

as fluid, megabytes of network traffic, commodities, currency, and so on. Each edge of the network has a given *capacity*, that limits the amount of stuff it is able to carry. The idea is to find out how much flow we can push from a designated source node to a designated sink node.

Although the network flow problem is defined in terms of the metaphor of pushing fluids, this problem and its many variations find remarkably diverse applications. These are often studied in the area of operations research. The network flow problem is also of interest because it is a restricted version of a more general optimization problem, called *linear programming*. A good understanding of network flows is helpful in obtaining a deeper understanding of linear programming.

Flow Networks: A *flow network* is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonnegative *capacity* $c(u, v) \geq 0$. (In our book, the capacity of edge e is denoted by c_e .) If $(u, v) \notin E$ we model this by setting $c(u, v) = 0$. There are two special vertices: a *source* s , and a *sink* t (see Fig. 59).

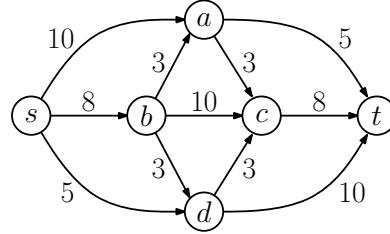


Fig. 59: A flow network.

We assume that there is no edge entering s and no edge leaving t . Such a network is sometimes called an *s-t network*. We also assume that every vertex lies on some path from the source to the sink.¹⁰ This implies that $m \geq n - 1$, where $n = |V|$ and $m = |E|$. It will also be convenient to assume that all capacities are integers. (We can assume more generally that the capacities are rational numbers, since we can convert them to integers by multiplying them by the least common multiple of the denominators.)

Flows, Capacities, and Conservation: Given an *s-t* network, a *flow* (also called an *s-t flow*) is a function f that maps each edge to a nonnegative real number and satisfies the following properties:

Capacity Constraint: For all $(u, v) \in E$, $f(u, v) \leq c(u, v)$.

Flow conservation (or flow balance): For all $v \in V \setminus \{s, t\}$, the sum of flow along edges into v equals the sum of flows along edges out of v .

We can state flow conservation more formally as follows. First off, let us make the assumption that if (u, v) is *not* an edge of E , then $f(u, v) = 0$. We then define the total flow into v and

¹⁰Neither of these is an essential requirement. Given a network that fails to satisfy these assumptions, we can easily generate an equivalent one that satisfies both.

total flow out of v as:

$$f^{\text{in}}(v) = \sum_{(u,v) \in E} f(u,v) \quad \text{and} \quad f^{\text{out}}(v) = \sum_{(v,w) \in V} f(v,w).$$

Then flow conservation states that $f^{\text{in}}(v) = f^{\text{out}}(v)$, for all $v \in V \setminus \{s, t\}$. Note that flow conservation *does not* apply to the source and sink, since we think of ourselves as pumping flow from s to t .

Two examples are shown in Fig. 60, where we use the notation f/c on each edge to denote the flow f and capacity c for this edge.

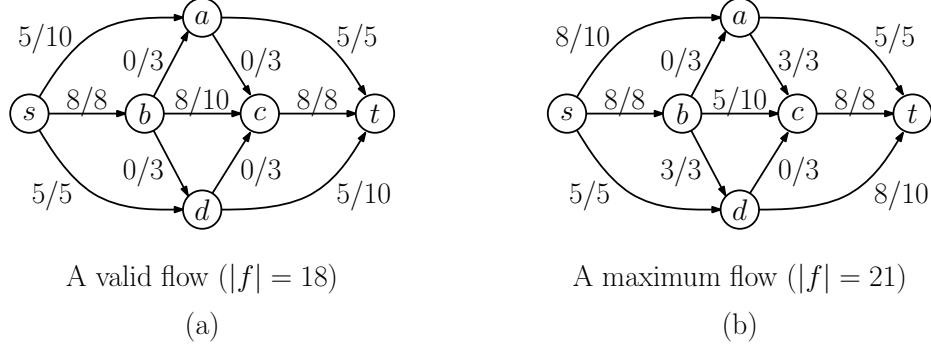


Fig. 60: A valid flow and a maximum flow.

The quantity $f(u, v)$ is called the *flow* along edge (u, v) . We are interested in defining the total flow, that is, the total amount of fluid flowing from s to t . The *value* of a flow f , denoted $|f|$, is defined as the sum of flows out of s , that is,

$$|f| = f^{\text{out}}(s) = \sum_{w \in V} f(s, w),$$

(For example, the value of the flow shown in Fig. 60(a) is $5 + 8 + 5 = 18$.) From flow conservation, it follows easily that this is also equal to the flow into t , that is, $f^{\text{in}}(t)$. We will prove this later.

Maximum Flow: Given an s - t network, an obvious optimization problem is to determine a flow of maximum value. More formally, the *maximum-flow problem* is, given a flow network $G = (V, E)$, and source and sink vertices s and t , find the flow of maximum value from s to t . (For example, in Fig. 60(b) we show flow of value $8 + 8 + 5 = 21$, which can be shown to be the maximum flow for this network.) Note that, although the value of the maximum flow is unique, there may generally be many different flow functions that achieve this value.

Path-Based Flows: The definition of flow we gave above is sometimes called the *edge-based* definition of flows. An alternative, but mathematically equivalent, definition is called the *path-based* definition of flows. Define an s - t *path* to be any simple path from s to t . For example, in Fig. 59, $\langle s, a, t \rangle$, $\langle s, b, a, c, t \rangle$ and $\langle s, d, c, t \rangle$ are all examples of s - t paths. There may generally be an exponential number of such paths (but that is alright, since this just a mathematical definition).

A *path-based flow* is a function that assigns each s - t path a nonnegative real number such that, for every edge $(u, v) \in E$, the sum of the flows on all the paths containing this edge is at most $c(u, v)$. Note that there is no need to provide a flow conservation constraint, because each path that carries a flow into a vertex (excluding s and t), carries an equivalent amount of flow out of that vertex. For example, in Fig. 61(b) we show a path-based flow that is equivalent to the edge-based flow of Fig. 61(a). The paths carrying zero flow are not shown.

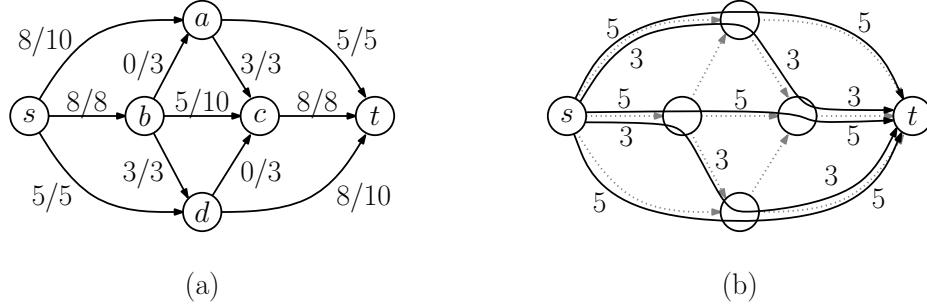


Fig. 61: (a) An edge-based flow and (b) its path-based equivalent.

The *value* of a path-based flow is defined to be the total sum of all the flows on all the s - t paths of the network. Although we will not prove it, the following claim is an easy consequence of the above definitions.

Claim: Given an s - t network G , under the assumption that there are no edges entering s or leaving t , G has an edge-based flow of value x if and only if G has a path-based flow of value x .

Multi-source, multi-sink networks: It may seem overly restrictive to require that there is only a single source and a single sink vertex. Many flow problems have situations in which many source vertices s_1, \dots, s_k and many sink vertices t_1, \dots, t_l . This can easily be modeled by just adding a special *super-source* s' and a *super-sink* t' , and attaching s' to all the s_i and attach all the t_j to t' . We let these edges have infinite capacity (see Fig. 62). Now by pushing the maximum flow from s' to t' we are effectively producing the maximum flow from all the s_i 's to all the t_j 's.

Note that we don't assume any correspondence between flows leaving source s_i and entering t_j . Flows from one source may flow into *any* sink vertex. In some cases, you would like to specify the flow from a certain source must arrive at a designated sink vertex. For example, imagine that the sources are manufacturing production centers and sinks are retail outlets, and you are told the amount of commodity from s_i to arrive at t_j . This variant of the flow problem, called the *multi-commodity flow problem*, is a much harder problem to solve (in fact, some formulations are NP-hard).

Lecture 18: Network Flows: The Ford-Fulkerson Algorithm

Network Flow: We continue discussion of the network flow problem. Last time, we introduced basic concepts, such the concepts s - t networks and flows. Today, we discuss the Ford-Fulkerson

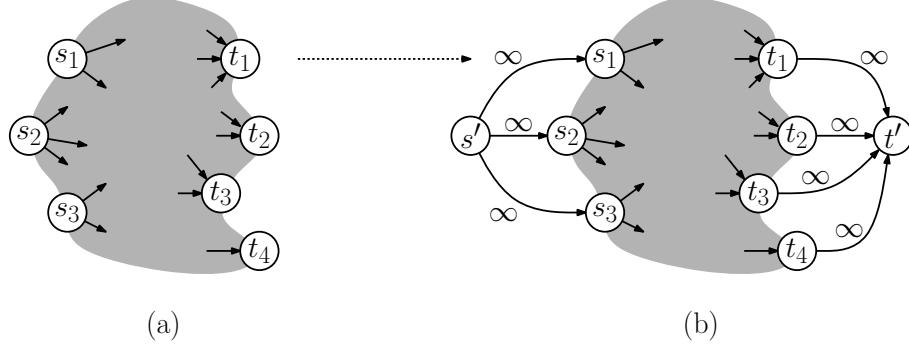


Fig. 62: Reduction from (a) multi-source/multi-sink to (b) single-source/single-sink.

Max Flow algorithm, cuts, and the relationship between flows and cuts.

Recall that a *flow network* is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonnegative *capacity* $c(u, v) \geq 0$, with a designated source vertex s and sink vertex t . We assume that there are no edges entering s or exiting t . A *flow* is a function f that maps each edge to a nonnegative real number that does not exceed the edge's capacity, and such that the total flow into any vertex other than s and t equals the total flow out of this vertex. The total *value* of a flow is equal to the sum of flows coming out of s (which, by flow conservation, is equal to the total flow entering t). The objective of the *max flow problem* is to compute a flow of maximum value. Today we present an algorithm for this problem.

Why Greedy Fails: Before considering our algorithm, we start by considering why a simple greedy scheme for computing the maximum flow does not work. The idea behind the greedy algorithm is motivated by the path-based notion of flow. (Recall this from the previous lecture.) Initially the flow on each edge is set to zero. Next, find any path P from s to t , such that the edge capacities on this path are all strictly positive. Let c_{\min} be the minimum capacity of any edge on this path. This quantity is called the *bottleneck capacity* of the path. Push c_{\min} units through this path. For each edge $(u, v) \in P$, set $f(u, v) \leftarrow c_{\min} + f(u, v)$, and decrease the capacity of (u, v) by c_{\min} . Repeat this until no s - t path (of positive capacity edges) remains in the network.

While this may seem to be a very reasonable algorithm, and will generally produce a valid flow, it may fail to compute the maximum flow. To see why, consider the network shown in Fig. 63(a). Suppose we push 5 units along the topmost path, 8 units along the middle path, and 5 units along the bottommost path. We have a flow of value 18. After adjusting the capacities (see Fig. 63(b)) we see that there is no path of positive capacity from s to t . Thus, greedy gets stuck.

Residual Network: The key insight to overcoming the problem with the greedy algorithm is to observe that, in addition to increasing flows on edges, it is possible to *decrease* flows on edges that already carry flow (as long as the flow never becomes negative). It may seem counterintuitive that this would help, but we shall see that it is exactly what is needed to obtain an optimal solution.

To make this idea clearer, we first need to define the notion of the residual network and

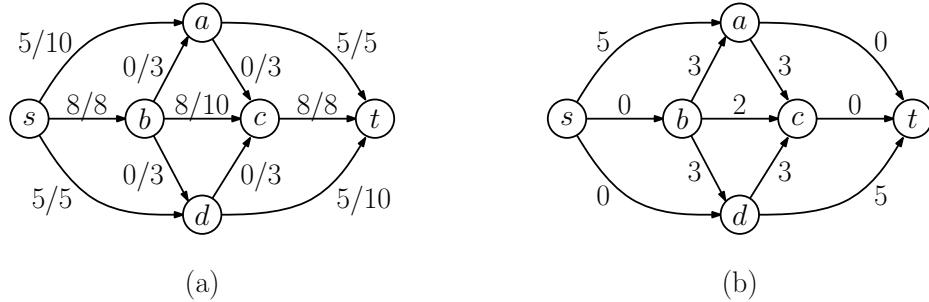


Fig. 63: The greedy flow algorithm can get stuck before finding the maximum flow.

augmenting paths. Given a flow network G and a flow f , define the *residual network*, denoted G_f , to be a network having the same vertex set and same source and sink, and whose edges are defined as follows:

Forward edges: For each edge (u, v) for which $f(u, v) < c(u, v)$, create an edge (u, v) in G_f and assign it the capacity $c_f(u, v) = c(u, v) - f(u, v)$. Intuitively, this edge signifies that we can add up to $c_f(u, v)$ additional units of flow to this edge without violating the original capacity constraint.

Backward edges: For each edge (u, v) for which $f(u, v) > 0$, create an edge (v, u) in G_f and assign it a capacity of $c_f(v, u) = f(u, v)$. Intuitively, this edge signifies that we can cancel up to $f(u, v)$ units of flow along (u, v) . Conceptually, by pushing positive flow along the reverse edge (v, u) we are decreasing the flow along the original edge (u, v) .

Observe that every edge of the residual network has *strictly positive* capacity. (This will be important later on.) Note that each edge in the original network may result in the generation of up to two new edges in the residual network. Thus, the residual network is of the same asymptotic size as the original network.

An example of a flow and the associated residual network are shown in Fig. 64(a) and (b), respectively. For example, the edge (b, c) of capacity 2 signifies that we can add up to 2 more units of flow to edge (b, c) and the edge (c, b) of capacity 8 signifies that we can cancel up to 8 units of flow from the edge (b, c) .

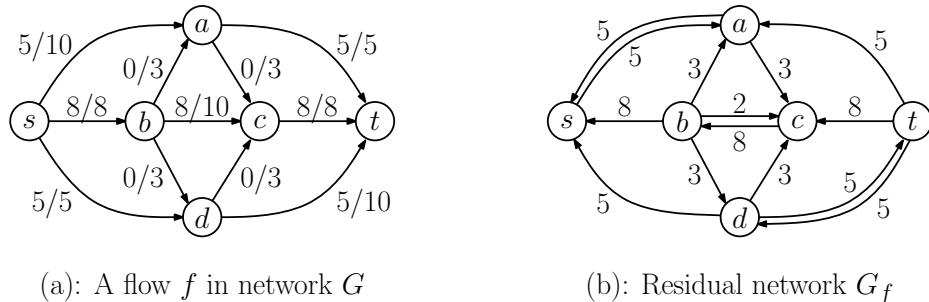


Fig. 64: A flow f and the residual network G_f .

The capacity of each edge in the residual network is called its *residual capacity*. The key observation about the residual network is that if we can push flow through the residual network then we can push this additional amount of flow through the original network. This is formalized in the following lemma. Given two flows f and f' , we define their *sum*, $f + f'$, in the natural way, by summing the flows along each edge. If $f'' = f + f'$, then $f''(u, v) = f(u, v) + f'(u, v)$. Clearly, the value of $f + f'$ is equal to $|f| + |f'|$.

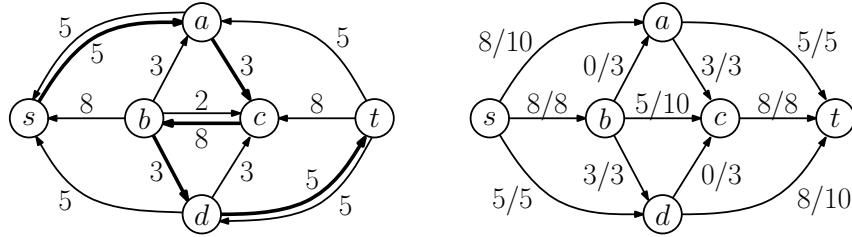
Lemma: Let f be a flow in G and let f' be a flow in G_f . Then $(f + f')$ is a flow in G .

Proof: (Sketch) To show that the resulting flow is valid, we need to show that it satisfies both the capacity constraints and flow conservation. It is easy to see that the capacities of G_f were exactly designed so that any flow along an edge of G_f when added to the flow f of G will satisfy G 's capacity constraints. Also, since both flows satisfy flow conservation, it is easy to see that their sum will as well. (More generally, any linear combination $\alpha f + \beta f'$ will satisfy flow conservation.)

This lemma suggests that all we need to do to increase the flow is to find any flow in the residual network. This leads to the notion of an augmenting path.

Augmenting Paths and Ford-Fulkerson: Consider a network G , let f be a flow in G , and let G_f be the associated residual network. An *augmenting path* is a simple path P from s to t in G_f . The *residual capacity* (also called the *bottleneck capacity*) of the path is the minimum capacity of any edge on the path. It is denoted $c_f(P)$. (Recall that all the edges of G_f are of strictly positive capacity, so $c_f(P) > 0$.) By pushing $c_f(P)$ units of flow along each edge of the path, we obtain a valid flow in G_f , and by the previous lemma, adding this to f results in a valid flow in G of strictly higher value.

For example, in Fig. 65(a) we show an augmenting path of capacity 3 in the residual network for the flow given earlier in Fig. 64. In (b), we show the result of adding this flow to every edge of the augmenting path. Observe that because of the backwards edge (c, b) , we have decreased the flow along edge (b, c) by 3, from 8 to 5.



(a): Augmenting path of capacity 3 (b): The flow after augmentation

Fig. 65: Augmenting path and augmentation.

How is this different from the greedy algorithm? The greedy algorithm only increases flow on edges. Since an augmenting path may increase flow on a backwards edge, it may actually *decrease* the flow on some edge of the original network.

This observation naturally suggests an algorithm for computing flows of ever larger value. Start with a flow of weight 0, and then repeatedly find an augmenting path. Repeat this

until no such path exists. This, in a nutshell, is the simplest and best known algorithm for computing flows, called the *Ford-Fulkerson method*. (We do not call it an “algorithm,” since the method of selecting the augmenting path is not specified. We will discuss this later.) It is summarized in the code fragment below.

Ford-Fulkerson Network Flow

```

ford-fulkerson-flow(G = (V, E, s, t)) {
    f = 0 (all edges carry zero flow)
    while (true) {
        G' = the residual-network of G for f
        if (G' has no s-t augmenting path)
            break                                // no augmentations left
        P = any-augmenting-path of G'             // augmenting path
        c = minimum capacity edge of P           // augmentation amount
        augment f by adding c to the flow on every edge of P
    }
    return f
}

```

There are three issues to consider before declaring this a reasonable algorithm.

- How efficiently can we perform augmentation?
- How many augmentations might be required until converging?
- If no more augmentations can be performed, have we found the max-flow?

Let us consider first the question of how to perform augmentation. First, given G and f , we need to compute the residual network, G_f . This is easy to do in $O(n + m)$ time, where $n = |V|$ and $m = |E|$. We assume that G_f contains only edges of strictly positive capacity. Next, we need to determine whether there exists an augmenting path from s to t in G_f . We can do this by performing either a DFS or BFS in the residual network starting at s and terminating as soon (if ever) t is reached. Let P be the resulting path. Clearly, this can be done in $O(n + m)$ time as well. Finally, we compute the minimum cost edge along P , and increase the flow f by this amount for every edge of P .

Two questions remain: What is the best way to select the augmenting path, and is this correct in the sense of converging to the maximum flow? Next, we consider the issue of correctness. Before doing this, we will need to introduce the concept of a cut.

Cuts: In order to show that Ford-Fulkerson leads to the maximum flow, we need to formalize the notion of a “bottleneck” in the network. Intuitively, the flow cannot be increased forever, because there is some subset of edges, whose capacities eventually become saturated with flow. Every path from s to t must cross one of these saturated edges, and so the sum of capacities of these edges imposes an upper bound on size of the maximum flow. Thus, these edges form a bottleneck.

We want to make this concept mathematically formal. Since such a set of edges lie on every path from s to t , their removal defines a partition separating the vertices that s can reach from the vertices that s cannot reach. This suggests the following concept.

Given a network G , define a *cut* (also called an *s-t cut*) to be a partition of the vertex set into two disjoint subsets $X \subseteq V$ and $Y = V \setminus X$, where $s \in X$ and $t \in Y$. We define the *net flow* from X to Y to be the sum of flows from X to Y minus the sum of flows from Y to X , that is,

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y) - \sum_{y \in Y} \sum_{x \in X} f(y, x).$$

Observe that $f(X, Y) = -f(Y, X)$.

For example, Fig. 66 shows a flow of value 17. It also shows a cut $(X, Y) = (\{s, a\}, \{b, c, d, t\})$, where $f(X, Y) = 17$.

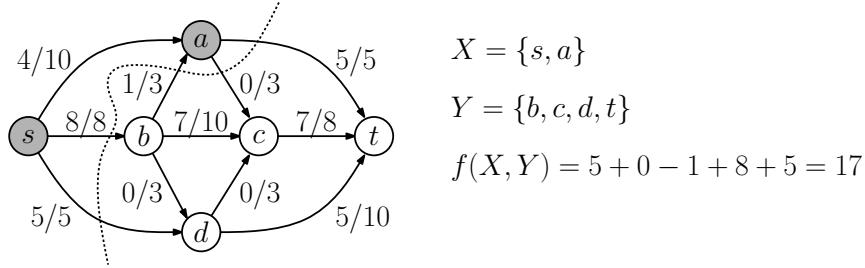


Fig. 66: Flow of value 17 across the cut $(\{s, a\}, \{b, c, d, t\})$.

Lemma: Let (X, Y) be any *s-t* cut in a network. Given any flow f , the value of f is equal to the net flow across the cut, that is, $f(X, Y) = |f|$.

