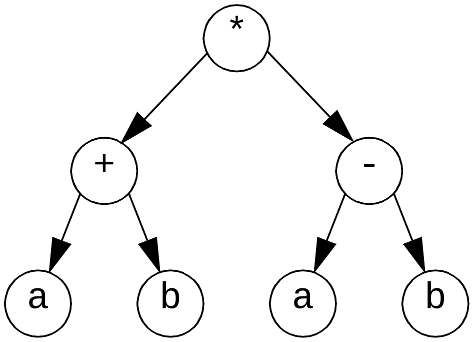
Various cases in operator precedence parsing

Infix: `1 - 2 ^ 3 ^ 3 - ( 4 + 5 * 6 ) * 7`



Postfix: `1 2 3 3 ^ ^ - 4 5 6 * + - 7 * -`

Infix to postfix conversion



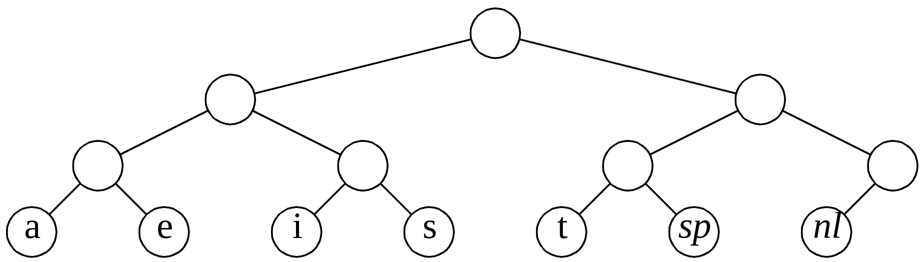
Expression tree for `(a+b)*(c-d)`

Chapter 12

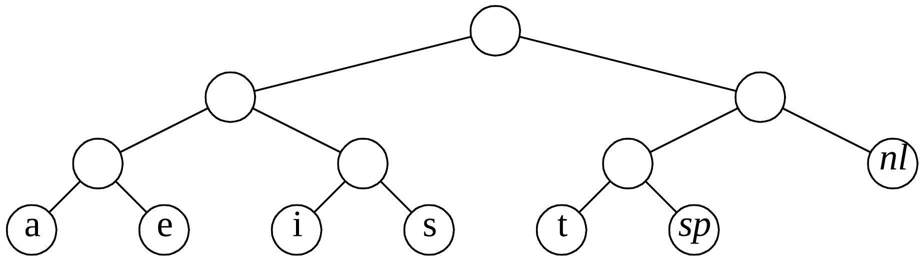
Utilities

Character	Code	Frequency	Total Bits
a	000	10	30
e	001	15	45
i	010	12	36
s	011	3	9
t	100	4	12
sp	101	13	39
nl	110	1	3
Total			**174**

