CONSTRUCTION AND OPTIMIZATION OF ALGORITHMS

Computational complexity theory

Computational complexity theory focuses on classifying computational problems according to their inherent difficulty, and relating these classes to each other.

Literature

Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein. Introduction to Algorithms.

N. Wirth. Algorithms and Data Structures.

Time complexity

The time complexity of the algorithm in the **worst, best or average** case. In some particular cases, we shall be interested in the average-case running time of an algorithm; we shall see the technique of **probabilistic analysis** applied to various algorithms.

1. The larger of the two natural numbers is 34. What should be the smaller number for the Euclidean algorithm to have as many steps as possible?
2. The larger of the two natural numbers is 55. What should be the smaller number for the Euclidean algorithm to have as many steps as possible?

Asymptotic complexity

$$\exists(C, C' > 0), n_0 : \forall(n > n_0)\ |Cg(n)| \leq |f(n)| \leq |C'g(n)|$$

$$\forall(C > 0), \exists n_0 : \forall(n > n_0)\ |f(n)| < |Cg(n)|$$

$$\forall(C > 0), \exists n_0 : \forall(n > n_0)\ |Cg(n)| < |f(n)|$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

Examples - ?

Logarithmic, linear, polynomial, exponential complexity.

Other measures of complexity

Other measures of complexity are also used, such as

the amount of communication (used in communication complexity),

the number of gates in a circuit (used in circuit complexity) and

the number of processors (used in parallel computing).

One of the roles of computational complexity theory is to determine the practical limits on what computers can and cannot do.

Other measures of complexity

If the created program is used only a few times, then **the cost of writing and debugging the program** will dominate the total cost of the program, that is, the actual execution time will not have a significant impact on the total cost. In this case, you should prefer the algorithm that is the easiest to realize.

Other measures of complexity

If the program will only work with **“small” input data**, the degree of growth in the execution time will be less important than the constant present in the asymptotic runtime formula. At the same time, the notion of “smallness” of the input data depends on the exact execution time of competing algorithms. There are algorithms, such as the algorithm of integer multiplication, which are asymptotically the most efficient, but which are never used in practice even for large tasks, since their proportionality constants far exceed those of other, simpler and less “efficient” algorithms.

Other measures of complexity

Sometimes there are **incorrect algorithms** that either get looped or sometimes give the wrong result. But they still apply, because in most cases they lead to the desired result. For example, Kramer’s rule or resolution method.

Calculate the complexity of the algorithm

Bubble Sort Idea

- Move smallest element in range 1,…,*n* to position 1 by a series of swaps
- Move smallest element in range 2,…,*n* to position 2 by a series of swaps
- Move smallest element in range 3,…,*n* to position 3 by a series of swaps
  - *etc.*

Calculate the complexity of the algorithm

Selection Sort Idea

Rearranged version of Bubble Sort:

- Are first 2 elements sorted? If not, swap.
- Are the first 3 elements sorted? If not, move the 3rd element to the left by series of swaps.
- Are the first 4 elements sorted? If not, move the 4th element to the left by series of swaps.
  - *etc.*

Why Selection (or Bubble) Sort is Slow

- *Inversion*: a pair (i,j) such that i<j but Array[i] > Array[j]
- Array of size N can have $\theta(N^2)$ inversions
- Selection/Bubble Sort only swaps adjacent elements
  - *Only removes 1 inversion at a time!*
- Worst case running time is $\theta(N^2)$

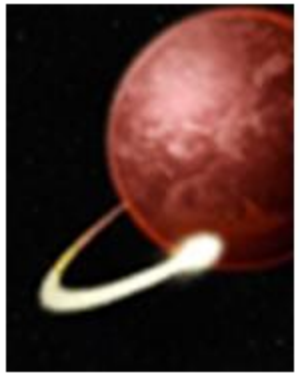
Merge Sort Running Time

Any difference best / worse case?

$T(1) = b$

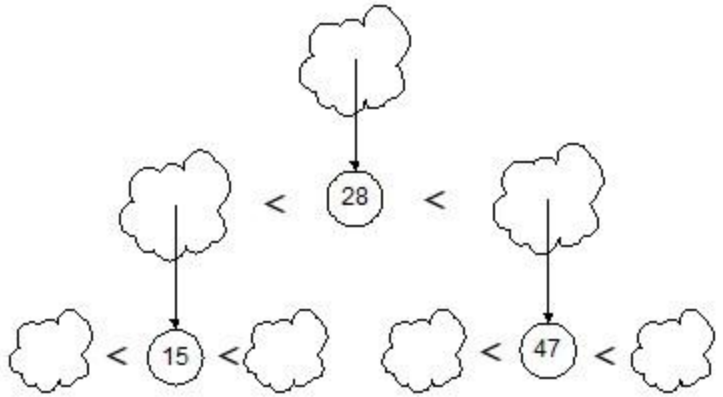
$T(n) = 2T(n/2) + cn \quad$ for $n>1$

$T(n) = 2T(n/2)+cn$	
$T(n) = 4T(n/4) +cn +cn$	substitute
$T(n) = 8T(n/8)+cn+cn+cn$	substitute
$T(n) = 2^kT(n/2^k)+kcn$	inductive leap
$T(n) = nT(1) + cn \log n$ where $k = \log n$	select value for k
$T(n) = \theta(n \log n)$	simplify