Proof: Recall that there are no edges leading into s , and so we have $|f| = f^{\text{out}}(s) = f^{\text{out}}(s) - f^{\text{in}}(s)$. Since all the other nodes of X must satisfy flow conservation it follows that

$$|f| = \sum_{x \in X} (f^{\text{out}}(x) - f^{\text{in}}(x))$$

Now, observe that every edge (u, v) where both u and v are in X contributes one positive term and one negative term of value $f(u, v)$ to the above sum, and so all of these cancel out. The only terms that remain are the edges that either go from X to Y (which contribute positively) and those from Y to X (which contribute negatively). Thus, it follows that the value of the sum is exactly $f(X, Y)$, and therefore $|f| = f(X, Y)$.

Define the *capacity* of the cut (X, Y) to be the sum of the capacities of the edges leading from X to Y , that is,

$$c(X, Y) = \sum_{x \in X} \sum_{y \in Y} c(x, y).$$

(Note that the capacities of edges from Y into X are ignored.) Clearly it is not possible to push more flow through a cut than its capacity. Combining this with the above lemma we have:

Lemma: Given any *s-t* cut (X, Y) and any flow f we have $|f| \leq c(X, Y)$.

The optimality of the Ford-Fulkerson method is based on the following famous theorem, called the *Max-Flow/Min-Cut Theorem*. Basically, it states that in any flow network the minimum capacity cut acts like a bottleneck to limit the maximum amount of flow. The Ford-Fulkerson method terminates when it finds this bottleneck, and hence on termination, it finds both the minimum cut and the maximum flow.

Max-Flow/Min-Cut Theorem: The following three conditions are equivalent.

- (i) f is a maximum flow in G ,
- (ii) The residual network G_f contains no augmenting paths,
- (iii) $|f| = c(X, Y)$ for some cut (X, Y) of G .

Proof:

- (i) \Rightarrow (ii): (by contradiction) If f is a max flow and there were an augmenting path in G_f , then by pushing flow along this path we would have a larger flow, a contradiction.
- (ii) \Rightarrow (iii): If there are no augmenting paths then s and t are not connected in the residual network. Let X be those vertices reachable from s in the residual network, and let Y be the rest. Clearly, (X, Y) forms a cut. Because each edge crossing the cut must be saturated with flow, it follows that the flow across the cut equals the capacity of the cut, thus $|f| = c(X, Y)$.
- (iii) \Rightarrow (i): (by contradiction) Suppose that there is a flow f' whose value exceeds $|f|$. Then we would have $|f'| > c(X, Y)$, which contradicts the previous lemma.

We have established that, on termination, Ford-Fulkerson generates the maximum flow. But, is it guaranteed to terminate and, if so, how long does it take? We will consider this question next time.

Lecture 19: More on Network Flow

Analysis of Ford-Fulkerson: We have established that, on termination, Ford-Fulkerson generates the maximum flow. But, is it guaranteed to terminate and, if so, how long does it take? First, it is easy to see that it will terminate. Recall that we assumed that all edge capacities are integers. Every augmentation increases the flow by an integer amount, and therefore, after a finite number of augmentations (at most the sum of all the capacities of edges incident to s) the algorithm must terminate.

Recall our convention that $n = |V|$ and $m = |E|$. Since we assume that every vertex is reachable from s , it follows that $m \geq n - 1$. Therefore, $n = O(m)$. Running times of the form $O(n + m)$ can be expressed more simply as $O(m)$.

As we saw last time, the residual network can be computed in $O(n + m) = O(m)$ time and an augmenting path can also be found in $O(m)$ time. Therefore, the running time of each augmentation step is $O(m)$. How many augmentations are needed? Unfortunately, the number could be very large. To see this, consider the example shown in Fig. 67. If the

algorithm were smart enough to send flow along the topmost and bottommost paths, each of capacity 100, the algorithm would terminate in just two augmenting steps to a total flow of value 200. However, suppose instead that it foolishly augments first through the path going through the center edge (see Fig. 67(b)). Then it would be limited to a bottleneck capacity of 1 unit. In the second augmentation, it could now route through the complementary path, this time undoing the flow on the center edge, and again with bottleneck capacity 1 (see Fig. 67(c)). Proceeding in this way, it will take 200 augmentations until we terminate with the final maximum flow (see Fig. 67(d)). Without increasing the network's size, we could replace the 100's with as large a number as we like and thus make the running time arbitrarily high.

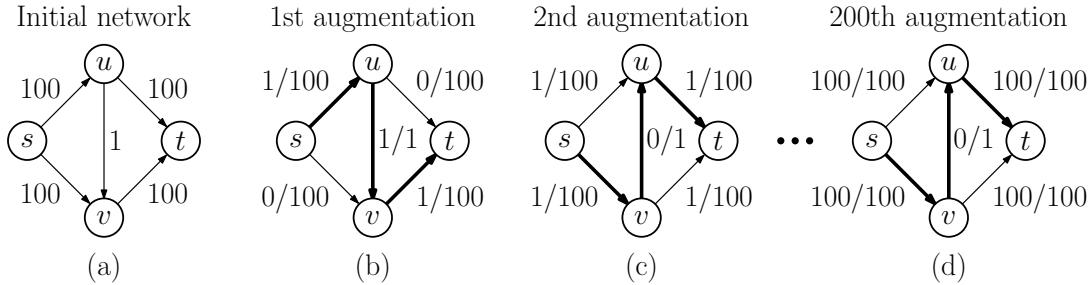


Fig. 67: Bad example for Ford-Fulkerson.

If we let F^* denote the final maximum flow value, the number of augmentation steps can be as high as F^* . If we make the reasonable assumption that each augmentation step takes at least $\Omega(m)$ time, the total running time can be as high as $\Omega(F^* \cdot m)$. Since F^* may be arbitrarily high (it depends neither on n or m), this running time could be arbitrarily high, as a function of n and m .

Faster Max-Flow Algorithms: We have shown that if the augmenting path was chosen in a bad way the algorithm could run for a very long time before converging on the final flow. There are a number of alternatives that result in considerably better running times, however. Below we sketch a few algorithms are more complex than Ford-Fulkerson, but may be superior with respect to asymptotic running times.

Scaling Algorithm: As we saw above, Ford-Fulkerson can perform very badly when the optimum flow is very high. But the above example indicates that we do badly when we augment along paths of very low capacity. What if we were to select paths of high capacity. We could attempt to find the path of maximum capacity, but it turns out that it is not necessary to be quite so greedy. Selecting any augmenting path whose residual capacity is within a constant of the maximum is good enough. This gives rise to something called the *scaling algorithm* for max flows.

The idea is to start with an upper bound on the maximum possible flow. The sum of capacities of the edges leaving s certainly suffices:

$$C = \sum_{(s,v) \in E} c(s, v).$$

Clearly, the maximum flow value cannot exceed C . Next, define Δ to be the largest power of 2, such that $\Delta \leq C$. Given any flow f (initially the flow of value 0), define $G_f(\Delta)$ to be the residual network consisting *only of edges of residual capacity at least Δ* . (That is, we ignore all edges of small capacity.) Repeatedly find an augmenting path in $G_f(\Delta)$, augment the flow along this path, and then compute the residual network $G_{f'}(\Delta)$ for the augmented flow f' . Repeat this until no augmenting paths remain.

Intuitively, each such augmentation has the advantage that it will make big progress, because each augmentation will increase the flow by at least Δ units. When no more augmenting paths remain, set $\Delta \leftarrow \lceil \Delta/2 \rceil$, compute $G_f(\Delta)$ for the new value of Δ , and repeat the process. Eventually, we will have $\Delta = 1$. When the algorithm terminates for $\Delta = 1$, we have the final maximum flow.

For example, consider the same network as in Fig. 67. We start with the upper bound of $C = 401$. (Let's just round this to 400 for simplicity.) As usual, let us start with f being the zero flow (see Fig. 68(a)).

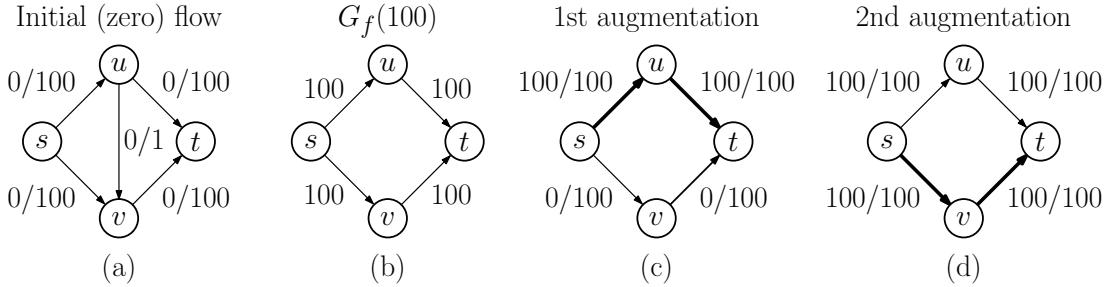


Fig. 68: Scaling algorithm for network flow.

The residual networks $G_f(400)$, $G_f(200)$ are empty, since there are no edges whose capacity exceeds these values. However, $G_f(100)$ includes the edges of capacity 100 (see Fig. 68(b)). They key is that is *does not* contain the edge of capacity 1, and therefore we cannot push flow through this edge at this time. After two augmentations, we obtain the flow shown in Fig. 68(d). The algorithm will then try all the remaining residual networks $G_f(50)$, $G_f(25)$, \dots , $G_f(1)$. The edge of capacity 1 will appear only in the final residual network, but by this time it serves no purpose because we have already found the maximum flow.

It can be shown that for each choice of Δ , the algorithm terminates after $O(m)$ augmentation steps. (This is not trivial. See our text for a proof.) Since each augmentation takes $O(m)$ time, the time spent for each value of Δ is $O(m^2)$. Finally, since we cut the value of Δ in half with each iteration, it is easy to see that we will consider $O(\log C)$ different values of Δ . Whenever C is sufficiently large (that is, when $C/\log C$ is asymptotically larger than m) the scaling algorithm will outperform the Ford-Fulkerson algorithm.

Perhaps more importantly, observe that the total number of bits needed to encode the weights of number of magnitude C is $O(\log C)$. Therefore, the total space needed to encode the input network is $O(m \log C)$. Although the running time of the scaling algorithm is not polynomial in n and m (which would be the ideal), it is polynomial in the *number of bits* needed to encode the input. Thus, it is in some sense a polynomial time algorithm. Algorithms that run in

time that is polynomial in the number of bits of input are said to run in *pseudo-polynomial time*. The scaling algorithm is an example of such an algorithm.

Edmonds-Karp Algorithm: Neither of the algorithms we have seen so far runs in “truly” polynomial time (that is, polynomial in n and m , irrespective of the magnitudes of the capacity.) Edmonds and Karp developed the first such algorithm. This algorithm uses Ford-Fulkerson as its basis, but with the change that When finding the augmenting path, we compute the $s-t$ path in the residual network having the *smallest number of edges*. Note that this can be accomplished by using BFS to compute the augmenting path, since BFS effectively finds shortest path based on the number of edges. It can be shown that the total number of augmenting steps using this method is $O(nm)$. (Again, this is not trivial. Our book does not give a proof, but one can be found in the algorithms book by Cormen, Leiserson, Rivest, and Stein.) Recall that each augmenting path can be computed in $O(m)$ time. Thus, the overall running time is $O(nm^2)$.

Even Faster Algorithms: The max-flow problem is widely studied, and there are many different algorithms. Our book discusses one algorithm, called the *pre-flow push algorithm*. There are a number of variants of this algorithm, but the simplest one runs in $O(n^3)$ time. The fastest known algorithm for network flow is a hybrid of two algorithms (depending on the ratio between the number of edges and the number of vertices). Its running time is $O(nm)$. The two algorithms are due to King, Rao, and Tarjan (1994) and Orlin (2013).

Applications of Max-Flow: The network flow problem has a huge number of applications. Many of these applications do not appear at first to have anything to do with networks or flows. This is a testament to the power of this problem. In this lecture and the next, we will present a few applications from our book. (If you need more convincing of this, however, see the exercises in Chapter 7 of KL. There are over 40 problems, most of which involve reductions to network flow.)

Maximum Matching in Bipartite Graphs: Earlier in the semester we talked about stable marriage. There are many applications where pairings are to be sought, and there are many criteria for what constitutes a good pairing. We will consider another one here. As in the stable marriage problem, we will present it in the form of a “dating game,” but there are many serious applications of this general problem.

Suppose you are running a dating service, and there are a set of men X and a set of women Y . Using a questionnaire you establish which men are compatible with which women. Your task is to pair up as many compatible pairs of men and women as possible, subject to the constraint that each man is paired with at most one woman, and vice versa. (It may be that some men are not paired with any woman.) Note that, unlike the stable marriage problem, there are no preferences here, only compatibility and incompatibility constraints.

Recall that an undirected graph $G = (V, E)$ is said to be *bipartite* if V can be partitioned into two sets X and Y , such that every edge has one endpoint in X and the other in Y . This problem can be modeled as an undirected, bipartite graph whose vertex set is $V = X \cup Y$ and whose edge set consists of pairs $\{u, v\}$, $u \in X$, $v \in Y$ such that u and v are compatible (see Fig. 69(a)). Given a graph, a *matching* is defined to be a subset of edges $M \subseteq E$ such that for each $v \in V$, there is at most one edge of M incident to v . Clearly, the objective to the

dating problem is to find a maximum matching in G that has the highest cardinality. Such a matching is called a *maximum matching* (see Fig. 69(b)).

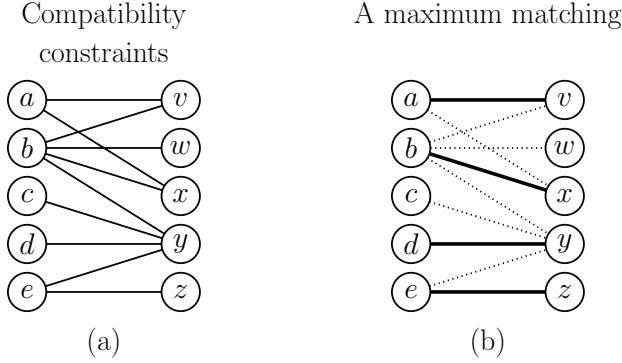


Fig. 69: A bipartite graph G and a maximum matching in G .

The resulting undirected graph has the property that its vertex set can be divided into two groups such that all its edges go from one group to the other. This problem is called the *maximum bipartite matching problem*.

We will now show a reduction from maximum bipartite matching to network flow. In particular, we will show that, given any bipartite graph G (see Fig. 70(a)) for which we want to solve the maximum matching problem, we can convert it into an instance of network flow G' , such that the maximum matching on G can be extracted from the maximum flow on G' .

To do this, we construct a flow network $G' = (V', E')$ as follows. Let s and t be two new vertices and let $V' = V \cup \{s, t\}$.

$$E' = \begin{cases} \{(s, u) \mid u \in X\} \cup & (\text{connect source to left-side vertices}) \\ \{(v, t) \mid v \in Y\} \cup & (\text{connect right-side vertices to sink}) \\ \{(u, v) \mid (u, v) \in E\} & (\text{direct } G\text{'s edges from left to right}). \end{cases}$$

Set the capacity of all edges in this network to 1 (see Fig. 70(b)).

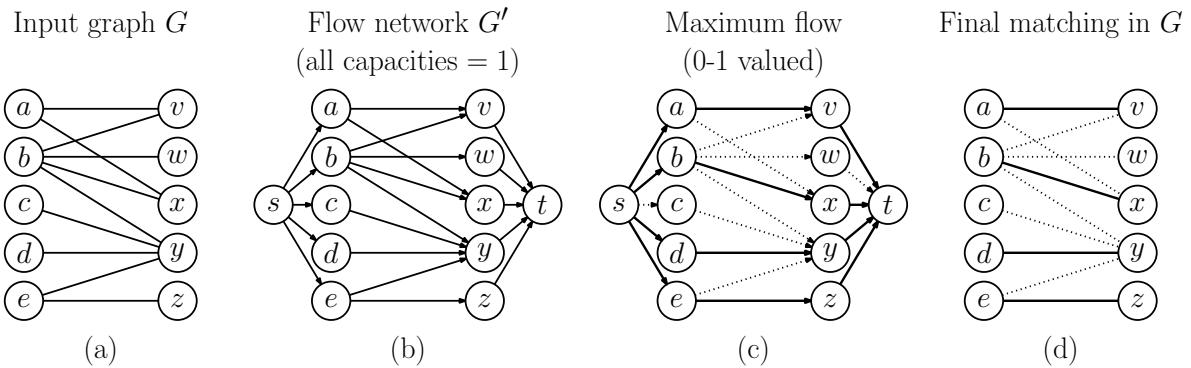


Fig. 70: Reducing bipartite matching to network flow.

Now, compute the maximum flow in G' (see Fig. 70(c)). Although in general it can be that

flows are real numbers, observe that the Ford-Fulkerson method will only assign integer value flows to the edges (and this is true of all existing network flow algorithms).

Since each vertex in X has exactly one incoming edge, it can have flow along at most one outgoing edge, and since each vertex in Y has exactly one outgoing edge, it can have flow along at most one incoming edge. Thus letting f denote the maximum flow, we can define a matching

$$M = \{(u, v) \mid u \in X, v \in Y, f(u, v) > 0\}$$

(see Fig. 70(d)).

We claim that this matching is maximum because for every matching there is a corresponding flow of equal value, and for every (integer) flow there is a matching of equal value. Thus by maximizing one we maximize the other.

Because the capacities are so low, we do not need to use a fancy implementation. Recall that Ford-Fulkerson runs in time $O(m \cdot F^*)$, where F^* is the final maximum flow. In our case F^* is not very large. In particular, the total capacity of the edges coming out of S is at most $|X| \leq |V| = n$. Therefore, the running time of Ford-Fulkerson on this instance is $O(m \cdot F^*) = O(nm)$.

There are other algorithms for maximum bipartite matching. The best is due to Hopcroft and Karp, and runs in $O(\sqrt{n} \cdot m)$ time.

Lecture 20: Extensions of Network Flow

Extensions of Network Flow: Network flow is an important problem because it is useful in a wide variety of applications. We will discuss two useful extensions to the network flow problem. We will show that these problems can be reduced to network flow, and thus a single algorithm can be used to solve both of them. Many computational problems that would seem to have little to do with flow of fluids through networks can be expressed as one of these two extended versions.

Circulation with Demands: There are many problems that are similar to network flow in which, rather than transporting flow from a single source to a single sink, we have a collection of *supply nodes* that want to ship flow (or products or goods) and a collection of *demand nodes* that want to receive flow. Each supply node is associated with the amount of product it wishes to ship and each demand node is associated with the amount that it wishes to receive. The question that arises is whether there is some way to get the products from the supply nodes to the demand nodes, subject to the capacity constraints. This is a *decision problem* (or *feasibility problem*), meaning that it has a yes-no answer, as opposed to maximum flow, which is an *optimization problem*.

We can model both supply and demand nodes elegantly by associating a single numeric value with each node, called its *demand*. If $v \in V$ is a demand node, let d_v the amount of this demand. If v is a supply node, we model this by assigning it a negative demand, so that $-d_v$

is its available supply. Intuitively, supplying x units of product is equivalent to demanding receipt of $-x$ units.¹¹ If v is neither a supply or demand node, we let $d_v = 0$.

Suppose that we are given a directed graph $G = (V, E)$ in which each edge (u, v) is associated with a positive capacity $c(u, v)$ and each vertex v is associated with a supply/demand value d_v . Let S denote the set of *supply nodes* ($d_v < 0$), and let T denote the set of *demand nodes* ($d_v > 0$). Note that vertices of S may have incoming edges and vertices of T may have outgoing edges. (For example, in Fig. 71(a), we show a network in which each node is each labeled with its demand.)

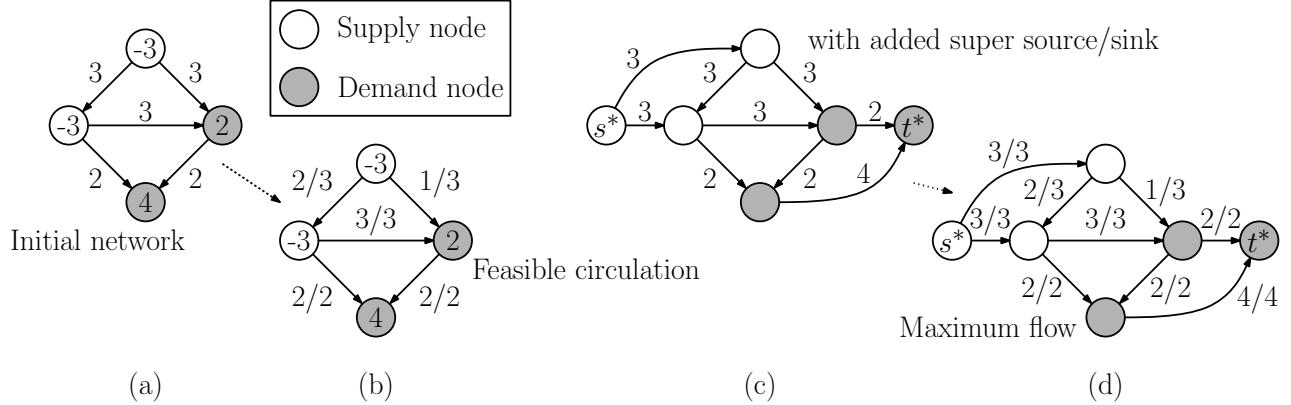


Fig. 71: Reducing the circulation problem to network flow.

Recall that, given a flow f and a node v , $f^{\text{in}}(v)$ is the sum of flows along incoming edges to v and $f^{\text{out}}(v)$ is the sum of flows along outgoing edges from v . We define a *circulation* in G to be a function f that assigns a nonnegative real number to each edge that satisfies the following two conditions.

Capacity constraints: For each $(u, v) \in E$, $0 \leq f(u, v) \leq c(u, v)$.

Supply/Demand constraints: For vertex $v \in V$, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

(In Fig. 71(b), we show a valid circulation for the network of part (a).) Observe that demand constraints correspond to the flow-balance in the original max flow problem, since if a vertex is not in S or T , then $d_v = 0$ and we have $f^{\text{in}}(v) = f^{\text{out}}(v)$. Also it is easy to see that the total demand must equal the total supply, otherwise we have no chance of finding a feasible circulation. That is, we require that

$$\sum_{v \in V} d_v = 0 \quad \text{or equivalently} \quad -\sum_{v \in S} d_v = \sum_{v \in T} d_v$$

Define $D = \sum_{v \in T} d_v$ denote the *total demand*. (Note that this is equal to the negation of the total supply, $\sum_{v \in S} d_v$.)