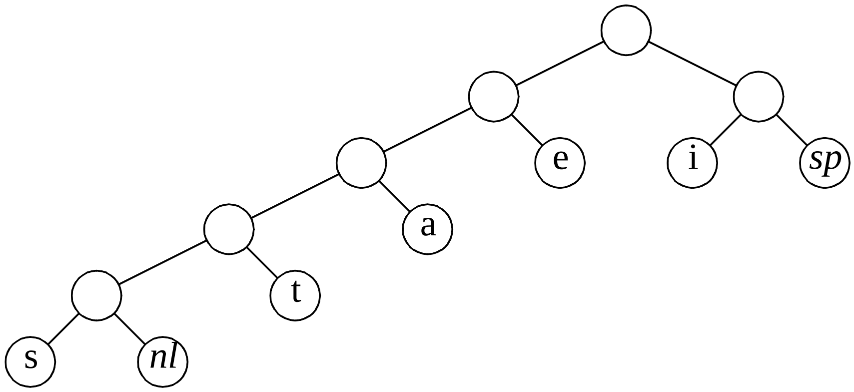
A standard coding scheme



Representation of the original code by a tree



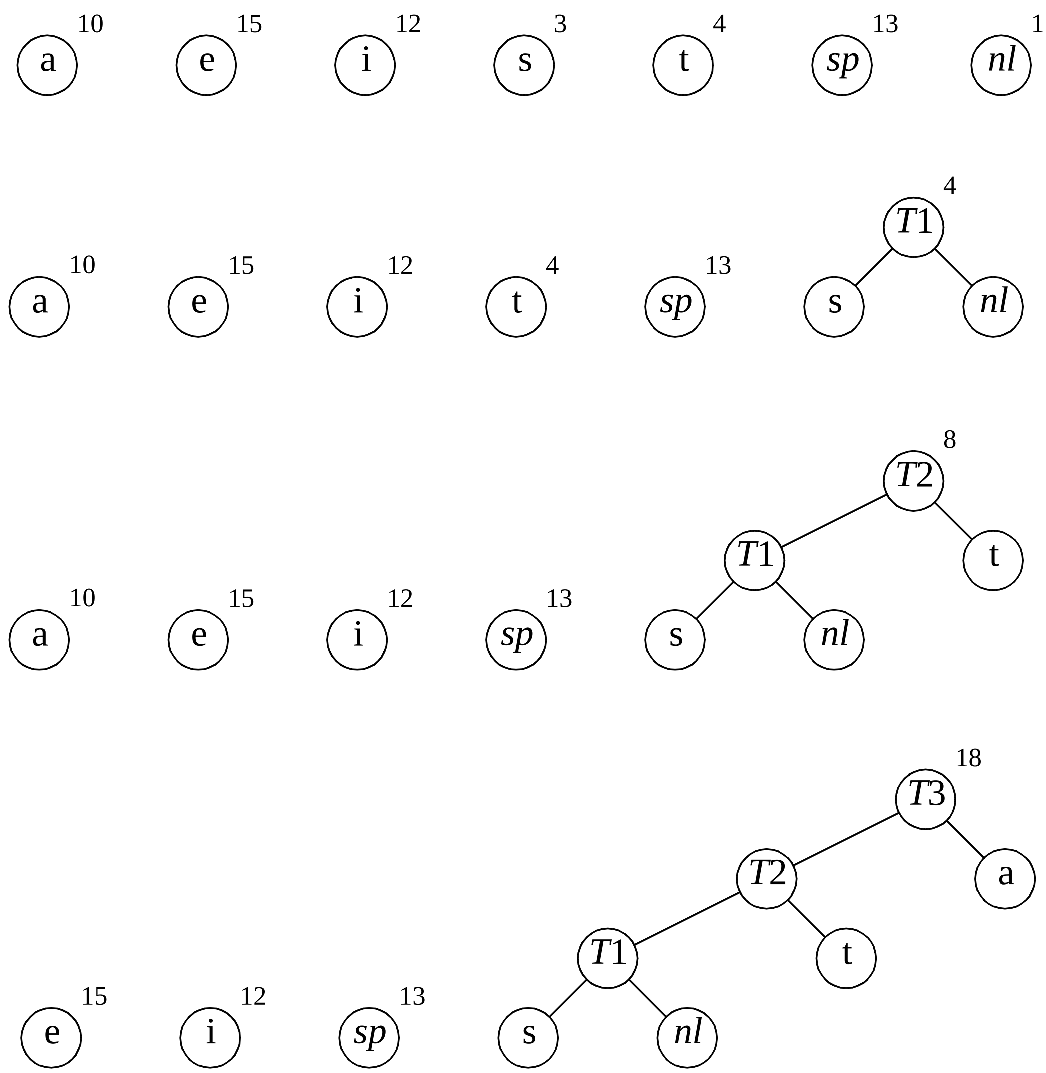
A slightly better tree



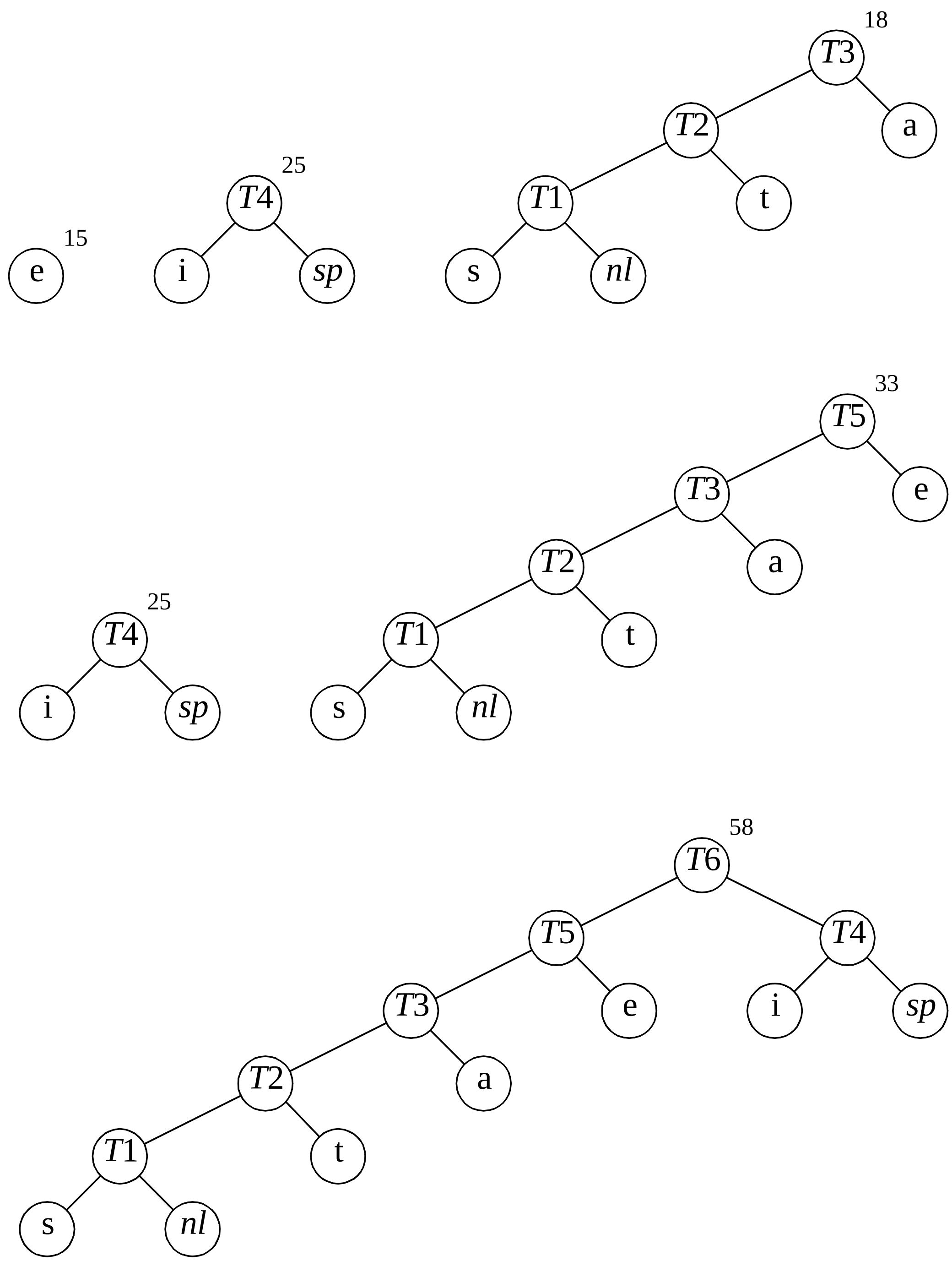
Optimal prefix code tree

Character	Code	Frequency	Total Bits
a	001	10	30
e	01	15	30
i	10	12	24
s	00000	3	15
t	0001	4	16
sp	11	13	26
nl	00001	1	5
Total			**146**