Picture from PhotoDisc.com

QuickSort



1. Pick a “pivot”.
2. Divide list into two lists:
   - One less-than-or-equal-to pivot value
   - One greater than pivot
3. Sort each sub-problem recursively
4. Answer is the concatenation of the two solutions

Problems

1. Suppose weare comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n\lg n$ steps. For which values of n does insertion sort beat merge sort?

2. What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

Problems

3. For each function f(n) and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes f(n) microseconds.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Algorithms on graphs

First examples

1. Connect six points with non-intersecting segments so that 3 points leave each point.

2. Connect 14 points with non-intersecting segments so that 4 points leave each point.

3. Draw a closed six-pattern broken line that intersects each of its links once.

4. The volleyball net has a size of 10 by 50 cells. What is the greatest number of ropes can be cut so that the grid does not fall apart into pieces?

Prim's Algorithm

- Prim's algorithm finds a minimum cost spanning tree by selecting edges from the graph one-by-one as follows:
- It starts with a tree, T, consisting of a single starting vertex, x.
- Then, it finds the shortest edge emanating from x that connects T to the rest of the graph (i.e., a vertex not in the tree T).
- It adds this edge and the new vertex to the tree T.
- It then picks the shortest edge emanating from the revised tree T that also connects T to the rest of the graph and repeats the process.

Prim's Algorithm Abstract

Consider a graph G=(V, E);

Let T be a tree consisting of only the starting vertex **x;**

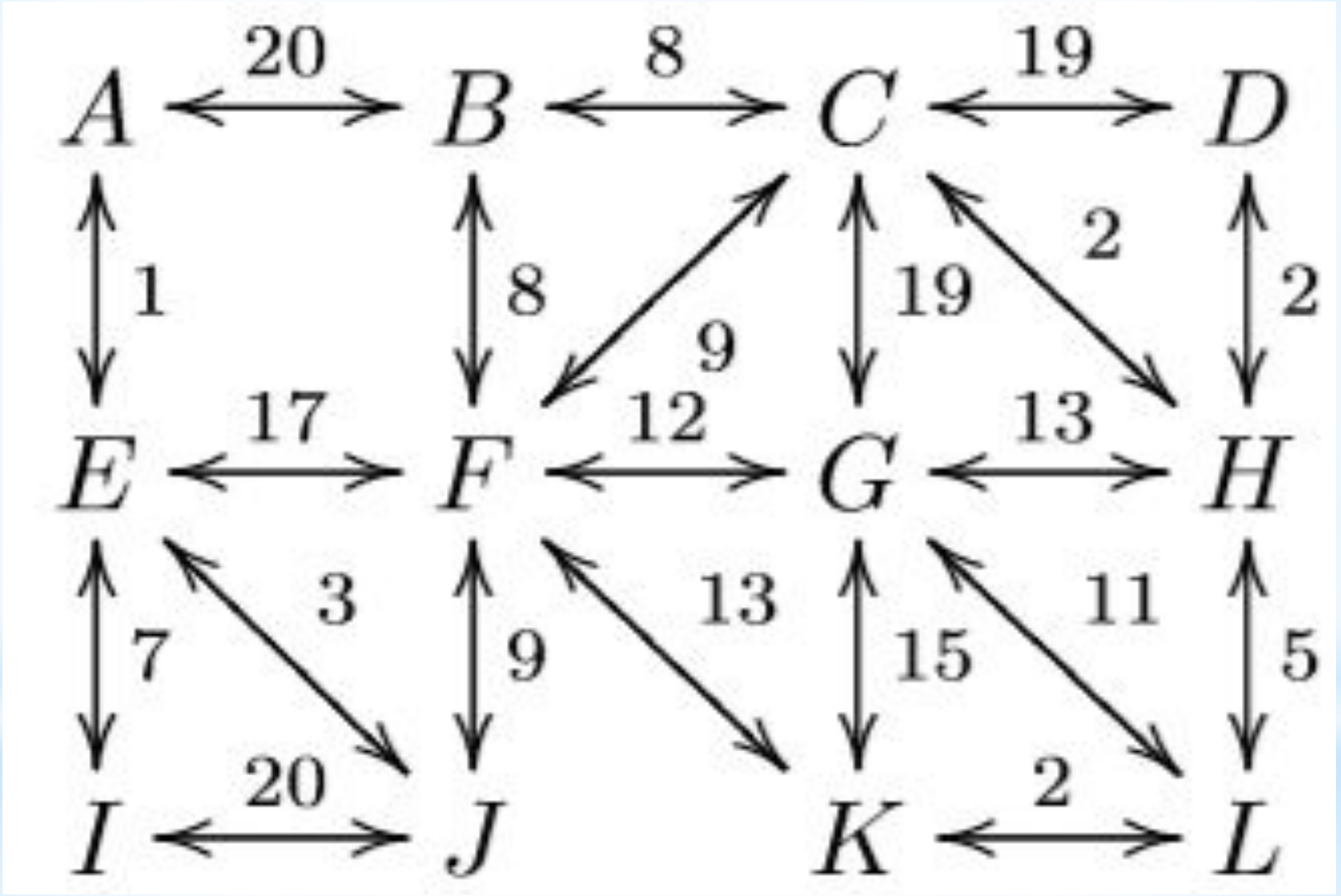
while (T has fewer than I V I vertices)