We claim that we can convert any instance G of the circulation problem to an equivalent network flow problem. We assume that total supply equals total demand (since if not we

¹¹I would not advise applying this in real life. I doubt that the IRS would appreciate it if your paid your \$100 tax bill by demanding that they send you $-\$100$ dollars.

can simply answer “no” immediately.) The reduction is similar in spirit to what we did for bipartite matching. Given the network $G = (V, E)$ as input (with vertex demands d_v and edges capacities $c(u, v)$):

- Create a new network $G' = (V', E')$ that has all the same vertices and edges as G (that is, $V' \leftarrow V$ and $E' \leftarrow E$)
- Add to V' a *super-source* s^* and a *super-sink* t^*
- For each supply node $v \in S$, we add a new edge (s^*, v) of capacity $-d_v$
- For each demand node $u \in T$, we add a new edge (u, t^*) of capacity d_u

The entire construction is illustrated in Fig. 71(c).

Intuitively, these new edges will be responsible for providing the necessary supply for vertices of S and draining off the excess demand from the vertices of T . Suppose that we now compute the maximum flow in G' (by whichever maximum flow algorithm you like). If the flow value is at least D , then intuitively, we have managed to push enough flow into the network and (by flow balance) enough flow out of the network to satisfy all the demand constraints (see Fig. 71(d)). The following lemma proves formally that this is a necessary and sufficient condition for a circulation to exist.

Lemma: There is a feasible circulation in G if and only if G' has an s^*-t^* flow of value D .

Proof: Suppose that there is a feasible circulation f in G . The value of this circulation (the net flow coming out of all supply nodes) is clearly D . We can create a flow f' of value D in G' , by saturating all the edges coming out of s^* and all the edges coming into t^* . We claim that this is a valid flow for G' . Clearly it satisfies all the capacity constraints. To see that it satisfies the flow balance constraints observe that for each vertex $v \in V$, we have one of three cases:

- ($v \in S$) The flow into v from s^* matches the supply coming out of v from the circulation.
- ($v \in T$) The flow out of v to t^* matches the demand coming into v from the circulation.
- ($v \in V \setminus (S \cup T)$) We have $d_v = 0$, which means that it already satisfied flow constraint.

Conversely, suppose that we have a flow f' of value D in G' . It must be that each edge leaving s^* and each edge entering t^* is saturated. Therefore, by flow balance of f' , all the supply nodes and all the demand nodes have achieved their desired supply/demand quotas. Therefore, by ignoring the flows along the edges incident to s^* and t^* , we have a feasible circulation f for G . This completes the proof.

Note that we have not presented a new algorithm. Rather, we have reduced the problem of computing circulations to the known problem of computing maximum flows. The running time of the overall solution is equal to the time to compute the reduction, which is $O(n + m)$ plus the time to compute max flows (which is $O(mn)$ time by the current best technology).

Circulations with Upper and Lower Capacity Bounds: Sometimes, in addition to having a certain maximum flow value, we would also like to impose minimum capacity constraints. That is, given a network $G = (V, E)$, for each edge $(u, v) \in E$ we would like to specify two constraints $\ell(u, v)$ and $c(u, v)$, where $0 \leq \ell(u, v) \leq c(u, v)$. A circulation function f must satisfy the same demand constraints as before, but must also satisfy both the upper and lower flow bounds:

(New) Capacity Constraints: For each $(u, v) \in E$, $\ell(u, v) \leq f(u, v) \leq c(u, v)$.

Demand Constraints: For vertex $v \in V$, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

Henceforth, we will use the term *upper flow bound* in place of *capacity* (since it doesn't make sense to talk about a lower bound as a capacity constraint). An example of such a network is shown in Fig. 72(a), and a valid circulation is shown in Fig. 72(b).

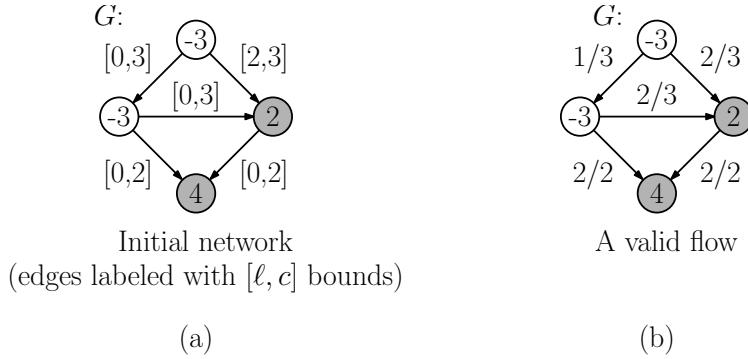


Fig. 72: (a) A network with both upper and lower flow bounds and (b) a valid circulation.

We will reduce this problem to a standard circulation problem (with just the usual upper capacity bounds). To help motivate our reduction, suppose (for conceptual purposes) that we generate an initial (possibly invalid) circulation f_0 that exactly satisfies all the lower flow bounds. In particular, we let $f_0(u, v) = \ell(u, v)$ (see Fig. 73(a)). This circulation may be invalid because f_0 need not satisfy the demand constraints (which, recall, provide for flow balance as well). We will modify the supply/demand values to compensate for this imbalance. Since the lower-bound constraints are all satisfied, it will be possible to apply a standard circulation algorithm (without lower flow bounds) to solve the problem.

To make this formal, for each $v \in V$, let L_v denote the *excess flow* coming into v in f_0 , that is

$$L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v) = \sum_{(u,v) \in E} \ell(u, v) - \sum_{(v,w) \in E} \ell(v, w).$$

(Note that this may be negative, which means that we have a flow deficit.) If we are lucky, then $L_v = d_v$, and v 's supply/demand needs are already met. Otherwise, we will adjust the supply and demand values so that by (1) computing any valid circulation f_1 for the adjusted values and (2) combining this with f_0 , we will obtain a flow that satisfies all the requirements.

How do we adjust the supply/demand values? We want to generate a net flow of d_v units coming into v and cancel out the excess L_v coming in. That is, we want f_1 to satisfy:

$$f_1^{\text{in}}(v) - f_1^{\text{out}}(v) = d_v - L_v.$$

(In particular, this means that if we combine f_0 and f_1 by summing their flows for each edge, then the final flow into v will be d_v , as desired.)

How do we determine whether there exists such a circulation f_1 ? Let's consider how to set the edge capacities. We have already sent $\ell(u, v)$ units of flow through the edge (u, v) , which implies that we have $c(u, v) - \ell(u, v)$ capacity remaining. (Note that unlike our definition of residual graphs, we do not want to allow for the possibility of “undoing” flow. Can you see why not?)

We are now ready to put the pieces together. Given the network $G = (V, E)$ as input (with vertex demands d_v and lower and upper flow bounds $\ell(u, v)$ and $c(u, v)$):

1. Create an initial pseudo-circulation f_0 by setting $f(u, v) = \ell(u, v)$ (see Fig. 73(a))
2. Create a new network $G' = (V', E')$ that has all the same vertices and edges as G (that is, $V' \leftarrow V$ and $E' \leftarrow E$)
3. For each $(u, v) \in E'$, set its adjusted capacity to $c'(u, v) \leftarrow c(u, v) - \ell(u, v)$
4. For each $v \in V'$, set its adjusted demand to $d'_v \leftarrow d_v - L_v$, where, $L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v)$ (see Fig. 73(b))

Now, invoke any (standard) circulation algorithm on G' (see Fig. 73(c)). Note that there is no need to consider lower flow bounds with G' , because f_0 has already taken care of those. If the algorithm reports that there is no valid circulation for G' , then we declare that there is no valid circulation for the original network G . If the algorithm returns a valid circulation f_1 for G' , then the final circulation for G is the combination $f_0 + f_1$ (see Fig. 73(d)).

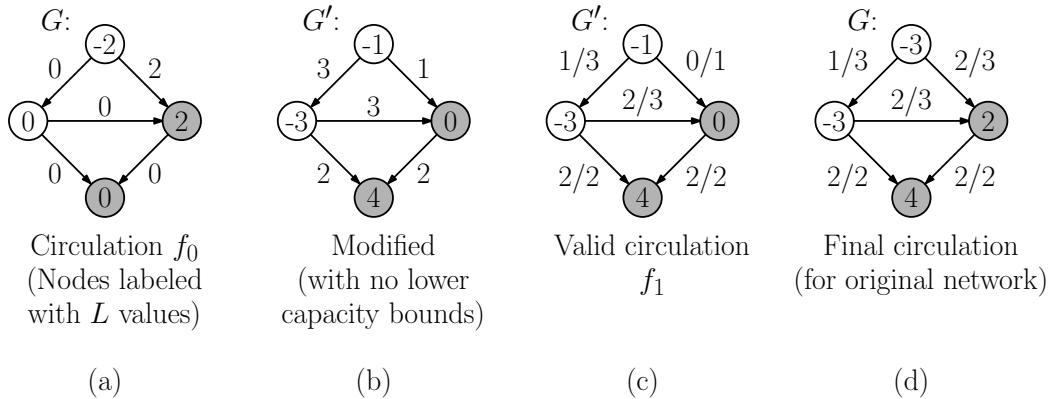


Fig. 73: Reducing the circulation problem with upper and lower flow bounds to a standard circulation problem.

To establish the correctness of our reduction, we prove below that its output $f_0 + f_1$ is a valid circulation for G (with lower flow bounds) if and only if f_1 is a valid circulation circulation for G' .

Lemma: The network G (with both lower and upper flow bounds) has a feasible circulation if and only if G' (with only upper capacity bounds) has a feasible circulation.

Proof: (Sketch. See KL for a formal proof.) Intuitively, if G' has a feasible circulation f' then the circulation $f(u, v) = f'(u, v) + \ell(u, v)$ can be shown to be a valid circulation for G and it satisfies the lower flow bounds. Conversely, if G has a feasible circulation (satisfying both the upper and lower flow bounds), then let $f'(u, v) = f(u, v) - \ell(u, v)$. As above, it can be shown that f' is a valid circulation for G' . (Think of f' as f_1 and f as $f_0 + f_1$.)

Note that (as in the original circulation problem) we have not presented a new algorithm. Instead, we have shown how to *reduce* the current problem (circulation with lower and upper flow bounds) to a problem we have already solved (circulation with only upper bounds). Again, the running time will be the sum of the time to perform the reduction, which is easily seen to be $O(n + m)$ plus the time to compute the circulation, which as we have seen reduces to the time to compute a maximum flow, which according to the current best technology is $O(nm)$ time.

Application: Survey Design: To demonstrate the usefulness of circulations with lower flow bounds, let us consider an application problem that arises in the area of data mining. A company sells k different products, and it maintains a database which stores which customers have bought which products recently. We want to send a survey to a subset of n customers. We will tailor each survey so it is appropriate for the particular customer it is sent to. Here are some guidelines that we want to satisfy:

- The survey sent to a customer will ask questions only about the products this customer has purchased.
- We want to get as much information as possible, but do not want to annoy the customer by asking too many questions. (Otherwise, they will simply not respond.) Based on our knowledge of how many products customer i has purchased, and easily they are annoyed, our marketing people have come up with two bounds $0 \leq c_i \leq c'_i$. We will ask the i th customer about at least c_i products they bought, but (to avoid annoying them) at most c'_i products.
- Again, our marketing people know that we want more information about some products (e.g., new releases) and less about others. To get a balanced amount of information about each product, for the j th product we have two bounds $0 \leq p_j \leq p'_j$, and we will ask at least p_j customers about this product and at most p'_j customers.

We can model this as a bipartite graph G , in which the customers form one of the parts of the network and products form the other part. There is an edge (i, j) if customer i has purchased product j . The flow through each customer node will reflect the number of products this customer is asked about. The flow through each product node will reflect the number of customers that are asked about this product.

This suggests the following network design. Given the bipartite graph G , we create a directed network as follows (see Fig. 74).

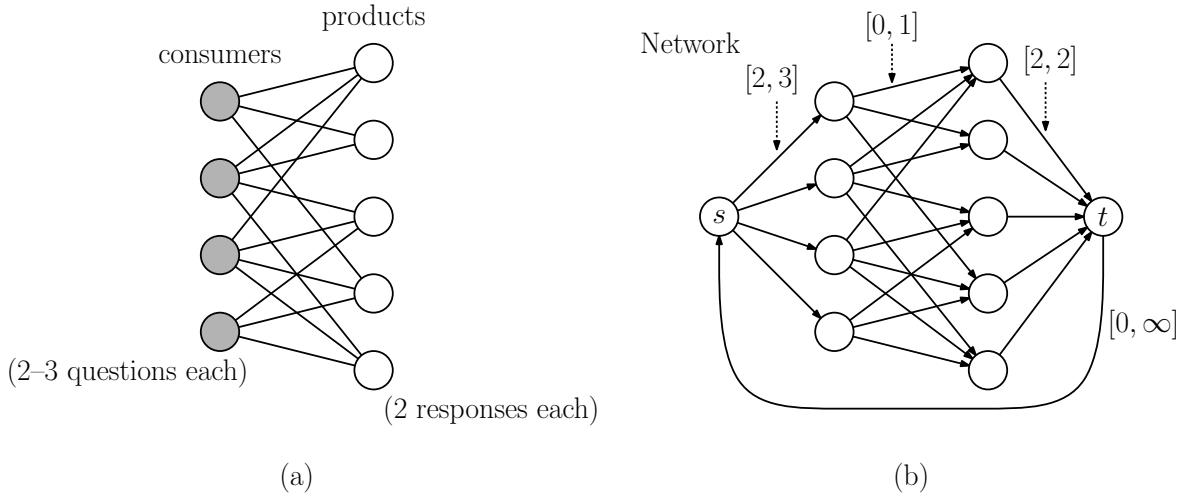


Fig. 74: Reducing the survey design problem to circulation with lower and upper flow bounds.

- For each customer i who purchased product j we create a directed edge (i, j) with an upper flow bounds of 1, respectively. This models the requirement that customer i will be surveyed at most once about product j , and customers will be asked only about products they purchased.
- We create a source vertex s and connect it to all the customers, where the edge from s to customer i has lower and upper flow bounds of c_i and c'_i , respectively. This models the requirement that customer i will be asked about at least c_i products and at most c'_i .
- We create a sink vertex t , and create an edge from product j to t with lower and upper flow bounds of p_j and p'_j . This models the requirement that there are at least p_j and at most p'_j customers will be asked about product j .
- We create an edge (s, t) . Its lower bound is set to zero and its upper bound can be set to any very large value. This is needed for technical reasons, since we want a circulation.
- All node demands are set to 0.

It is easy to see that if G has a valid (integer valued) circulation. There is a flow of one unit along edge (i, j) if customer i is surveyed about product j . From our capacity constraints, it follows that customer i receives somewhere between c_i and c'_i products to answer questions about, and each product j is asked about to between p_j and p'_j customers. Since the node demands are all 0, it follows that the flows through every vertex (including s and t) satisfy flow conservation. This implies that the total number of surveys sent to all the customers (the flow out of s) equals the total number of surveys received on all the products (the flow into t). The converse is also easy to show, namely that a valid survey design implies the existence of a circulation in G . Therefore, there exists a valid circulation in G' if and only there is a valid survey design (see Fig. 75).

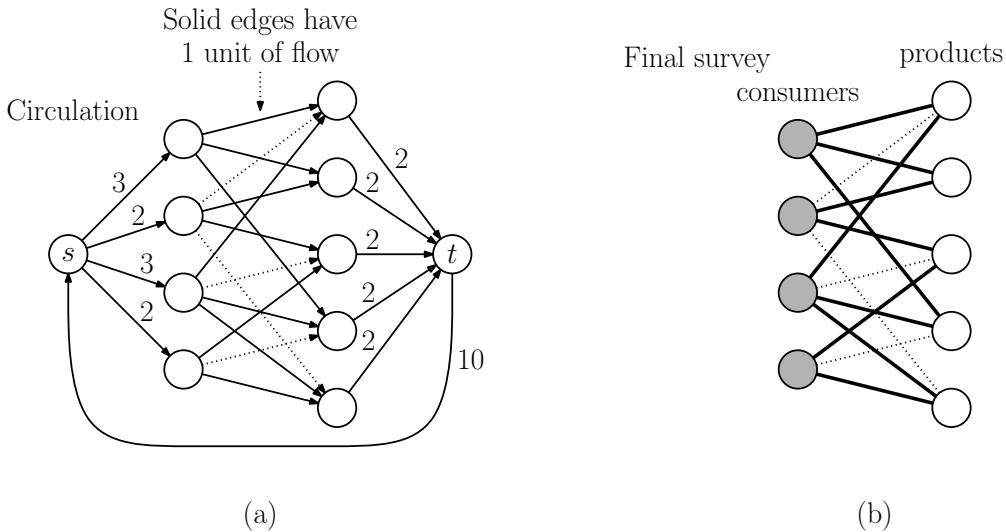


Fig. 75: Reducing the survey design problem to circulation with lower and upper flow bounds.

Lecture 21: NP-Completeness: General Definitions

Efficiency and Polynomial Time: Up to this point of the semester we have been building up your “bag of tricks” for solving algorithmic problems efficiently. Hopefully when presented with a problem you now have a little better idea of how to go about solving the problem. What sort of design paradigm should be used (divide-and-conquer, DFS, greedy, dynamic programming, etc.), what sort of data structures might be relevant (trees, priority queues, graphs) and what representations would be best (adjacency list, adjacency matrices), what is the running time of your algorithm.

The notion of what we mean by efficient is quite vague. If n is small, a running time of 2^n may be just fine, but when n is huge, even n^2 may be unacceptably slow. In an effort put matters on a clear mathematical basis, algorithm designers observed that there are two very general classes of combinatorial problems: those that can be solved by an intelligent search process and those that involve simple brute-force search. Since most combinatorial problems involve choosing from an exponential set of possibilities, the key distinguishing feature in most cases was whether there existed a *polynomial time algorithm* for solving the problem.

Recall that an algorithm is said to run in *polynomial time* if its worst-case running time is $O(n^c)$, where c is a nonnegative constant. (Note that running times like $O(n \log n)$ are also polynomial time, since $n \log n = O(n^2)$.) A computational problem is said to be solved *efficiently* if it is solvable in polynomial time. Higher worst-case running times, such as 2^n , $n!$, and n^n are not polynomial time.

You can't be serious! You would be quite right to object to this “definition” of efficiently solvable for a number of reasons. First off, if you are interested only in small values of n , a running time of 2^n with a small constant factor may be vastly superior to an algorithm that runs in $O(n^{20})$ and/or where the asymptotic notation hides huge constant factors. There are many problems for which good *average case* solutions exist, but the worst case complexity, which

5 Orthogonal Range Searching

Querying a Database

At first sight it seems that databases have little to do with geometry. Nevertheless, many types of questions—from now on called *queries*—about data in a database can be interpreted geometrically. To this end we transform records in a database into points in a multi-dimensional space, and we transform the queries about the records into queries on this set of points. Let’s demonstrate this with an example.

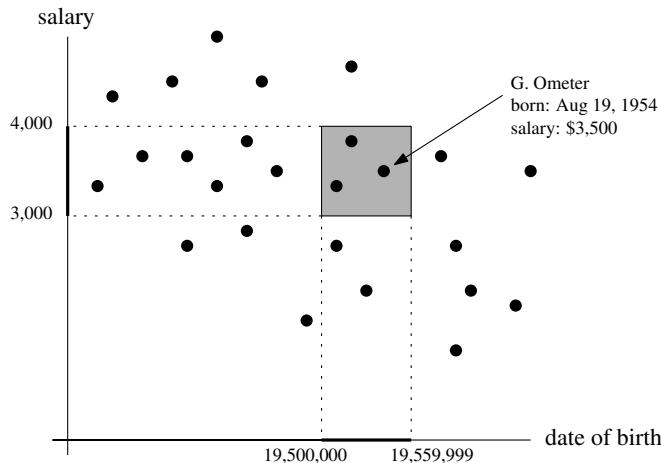
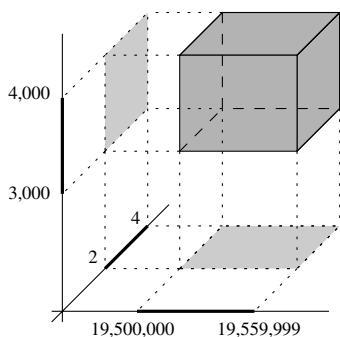


Figure 5.1
Interpreting a database query
geometrically

Consider a database for personnel administration. In such a database the name, address, date of birth, salary, and so on, of each employee are stored. A typical query one may want to perform is to report all employees born between 1950 and 1955 who earn between \$3,000 and \$4,000 a month. To formulate this as a geometric problem we represent each employee by a point in the plane. The first coordinate of the point is the date of birth, represented by the integer $10,000 \times year + 100 \times month + day$, and the second coordinate is the monthly salary. With the point we also store the other information we have about the employee, such as name and address. The database query asking for all employees born between 1950 and 1955 who earn between \$3,000 and



\$4,000 transforms into the following geometric query: report all points whose first coordinate lies between 19,500,000 and 19,559,999, and whose second coordinate lies between 3,000 and 4,000. In other words, we want to report all the points inside an axis-parallel query rectangle—see Figure 5.1.

What if we also have information about the number of children of each employee, and we would like to be able to ask queries like “report all employees born between 1950 and 1955 who earn between \$3,000 and \$4,000 a month and have between two and four children”? In this case we represent each employee by a point in 3-dimensional space: the first coordinate represents the date of birth, the second coordinate the salary, and the third coordinate the number of children. To answer the query we now have to report all points inside the axis-parallel box $[19,500,000 : 19,559,999] \times [3,000 : 4,000] \times [2 : 4]$. In general, if we are interested in answering queries on d fields of the records in our database, we transform the records to points in d -dimensional space. A query asking to report all records whose fields lie between specified values then transforms to a query asking for all points inside a d -dimensional axis-parallel box. Such a query is called a *rectangular range query*, or an *orthogonal range query*, in computational geometry. In this chapter we shall study data structures for such queries.