Optimal prefix code



Huffman’s algorithm after each of first three merges



Huffman’s algorithm after each of last three merges

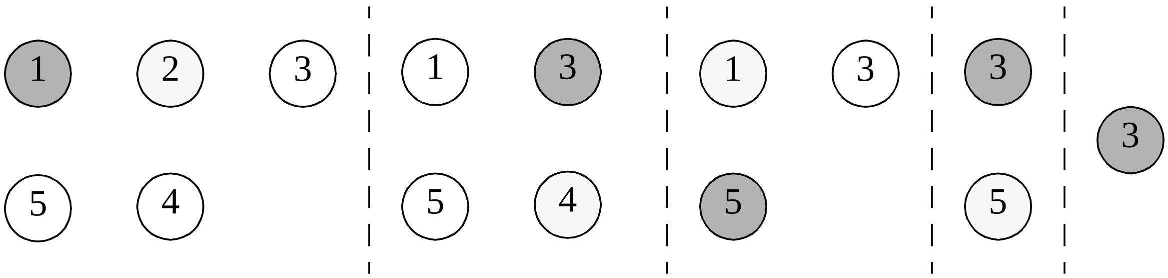
	Character	Weight	Parent	Child Type
0	a	10	9	1
1	e	15	11	1
2	i	12	10	0
3	s	3	7	0
4	t	4	8	1
5	sp	13	10	1
6	nl	1	7	1
7	T1	4	8	0
8	T2	8	9	0
9	T3	18	11	0
10	T4	25	12	1
11	T5	33	12	0
12	T6	58	0	

Encoding table (numbers on left are array indices)

Chapter 13

Simulation

1. At the start, the potato is at player 1; after one pass it is at player 2.
2. Player 2 is eliminated, player 3 picks up the potato, and after one pass it is at player 4.
3. Player 4 is eliminated, player 5 picks up the potato and passes it to player 1.
4. Player 1 is eliminated, player 3 picks up the potato, and passes it to player 5.
5. Player 5 is eliminated, so player 3 wins.



(a) (b) (c) (d) (e)

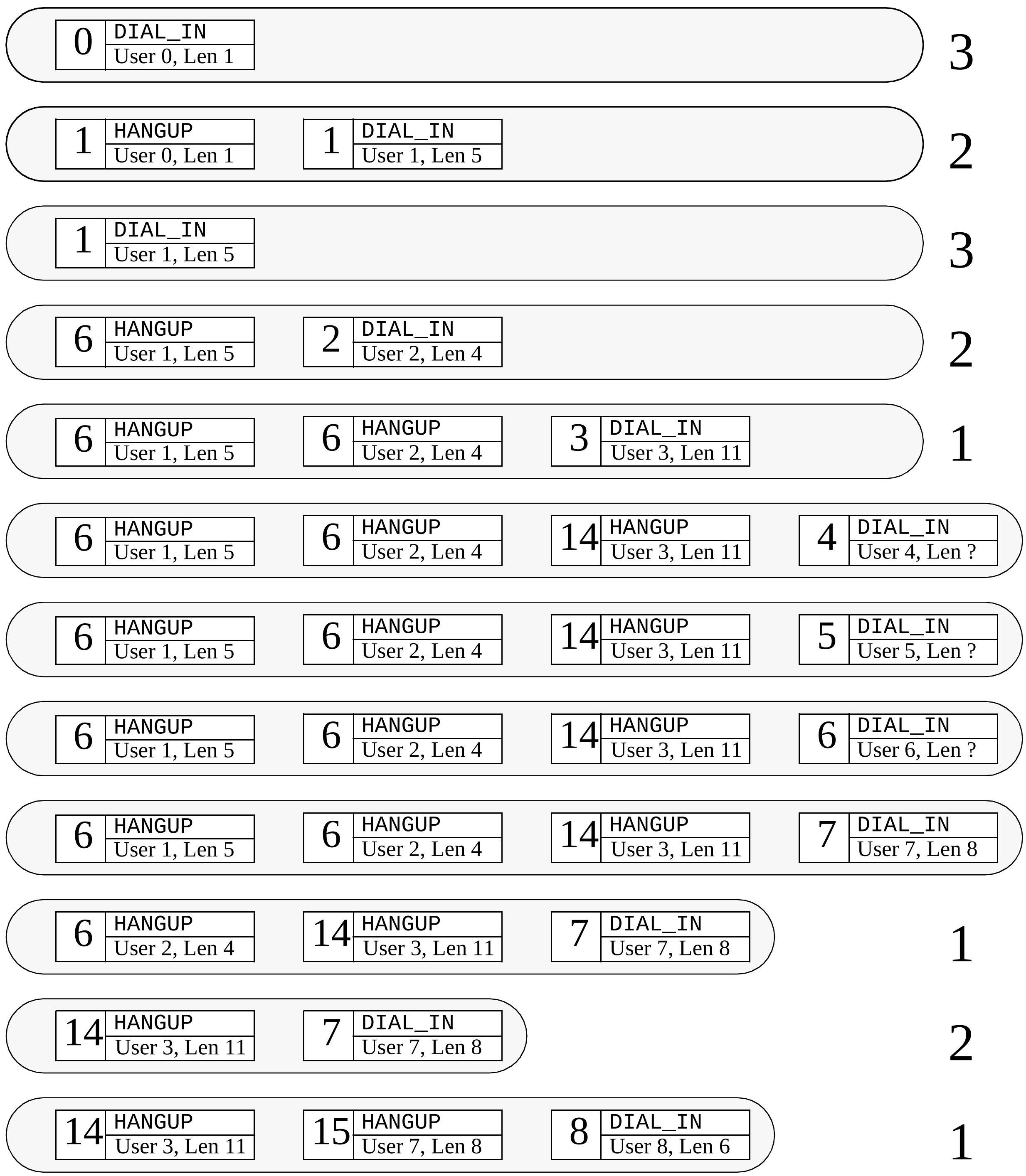
The Josephus problem

```
User 0 dials in at time 0 and connects for 1 minutes
User 0 hangs up at time 1
User 1 dials in at time 1 and connects for 5 minutes
User 2 dials in at time 2 and connects for 4 minutes
User 3 dials in at time 3 and connects for 11 minutes
User 4 dials in at time 4 but gets busy signal
User 5 dials in at time 5 but gets busy signal
User 6 dials in at time 6 but gets busy signal
User 1 hangs up at time 6
User 2 hangs up at time 6
User 7 dials in at time 7 and connects for 8 minutes
User 8 dials in at time 8 and connects for 6 minutes
User 9 dials in at time 9 but gets busy signal
User 10 dials in at time 10 but gets busy signal
User 11 dials in at time 11 but gets busy signal
User 12 dials in at time 12 but gets busy signal
User 13 dials in at time 13 but gets busy signal
User 3 hangs up at time 14
User 14 dials in at time 14 and connects for 6 minutes
User 8 hangs up at time 14
User 15 dials in at time 15 and connects for 3 minutes
User 7 hangs up at time 15
User 16 dials in at time 16 and connects for 5 minutes
User 17 dials in at time 17 but gets busy signal
User 15 hangs up at time 18
User 18 dials in at time 18 and connects for 7 minutes
User 19 dials in at time 19 but gets busy signal
```