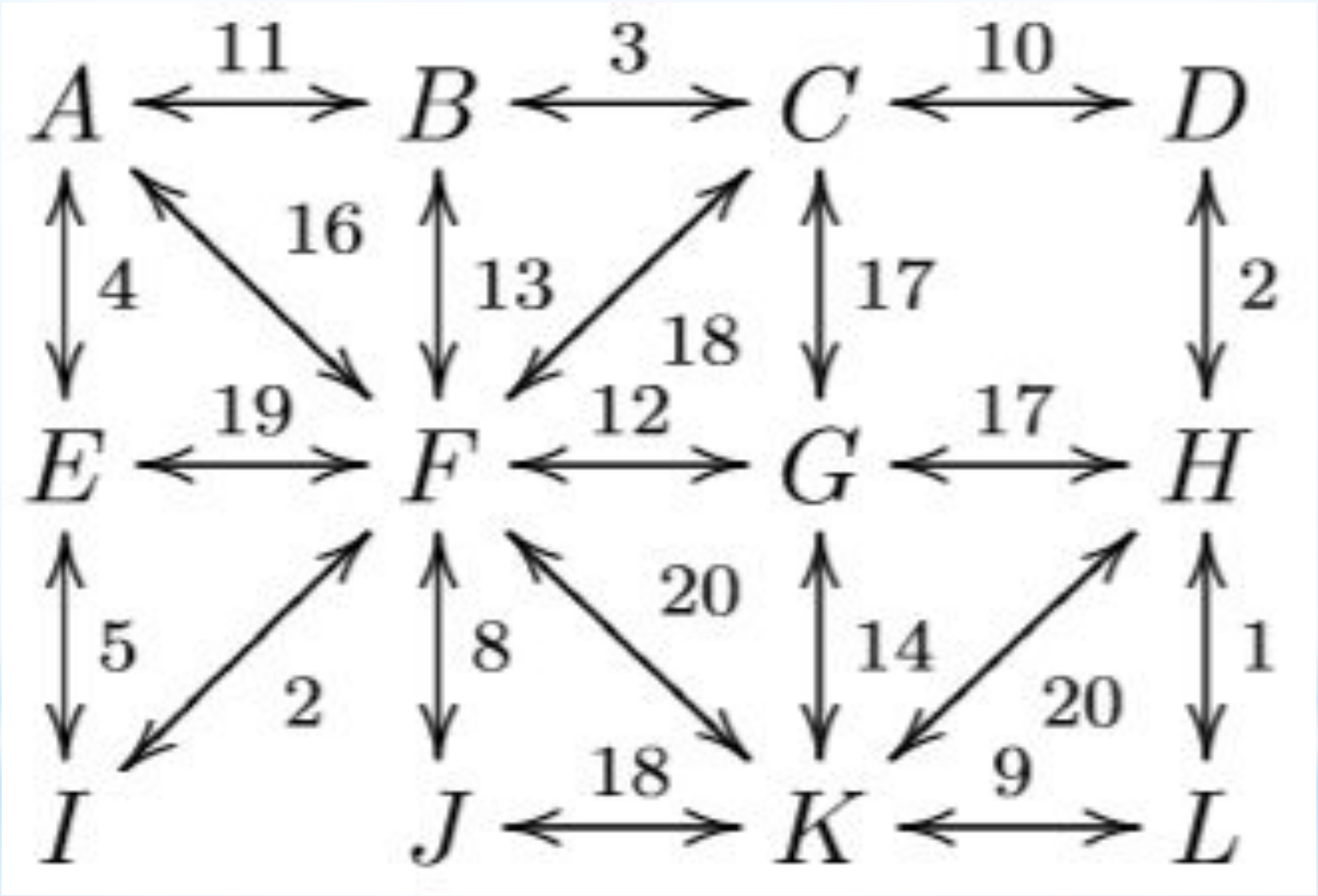
{

find a smallest edge connecting T to G-T;

add it to T;