5.1 1-Dimensional Range Searching

Before we try to tackle the 2- or higher-dimensional rectangular range searching problem, let’s have a look at the 1-dimensional version. The data we are given is a set of points in 1-dimensional space—in other words, a set of real numbers. A query asks for the points inside a 1-dimensional query rectangle—in other words, an interval $[x : x']$.

Let $P := \{p_1, p_2, \dots, p_n\}$ be the given set of points on the real line. We can solve the 1-dimensional range searching problem efficiently using a well-known data structure: a balanced binary search tree \mathcal{T} . (A solution that uses an array is also possible. This solution does not generalize to higher dimensions, however, nor does it allow for efficient updates on P .) The leaves of \mathcal{T} store the points of P and the internal nodes of \mathcal{T} store splitting values to guide the search. We denote the splitting value stored at a node v by x_v . We assume that the left subtree of a node v contains all the points smaller than or equal to x_v , and that the right subtree contains all the points strictly greater than x_v .

To report the points in a query range $[x : x']$ we proceed as follows. We search with x and x' in \mathcal{T} . Let μ and μ' be the two leaves where the searches end, respectively. Then the points in the interval $[x : x']$ are the ones stored in the leaves in between μ and μ' plus, possibly, the point stored at μ and the point stored at μ' . When we search with the interval $[18 : 77]$ in the tree of Figure 5.2, for instance, we have to report all the points stored in the dark grey leaves, plus the point stored in the leaf μ . How can we find the leaves in between μ and μ' ? As Figure 5.2 already suggests, they are the leaves of certain subtrees in between the search paths to μ and μ' . (In Figure 5.2, these subtrees are dark

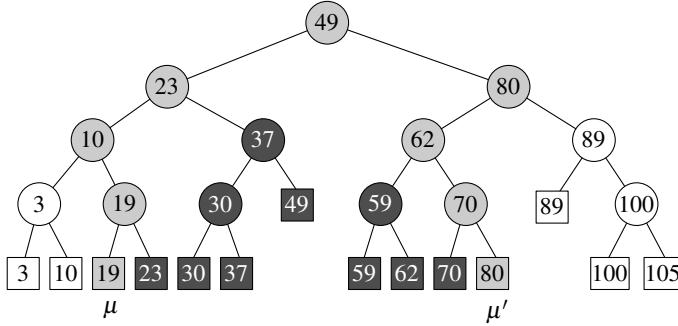


Figure 5.2
A 1-dimensional range query in a binary search tree

grey, whereas the nodes on the search paths are light grey.) More precisely, the subtrees that we select are rooted at nodes v in between the two search paths whose parents are on the search path. To find these nodes we first search for the node v_{split} where the paths to x and x' split. This is done with the following subroutine. Let $lc(v)$ and $rc(v)$ denote the left and right child, respectively, of a node v .

FINDSPLITNODE(\mathcal{T}, x, x')

Input. A tree \mathcal{T} and two values x and x' with $x \leq x'$.

Output. The node v where the paths to x and x' split, or the leaf where both paths end.

1. $v \leftarrow \text{root}(\mathcal{T})$
2. **while** v is not a leaf **and** ($x' \leq x_v$ **or** $x > x_v$)
3. **do if** $x' \leq x_v$
4. **then** $v \leftarrow lc(v)$
5. **else** $v \leftarrow rc(v)$
6. **return** v

Starting from v_{split} we then follow the search path of x . At each node where the path goes left, we report all the leaves in the right subtree, because this subtree is in between the two search paths. Similarly, we follow the path of x' and we report the leaves in the left subtree of nodes where the path goes right. Finally, we have to check the points stored at the leaves where the paths end; they may or may not lie in the range $[x : x']$.

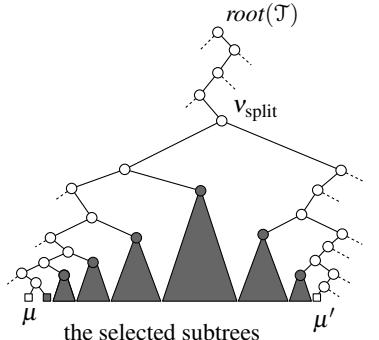
Next we describe the query algorithm in more detail. It uses a subroutine REPORTSUBTREE, which traverses the subtree rooted at a given node and reports the points stored at its leaves. Since the number of internal nodes of any binary tree is less than its number of leaves, this subroutine takes an amount of time that is linear in the number of reported points.

Algorithm 1DRANGEQUERY($\mathcal{T}, [x : x']$)

Input. A binary search tree \mathcal{T} and a range $[x : x']$.

Output. All points stored in \mathcal{T} that lie in the range.

1. $v_{\text{split}} \leftarrow \text{FINDSPLITNODE}(\mathcal{T}, x, x')$
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported.



```

4.   else (* Follow the path to  $x$  and report the points in subtrees right of the
      path. *)
5.      $v \leftarrow lc(v_{\text{split}})$ 
6.     while  $v$  is not a leaf
7.       do if  $x \leqslant x_v$ 
8.         then REPORTSUBTREE( $rc(v)$ )
9.          $v \leftarrow lc(v)$ 
10.        else  $v \leftarrow rc(v)$ 
11.     Check if the point stored at the leaf  $v$  must be reported.
12.     Similarly, follow the path to  $x'$ , report the points in subtrees left of
        the path, and check if the point stored at the leaf where the path
        ends must be reported.

```

We first prove the correctness of the algorithm.

Lemma 5.1 *Algorithm 1DRANGEQUERY reports exactly those points that lie in the query range.*

Proof. We first show that any reported point p lies in the query range. If p is stored at the leaf where the path to x or to x' ends, then p is tested explicitly for inclusion in the query range. Otherwise, p is reported in a call to REPORTSUBTREE. Assume this call was made when we followed the path to x . Let v be the node on the path such that p was reported in the call REPORTSUBTREE($rc(v)$). Since v and, hence, $rc(v)$ lie in the left subtree of v_{split} , we have $p \leqslant x_{v_{\text{split}}}$. Because the search path of x' goes right at v_{split} this means that $p < x'$. On the other hand, the search path of x goes left at v and p is in the right subtree of v , so $x < p$. It follows that $p \in [x : x']$. The proof that p lies in the range when it is reported while following the path to x' is symmetrical.

It remains to prove that any point p in the range is reported. Let μ be the leaf where p is stored, and let v be the lowest ancestor of μ that is visited by the query algorithm. We claim that $v = \mu$, which implies that p is reported. Assume for a contradiction that $v \neq \mu$. Observe that v cannot be a node visited in a call to REPORTSUBTREE, because all descendants of such a node are visited. Hence, v is either on the search path to x , or on the search path to x' , or both. Because all three cases are similar, we only consider the third case. Assume first that μ is in the left subtree of v . Then the search path of x goes right at v (otherwise v would not be the lowest visited ancestor). But this implies that $p < x$. Similarly, if μ is in the right subtree of v , then the path of x' goes left at v , and $p > x'$. In both cases, the assumption that p lies in the range is contradicted. \square

We now turn our attention to the performance of the data structure. Because it is a balanced binary search tree, it uses $O(n)$ storage and it can be built in $O(n \log n)$ time. What about the query time? In the worst case all the points could be in the query range. In this case the query time will be $\Theta(n)$, which seems bad. Indeed, we do not need any data structure to achieve $\Theta(n)$ query time; simply checking all the points against the query range leads to the same result. On the other hand, a query time of $\Theta(n)$ cannot be avoided when we have to report all the points. Therefore we shall give a more refined analysis

of the query time. The refined analysis takes not only n , the number of points in the set P , into account, but also k , the number of reported points. In other words, we will show that the query algorithm is *output-sensitive*, a concept we already encountered in Chapter 2.

Recall that the time spent in a call to REPORTSUBTREE is linear in the number of reported points. Hence, the total time spent in all such calls is $O(k)$. The remaining nodes that are visited are nodes on the search path of x or x' . Because \mathcal{T} is balanced, these paths have length $O(\log n)$. The time we spend at each node is $O(1)$, so the total time spent in these nodes is $O(\log n)$, which gives a query time of $O(\log n + k)$.

The following theorem summarizes the results for 1-dimensional range searching:

Theorem 5.2 *Let P be a set of n points in 1-dimensional space. The set P can be stored in a balanced binary search tree, which uses $O(n)$ storage and has $O(n \log n)$ construction time, such that the points in a query range can be reported in time $O(k + \log n)$, where k is the number of reported points.*

5.2 Kd-Trees

Now let's go to the 2-dimensional rectangular range searching problem. Let P be a set of n points in the plane. In the remainder of this section we assume that no two points in P have the same x -coordinate, and no two points have the same y -coordinate. This restriction is not very realistic, especially not if the points represent employees and the coordinates are things like salary or number of children. Fortunately, the restriction can be overcome with a nice trick that we describe in Section 5.5.

A 2-dimensional rectangular range query on P asks for the points from P lying inside a query rectangle $[x : x'] \times [y : y']$. A point $p := (p_x, p_y)$ lies inside this rectangle if and only if

$$p_x \in [x : x'] \quad \text{and} \quad p_y \in [y : y'].$$

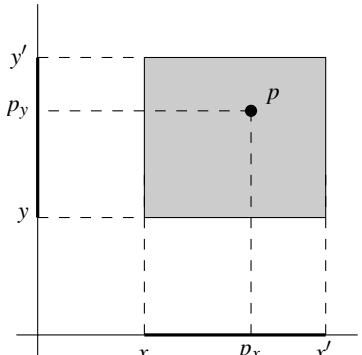
We could say that a 2-dimensional rectangular range query is composed of two 1-dimensional sub-queries, one on the x -coordinate of the points and one on the y -coordinate.

In the previous section we saw a data structure for 1-dimensional range queries. How can we generalize this structure—which was just a binary search tree—to 2-dimensional range queries? Let's consider the following recursive definition of the binary search tree: the set of (1-dimensional) points is split into two subsets of roughly equal size; one subset contains the points smaller than or equal to the splitting value, the other subset contains the points larger than the splitting value. The splitting value is stored at the root, and the two subsets are stored recursively in the two subtrees.

In the 2-dimensional case each point has two values that are important: its x - and its y -coordinate. Therefore we first split on x -coordinate, next on

Section 5.2

KD-TREES



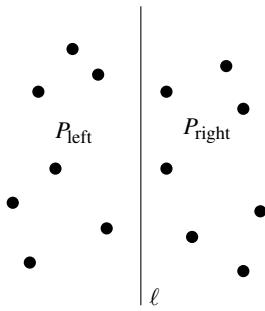
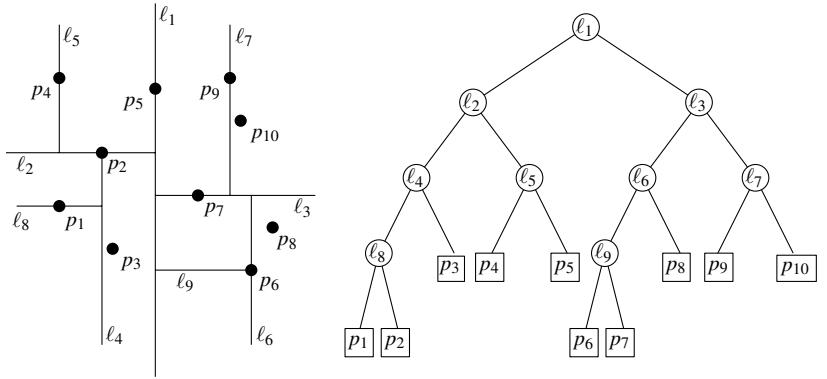


Figure 5.3

A kd-tree: on the left the way the plane is subdivided and on the right the corresponding binary tree



stood for k -dimensional tree; the tree we described above would be a 2d-tree. Nowadays, the original meaning is lost, and what used to be called a 2d-tree is now called a 2-dimensional kd-tree.

We can construct a kd-tree with the recursive procedure described below. This procedure has two parameters: a set of points and an integer. The first parameter is the set for which we want to build the kd-tree; initially this is the set P . The second parameter is depth of recursion or, in other words, the depth of the root of the subtree that the recursive call constructs. The depth parameter is zero at the first call. The depth is important because, as explained above, it determines whether we must split with a vertical or a horizontal line. The procedure returns the root of the kd-tree.

Algorithm BUILDKDTREE(P, depth)

Input. A set of points P and the current depth depth .

Output. The root of a kd-tree storing P .

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** depth is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x -coordinate of the points in P . Let P_1 be the set of

- points to the left of ℓ or on ℓ , and let P_2 be the set of points to the right of ℓ .
5. **else** Split P into two subsets with a horizontal line ℓ through the median y -coordinate of the points in P . Let P_1 be the set of points below ℓ or on ℓ , and let P_2 be the set of points above ℓ .
 6. $v_{\text{left}} \leftarrow \text{BUILDKDTREE}(P_1, depth + 1)$
 7. $v_{\text{right}} \leftarrow \text{BUILDKDTREE}(P_2, depth + 1)$
 8. Create a node v storing ℓ , make v_{left} the left child of v , and make v_{right} the right child of v .
 9. **return** v

The algorithm uses the convention that the point on the splitting line—the one determining the median x - or y -coordinate—belongs to the subset to the left of, or below, the splitting line. For this to work correctly, the median of a set of n numbers should be defined as the $\lceil n/2 \rceil$ -th smallest number. This means that the median of two values is the smaller one, which ensures that the algorithm terminates.

Before we come to the query algorithm, let's analyze the construction time of a 2-dimensional kd-tree. The most expensive step that is performed at every recursive call is finding the splitting line. This requires determining the median x -coordinate or the median y -coordinate, depending on whether the depth is even or odd. Median finding can be done in linear time. Linear time median finding algorithms, however, are rather complicated. A better approach is to presort the set of points both on x - and on y -coordinate. The parameter set P is now passed to the procedure in the form of two sorted lists, one on x -coordinate and one on y -coordinate. Given the two sorted lists, it is easy to find the median x -coordinate (when the depth is even) or the median y -coordinate (when the depth is odd) in linear time. It is also easy to construct the sorted lists for the two recursive calls in linear time from the given lists. Hence, the building time $T(n)$ satisfies the recurrence

$$T(n) = \begin{cases} O(1), & \text{if } n = 1, \\ O(n) + 2T(\lceil n/2 \rceil), & \text{if } n > 1, \end{cases}$$

which solves to $O(n \log n)$. This bound subsumes the time we spend for presorting the points on x - and y -coordinate.

To bound the amount of storage we note that each leaf in the kd-tree stores a distinct point of P . Hence, there are n leaves. Because a kd-tree is a binary tree, and every leaf and internal node uses $O(1)$ storage, this implies that the total amount of storage is $O(n)$. This leads to the following lemma.

Lemma 5.3 *A kd-tree for a set of n points uses $O(n)$ storage and can be constructed in $O(n \log n)$ time.*

We now turn to the query algorithm. The splitting line stored at the root partitions the plane into two half-planes. The points in the left half-plane are stored in the left subtree, and the points in the right half-plane are stored in the

right subtree. In a sense, the left child of the root corresponds to the left half-plane and the right child corresponds to the right half-plane. (The convention used in `BUILDKDTREE` that the point on the splitting line belongs to the left subset implies that the left half-plane is closed to the right and the right half-plane is open to the left.) The other nodes in a kd-tree correspond to a region of the plane as well. The left child of the left child of the root, for instance, corresponds to the region bounded to the right by the splitting line stored at the root and bounded from above by the line stored at the left child of the root. In general, the region corresponding to a node v is a rectangle, which can be unbounded on one or more sides. It is bounded by splitting lines stored at ancestors of v —see Figure 5.4. We denote the region corresponding to a node

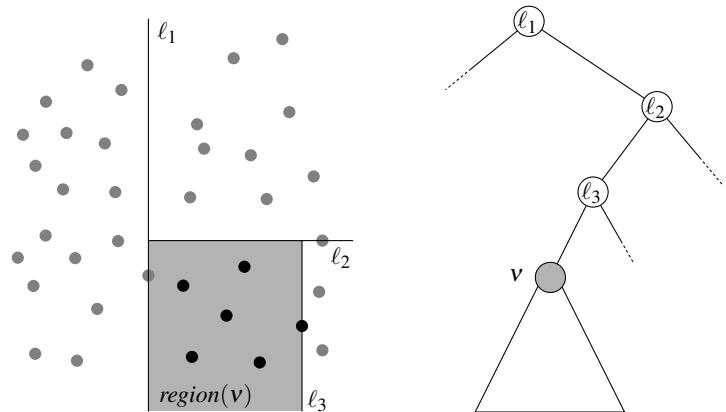


Figure 5.4

Correspondence between nodes in a kd-tree and regions in the plane

v by $region(v)$. The region of the root of a kd-tree is simply the whole plane. Observe that a point is stored in the subtree rooted at a node v if and only if it lies in $region(v)$. For instance, the subtree of the node v in Figure 5.4 stores the points indicated as black dots. Therefore we have to search the subtree rooted at v only if the query rectangle intersects $region(v)$. This observation leads to the following query algorithm: we traverse the kd-tree, but visit only nodes whose region is intersected by the query rectangle. When a region is fully contained in the query rectangle, we can report all the points stored in its subtree. When the traversal reaches a leaf, we have to check whether the point stored at the leaf is contained in the query region and, if so, report it. Figure 5.5 illustrates the query algorithm. (Note that the kd-tree of Figure 5.5 could not have been constructed by Algorithm `BUILDKDTREE`; the median wasn't always chosen as the split value.) The grey nodes are visited when we query with the grey rectangle. The node marked with a star corresponds to a region that is completely contained in the query rectangle; in the figure this rectangular region is shown darker. Hence, the dark grey subtree rooted at this node is traversed and all points stored in it are reported. The other leaves that are visited correspond to regions that are only partially inside the query rectangle. Hence, the points stored in them must be tested for inclusion in the query range; this results in points p_6 and p_{11} being reported, and points p_3 , p_{12} , and p_{13} not being reported. The query algorithm is described by the following recursive

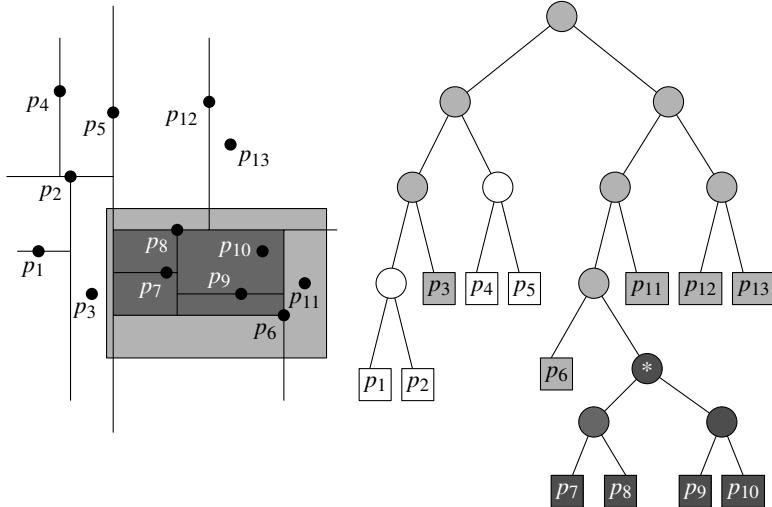


Figure 5.5
A query on a kd-tree

procedure, which takes as arguments the root of a kd-tree and the query range R . It uses a subroutine REPORTSUBTREE(v), which traverses the subtree rooted at a node v and reports all the points stored at its leaves. Recall that $lc(v)$ and $rc(v)$ denote the left and right child of a node v , respectively.

Algorithm SEARCHKDTREE(v, R)

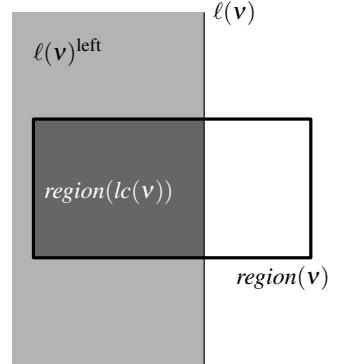
Input. The root of (a subtree of) a kd-tree, and a range R .
Output. All points at leaves below v that lie in the range.

1. **if** v is a leaf
 2. **then** Report the point stored at v if it lies in R .
 3. **else if** $region(lc(v))$ is fully contained in R
 4. **then** REPORTSUBTREE($lc(v)$)
 5. **else if** $region(lc(v))$ intersects R
 6. **then** SEARCHKDTREE($lc(v), R$)
 7. **if** $region(rc(v))$ is fully contained in R
 8. **then** REPORTSUBTREE($rc(v)$)
 9. **else if** $region(rc(v))$ intersects R
 10. **then** SEARCHKDTREE($rc(v), R$)

The main test the query algorithm performs is whether the query range R intersects the region corresponding to some node v . To be able to do this test we can compute $region(v)$ for all nodes v during the preprocessing phase and store it, but this is not necessary: one can maintain the current region through the recursive calls using the lines stored in the internal nodes. For instance, the region corresponding to the left child of a node v at even depth can be computed from $region(v)$ as follows:

$$region(lc(v)) = region(v) \cap \ell(v)^{\text{left}},$$

where $\ell(v)$ is the splitting line stored at v , and $\ell(v)^{\text{left}}$ is the half-plane to the left of and including $\ell(v)$.



Observe that the query algorithm above never assumes that the query range R is a rectangle. Indeed, it works for any other query range as well.

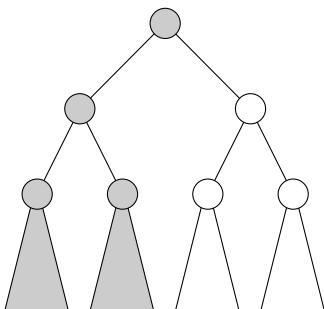
We now analyze the time a query with a rectangular range takes.