Sample output for the modem bank simulation: 3 modems; a dial-in is attempted every minute; average connect time is 5 minutes; simulation is run for 19 minutes

1. The first `DIAL_IN` request is inserted
2. After `DIAL_IN` is removed, the request is connected resulting in a `HANGUP` and a replacement `DIAL_IN` request
3. A `HANGUP` request is processed
4. A `DIAL_IN` request is processed resulting in a connect. Thus both a `HANGUP` and `DIAL_IN` event are added (three times)
5. A `DIAL_IN` request fails; a replacement `DIAL_IN` is generated (three times)
6. A `HANGUP` request is processed (twice)
7. A `DIAL_IN` request succeeds, `HANGUP` and `DIAL_IN` are added.

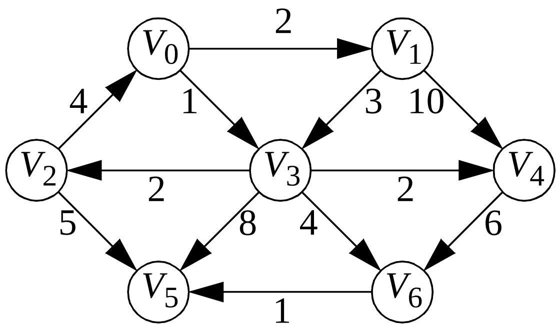
Steps in the simulation



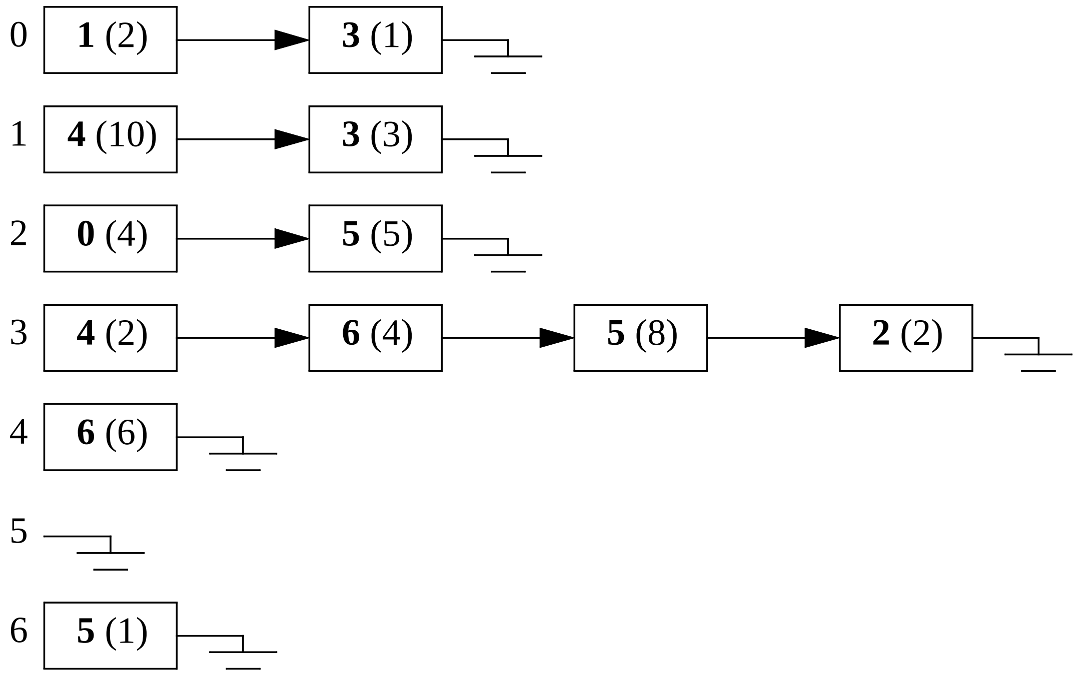
Priority queue for modem bank after each step

Chapter 14

Graphs and Paths



A directed graph



Adjacency list representation of graph in Figure 14.1; nodes in list *i* represent vertices adjacent to *i* and the cost of the connecting edge

- `dist`: The length of the shortest path (either weighted or unweighted, depending on the algorithm) from the starting vertex to this vertex. This value is computed by the shortest path algorithm.
- `prev`: The previous vertex on the shortest path to this vertex.
- `name`: The name corresponding to this vertex. This is established when the vertex is placed into the dictionary and will never change. None of the shortest path algorithms examine this member. It is only used to print a final path.
- `adj`: A list of adjacent vertices. This is established when the graph is read. None of the shortest path algorithms will change the pointer or the linked list.

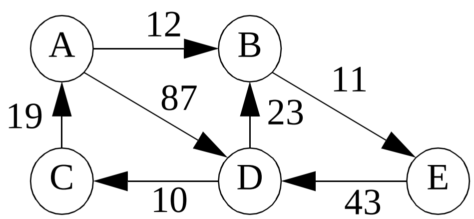
Information maintained by the Graph table


```
D C 10
A B 12
D B 23
A D 87
E D 43
B E 11
C A 19
```

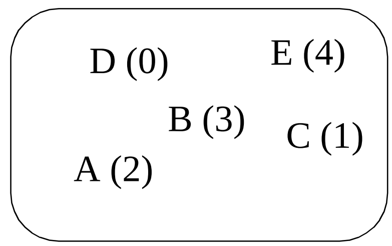
Input

	dist	prev	name	adj
0	66	4	D	**3** (23), **1** (10)
1	76	0	C	**2** (19)
2	0	-1	A	**0** (87), **3** (12)
3	12	2	B	**4** (11)
4	23	3	E	**0** (43)