}


A
B
C
D
E
F
G
H
I
J
K
L
20
8
19
1
8
19
2
2
9
17
12
13
7
3
9
13
15
11
5
20
2


A
B
C
D
E
F
G
H
I
J
K
L
11
3
10
16
4
13
17
2
18
19
12
17
20
5
8
14
1
2
20
18
9

Kruskal's Algorithm

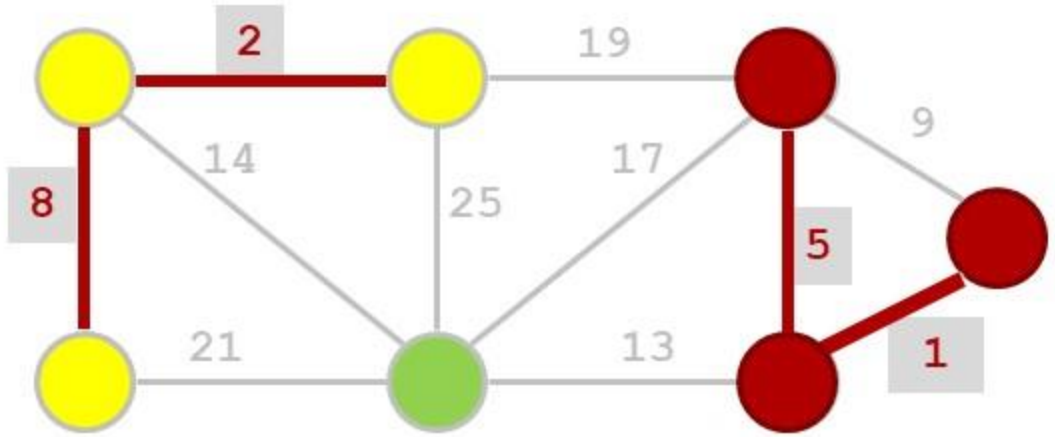
- Greedy algorithm to choose the edges as follows.

Step 1	**First edge: choose any edge with the minimum weight.**
Step 2	**Next edge: choose any edge with minimum weight from *those not yet selected*. (The subgraph can look disconnected at this stage.)**
Step 3	**Continue to choose edges of minimum weight from those not yet selected, except *do not select any edge that creates a cycle* in the subgraph.**
Step 4	**Repeat step 3 until the subgraph connects all vertices of the original graph.**