Lemma 5.4 *A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in $O(\sqrt{n} + k)$ time, where k is the number of reported points.*

Proof. First of all, note that the time to traverse a subtree and report the points stored in its leaves is linear in the number of reported points. Hence, the total time required for traversing subtrees in steps 4 and 8 is $O(k)$, where k is the total number of reported points. It remains to bound the number of nodes visited by the query algorithm that are not in one of the traversed subtrees. (These are the light grey nodes in Figure 5.5.) For each such node v , the query range properly intersects $\text{region}(v)$, that is, $\text{region}(v)$ is intersected by, but not fully contained in the range. In other words, the boundary of the query range intersects $\text{region}(v)$. To analyze the number of such nodes, we shall bound the number of regions intersected by any vertical line. This will give us an upper bound on the number of regions intersected by the left and right edge of the query rectangle. The number of regions intersected by the bottom and top edges of the query range can be bounded in the same way.

Let ℓ be a vertical line, and let \mathcal{T} be a kd-tree. Let $\ell(\text{root}(\mathcal{T}))$ be the splitting line stored at the root of the kd-tree. The line ℓ intersects either the region to the left of $\ell(\text{root}(\mathcal{T}))$ or the region to the right of $\ell(\text{root}(\mathcal{T}))$, but not both. This observation seems to imply that $Q(n)$, the number of intersected regions in a kd-tree storing a set of n points, satisfies the recurrence $Q(n) = 1 + Q(n/2)$. But this is not true, because the splitting lines are horizontal at the children of the root. This means that if the line ℓ intersects for instance $\text{region}(\text{lc}(\text{root}(\mathcal{T})))$, then it will always intersect the regions corresponding to both children of $\text{lc}(\text{root}(\mathcal{T}))$. Hence, the recursive situation we get is not the same as the original situation, and the recurrence above is incorrect. To overcome this problem we have to make sure that the recursive situation is exactly the same as the original situation: the root of the subtree must contain a vertical splitting line. This leads us to redefine $Q(n)$ as the number of intersected regions in a kd-tree storing n points whose root contains a vertical splitting line. To write a recurrence for $Q(n)$ we now have to go down two steps in the tree. Each of the four nodes at depth two in the tree corresponds to a region containing $n/4$ points. (To be precise, a region can contain at most $\lceil \lceil n/2 \rceil / 2 \rceil = \lceil n/4 \rceil$ points, but asymptotically this does not influence the outcome of the recurrence below.) Two of the four nodes correspond to intersected regions, so we have to count the number of intersected regions in these subtrees recursively. Moreover, ℓ intersects the region of the root and of one of its children. Hence, $Q(n)$ satisfies the recurrence

$$Q(n) = \begin{cases} O(1), & \text{if } n = 1, \\ 2 + 2Q(n/4), & \text{if } n > 1. \end{cases}$$



This recurrence solves to $Q(n) = O(\sqrt{n})$. In other words, any vertical line intersects $O(\sqrt{n})$ regions in a kd-tree. In a similar way one can prove that the

total number of regions intersected by a horizontal line is $O(\sqrt{n})$. The total number of regions intersected by the boundary of a rectangular query range is bounded by $O(\sqrt{n})$ as well. \square

Section 5.3
RANGE TREES

The analysis of the query time that we gave above is rather pessimistic: we bounded the number of regions intersecting an edge of the query rectangle by the number of regions intersecting the line through it. In many practical situations the range will be small. As a result, the edges are short and will intersect much fewer regions. For example, when we search with a range $[x : x] \times [y : y]$ —this query effectively asks whether the point (x, y) is in the set—the query time is bounded by $O(\log n)$.

The following theorem summarizes the performance of kd-trees.

Theorem 5.5 A kd-tree for a set P of n points in the plane uses $O(n)$ storage and can be built in $O(n \log n)$ time. A rectangular range query on the kd-tree takes $O(\sqrt{n} + k)$ time, where k is the number of reported points.

Kd-trees can also be used for point sets in 3- or higher-dimensional space. The construction algorithm is very similar to the planar case: At the root, we split the set of points into two subsets of roughly the same size by a hyperplane perpendicular to the x_1 -axis. In other words, at the root the point set is partitioned based on the first coordinate of the points. At the children of the root the partition is based on the second coordinate, at nodes at depth two on the third coordinate, and so on, until at depth $d - 1$ we partition on the last coordinate. At depth d we start all over again, partitioning on first coordinate. The recursion stops when there is only one point left, which is then stored at a leaf. Because a d -dimensional kd-tree for a set of n points is a binary tree with n leaves, it uses $O(n)$ storage. The construction time is $O(n \log n)$. (As usual, we assume d to be a constant.)

Nodes in a d -dimensional kd-tree correspond to regions, as in the plane. The query algorithm visits those nodes whose regions are properly intersected by the query range, and traverses subtrees (to report the points stored in the leaves) that are rooted at nodes whose region is fully contained in the query range. It can be shown that the query time is bounded by $O(n^{1-1/d} + k)$.

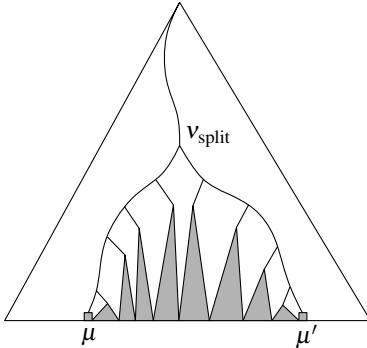
5.3 Range Trees

Kd-trees, which were described in the previous section, have $O(\sqrt{n} + k)$ query time. So when the number of reported points is small, the query time is relatively high. In this section we shall describe another data structure for rectangular range queries, the *range tree*, which has a better query time, namely $O(\log^2 n + k)$. The price we have to pay for this improvement is an increase in storage from $O(n)$ for kd-trees to $O(n \log n)$ for range trees.

As we observed before, a 2-dimensional range query is essentially composed of two 1-dimensional sub-queries, one on the x -coordinate of the points and one

on the y -coordinate. This gave us the idea to split the given point set alternately on x - and y -coordinate, leading to the kd-tree. To obtain the range tree, we shall use this observation in a different way.

Let P be a set of n points in the plane that we want to preprocess for rectangular range queries. Let $[x : x'] \times [y : y']$ be the query range. We first concentrate on finding the points whose x -coordinate lies in $[x : x']$, the x -interval of the query rectangle, and worry about the y -coordinate later. If we only care about the x -coordinate then the query is a 1-dimensional range query. In Section 5.1 we have seen how to answer such a query: with a binary search tree on the x -coordinate of the points. The query algorithm was roughly as follows. We search with x and x' in the tree until we get to a node v_{split} where the search paths split. From the left child of v_{split} we continue the search with x , and at every node v where the search path of x goes left, we report all points in the right subtree of v . Similarly, we continue the search with x' at the right child of v_{split} , and at every node v where the search path of x' goes right we report all points in the left subtree of v . Finally, we check the leaves μ and μ' where the two paths end to see if they contain a point in the range. In effect, we select a collection of $O(\log n)$ subtrees that together contain exactly the points whose x -coordinate lies in the x -interval of the query rectangle.



Let's call the subset of points stored in the leaves of the subtree rooted at a node v the *canonical subset* of v . The canonical subset of the root of the tree, for instance, is the whole set P . The canonical subset of a leaf is simply the point stored at that leaf. We denote the canonical subset of node v by $P(v)$. We have just seen that the subset of points whose x -coordinate lies in a query range can be expressed as the disjoint union of $O(\log n)$ canonical subsets; these are the sets $P(v)$ of the nodes v that are the roots of the selected subtrees. We are not interested in all the points in such a canonical subset $P(v)$, but only want to report the ones whose y -coordinate lies in the interval $[y : y']$. This is another 1-dimensional query, which we can solve, provided we have a binary search tree on the y -coordinate of the points in $P(v)$ available. This leads to the following data structure for rectangular range queries on a set P of n points in the plane.

- The main tree is a balanced binary search tree \mathcal{T} built on the x -coordinate of the points in P .
- For any internal or leaf node v in \mathcal{T} , the canonical subset $P(v)$ is stored in a balanced binary search tree $\mathcal{T}_{\text{assoc}}(v)$ on the y -coordinate of the points. The node v stores a pointer to the root of $\mathcal{T}_{\text{assoc}}(v)$, which is called the *associated structure* of v .

This data structure is called a range tree. Figure 5.6 shows the structure of a range tree. Data structures where nodes have pointers to associated structures are often called *multi-level data structures*. The main tree \mathcal{T} is then called the *first-level tree*, and the associated structures are *second-level trees*. Multi-level data structures play an important role in computational geometry; more examples can be found in Chapters 10 and 16.

A range tree can be constructed with the following recursive algorithm, which receives as input the set $P := \{p_1, \dots, p_n\}$ of points sorted on x -coordinate and

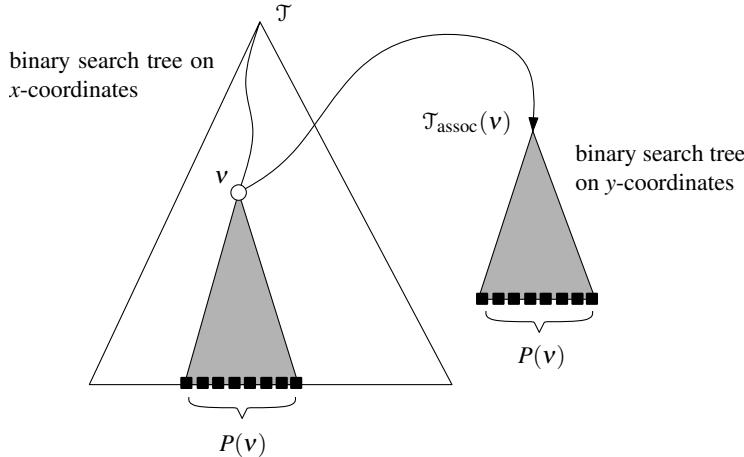


Figure 5.6
A 2-dimensional range tree

returns the root of a 2-dimensional range tree \mathcal{T} of P . As in the previous section, we assume that no two points have the same x - or y -coordinate. We shall get rid of this assumption in Section 5.5.

Algorithm BUILD2DRANGETREE(P)

Input. A set P of points in the plane.

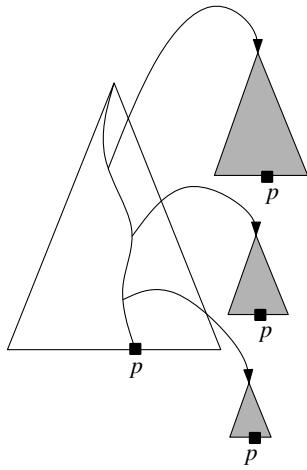
Output. The root of a 2-dimensional range tree.

1. Construct the associated structure: Build a binary search tree $\mathcal{T}_{\text{assoc}}$ on the set P_y of y -coordinates of the points in P . Store at the leaves of $\mathcal{T}_{\text{assoc}}$ not just the y -coordinate of the points in P_y , but the points themselves.
2. **if** P contains only one point
3. **then** Create a leaf v storing this point, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v .
4. **else** Split P into two subsets; one subset P_{left} contains the points with x -coordinate less than or equal to x_{mid} , the median x -coordinate, and the other subset P_{right} contains the points with x -coordinate larger than x_{mid} .
5. $v_{\text{left}} \leftarrow \text{BUILD2DRANGETREE}(P_{\text{left}})$
6. $v_{\text{right}} \leftarrow \text{BUILD2DRANGETREE}(P_{\text{right}})$
7. Create a node v storing x_{mid} , make v_{left} the left child of v , make v_{right} the right child of v , and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v .
8. **return** v

Note that in the leaves of the associated structures we do not just store the y -coordinate of the points but the points themselves. This is important because, when searching the associated structures, we need to report the points and not just the y -coordinates.

Lemma 5.6 A range tree on a set of n points in the plane requires $O(n \log n)$ storage.

Proof. A point p in P is stored only in the associated structure of nodes on the path in \mathcal{T} towards the leaf containing p . Hence, for all nodes at a given depth of \mathcal{T} ,



the point p is stored in exactly one associated structure. Because 1-dimensional range trees use linear storage it follows that the associated structures of all nodes at any depth of \mathcal{T} together use $O(n)$ storage. The depth of \mathcal{T} is $O(\log n)$. Hence, the total amount of storage required is bounded by $O(n \log n)$. \square

Algorithm BUILD2DRANGETREE as it is described will not result in the optimal construction time of $O(n \log n)$. To obtain this we have to be a bit careful. Constructing a binary search tree on an unsorted set of n keys takes $O(n \log n)$ time. This means that constructing the associated structure in line 1 would take $O(n \log n)$ time. But we can do better if the points in P_y are presorted on y -coordinate; then the binary search tree can be constructed bottom-up in linear time. During the construction algorithm we therefore maintain the set of points in two lists, one sorted on x -coordinate and one sorted on y -coordinate. This way the time we spend at a node in the main tree \mathcal{T} is linear in the size of its canonical subset. This implies that the total construction time is the same as the amount of storage, namely $O(n \log n)$. Since the presorting takes $O(n \log n)$ time as well, the total construction time is again $O(n \log n)$.

The query algorithm first selects $O(\log n)$ canonical subsets that together contain the points whose x -coordinate lie in the range $[x : x']$. This can be done with the 1-dimensional query algorithm. Of those subsets, we then report the points whose y -coordinate lie in the range $[y : y']$. For this we also use the 1-dimensional query algorithm; this time it is applied to the associated structures that store the selected canonical subsets. Thus the query algorithm is virtually the same as the 1-dimensional query algorithm 1DRANGEQUERY; the only difference is that calls to REPORTSUBTREE are replaced by calls to 1DRANGEQUERY.

Algorithm 2DRANGEQUERY($\mathcal{T}, [x : x'] \times [y : y']$)

Input. A 2-dimensional range tree \mathcal{T} and a range $[x : x'] \times [y : y']$.

Output. All points in \mathcal{T} that lie in the range.

1. $v_{\text{split}} \leftarrow \text{FINDSPLITNODE}(\mathcal{T}, x, x')$
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported.
4. **else** (* Follow the path to x and call 1DRANGEQUERY on the subtrees right of the path. *)
5. $v \leftarrow lc(v_{\text{split}})$
6. **while** v is not a leaf
7. **do if** $x \leqslant x_v$
8. **then** 1DRANGEQUERY($\mathcal{T}_{\text{assoc}}(rc(v)), [y : y']$)
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$
11. Check if the point stored at v must be reported.
12. Similarly, follow the path from $rc(v_{\text{split}})$ to x' , call 1DRANGEQUERY with the range $[y : y']$ on the associated structures of subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.

Lemma 5.7 A query with an axis-parallel rectangle in a range tree storing n points takes $O(\log^2 n + k)$ time, where k is the number of reported points.

Proof. At each node v in the main tree \mathcal{T} we spend constant time to decide where the search path continues, and we possibly call 1DRANGEQUERY. Theorem 5.2 states that the time we spend in this recursive call is $O(\log n + k_v)$, where k_v is the number of points reported in this call. Hence, the total time we spend is

$$\sum_v O(\log n + k_v),$$

where the summation is over all nodes in the main tree \mathcal{T} that are visited. Notice that the sum $\sum_v k_v$ equals k , the total number of reported points. Furthermore, the search paths of x and x' in the main tree \mathcal{T} have length $O(\log n)$. Hence, $\sum_v O(\log n) = O(\log^2 n)$. The lemma follows. \square

The following theorem summarizes the performance of 2-dimensional range trees.

Theorem 5.8 Let P be a set of n points in the plane. A range tree for P uses $O(n \log n)$ storage and can be constructed in $O(n \log n)$ time. By querying this range tree one can report the points in P that lie in a rectangular query range in $O(\log^2 n + k)$ time, where k is the number of reported points.

The query time stated in Theorem 5.8 can be improved to $O(\log n + k)$ by a technique called *fractional cascading*. This is described in Section 5.6.

5.4 Higher-Dimensional Range Trees

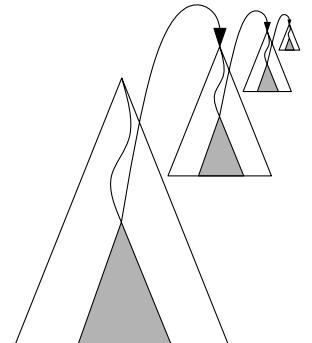
It is fairly straightforward to generalize 2-dimensional range trees to higher-dimensional range trees. We only describe the global approach.

Let P be a set of points in d -dimensional space. We construct a balanced binary search tree on the first coordinate of the points. The canonical subset $P(v)$ of a node v in this first-level tree, the main tree, consists of the points stored in the leaves of the subtree rooted at v . For each node v we construct an associated structure $\mathcal{T}_{\text{assoc}}(v)$; the second-level tree $\mathcal{T}_{\text{assoc}}(v)$ is a $(d - 1)$ -dimensional range tree for the points in $P(v)$, restricted to their last $d - 1$ coordinates. This $(d - 1)$ -dimensional range tree is constructed recursively in the same way: it is a balanced binary search tree on the second coordinate of the points, in which each node has a pointer to a $(d - 2)$ -dimensional range tree of the points in its subtree, restricted to the last $(d - 2)$ coordinates. The recursion stops when we are left with points restricted to their last coordinate; these are stored in a 1-dimensional range tree—a balanced binary search tree.

The query algorithm is also very similar to the 2-dimensional case. We use the first-level tree to locate $O(\log n)$ nodes whose canonical subsets together contain all the points whose first coordinates are in the correct range. These canonical subsets are queried further by performing a range query on the corresponding second-level structures. In each second-level structure we select

Section 5.4

HIGHER-DIMENSIONAL RANGE TREES



$O(\log n)$ canonical subsets. This means there are $O(\log^2 n)$ canonical subsets in the second-level structures in total. Together, they contain all points whose first and second coordinate lie in the correct ranges. The third-level structures storing these canonical subsets are then queried with the range for the third coordinate, and so on, until we reach the 1-dimensional trees. In these trees we find the points whose last coordinate lies in the correct range and report them. This approach leads to the following result.

Theorem 5.9 *Let P be a set of n points in d -dimensional space, where $d \geq 2$. A range tree for P uses $O(n \log^{d-1} n)$ storage and it can be constructed in $O(n \log^{d-1} n)$ time. One can report the points in P that lie in a rectangular query range in $O(\log^d n + k)$ time, where k is the number of reported points.*

Proof. Let $T_d(n)$ denote the construction time for a range tree on a set of n points in d -dimensional space. By Theorem 5.8 we know that $T_2(n) = O(n \log n)$. The construction of a d -dimensional range tree consists of building a balanced binary search tree, which takes time $O(n \log n)$, and the construction of associated structures. At the nodes at any depth of the first-level tree, each point is stored in exactly one associated structure. The time required to build all associated structures of the nodes at some depth is $O(T_{d-1}(n))$, the time required to build the associated structure of the root. This follows because the building time is at least linear. Hence, the total construction time satisfies

$$T_d(n) = O(n \log n) + O(\log n) \cdot T_{d-1}(n).$$

Since $T_2(n) = O(n \log n)$, this recurrence solves to $O(n \log^{d-1} n)$. The bound on the amount of storage follows in the same way.

Let $Q_d(n)$ denote the time spent in querying a d -dimensional range tree on n points, not counting the time to report points. Querying the d -dimensional range tree involves searching in a first-level tree, which takes time $O(\log n)$, and querying a logarithmic number of $(d - 1)$ -dimensional range trees. Hence,

$$Q_d(n) = O(\log n) + O(\log n) \cdot Q_{d-1}(n),$$

where $Q_2(n) = O(\log^2 n)$. This recurrence easily solves to $Q_d(n) = O(\log^d n)$. We still have to add the time needed to report points, which is bounded by $O(k)$. The bound on the query time follows. \square

As in the 2-dimensional case, the query time can be improved by a logarithmic factor—see Section 5.6.

5.5 General Sets of Points

Until now we imposed the restriction that no two points have equal x - or y -coordinate, which is highly unrealistic. Fortunately, this is easy to remedy. The crucial observation is that we never assumed the coordinate values to be real numbers. We only need that they come from a totally ordered universe, so that

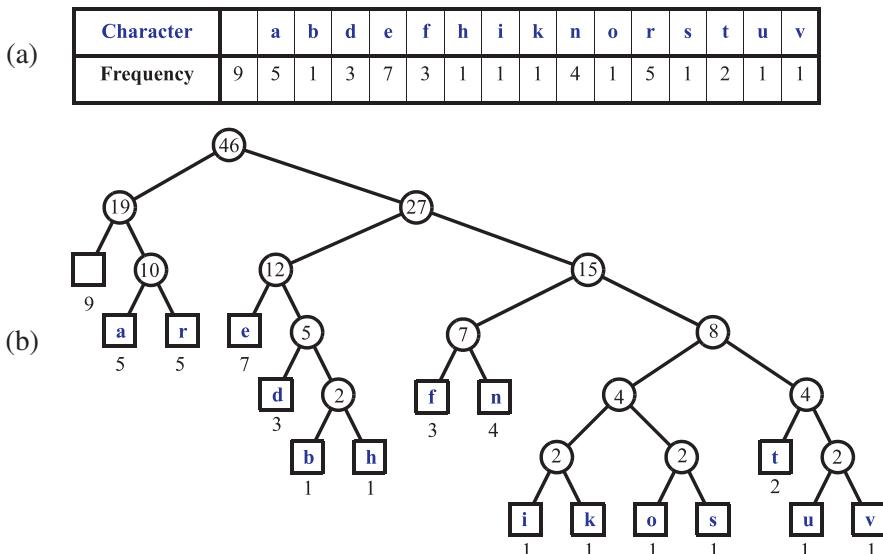


Figure 12.8: An example Huffman code for the input string $X = \text{"a fast runner need never be afraid of the dark"}$: (a) frequency of each character of X ; (b) Huffman tree T for string X . The code for a character c is obtained by tracing the path from the root of T to the external node where c is stored, and associating a left child with 0 and a right child with 1. For example, the code for “a” is 010, and the code for “f” is 1100.

12.4.1 The Huffman-Coding Algorithm

The Huffman-coding algorithm begins with each of the d distinct characters of the string X to encode being the root node of a single-node binary tree. The algorithm proceeds in a series of rounds. In each round, the algorithm takes the two binary trees with the smallest frequencies and merges them into a single binary tree. It repeats this process until only one tree is left. (See Code Fragment 12.9.)