Graph table

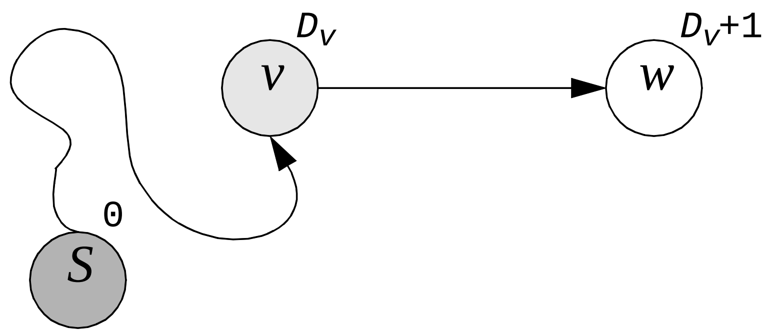


Visual representation of graph

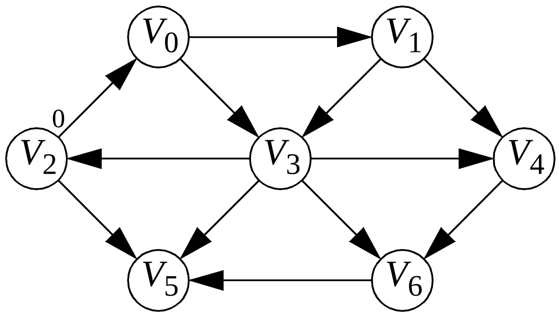


Dictionary

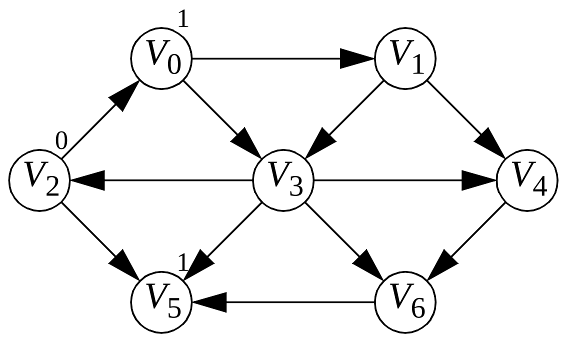
Data structures used in a shortest path calculation, with input graph taken from a file: shortest weighted path from A to C is: A to B to E to D to C (cost 76)



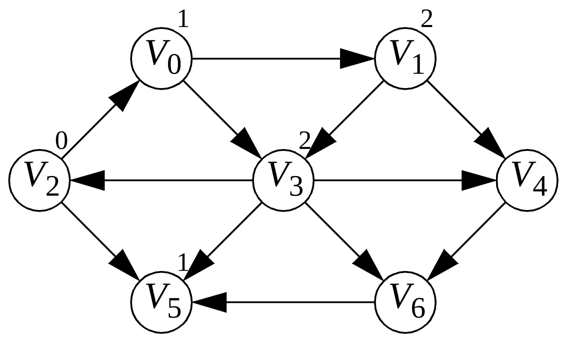
If *w* is adjacent to *v* and there is a path to *v*, then there is a path to *w*



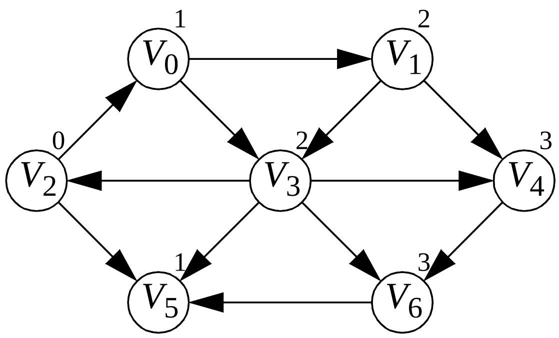
Graph after marking the start node as reachable in zero edges



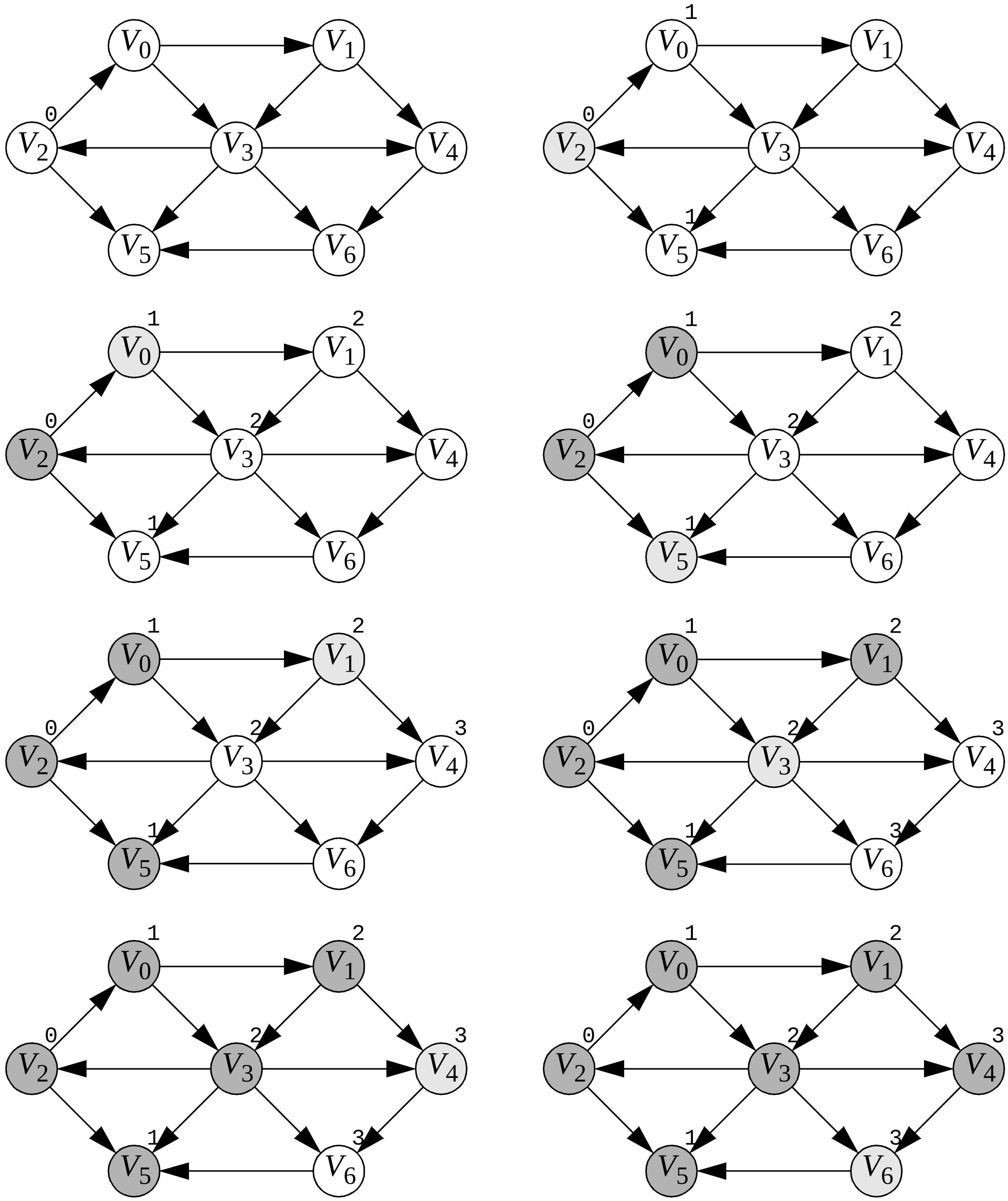
Graph after finding all vertices whose path length from the start is 1