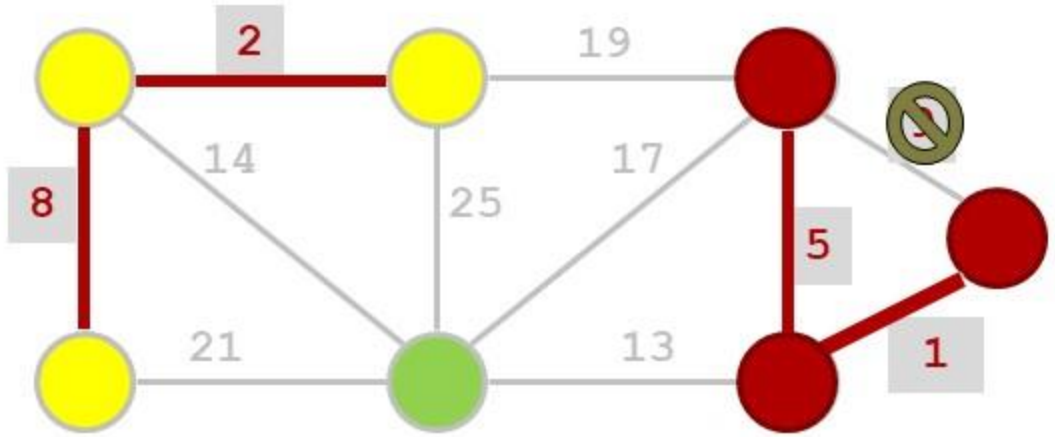
Kruskal's Algorithm

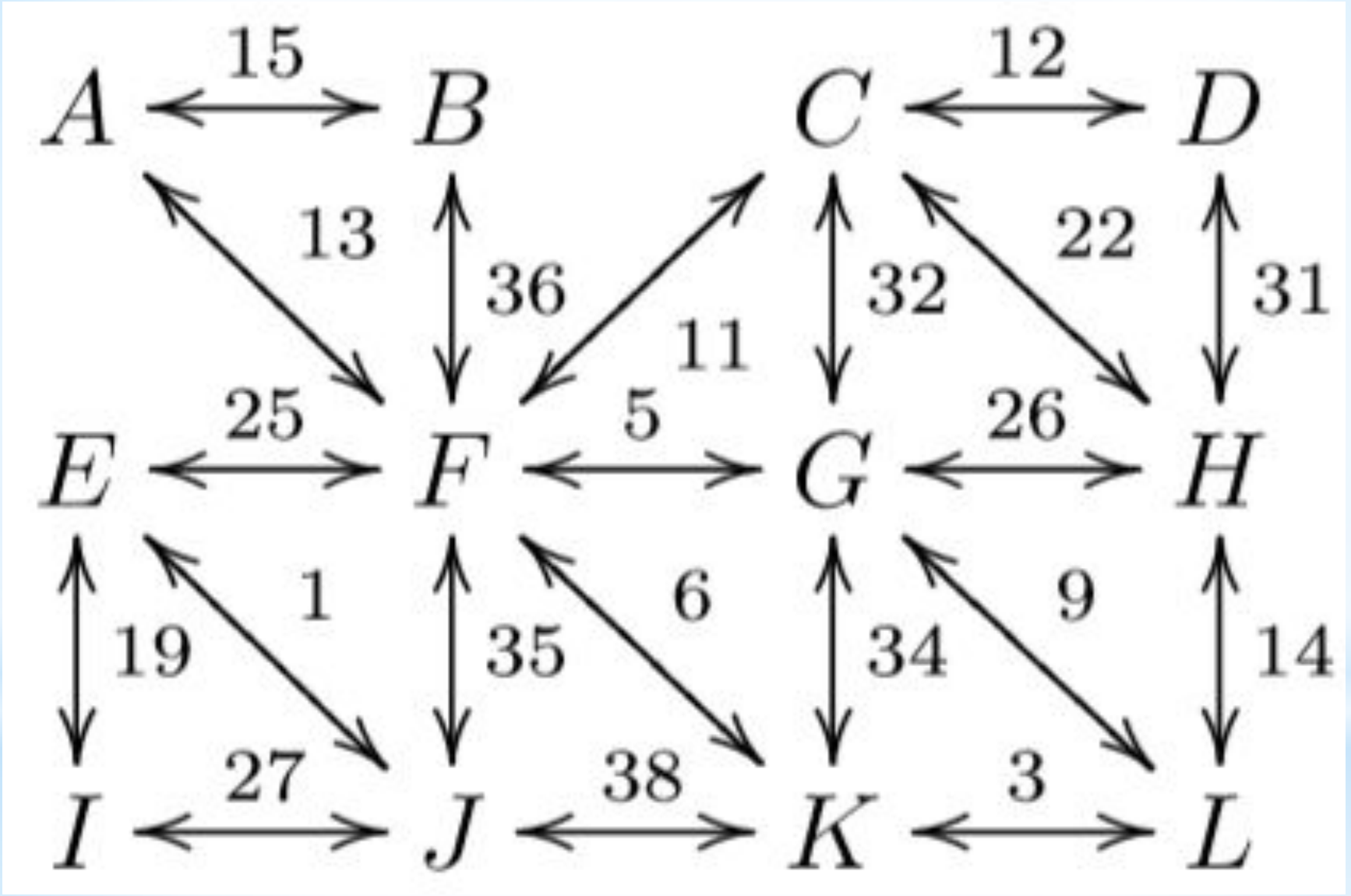

```
Build a priority queue (min-based) with all of the edges of G.
T = ϕ ;
while(queue is not empty){
   get minimum edge e from priorityQueue;
   if(e does not create a cycle with edges in T)
      add e to T;
}
return T;
```