Each iteration of the **while** loop in Huffman’s algorithm can be implemented in $O(\log d)$ time using a priority queue represented with a heap. In addition, each iteration takes two nodes out of Q and adds one in, a process that is repeated $d - 1$ times before exactly one node is left in Q . Thus, this algorithm runs in $O(n + d \log d)$ time. Although a full justification of this algorithm’s correctness is beyond our scope, we note that its intuition comes from a simple idea—any optimal code can be converted into an optimal code in which the code words for the two lowest-frequency characters, a and b , differ only in their last bit. Repeating the argument for a string with a and b replaced by a character c , gives the following.

Proposition 12.7: *Huffman’s algorithm constructs an optimal prefix code for a string of length n with d distinct characters in $O(n + d \log d)$ time.*

Algorithm $\text{Huffman}(X)$:

Input: String X of length n with d distinct characters

Output: Coding tree for X

Compute the frequency $f(c)$ of each character c of X .

Initialize a priority queue Q .

for each character c in X **do**

 Create a single-node binary tree T storing c .

 Insert T into Q with key $f(c)$.

while $Q.\text{size}() > 1$ **do**

$f_1 \leftarrow Q.\text{min}()$

$T_1 \leftarrow Q.\text{removeMin}()$

$f_2 \leftarrow Q.\text{min}()$

$T_2 \leftarrow Q.\text{removeMin}()$

 Create a new binary tree T with left subtree T_1 and right subtree T_2 .

 Insert T into Q with key $f_1 + f_2$.

return tree $Q.\text{removeMin}()$

Code Fragment 12.9: Huffman-coding algorithm.

12.4.2 The Greedy Method

Huffman's algorithm for building an optimal encoding is an example application of an algorithmic design pattern called the **greedy method**. This design pattern is applied to optimization problems, where we are trying to construct some structure while minimizing or maximizing some property of that structure.

The general formula for the greedy method pattern is almost as simple as that for the brute-force method. In order to solve a given optimization problem using the greedy method, we proceed by a sequence of choices. The sequence starts from some well-understood starting condition, and computes the cost for that initial condition. The pattern then asks that we iteratively make additional choices by identifying the decision that achieves the best cost improvement from all of the choices that are currently possible. This approach does not always lead to an optimal solution.

But there are several problems that it does work for, and such problems are said to possess the **greedy-choice** property. This is the property that a global optimal condition can be reached by a series of locally optimal choices (that is, choices that are each the current best from among the possibilities available at the time), starting from a well-defined starting condition. The problem of computing an optimal variable-length prefix code is just one example of a problem that possesses the greedy-choice property.

12.5 Tries

The pattern matching algorithms presented in the previous section speed up the search in a text by preprocessing the pattern (to compute the failure function in the KMP algorithm or the last function in the BM algorithm). In this section, we take a complementary approach, namely, we present string searching algorithms that preprocess the text. This approach is suitable for applications where a series of queries is performed on a fixed text, so that the initial cost of preprocessing the text is compensated by a speedup in each subsequent query (for example, a Web site that offers pattern matching in Shakespeare’s *Hamlet* or a search engine that offers Web pages on the *Hamlet* topic).

A *trie* (pronounced “try”) is a tree-based data structure for storing strings in order to support fast pattern matching. The main application for tries is in information retrieval. Indeed, the name “trie” comes from the word “retrieval.” In an information retrieval application, such as a search for a certain DNA sequence in a genomic database, we are given a collection S of strings, all defined using the same alphabet. The primary query operations that tries support are pattern matching and **prefix matching**. The latter operation involves being given a string X , and looking for all the strings in S that contain X as a prefix.

12.5.1 Standard Tries

Let S be a set of s strings from alphabet Σ such that no string in S is a prefix of another string. A **standard trie** for S is an ordered tree T with the following properties (see Figure 12.9):

- Each node of T , except the root, is labeled with a character of Σ .
- The ordering of the children of an internal node of T is determined by a canonical ordering of the alphabet Σ .
- T has s external nodes, each associated with a string of S , such that the concatenation of the labels of the nodes on the path from the root to an external node v of T yields the string of S associated with v .

Thus, a trie T represents the strings of S with paths from the root to the external nodes of T . Note the importance of assuming that no string in S is a prefix of another string. This ensures that each string of S is uniquely associated with an external node of T . We can always satisfy this assumption by adding a special character that is not in the original alphabet Σ at the end of each string.

An internal node in a standard trie T can have anywhere between 1 and d children, where d is the size of the alphabet. There is an edge going from the root r to one of its children for each character that is first in some string in the collection S . In addition, a path from the root of T to an internal node v at depth i corresponds to

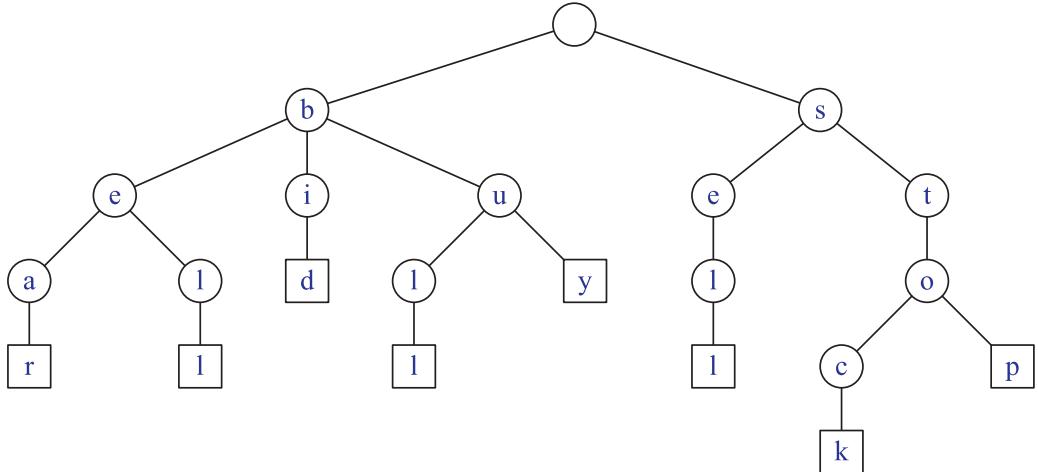


Figure 12.9: Standard trie for the strings $\{\text{bear}, \text{bell}, \text{bid}, \text{bull}, \text{buy}, \text{sell}, \text{stock}, \text{stop}\}$.

an i -character prefix $X[0..i-1]$ of a string X of S . In fact, for each character c that can follow the prefix $X[0..i-1]$ in a string of the set S , there is a child of v labeled with character c . In this way, a trie concisely stores the common prefixes that exist among a set of strings.

If there are only two characters in the alphabet, then the trie is essentially a binary tree, with some internal nodes possibly having only one child (that is, it may be an improper binary tree). In general, if there are d characters in the alphabet, then the trie will be a multi-way tree where each internal node has between 1 and d children. In addition, there are likely to be several internal nodes in a standard trie that have fewer than d children. For example, the trie shown in Figure 12.9 has several internal nodes with only one child. We can implement a trie with a tree storing characters at its nodes.

The following proposition provides some important structural properties of a standard trie.

Proposition 12.8: A standard trie storing a collection S of s strings of total length n from an alphabet of size d has the following properties:

- Every internal node of T has at most d children
- T has s external nodes
- The height of T is equal to the length of the longest string in S
- The number of nodes of T is $O(n)$

The worst case for the number of nodes of a trie occurs when no two strings share a common nonempty prefix; that is, except for the root, all internal nodes have one child.

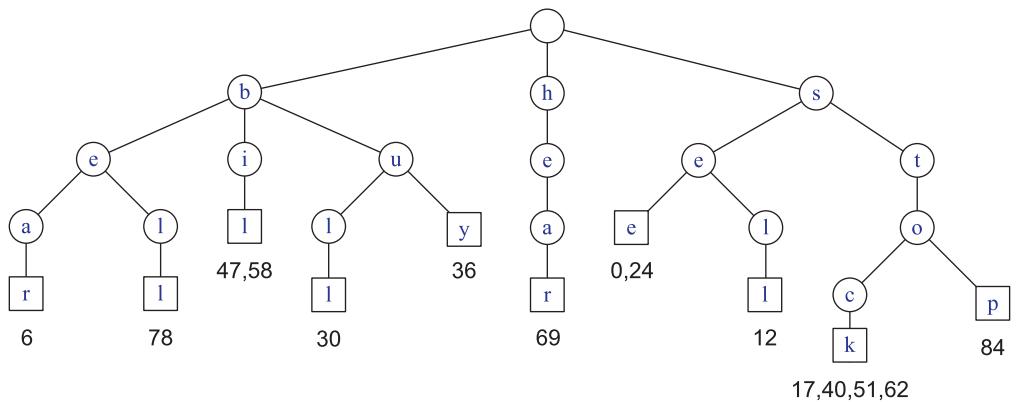
A trie T for a set S of strings can be used to implement a dictionary whose keys are the strings of S . Namely, we perform a search in T for a string X by tracing down from the root the path indicated by the characters in X . If this path can be traced and terminates at an external node, then we know X is in the dictionary. For example, in the trie in Figure 12.9, tracing the path for “bull” ends up at an external node. If the path cannot be traced or the path can be traced but terminates at an internal node, then X is not in the dictionary. In the example in Figure 12.9, the path for “bet” cannot be traced and the path for “be” ends at an internal node. Neither such word is in the dictionary. Note that in this implementation of a dictionary, single characters are compared instead of the entire string (key). It is easy to see that the running time of the search for a string of size m is $O(dm)$, where d is the size of the alphabet. Indeed, we visit at most $m + 1$ nodes of T and we spend $O(d)$ time at each node. For some alphabets, we may be able to improve the time spent at a node to be $O(1)$ or $O(\log d)$ by using a dictionary of characters implemented in a hash table or search table. However, since d is a constant in most applications, we can stick with the simple approach that takes $O(d)$ time per node visited.

From the discussion above, it follows that we can use a trie to perform a special type of pattern matching, called ***word matching***, where we want to determine whether a given pattern matches one of the words of the text exactly. (See Figure 12.10.) Word matching differs from standard pattern matching since the pattern cannot match an arbitrary substring of the text, but only one of its words. Using a trie, word matching for a pattern of length m takes $O(dm)$ time, where d is the size of the alphabet, independent of the size of the text. If the alphabet has constant size (as is the case for text in natural languages and DNA strings), a query takes $O(m)$ time, proportional to the size of the pattern. A simple extension of this scheme supports prefix matching queries. However, arbitrary occurrences of the pattern in the text (for example, the pattern is a proper suffix of a word or spans two words) cannot be efficiently performed.

To construct a standard trie for a set S of strings, we can use an incremental algorithm that inserts the strings one at a time. Recall the assumption that no string of S is a prefix of another string. To insert a string X into the current trie T , we first try to trace the path associated with X in T . Since X is not already in T and no string in S is a prefix of another string, we stop tracing the path at an ***internal*** node v of T before reaching the end of X . We then create a new chain of node descendants of v to store the remaining characters of X . The time to insert X is $O(dm)$, where m is the length of X and d is the size of the alphabet. Thus, constructing the entire trie for set S takes $O(dn)$ time, where n is the total length of the strings of S .

s	e	e		a		b	e	a	r	?		s	e	l	l		s	t	o	c	k	!	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
s	e	e		a		b	u	l	l	?		b	u	y		s	t	o	c	k	!		
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	
b	i	d		s	t	o	c	k	!		b	i	d		s	t	o	c	k	!			
47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68		
h	e	a	r		t	h	e		b	e	l	1	?		s	t	o	p	!				
69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88				

(a)



(b)

Figure 12.10: Word matching and prefix matching with a standard trie: (a) text to be searched; (b) standard trie for the words in the text (articles and prepositions, which are also known as *stop words*, excluded), with external nodes augmented with indications of the word positions.

There is a potential space inefficiency in the standard trie that has prompted the development of the *compressed trie*, which is also known (for historical reasons) as the *Patricia trie*. Namely, there are potentially a lot of nodes in the standard trie that have only one child, and the existence of such nodes is a waste. We discuss the compressed trie next.

12.5.2 Compressed Tries

A **compressed trie** is similar to a standard trie but it ensures that each internal node in the trie has at least two children. It enforces this rule by compressing chains of single-child nodes into individual edges. (See Figure 12.11.) Let T be a standard trie. We say that an internal node v of T is **redundant** if v has one child and is not the root. For example, the trie of Figure 12.9 has eight redundant nodes. Let us also say that a chain of $k \geq 2$ edges

$$(v_0, v_1)(v_1, v_2) \cdots (v_{k-1}, v_k),$$

is **redundant** if:

- v_i is redundant for $i = 1, \dots, k - 1$
- v_0 and v_k are not redundant

We can transform T into a compressed trie by replacing each redundant chain $(v_0, v_1) \cdots (v_{k-1}, v_k)$ of $k \geq 2$ edges into a single edge (v_0, v_k) , relabeling v_k with the concatenation of the labels of nodes v_1, \dots, v_k .

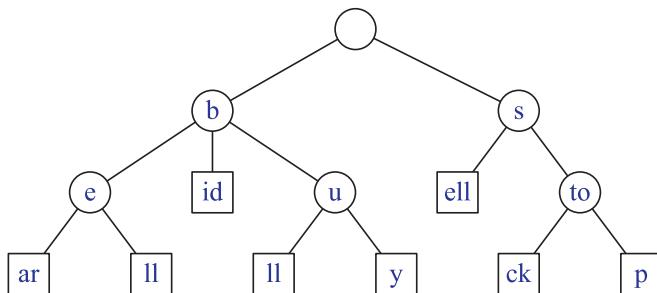


Figure 12.11: Compressed trie for the strings {bear, bell, bid, bull, buy, sell, stock, stop}. Compare this with the standard trie shown in Figure 12.9.

Thus, nodes in a compressed trie are labeled with strings, which are substrings of strings in the collection, rather than with individual characters. The advantage of a compressed trie over a standard trie is that the number of nodes of the compressed trie is proportional to the number of strings and not to their total length, as shown in the following proposition (compare with Proposition 12.8).

Proposition 12.9: A compressed trie storing a collection S of s strings from an alphabet of size d has the following properties:

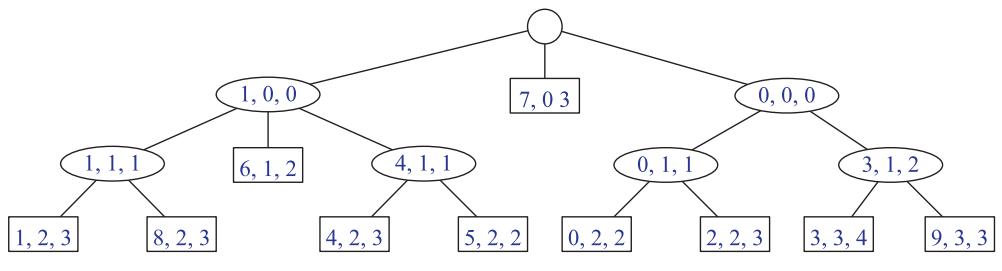
- Every internal node of T has at least two children and most d children
- T has s external nodes
- The number of nodes of T is $O(s)$

The attentive reader may wonder whether the compression of paths provides any significant advantage, since it is offset by a corresponding expansion of the node labels. Indeed, a compressed trie is truly advantageous only when it is used as an *auxiliary* index structure over a collection of strings already stored in a primary structure, and is not required to actually store all the characters of the strings in the collection.

Suppose, for example, that the collection S of strings is an array of strings $S[0], S[1], \dots, S[s - 1]$. Instead of storing the label X of a node explicitly, we represent it implicitly by a triplet of integers (i, j, k) , such that $X = S[i][j..k]$; that is, X is the substring of $S[i]$ consisting of the j th to the k th included. (See the example in Figure 12.12. Also compare with the standard trie of Figure 12.10.)

	0	1	2	3	4		0	1	2	3		0	1	2	3
$S[0] =$	s	e	e				b	u	l	l		h	e	a	r
$S[1] =$	b	e	a	r			b	u	y			b	e	l	l
$S[2] =$	s	e	l	l			b	i	d			s	t	o	p
$S[3] =$	s	t	o	c	k										

(a)



(b)

Figure 12.12: (a) Collection S of strings stored in an array. (b) Compact representation of the compressed trie for S .

This additional compression scheme allows us to reduce the total space for the trie itself from $O(n)$ for the standard trie to $O(s)$ for the compressed trie, where n is the total length of the strings in S and s is the number of strings in S . We must still store the different strings in S , of course, but we nevertheless reduce the space for the trie.

we process elements in pairs. We compare pairs of elements from the input first *with each other*, and then we compare the smaller with the current minimum and the larger to the current maximum, at a cost of 3 comparisons for every 2 elements.

How we set up initial values for the current minimum and maximum depends on whether n is odd or even. If n is odd, we set both the minimum and maximum to the value of the first element, and then we process the rest of the elements in pairs. If n is even, we perform 1 comparison on the first 2 elements to determine the initial values of the minimum and maximum, and then process the rest of the elements in pairs as in the case for odd n .

Let us analyze the total number of comparisons. If n is odd, then we perform $3 \lfloor n/2 \rfloor$ comparisons. If n is even, we perform 1 initial comparison followed by $3(n - 2)/2$ comparisons, for a total of $3n/2 - 2$. Thus, in either case, the total number of comparisons is at most $3 \lfloor n/2 \rfloor$.

Exercises

9.1-1

Show that the second smallest of n elements can be found with $n + \lceil \lg n \rceil - 2$ comparisons in the worst case. (*Hint:* Also find the smallest element.)

9.1-2 *

Prove the lower bound of $\lceil 3n/2 \rceil - 2$ comparisons in the worst case to find both the maximum and minimum of n numbers. (*Hint:* Consider how many numbers are potentially either the maximum or minimum, and investigate how a comparison affects these counts.)

9.2 Selection in expected linear time

The general selection problem appears more difficult than the simple problem of finding a minimum. Yet, surprisingly, the asymptotic running time for both problems is the same: $\Theta(n)$. In this section, we present a divide-and-conquer algorithm for the selection problem. The algorithm RANDOMIZED-SELECT is modeled after the quicksort algorithm of Chapter 7. As in quicksort, we partition the input array recursively. But unlike quicksort, which recursively processes both sides of the partition, RANDOMIZED-SELECT works on only one side of the partition. This difference shows up in the analysis: whereas quicksort has an expected running time of $\Theta(n \lg n)$, the expected running time of RANDOMIZED-SELECT is $\Theta(n)$, assuming that the elements are distinct.

RANDOMIZED-SELECT uses the procedure RANDOMIZED-PARTITION introduced in Section 7.3. Thus, like RANDOMIZED-QUICKSORT, it is a randomized algorithm, since its behavior is determined in part by the output of a random-number generator. The following code for RANDOMIZED-SELECT returns the i th smallest element of the array $A[p \dots r]$.

```
RANDOMIZED-SELECT( $A, p, r, i$ )
1  if  $p == r$ 
2    return  $A[p]$ 
3   $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
4   $k = q - p + 1$ 
5  if  $i == k$           // the pivot value is the answer
6    return  $A[q]$ 
7  elseif  $i < k$ 
8    return RANDOMIZED-SELECT( $A, p, q - 1, i$ )
9  else return RANDOMIZED-SELECT( $A, q + 1, r, i - k$ )
```

The RANDOMIZED-SELECT procedure works as follows. Line 1 checks for the base case of the recursion, in which the subarray $A[p \dots r]$ consists of just one element. In this case, i must equal 1, and we simply return $A[p]$ in line 2 as the i th smallest element. Otherwise, the call to RANDOMIZED-PARTITION in line 3 partitions the array $A[p \dots r]$ into two (possibly empty) subarrays $A[p \dots q - 1]$ and $A[q + 1 \dots r]$ such that each element of $A[p \dots q - 1]$ is less than or equal to $A[q]$, which in turn is less than each element of $A[q + 1 \dots r]$. As in quicksort, we will refer to $A[q]$ as the *pivot* element. Line 4 computes the number k of elements in the subarray $A[p \dots q]$, that is, the number of elements in the low side of the partition, plus one for the pivot element. Line 5 then checks whether $A[q]$ is the i th smallest element. If it is, then line 6 returns $A[q]$. Otherwise, the algorithm determines in which of the two subarrays $A[p \dots q - 1]$ and $A[q + 1 \dots r]$ the i th smallest element lies. If $i < k$, then the desired element lies on the low side of the partition, and line 8 recursively selects it from the subarray. If $i > k$, however, then the desired element lies on the high side of the partition. Since we already know k values that are smaller than the i th smallest element of $A[p \dots r]$ —namely, the elements of $A[p \dots q]$ —the desired element is the $(i - k)$ th smallest element of $A[q + 1 \dots r]$, which line 9 finds recursively. The code appears to allow recursive calls to subarrays with 0 elements, but Exercise 9.2-1 asks you to show that this situation cannot happen.

The worst-case running time for RANDOMIZED-SELECT is $\Theta(n^2)$, even to find the minimum, because we could be extremely unlucky and always partition around the largest remaining element, and partitioning takes $\Theta(n)$ time. We will see that

the algorithm has a linear expected running time, though, and because it is randomized, no particular input elicits the worst-case behavior.