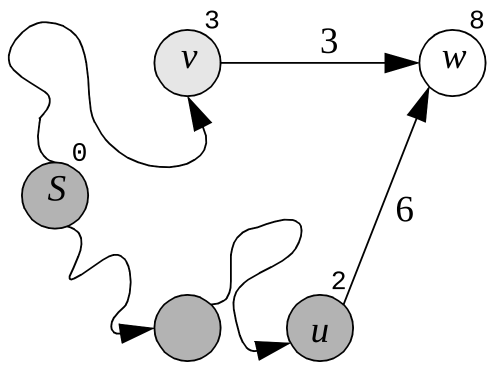
Graph after finding all vertices whose shortest path from the start is 2



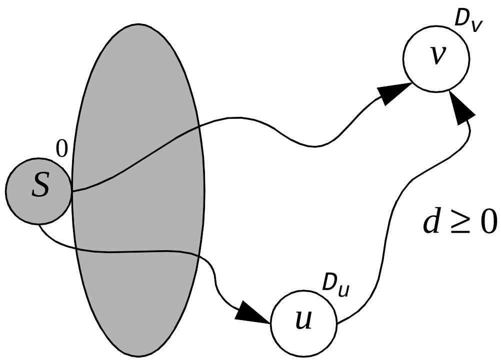
Final shortest paths



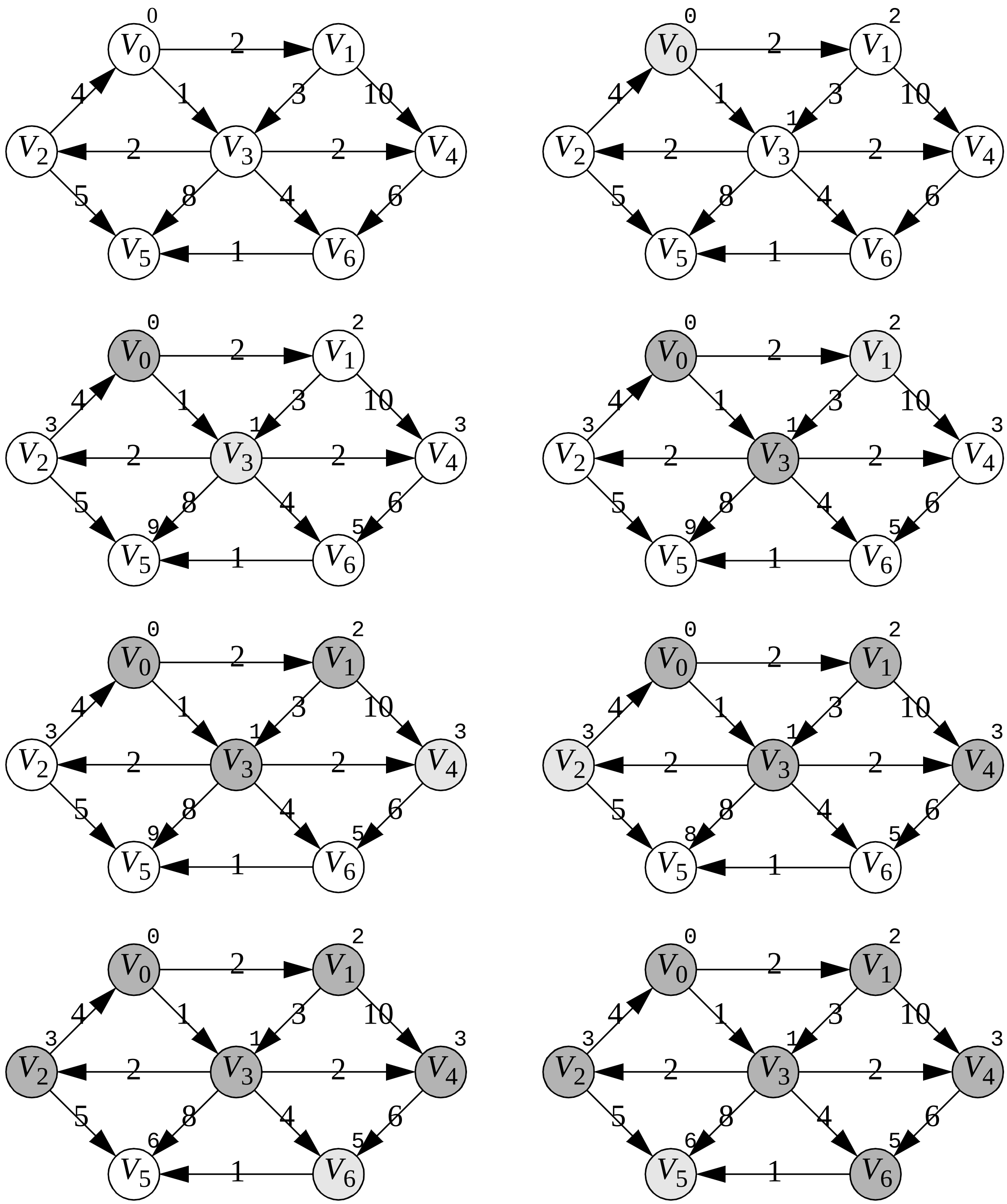
How the graph is searched in unweighted shortest path computation