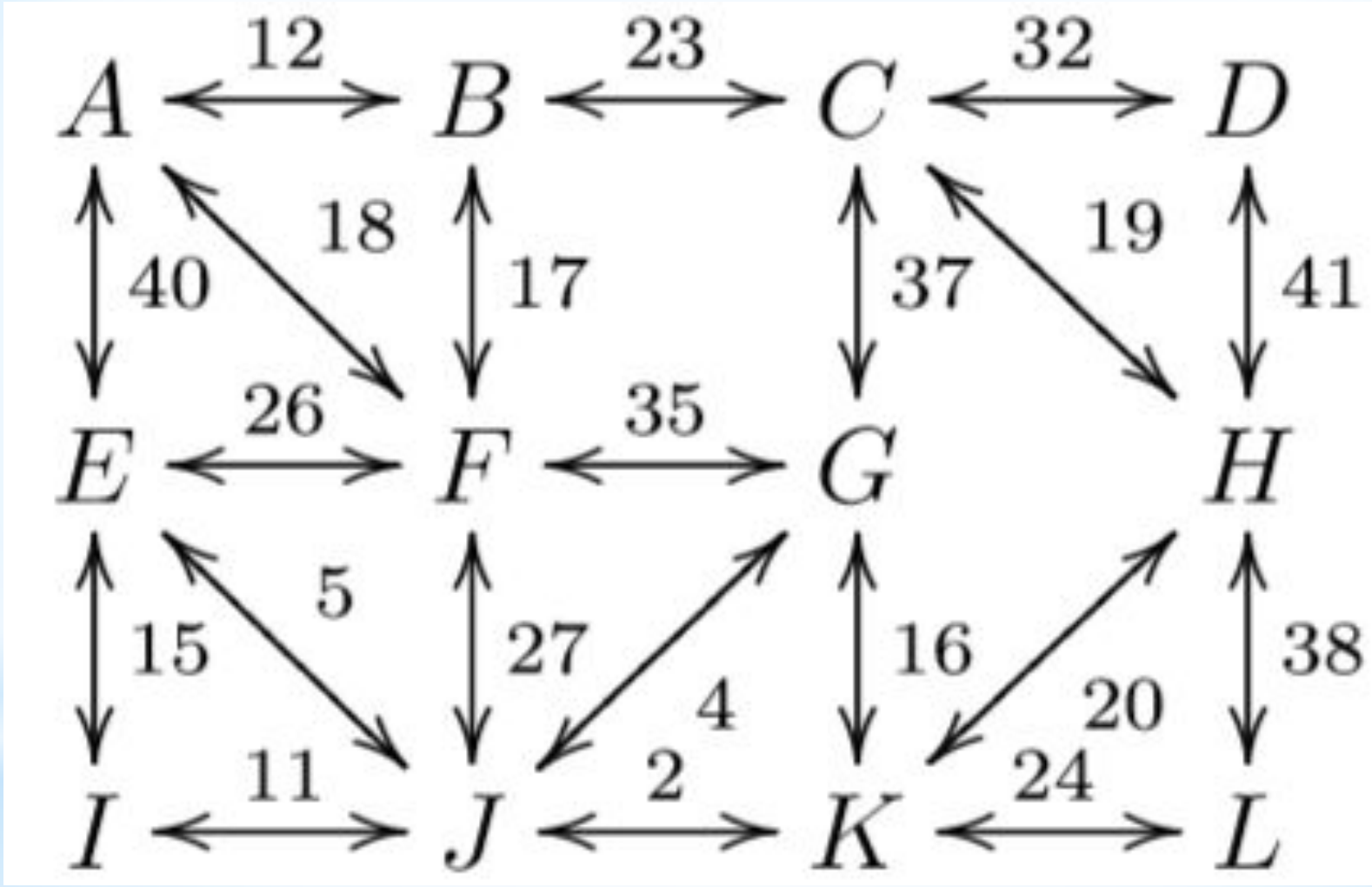
Kruskal's Algorithm



Kruskal's Algorithm




A
15
B
C
12
D
13
36
11
32
22
31
25
5
26
E
F
G
H
1
6
9
19
35
34
14
27
38
3
I
J
K
L


A
12
B
23
C
32
D
40
18
17
37
19
41
E
26
F
35
G
H
15
5
27
16
4
38
20
I
11
J
2
K
24
L

Dijkstra's Algorithm

- Finds the shortest path to all nodes from the start node
- Performs a modified BFS that accounts for cost of vertices
  - The *cost of a vertex* (to reach a start vertex) is weight of the shortest path from the start node to the vertex ***using only the vertices which are already visited*** (except for the last one)
  - Selects the node with the least cost from unvisited vertices
  - In an unweighted graph (weight of each edge is 1) this reduces to a BFS
- To be able to quickly find an unvisited vertex with the smallest cost, we will use a priority queue (highest priority = smallest cost)
  - When we visit a vertex and remove it from the queue, we might need to update the cost of some vertices (neighbors of the vertex) in queue (decrease it)
- The shortest path to any node can be found by backtracking in the `results` array (the `results` array contains, for each node, the minumum cost and a "parent" node from which one can get to this node achieving the minimum cost)