To analyze the expected running time of RANDOMIZED-SELECT, we let the running time on an input array $A[p \dots r]$ of n elements be a random variable that we denote by $T(n)$, and we obtain an upper bound on $E[T(n)]$ as follows. The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot. Therefore, for each k such that $1 \leq k \leq n$, the subarray $A[p \dots q]$ has k elements (all less than or equal to the pivot) with probability $1/n$. For $k = 1, 2, \dots, n$, we define indicator random variables X_k where

$$X_k = I\{\text{the subarray } A[p \dots q] \text{ has exactly } k \text{ elements}\} ,$$

and so, assuming that the elements are distinct, we have

$$E[X_k] = 1/n . \quad (9.1)$$

When we call RANDOMIZED-SELECT and choose $A[q]$ as the pivot element, we do not know, a priori, if we will terminate immediately with the correct answer, recurse on the subarray $A[p \dots q - 1]$, or recurse on the subarray $A[q + 1 \dots r]$. This decision depends on where the i th smallest element falls relative to $A[q]$. Assuming that $T(n)$ is monotonically increasing, we can upper-bound the time needed for the recursive call by the time needed for the recursive call on the largest possible input. In other words, to obtain an upper bound, we assume that the i th element is always on the side of the partition with the greater number of elements. For a given call of RANDOMIZED-SELECT, the indicator random variable X_k has the value 1 for exactly one value of k , and it is 0 for all other k . When $X_k = 1$, the two subarrays on which we might recurse have sizes $k - 1$ and $n - k$. Hence, we have the recurrence

$$\begin{aligned} T(n) &\leq \sum_{k=1}^n X_k \cdot (T(\max(k - 1, n - k)) + O(n)) \\ &= \sum_{k=1}^n X_k \cdot T(\max(k - 1, n - k)) + O(n) . \end{aligned}$$

Taking expected values, we have

$$\begin{aligned}
 \mathbb{E}[T(n)] &\leq \mathbb{E}\left[\sum_{k=1}^n X_k \cdot T(\max(k-1, n-k)) + O(n)\right] \\
 &= \sum_{k=1}^n \mathbb{E}[X_k \cdot T(\max(k-1, n-k))] + O(n) \quad (\text{by linearity of expectation}) \\
 &= \sum_{k=1}^n \mathbb{E}[X_k] \cdot \mathbb{E}[T(\max(k-1, n-k))] + O(n) \quad (\text{by equation (C.24)}) \\
 &= \sum_{k=1}^n \frac{1}{n} \cdot \mathbb{E}[T(\max(k-1, n-k))] + O(n) \quad (\text{by equation (9.1)}).
 \end{aligned}$$

In order to apply equation (C.24), we rely on X_k and $T(\max(k-1, n-k))$ being independent random variables. Exercise 9.2-2 asks you to justify this assertion.

Let us consider the expression $\max(k-1, n-k)$. We have

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil, \\ n-k & \text{if } k \leq \lceil n/2 \rceil. \end{cases}$$

If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation, and if n is odd, all these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once. Thus, we have

$$\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)] + O(n).$$

We show that $\mathbb{E}[T(n)] = O(n)$ by substitution. Assume that $\mathbb{E}[T(n)] \leq cn$ for some constant c that satisfies the initial conditions of the recurrence. We assume that $T(n) = O(1)$ for n less than some constant; we shall pick this constant later. We also pick a constant a such that the function described by the $O(n)$ term above (which describes the non-recursive component of the running time of the algorithm) is bounded from above by an for all $n > 0$. Using this inductive hypothesis, we have

$$\begin{aligned}
 \mathbb{E}[T(n)] &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an \\
 &= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an
 \end{aligned}$$

$$\begin{aligned}
&= \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an \\
&\leq \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2} \right) + an \\
&= \frac{2c}{n} \left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \\
&= \frac{c}{n} \left(\frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an \\
&= c \left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \\
&\leq \frac{3cn}{4} + \frac{c}{2} + an \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an \right).
\end{aligned}$$

In order to complete the proof, we need to show that for sufficiently large n , this last expression is at most cn or, equivalently, that $cn/4 - c/2 - an \geq 0$. If we add $c/2$ to both sides and factor out n , we get $n(c/4 - a) \geq c/2$. As long as we choose the constant c so that $c/4 - a > 0$, i.e., $c > 4a$, we can divide both sides by $c/4 - a$, giving

$$n \geq \frac{c/2}{c/4 - a} = \frac{2c}{c - 4a}.$$

Thus, if we assume that $T(n) = O(1)$ for $n < 2c/(c-4a)$, then $E[T(n)] = O(n)$. We conclude that we can find any order statistic, and in particular the median, in expected linear time, assuming that the elements are distinct.

Exercises

9.2-1

Show that RANDOMIZED-SELECT never makes a recursive call to a 0-length array.

9.2-2

Argue that the indicator random variable X_k and the value $T(\max(k-1, n-k))$ are independent.

9.2-3

Write an iterative version of RANDOMIZED-SELECT.

16. Given an unsorted array of 50 elements, design an algorithm with $O(n)$ time complexity to print the least 25 elements.

We need the 25 smallest elements from 50 numbers in $O(n)$.

Idea

- Use the **Quickselect algorithm** (selection algorithm based on partitioning, same as in Quicksort).
- Quickselect can find the k -th smallest element in expected $O(n)$.
- Then, partitioning gives us all elements smaller than or equal to that pivot.

Algorithm (Quickselect for 25 smallest)

1. Input array $A[1..50]$.
2. Use **Quickselect(A, k=25)** to find the 25-th smallest element x .
 - a. Partition array around pivot until pivot is the 25-th element.
3. Traverse A , print all elements $\leq x$.
4. (If duplicates around x cause more than 25 outputs, print exactly 25 smallest by selecting from that subset.)

Complexity

- Quickselect runs in expected $O(n)$.
- One additional linear scan prints results.
- Total: $O(n)$.

Answer:

Use Quickselect to find the 25-th smallest element in the array, then print all elements \leq it. This algorithm runs in expected **linear time $O(n)$** .

NETWORK FLOW

1. Define Flow in Flow-networks.

Flow Networks:

A flow network is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. There are two special vertices: a source s , and a sink t .

Flows:

Given an s - t network, a flow (also called an s - t flow) is a function f that maps each edge to a nonnegative real number and satisfies the following properties:

Capacity Constraint: For all $(u, v) \in E$, $f(u, v) \leq c(u, v)$.

Flow conservation (or flow balance): For all $v \in V - \{s, t\}$, the sum of flow along edges into v equals the sum of flows along edges out of v .

$$f_{in}(v) = f_{out}(v), \text{ for all } v \in V - \{s, t\}.$$

3. Given a flow network $G=(V,E)$ where $V=\{s,u,t,v\}$ and $E=\{(s,u,2),(u,t,4),(s,v,4),(v,t,2)\}$ Find the **maximum flow** in the network G , showing all steps.

Deepseek

1. State and prove the **Max-flow Min-cut Theorem**.

2. Give the time complexity of the **Ford-Fulkerson algorithm** and justify it.

The **Ford-Fulkerson algorithm** finds the maximum flow in a flow network by repeatedly finding augmenting paths in the residual graph and pushing flow along them until no augmenting path exists.

1. Time Complexity Statement

The time complexity is: $[O(|E| \cdot f_{max})]$

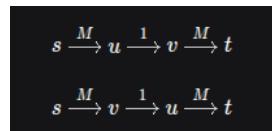
where: ($|E|$) = number of edges, (f_{max}) = value of the maximum flow

2. Justification

- **Each augmentation** increases the flow by at least 1 unit (if capacities are integers).
- So, there are at most (f_{max}) augmentations.
- **Finding an augmenting path** using DFS or BFS takes ($O(|E|)$) time in a graph with ($|V|$) vertices and ($|E|$) edges (since BFS/DFS is ($O(|V| + |E|)$) and typically ($|E| \geq |V|$) in flow networks).
- Hence total time = ($O(|E| \cdot f_{max})$).

3. Example to Illustrate Dependency on (f_{max})

Consider a graph:



If (M) is large, Ford-Fulkerson might repeatedly alternate flow between the two central edges ($u \rightarrow v$) and ($v \rightarrow u$), taking ($O(M)$) iterations, even though ($|E|$) is small. Here ($f_{max} = 2M$), so time is ($O(M \cdot |E|)$), which is large if (M) is large.

3. Provide an example of a **flow network** where the *Ford-Fulkerson algorithm* takes the **maximum number of iterations**.

To see this, consider the example shown in Fig. 56. If the algorithm were smart enough to send flow along the topmost and bottommost paths, each of capacity 100, the algorithm would terminate in just two augmenting steps to a total flow of value 200. However, suppose instead that it foolishly augments first through the path going through the center edge. Then it would be limited to a bottleneck capacity of 1 unit. In the second augmentation, it could now route through the complementary path, this time undoing the flow on the center edge, and again with bottleneck capacity 1. Proceeding in this way, it will take 200 augmentations until we terminate with the final maximum flow.

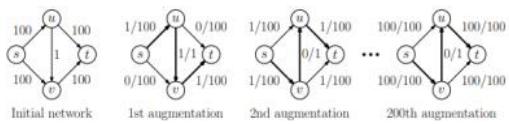


Fig. 56: Bad example for Ford-Fulkerson.

4. Let f and f' be two feasible (s,t) -flows in a graph G , with $|f'| > |f|$.

Prove that there exists a feasible (s,t) -flow of value $|f'| - |f|$ in the **residual graph** G_f .

Flow Difference:

Let $\Delta f = f' - f$ be the flow difference between f' and f . Clearly, Δf is also a feasible flow in the network because both f and f' are feasible, and subtracting one feasible flow from another yields another feasible flow.

Residual Network:

The residual network G_f for a flow f is constructed by considering the residual capacities for each edge. The residual capacity on an edge (u, v) in the residual network is defined as:

$$C_f(u, v) = c(u, v) - f(u, v)$$

where $c(u, v)$ is the original capacity of the edge (u, v) and $f(u, v)$ is the flow through that edge.

In the residual network G_f , there are two types of edges:

Forward edges: If there is a flow $f(u, v)$ the residual capacity is $c(u, v) - f(u, v)$.

Backward edges: If there is a flow $f(v, u)$, the residual capacity is $f(v, u)$ on the reverse edge (v, u) .

Feasibility of Δf : To show that Δf is a feasible flow in the residual network G_f , we need to check the flow conservation and capacity constraints:

Capacity constraint: For each edge (u, v) , the flow $\Delta f(u, v)$ must respect the residual capacity, meaning:

$$0 \leq \Delta f(u, v) \leq C_f(u, v)$$

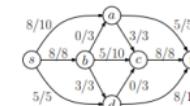
Since f' and f are both feasible flows, it follows that the flow difference $\Delta f = f' - f$ does not violate the capacity constraints.

Flow conservation: For each vertex v , the flow conservation condition must hold for Δf . This means that the total incoming flow to v should equal the total outgoing flow from v . Since f and f' are both feasible flows, their difference Δf will also satisfy flow conservation at each vertex.

5. **True or False:** Let G be a flow network with source s , sink t , and integer capacities. If f is a maximum $s-t$ flow, then every edge out of s must be saturated with flow.

In a flow network, having a maximum flow f_f does not necessarily mean that every edge out of the source s is saturated (i.e., has flow equal to its capacity). The maximum flow f_f is defined as the largest amount of flow that can be sent from the source s to the sink t , subject to the capacity constraints on the edges.

Saturation of Edges: An edge is said to be saturated if the flow through that edge equals its capacity. While it is true that some edges leaving the source may be saturated in a maximum flow scenario, it is not required for all edges out of s to be saturated. The flow can be distributed among multiple edges, and some edges may carry less than their full capacity.



6. Given $G(V,E)=\{s,u,v,t\}$ and edge set $E=\{(s,u,1),(s,v,1),(u,t,1),(v,t,1),(u,v,1)\}$ list all minimum $s-t$ cuts and prove that **Min-cut = Max-flow**.

Deepseek

7. Provide an example in which the Ford-Fulkerson algorithm makes $O(f)$ iterations, where f is the maximum flow.

We want an example where Ford-Fulkerson, using a *poor choice* of augmenting paths, requires $O(f)$ iterations, where (f) is the value of the maximum flow.

1. Understanding the requirement

Ford-Fulkerson's worst-case time is ($O(|E| \cdot f_{\max})$).

This bound is tight if each augmentation increases the flow by only 1 unit, and there are (f_{\max}) augmentations.

So we need a graph and a sequence of augmenting paths such that:

- Each path pushes only 1 unit of flow.
- The number of iterations = (f_{\max}).

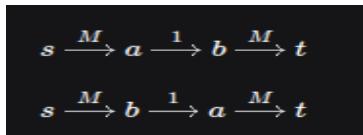
2. Classic “bad” example

Consider this graph: Vertices: $\{s, u, v, t\}$

Edges: $\{(s, u, 1), (u, v, 1), (v, t, 1), (s, v, 1), (u, t, 1)\}$

But that's not the worst case — let's use the standard example from textbooks:

Graph:



Where (M) is a large integer.

Vertices: (s, a, b, t)

Edges with capacities:

1. $((s, a, M))$
2. $((s, b, M))$
3. $((a, b, 1))$
4. $((b, a, 1))$
5. $((a, t, M))$
6. $((b, t, M))$

3. Poor path selection

Max flow = ($2M$) (can send (M) along ($s \rightarrow a \rightarrow t$) and (M) along ($s \rightarrow b \rightarrow t$)).

But if we choose augmenting paths that alternate flow through the middle edges ((a, b)) and ((b, a)), each path pushes only 1 unit.

Step-by-step bad iteration sequence:

Start: flow = 0.

1. Augmenting path: $(s \rightarrow a \rightarrow b \rightarrow t)$

Push 1 unit.

Residual: $((a, b)) \text{ cap } 0, \text{ reverse } ((b, a)) \text{ cap } 1.$

2. Augmenting path: $(s \rightarrow b \rightarrow a \rightarrow t)$

Push 1 unit.

Residual: $((b, a)) \text{ cap } 0, \text{ reverse } ((a, b)) \text{ cap } 1.$

3. Augmenting path: $(s \rightarrow a \rightarrow b \rightarrow t)$

Push 1 unit.

Residual: $((a, b)) \text{ cap } 0, \text{ reverse } ((b, a)) \text{ cap } 1.$

4. Augmenting path: $(s \rightarrow b \rightarrow a \rightarrow t)$

Push 1 unit.

... and so on.

Each odd iteration uses ($a \rightarrow b$), each even iteration uses ($b \rightarrow a$), each pushing 1 unit.

After 2 iterations, total flow = 2.

We need ($2M$) iterations to reach max flow ($2M$).

Thus **number of iterations = ($2M = f_{\max}$)**.

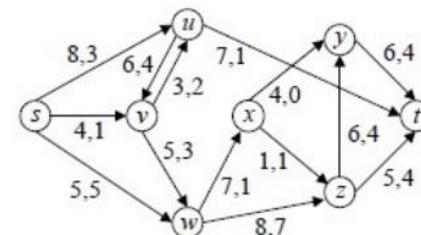
4. Conclusion

This matches ($O(f)$) iterations exactly.

[See graph with edges: $(s, a, M), (s, b, M), (a, b, 1), (b, a, 1), (a, t, M), (b, t, M)$] With alternating paths ($s \rightarrow a \rightarrow b \rightarrow t$) and ($s \rightarrow b \rightarrow a \rightarrow t$), Ford–Fulkerson takes ($2M$) iterations for max flow ($2M$).

8. Consider the given network flow (with source s and sink t). The capacity $c(e)$ and current flow amount $f(e)$ are shown for each edge e .

Run the Ford–Fulkerson algorithm until the maximum flow is computed.



9. Give an example of a small network where the Ford–Fulkerson algorithm can take exactly mC iterations, where m is the number of edges and C is the maximum capacity.

- $m = \text{number of edges}$
- $C = \text{maximum capacity of an edge}$
- Each iteration pushes 1 unit of flow

1. Understanding the bound

Ford–Fulkerson's worst-case number of iterations can be $O(m.C)$ if each augmenting path increases flow by 1 and the max flow is $m.C$.

But max flow f_{\max} might be less than $m.C$, so we need $f_{\max} = m.C$ for the worst case.

So we need:

- m edges, Each with capacity C , Max flow = mC , Each augmentation sends 1 unit

2. Constructing the example

Let's take a simple graph:

Vertices: (s, t) and m parallel edges from s to t , each with capacity C . Then max flow = mC .

If we choose augmenting paths poorly: Always use one edge at a time, sending 1 unit each time, we need mC iterations. But here each augmenting path uses only one edge — that's valid.

Example: Let $m = 2$, $C = 3$.

Edges:

e_1: $s \rightarrow t$, capacity 3

e_2: $s \rightarrow t$, capacity 3

Max flow = 6.

Bad sequence:

Iteration 1: push 1 along e_1

Iteration 2: push 1 along e_2

Iteration 3: push 1 along e_1

Iteration 4: push 1 along e_2

...

After 6 iterations, flow = 6.

Number of iterations = $mC = 2 \cdot 3 = 6$.

3. Generalization

For m edges $s \rightarrow t$, each capacity C ,

max flow = mC .

If each augmentation sends 1 unit and cycles through edges one by one, iterations = mC

So: $[G = (V = \{s, t\}, E = \{(s, t, C)\} \text{ repeated } m \text{ times}]$

With augmentation sequence: repeatedly pick edges in round-robin fashion, sending 1 unit each time.

10. Prove or disprove the statement: "If all capacities in a network are distinct, then there exists a unique flow function that gives the maximum flow."

Deepseek

11. State the properties that must be satisfied by a flow in a flow network.

In a flow network ($G = (V, E)$) with source (s), sink (t), and capacity function ($c: E \rightarrow \{R \geq 0\}$), a **flow** ($f: E \rightarrow R \geq 0$) must satisfy the following properties:

1. Capacity Constraint

For every edge $((u, v) \in E)$: $[0 \leq f(u, v) \leq c(u, v)]$

The flow on an edge cannot exceed its capacity and must be non-negative.

2. Flow Conservation

For every vertex ($u \in V \setminus \{s, t\}$):

$$\sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u)$$

That is, the total flow entering (u) equals the total flow leaving (u) (for intermediate nodes).

3. Flow Value

The **value** of the flow ($|f|$) is defined as:

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

Net flow out of source (s):

Equivalently, net flow into sink (t):

$$|f| = \sum_{v \in V} f(v, t) - \sum_{v \in V} f(t, v)$$

12. Given a flow F in a network, prove that the flow out of the source equals the flow into the sink.

For any flow (F) in a network, the **net flow out of the source** equals the **net flow into the sink**.

1. Definitions

Let ($G = (V, E)$) be a flow network with source (s), sink (t), and flow ($f: E \rightarrow R \geq 0$).

Net flow out of (s):

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

Net flow into (t):

$$\sum_{v \in V} f(v, t) - \sum_{v \in V} f(t, v)$$

2. Flow conservation for all vertices except (s) and (t)

For all ($u \in V \setminus \{s, t\}$):

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

3. Sum over all vertices

Consider the sum over all vertices ($u \in V$) of:

$$\left(\sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) \right)$$

By flow conservation, for (u not equal to s, t), this difference is 0.

For ($u = s$): this difference is ($|f|$) (net flow out of (s)).

For $u = t$:

Outflow from $t = \sum_v f(t, v)$

Inflow to $t = \sum_v f(v, t)$

So difference = $\sum_v f(t, v) - \sum_v f(v, t)$ = negative of net flow into t .

Let $\text{net_into}(t) = \sum_v f(v, t) - \sum_v f(t, v)$.

Then for $u = t$, the difference out - in = -net_into(t).

4. Total sum is zero

The sum over all ($u \in V$) of (outflow - inflow) = 0, because each ($f(x, y)$) appears once as outflow from (x) and once as inflow to (y) with opposite signs.

So:

$$\sum_{u \in V} \left[\sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) \right] = 0$$

But this equals:

$$|f| + (-\text{net_into}(t)) + \sum_{u \neq s, t} 0 = 0$$

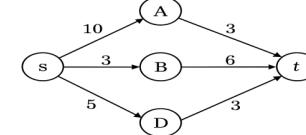
Thus:

$$|f| - \text{net_into}(t) = 0$$

$$|f| = \text{net_into}(t)$$

This is a fundamental property of flow networks, arising from flow conservation at all intermediate nodes and the antisymmetry of the flow sum over all vertices.

13. Given a network flow diagram, identify all minimum $s-t$ cuts and verify that the value of the max-flow equals the capacity of the min-cut.



Edges (capacity): $s \rightarrow A: 10; A \rightarrow t: 3; s \rightarrow B: 3; B \rightarrow t: 6; s \rightarrow D: 5; D \rightarrow t: 3$.

1) Produce a feasible flow and its value (upper bound on max flow).

Send flows along the three simple s -to- t paths:

- $s \rightarrow A \rightarrow t$: send 3 (limited by $A \rightarrow t = 3$).
- $s \rightarrow B \rightarrow t$: send 3 (limited by $s \rightarrow B = 3$).
- $s \rightarrow D \rightarrow t$: send 3 (limited by $D \rightarrow t = 3$).

All other direct paths to t are unavailable. Total flow value = $3+3+3=9$.

2) Show this flow is maximum by finding a cut of capacity 9.

Construct residual graph after the flow above:

- Remaining forward capacity: $s \rightarrow A$ has $10-3=7$, $s \rightarrow B$ has $3-3=0$, $s \rightarrow D$ has $5-3=2$.
- $A \rightarrow t$ and $D \rightarrow t$ are saturated (0 forward), $B \rightarrow t$ has $6-3=3$ forward. Reverse edges from A, B, D to s carry the 3 units of flow back.

Find vertices reachable from s in the residual graph: starting at s you can go to A via 7 and to D via 2 but not to B (its forward capacity is 0). So $R=\{s, A, D\}$.

Find vertices that can reach t in the residual graph: B can reach t (forward capacity 3), so $W=\{B, t\}$.

The canonical cut from the max-flow residual analysis is $(R, V \setminus R) = \{s, A, D\}, \{B, t\}$. Its capacity equals the sum of capacities of all original edges from R to $(V \setminus R)$:

- $s \rightarrow B$ 3, $A \rightarrow t$ 3, $D \rightarrow t$ 3. No other edges go from R to $\{B, t\}$.

So cut capacity = $3+3+3=9$.

3) Conclude max-flow = min-cut and list all minimum cuts.

By the max-flow / min-cut theorem the flow value 9 equals the capacity of the cut $\{s, A, D\}, \{B, t\}$. From the residual reachability argument all minimum $s-t$ cuts are exactly the sets (S) with

$$R \subseteq S \subseteq V \setminus W.$$

Here $R=\{s, A, D\}$ and $(V \setminus W)=\{s, A, D\}$, so the only set satisfying $R \subseteq S \subseteq V \setminus W$ is $S=R$.

Therefore the cut $\{s, A, D\}, \{B, t\}$ is the unique minimum $s-t$ cut, and its capacity equals the maximum flow 9.

14. Suppose that flow network G contains an edge (u, v) and we create a flow network G' by creating a new vertex x and replacing (u, v) with new edges (u, x) and (x, v) with capacity $c(u, x)=c(x, v)=c(u, v)$. Does a maximum flow in G' have the same value as the maximum flow in G ? Justify your answer.

1. Understanding the transformation

We start with G , containing edge (u, v) with capacity $c(u, v)$.