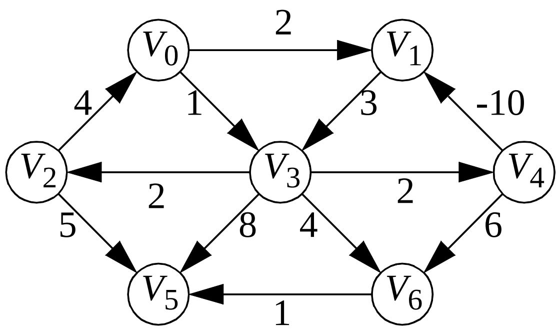
Eyeball is at v; w is adjacent; D_w should be lowered to 6



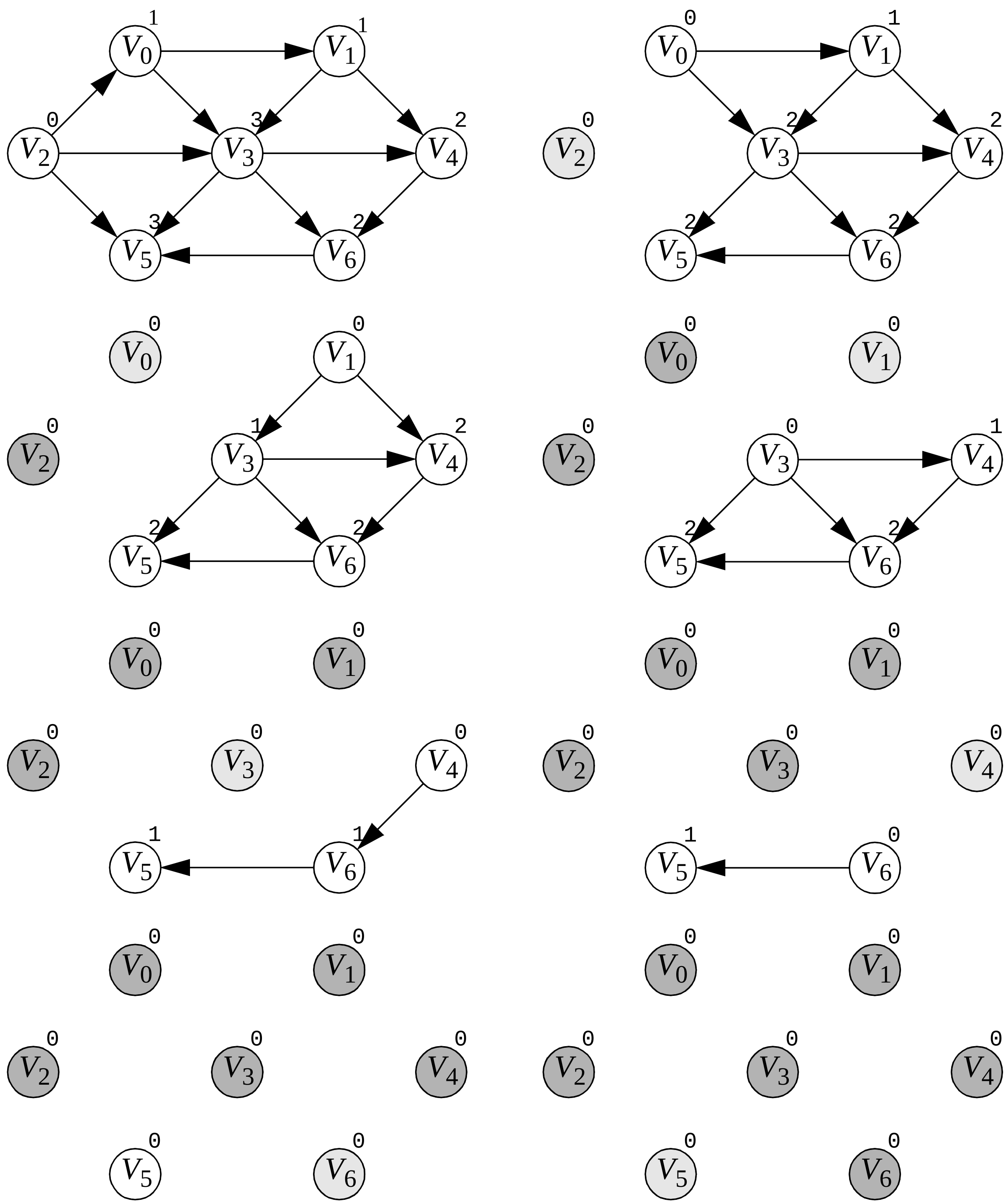
If D_v is minimal among all unseen vertices and all edge costs are nonnegative, then it represents the shortest path