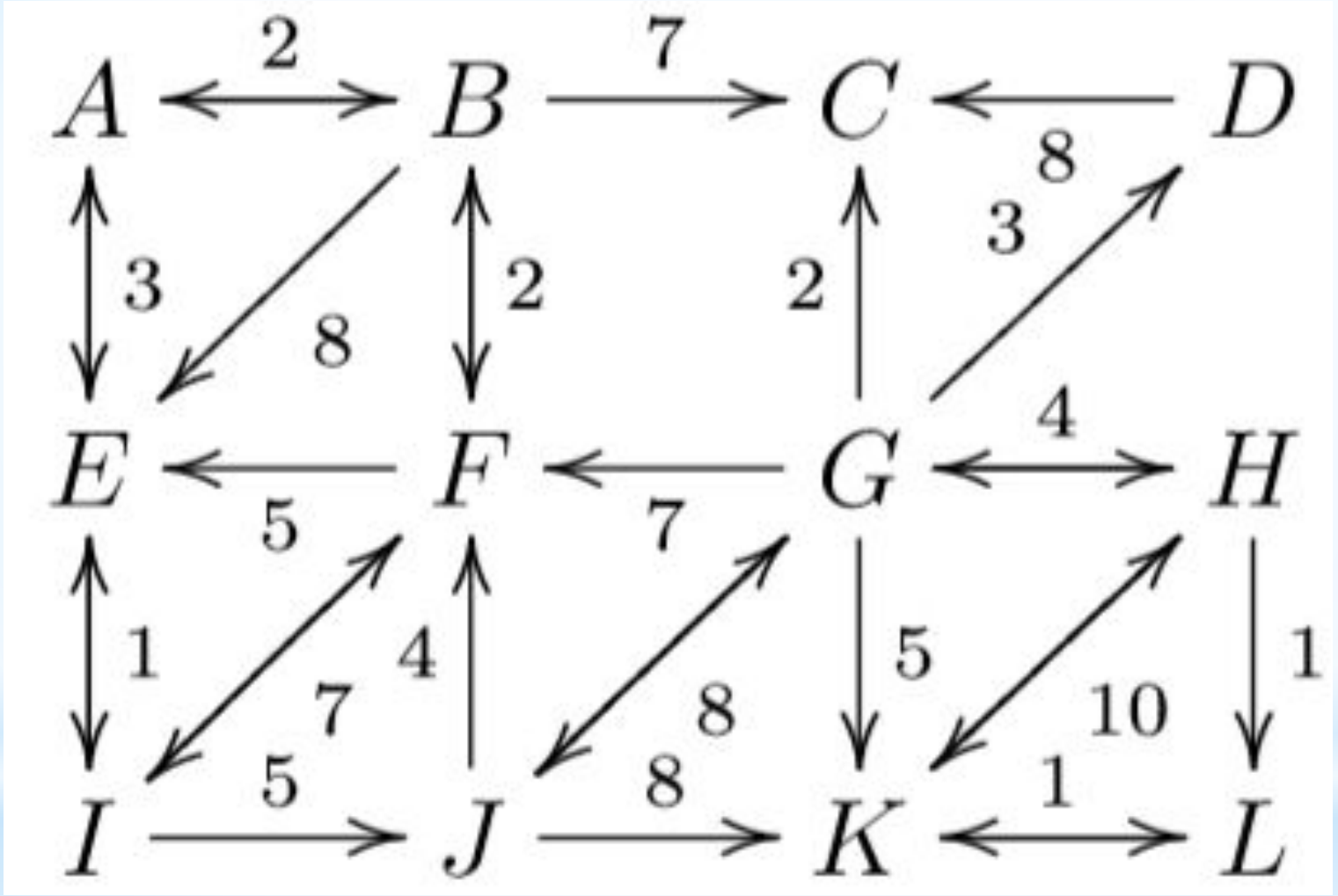
Dijkstra's Algorithm – Initialization

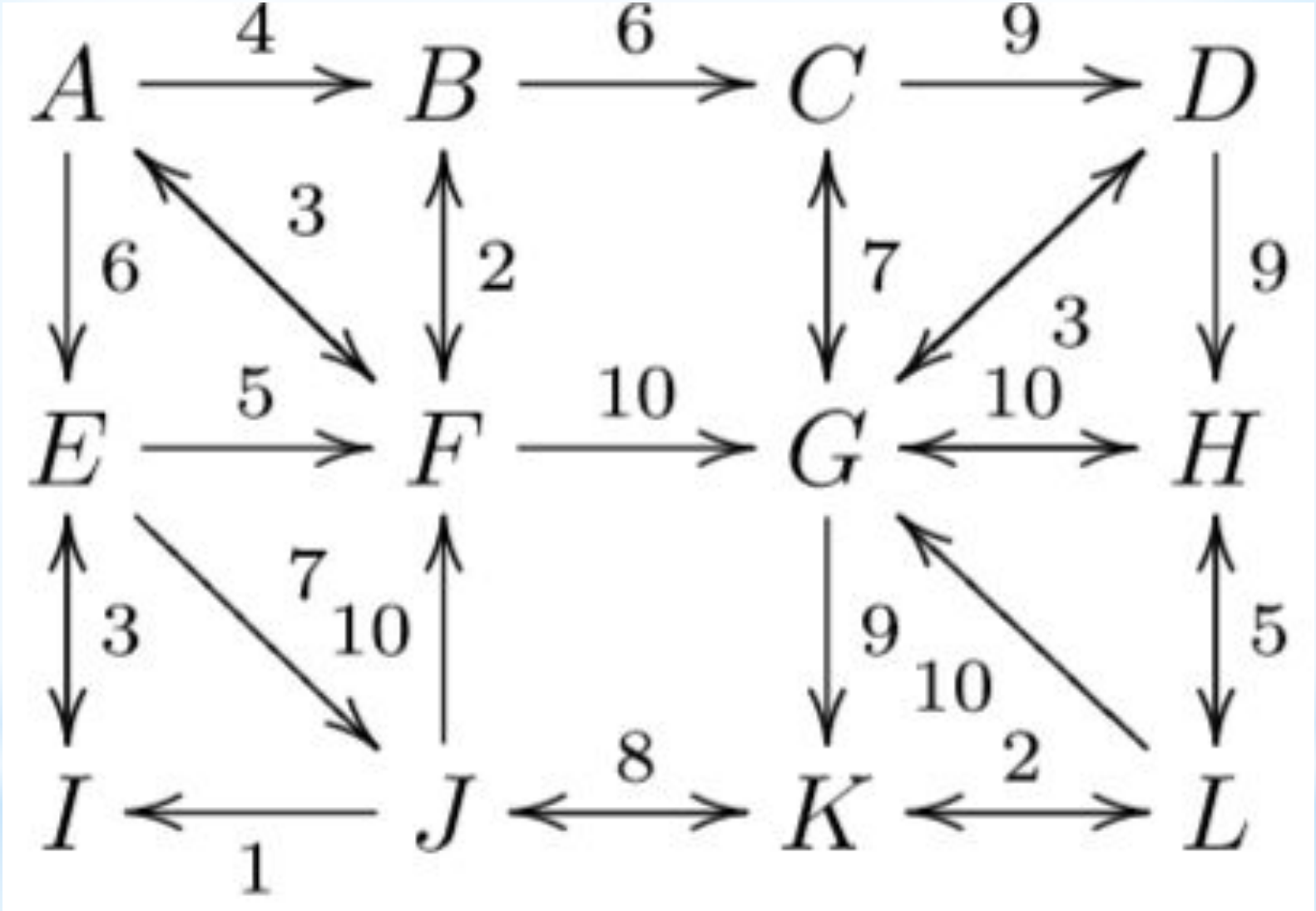
- Initialization – insert all vertices in a priority queue (*PQ*)
  - Set the cost of the start vertex to zero
  - Set the costs of all other vertices to infinity and their parent vertices to the start node
  - Note that because the cost to reach the start vertex is zero it will be at the head of the *PQ*

- Special requirement on priority queue *PQ*:
  - we can use min-heap
  - a cost of an item (vertex) can decrease, and in such case we need to bubble-up the item (in time *O(log n)*)
  - another complication is that we need to locate the item in the queue which cost has changed, but we know its value (vertex number) but not its location in the queue – therefore, we need to keep reversed index array mapping vertices to positions in the heap

Dijkstra's Algorithm – Main Loop

- Until *PQ* is empty
  - Remove the vertex with the least cost and insert it in a `results` array, make that the current vertex (*cv*) [it can be proved that the cost of this vertex is optimal]
  - Search the adjacency list of *cv* for neighbors which are still in *PQ*
  - For each such vertex, *v*, perform the following comparison
    - If **cost[*cv*] + weight(*cv*,*v*) < cost[*v*]** change *v*'s cost recorded in the *PQ* to **cost[*cv*] + weight(*cv*,*v*)** and change *v*'s parent vertex to *cv*
  - Repeat with the next vertex at the head of *PQ*


A
2
B
7
C
D
8
3
3
8
2
2
E
4
F
G
H
5
7
1
4
7
5
1
8
10
8
5
1
I
J
K
L


A
4
B
6
C
9
D
6
3
2
7
3
9
E
5
F
10
G
10
H
3
7
10
9
10
5
I
1
J
8
K
2
L