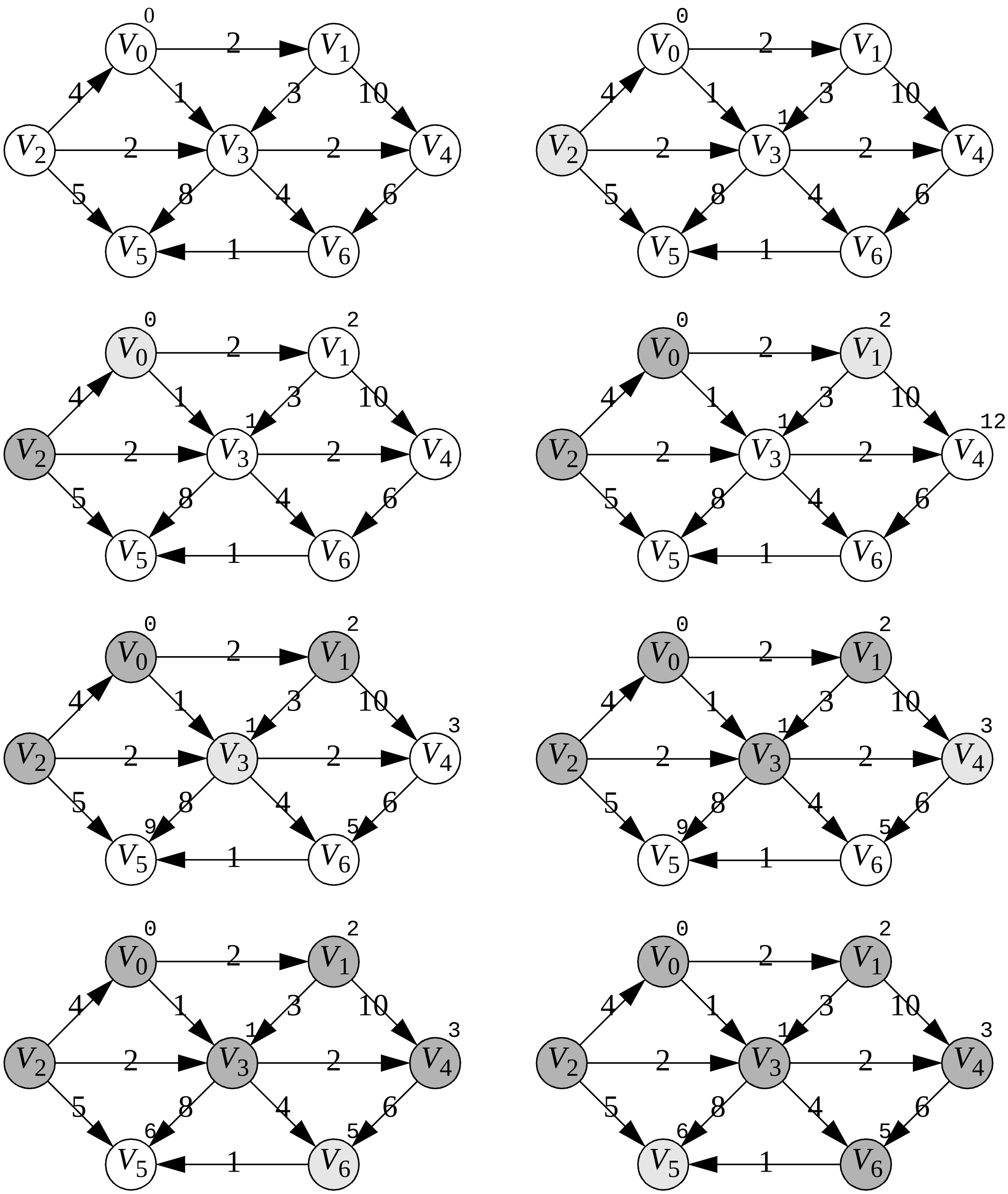
Stages of Dijkstra’s algorithm



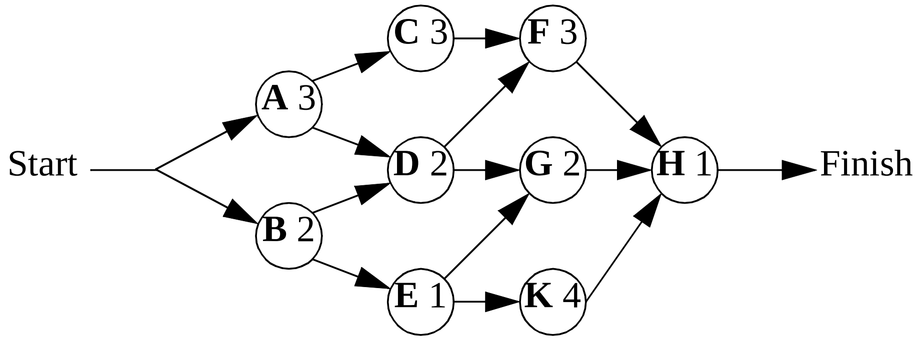
Graph with negative cost cycle



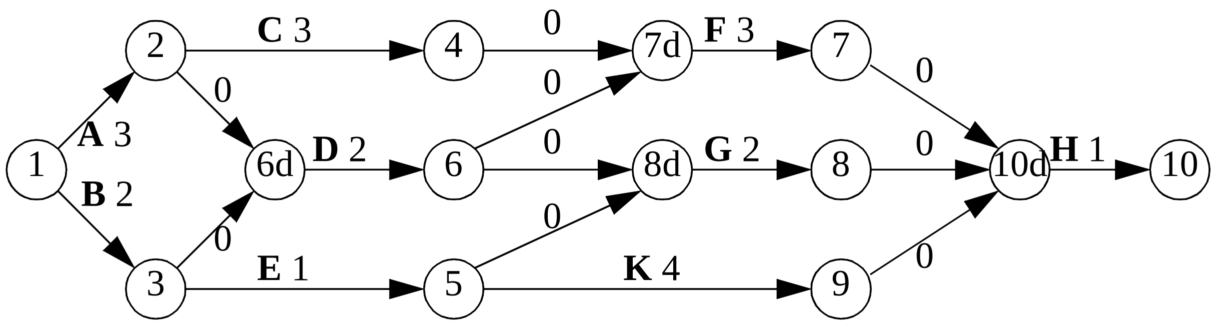
Topological sort

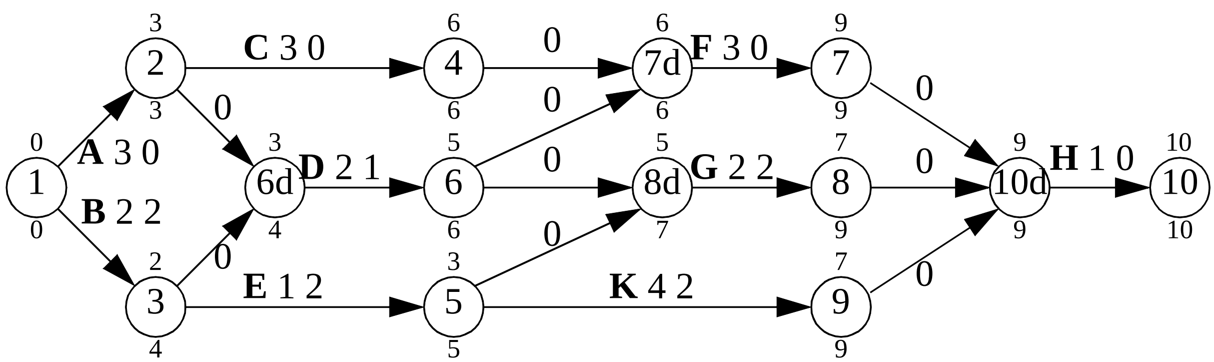


Stages of acyclic graph algorithm



Activity-node graph





Top: Event node grap; Bottom: Earliest completion time, latest completion time, and slack (additional edge item)