We create (G') by:

- Adding a new vertex (x)
- Removing (u, v)
- Adding (u, x) with capacity $c(u, v)$
- Adding (x, v) with capacity $c(u, v)$

So, the capacity through the “chain” $u \rightarrow x \rightarrow v$ is limited by $c(u, v)$ on both edges.

2. Intuition

The transformation doesn’t change the **bottleneck capacity** between u and v , because in G' , the flow from $u \rightarrow v$ is still constrained by $c(u, v)$ (both edges have that capacity, so effectively the $u-v$ capacity remains $c(u, v)$).

So any flow in G that uses $k \leq c(u, v)$ units on (u, v) can be routed in G' as k units along $(u \rightarrow x \rightarrow v)$, and vice versa.

3. Proof of equivalence

Part 1: Max flow in $G \leq$ Max flow in G'

Let f be a max flow in G . Define a flow f' in G' as:

- $f'(e) = f(e)$ for all edges e in G except (u, v)
- $f'(u, x) = f(u, v)$
- $f'(x, v) = f(u, v)$

Check capacity constraints in G' :

- $f'(u, x) = f(u, v) \leq c(u, v) = c'(u, x)$
- $f'(x, v) = f(u, v) \leq c(u, v) = c'(x, v)$

Flow conservation at x : inflow $f'(u, x)$ = outflow $f'(x, v)$, so satisfied.

Thus f' is valid in G' with same value as f .

So $\text{max-flow}(G) \leq \text{max-flow}(G')$.

Part 2: Max flow in $G' \leq$ Max flow in G

Let f' be a max flow in G' . Define a flow f in G as:

- $f(e) = f'(e)$ for all edges e in G except (u, v)
- $f(u, v) = f'(u, x)$ (which equals $f'(x, v)$ by flow conservation at (x))

Check capacity: $f(u, v) = f'(u, x) \leq c'(u, x) = c(u, v)$, so capacity satisfied.

Flow conservation in G at u and v is preserved because the net flow between u and v is the same as in G' :

In G' : flow from u to v = $f'(u, v)$.

In G : flow from u to v = $f(u, v) = f'(u, v)$.

So f is valid in G with same value as f' .

Thus $\text{max-flow}(G') \leq \text{max-flow}(G)$.

4. Edge cases

- If (u, v) is not used in some max flow of G , then in G' we can set $f'(u, x) = f'(x, v) = 0$.
- The transformation preserves all s-t paths and their bottleneck capacities.

Yes, The maximum flow value in G' is the same as in G .

Example:

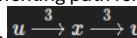
Graph (G) with nodes $\{s, u, v, t\}$ and capacities:

$c(s \rightarrow u) = 4; c(u \rightarrow v) = 3; c(v \rightarrow t) = 5; c(s \rightarrow v) = 2; c(u \rightarrow t) = 1$.

One maximum flow in (G) is

- $f(s \rightarrow u) = 4$ split as $f(u \rightarrow v) = 3; f(u \rightarrow t) = 1$,
- $f(s \rightarrow v) = 2; f(v \rightarrow t) = 3 + 2 = 5$.

Flow value $(=3+2+1=6)$. (No augmenting path remains.)



Form G' by splitting edge (u, v) into $u \xrightarrow{3} x \xrightarrow{3} v$ (both capacities $= 3$). Use the flow

- $f(s \rightarrow u) = 4; f(u \rightarrow x) = 3; f(x \rightarrow v) = 3; f(u \rightarrow t) = 1; f(s \rightarrow v) = 2; f(v \rightarrow t) = 5$.

This f' is feasible in G' (conservation holds at (x) since $(3=3)$) and has value (6) . The same one-to-one mapping argument in the proof applies in both directions, so $\text{max-flow}(G) = \text{max-flow}(G') = 6$.

Lemma: Let (X, Y) be any $s-t$ cut in a network. Given any flow f , the value of f is equal to the net flow across the cut, that is, $f(X, Y) = |f|$.

Proof: Recall that there are no edges leading into s , and so we have $|f| = f^{\text{out}}(s) = f^{\text{out}}(s) - f^{\text{in}}(s)$. Since all the other nodes of X must satisfy flow conservation it follows that

$$|f| = \sum_{x \in X} (f^{\text{out}}(x) - f^{\text{in}}(x))$$

Now, observe that every edge (u, v) where both u and v are in X contributes one positive term and one negative term of value $f(u, v)$ to the above sum, and so all of these cancel out. The only terms that remain are the edges that either go from X to Y (which contribute positively) and those from Y to X (which contribute negatively). Thus, it follows that the value of the sum is exactly $f(X, Y)$, and therefore $|f| = f(X, Y)$.

5. Given vertices u and v in a flow network, where capacity $c(u, v) = 5$ and $c(v, u) = 8$, suppose that

3 units of flow are shipped from u to v and 4 units are shipped from v to u . Compute the net flow from u to v by giving a diagram.



The net flow from vertex u to vertex v is -1 , indicating that there is effectively a surplus of flow moving back from v to u .

Questions on Circulation and Extensions

15. Give the reduction of the circulation problem to a network flow problem.

David mount pdf(p-110)

16. In the context of **flow networks with upper and lower bounds**, explain how the feasibility condition can be checked using a flow model.

David mount pdf(p-113)

17. Define the reduction of circulation problem to network flow problem.

A circulation problem can be reduced to a maximum flow problem by adding a super-source s and super-sink t .

For every node v , if its net demand is $b(v)$, then add an edge $s \rightarrow v$ with capacity $b(v)$ when $b(v) > 0$, and an edge $v \rightarrow t$ with capacity $-b(v)$ when $b(v) < 0$.

Finally, compute max-flow from s to t ; a feasible circulation exists iff all edges out of s are saturated.

Example Circulation Problem

You have a directed graph with demands:

- Node **A** has demand **+3** (needs 3 units of inflow).
- Node **B** has demand **-3** (must send out 3 units).

Edges:

- B** \rightarrow **A** with capacity 5.

Goal: Check if a feasible circulation exists.

Step-by-Step Reduction to Max-Flow

1. Add a Super-Source (s)

For every node with **positive demand**:

- A has demand **+3** \rightarrow add edge ($s \rightarrow A$) with capacity 3.

2. Add a Super-Sink (t)

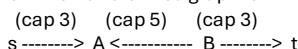
For every node with **negative demand**:

- B has demand **-3** \rightarrow add edge ($B \rightarrow t$) with capacity 3.

3. Keep Original Edges

- Keep $B \rightarrow A$ (capacity = 5).

Now the transformed graph is:



Run Max-Flow from s to t

Possible flow:

- $s \rightarrow A : 3$
- $A \rightarrow B : 3$ (uses part of available 5)
- $B \rightarrow t : 3$

All edges from s are saturated \Rightarrow **feasible circulation exists**.

Final Conclusion

If max-flow saturates all edges outgoing from s , then the original circulation problem has a solution.

If not, no feasible circulation exists.

Question on Bipartite Matching and Reductions

18. Give a **polynomial-time reduction** algorithm from the **Maximum Bipartite Matching** problem to the **Maximum Flow** problem.

1. Problem definitions

- Maximum Bipartite Matching:**

Given bipartite graph $G = (X \cup Y, E)$, find a maximum cardinality set of edges $M \subseteq E$ such that no two edges share a vertex.

- Maximum Flow:**

Given directed capacitated graph G' with source s and sink t , find a flow of maximum value.

2. Reduction construction

From $G = (X \cup Y, E)$ (undirected, bipartite):

- Create a new directed graph G' :

- Keep all vertices $X \cup Y$.
- Add a **source** s and a **sink** t .

- Edges and capacities:**

- Add directed edges from s to every vertex in X , each with capacity 1.
- For each original edge $\{x, y\} \in E$ with $x \in X, y \in Y$, add a directed edge $x \rightarrow y$ with capacity 1.
- Add directed edges from every vertex in Y to t , each with capacity 1.

3. Why it works

- Each unit of flow from s to t corresponds to choosing one matching edge:
 - $s \rightarrow x$ ensures $x \in X$ is used at most once.
 - $x \rightarrow y$ corresponds to selecting edge $\{x, y\}$.
 - $y \rightarrow t$ ensures $y \in Y$ is used at most once.
- Capacity 1 on all edges ensures that each $x \in X$ and each $y \in Y$ is matched at most once.
- Integer capacities \Rightarrow there exists an integer max flow (0/1 on edges), which corresponds directly to a matching.
- The size of the maximum matching = the value of the maximum flow.

4. Polynomial time

The reduction constructs G' in $O(|V| + |E|)$ time:

- $|V'| = |X| + |Y| + 2$
- $|E'| = |X| + |E| + |Y|$

Max flow in G' can be found in polynomial time (e.g., Edmonds-Karp: $O(|V| \cdot |E|^2)$).

5. Example

Bipartite graph: $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$

Edges: $(x_1, y_1), (x_1, y_2), (x_2, y_1)$

Flow network:

$s \rightarrow x_1$ (cap 1), $s \rightarrow x_2$ (cap 1)

$x_1 \rightarrow y_1$ (cap 1), $x_1 \rightarrow y_2$ (cap 1), $x_2 \rightarrow y_1$ (cap 1)

$y_1 \rightarrow t$ (cap 1), $y_2 \rightarrow t$ (cap 1)

Max flow = 2 (e.g., $(x_1 \rightarrow y_2, x_2 \rightarrow y_1)$), matching size = 2.

Construct flow network with source connected to X, X \rightarrow Y edges from E, Y connected to sink, all capacities 1.

19. Define a bipartite matching problem and illustrate how it can be transformed into a flow network to find the maximum matching.

David Mount pdf(p-110)

20. Define the general matching problem and bipartite matching problem.

Then, give a polynomial-time reduction algorithm to reduce the bipartite matching problem to a network flow problem, justifying the time complexity.

David Mount pdf(p-110)

21. Given a bipartite graph $G = (U \cup V, E)$ where

$U = \{a, c, e, g, i\}$ and $V = \{b, d, f, h, j\}$

$E = \{(a,d), (a,f), (b,c), (b,e), (c,d), (c,f), (c,h), (d,e), (d,g), (e,h), (f,g), (h,i), (i,j)\}$ find the maximum matching using the Hungarian Tree method.

We'll find the maximum bipartite matching using the Hungarian tree method (augmenting path search via BFS from unmatched vertices in (U)).

1. Initial matching

Let's start with an empty matching $M = \{\}$.

2. Step 1 — Find augmenting path from ($a \in U$)

BFS tree from a:

- a unmatched.
- Neighbors: d, f (both free).
- Pick d: a-d is augmenting path.

Augment: $M = \{(a,d)\}$.

3. Step 2 — Find augmenting path from ($b \in U$)

b unmatched.

Neighbors: c, e (both free).

Pick c: b-c is augmenting path.

Augment: $M = \{(a,d), (b,c)\}$.

4. Step 3 — Find augmenting path from ($e \in U$)

e unmatched.

Neighbors: d, h.

- d is matched to a.
- h is free.

So path: e-h is augmenting.

Augment: $M = \{(a,d), (b,c), (e,h)\}$.

5. Step 4 — Find augmenting path from ($g \in U$)

g unmatched.

Neighbors: d, f.

- d matched to a.
- f free.

So path: g-f is augmenting.

Augment: $M = \{(a,d), (b,c), (e,h), (g,f)\}$.

6. Step 5 — Find augmenting path from ($i \in U$)

i unmatched.

Neighbors: h, j.

- h matched to e.
- j free.

So path: i-j is augmenting.

Augment: $M = \{(a,d), (b,c), (e,h), (g,f), (i,j)\}$.

7. Check

All vertices in U are matched:

$(a \rightarrow d), (b \rightarrow c), (e \rightarrow h), (g \rightarrow f), (i \rightarrow j)$.

So maximum matching size = (5).

5 Matching edges: $(a,d), (b,c), (e,h), (g,f), (i,j)$.

22. Prove or disprove: The Max-flow and Bipartite Matching problems are polynomial-time reducible to each other.

We need to check whether:

1. Max-flow $\leq_{(P)}$ Bipartite Matching
2. Bipartite Matching $\leq_{(P)}$ Max-flow

1. Bipartite Matching $\leq_{(P)}$ Max-flow

This is true and well-known:

- Given a bipartite graph $G = (U \cup V, E)$, construct a flow network:
 - Source s connected to each $u \in U$ with capacity 1.
 - Each original edge (u, v) directed $u \rightarrow v$ with capacity 1.
 - Each $v \in V$ connected to sink t with capacity 1.
- The value of the maximum flow equals the size of the maximum matching.
- Reduction is polynomial (linear in graph size).

So Bipartite Matching is polynomial-time reducible to Max-flow.

2. Max-flow $\leq_{(P)}$ Bipartite Matching

This would mean: given any flow network, we can construct a bipartite graph whose maximum matching size equals the max flow value.

This is false in general because:

- **Max-flow** is a more general problem: it can have arbitrary capacities, multiple paths, and is not restricted to bipartite structure or unit capacities.
- If such a reduction existed, we could solve general max-flow by solving bipartite matching, but bipartite matching is a special case of max-flow with capacities 1 and a certain structure.
- Known fact: Bipartite matching corresponds to max-flow in a unit-capacity, layered network. But general max-flow can have large capacities and complex cycles not expressible as a bipartite matching.

Counterexample idea:

Take a max-flow instance with source s , sink t , and one edge (s, t) with capacity $k > 1$.

Max flow = k .

A bipartite matching problem's matching size is at most $\min(|U|, |V|)$, but here k can be arbitrarily large independent of graph size — impossible to encode in a bipartite matching instance of size polynomial in original network unless we "simulate" capacities by copying nodes, but that would require k copies, which is not polynomial if k is large (pseudo-polynomial).

Thus, no general polynomial-time reduction from Max-flow to Bipartite Matching unless $P = NP$ (which is open and unlikely).

Actually, known complexity:

- Bipartite Matching is in P.
- Max-flow is in P.

But a poly-time reduction from Max-flow to Bipartite Matching is not known and believed impossible because Bipartite Matching is a special case.

3. Conclusion

- Bipartite Matching \rightarrow Max-flow: YES (standard construction).
- Max-flow \rightarrow Bipartite Matching: NO, unless we allow pseudo-polynomial blow-up (copying edges capacity times), which is not polynomial-time.

Thus, they are **not** polynomial-time reducible to each other in **both directions**.

Disproved: Only one direction is polynomial-time reducible.

Questions on Applications and Complex Scenarios

23. In a hospital with n doctors and m patients, each doctor can treat a limited number of patients daily.

Design an **algorithm using network flow principles** to maximize the number of treated patients and analyze its complexity.

Problem setup

- n doctors D_1, D_2, \dots, D_n
- m patients P_1, P_2, \dots, P_m
- Each doctor D_i can treat up to c_i patients.
- Each patient can be treated by some subset of doctors (given by eligibility list).

Goal \rightarrow maximize the number of patients treated.

Algorithm (using Max-Flow)

1. Create a flow network $G=(V,E)$:

- Add a **source** s and **sink** t .
- For each doctor D_i : add edge (s, D_i) with capacity c_i .
- For each patient P_j : add edge (P_j, t) with capacity 1.
- For every eligible doctor-patient pair (D_i, P_j) : add edge (D_i, P_j) with capacity 1.

2. Run a Max-Flow algorithm (e.g., Edmonds-Karp or Dinic) on this network.

3. Interpret the flow:

- Each unit of flow from $D_i \rightarrow P_j$ represents doctor D_i treating patient P_j .
- The total flow value = total number of treated patients (maximum possible).

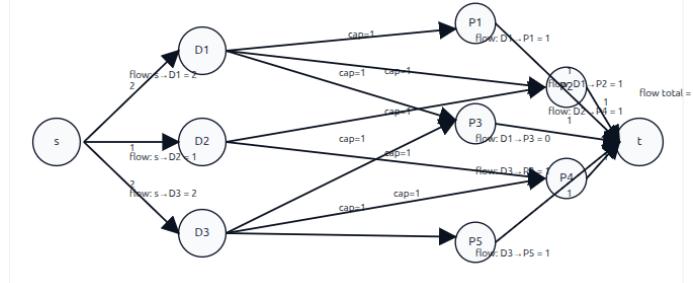
Complexity analysis

- Graph has $O(n + m)$ vertices and $O(nm)$ edges in the worst case.
- Using **Edmonds-Karp**, runtime $O(VE^2) = O((n+m)(nm)^2)$ — high but polynomial.
- Using **Dinic's algorithm**, time $O(\sqrt{V}E) = O(\sqrt{n+m} \cdot nm)$.
- If all capacities are integers ≤ 1 (unit capacities), the special case runs in $O(E\sqrt{V})$ — efficient in practice.

Conclusion:

Model doctors and patients as a bipartite flow network with capacities on doctor edges.

The **maximum flow = maximum number of patients treated**, and the **time complexity** is polynomial — $(O(nm))$ to $(O(E\sqrt{V}))$ depending on the max-flow algorithm used.



24. Construct an example where **circulations with lower bounds** are applied in an allocation or scheduling scenario such as survey or task assignment.

David Mount pdf(p-115)

25. In a hospital, there are n doctors and two shifts (morning & evening).

Each doctor can work in at most one shift per day and at most two days in total.
Design a network flow model to assign doctors to shifts efficiently.

We can model this as a **maximum flow problem** in a suitably constructed flow network.

1. Define the problem in constraints

We have:

- n doctors D_1, D_2, ..., D_n
- 2 shifts per day: morning M, evening E
- Total days available: let's say k days (problem doesn't specify days, but we can generalize: suppose there are m days).
- Each doctor can work **at most one shift per day** (so cannot do both M and E on same day).
- Each doctor can work **at most two days in total** (over all days).

We want to **maximize the number of shift assignments** (or ensure some required number if given).

2. Flow network construction

We build a directed graph with capacities:

Nodes: Source s, Sink t, One node for each doctor: (d_1, ..., d_n), One node for each shift on each day: e.g., (M_1, E_1, M_2, E_2, ..., M_m, E_m) if m days.

Edges:

1. s -> d_i: capacity 2 (each doctor works at most 2 days total; each day they can work 1 shift, so 2 capacity = 2 days max).
2. d_i -> M_j: capacity 1 (doctor i can work morning of day j at most once).
3. d_i -> E_j: capacity 1 (doctor i can work evening of day j at most once).
4. M_j -> t: capacity = number of doctors needed for morning shift on day j (or large if we just want to maximize).
5. E_j -> t: capacity = number of doctors needed for evening shift on day j (or large if maximizing).

3. Why it works

- Each unit of flow from s to t represents one doctor assigned to one shift on one day.
- Capacity 2 from s to d_i enforces "at most two days total" because each day the doctor can only work 1 shift (so 2 flow units = 2 days).
- The edges (d_i -> M_j) and (d_i -> E_j) have capacity 1, ensuring the doctor doesn't work both M and E on same day? Wait — careful: Actually, if a doctor works both M and E on same day, that would require 2 flow units from d_i to day j's shifts. But that's not allowed by "at most one shift per day". So we must prevent that.

4. Preventing same-day both shifts

We can enforce "at most one shift per day per doctor" by **splitting each day-node**:

Better model:

Nodes: s, Doctor nodes (d_1, ..., d_n), For each day j, a "day-doctor" node day_{i,j}) for each doctor? That's too big. Instead:

Actually simpler:

We can keep the earlier model but add **day nodes** day_j for each day j:

- s -> d_i : cap 2
- d_i -> day_j : cap 1 (doctor works day j at most once — i.e., one shift)
- day_j -> M_j : cap = large (or required morning slots)
- day_j -> E_j : cap = large (or required evening slots)
- M_j -> t : cap = morning requirement
- E_j -> t : cap = evening requirement

But this still allows a doctor to be assigned to both M and E through day_j if capacities allow? No — because d_i -> day_j has capacity 1, so doctor can only be assigned to one shift on that day.

Yes, this works.

5. Final network structure

1. s -> d_i : capacity 2
2. d_i -> Day_j : capacity 1 for all i, j (if doctor can work that day; omit if not)
3. Day_j -> M_j : capacity = ∞ or morning requirement
4. Day_j -> E_j : capacity = ∞ or evening requirement
5. M_j -> t : capacity = morning requirement a_j
6. E_j -> t : capacity = evening requirement b_j

If we just want to maximize total shifts assigned, set a_j, b_j large enough (e.g., n).

6. Use case

Compute max flow from s to t.

If max flow = $\sum_j (a_j + b_j)$, all shifts filled.

Otherwise, some shifts unfilled due to doctor availability constraints.

Model as max flow with doctor nodes connected to day nodes (cap 1), day nodes to shifts, source to doctors (cap 2), shifts to sink as per requirements.

26. Given a scenario of resource allocation or scheduling with constraints, formulate the problem using a circulation network and verify feasibility using a flow transformation.

27. The following scenario is given: After a flood, food must be distributed to several locations using available road networks with limited capacities.

Design a flow network and algorithm to decide if supplies meet demands, explaining whether total supply equals total demand.

We can model this as a **multi-source, multi-sink flow problem** and reduce it to a standard **single-source, single-sink max flow problem**.

1. Problem restatement

We have:

- **Supply nodes:** locations with supplies (e.g., warehouses) — each has a fixed supply amount S_i .

- **Demand nodes:** flood-affected locations needing food — each has a fixed demand D_{-j} .
- **Road network:** edges between locations with **capacity** $c(u, v) = \max$ food units that can be transported along that road per day.
- Question: Can we route supplies to meet **all demands** exactly?

Or: Does **total supply \geq total demand** and can we distribute accordingly?

2. Flow network construction

We build a directed graph G' for max flow:

Nodes:

- Keep all original locations as nodes.
- Add a **super-source** s .
- Add a **super-sink** t .

Edges and capacities:

1. From s to each supply node i : capacity S_{-i} (the total supply available at i).
2. Original road edges: directed or undirected?

If roads are two-way, we can put two directed edges each with the same capacity $c(u, v)$.

If one-way, one directed edge.

3. From each demand node j to t : capacity D_{-j} (the demand at j).

3. Check feasibility condition

Let $\text{TotalSupply} = \sum_i S_{-i}$

Let $\text{TotalDemand} = \sum_j D_{-j}$

Necessary condition:

$$\text{TotalSupply} \geq \text{TotalDemand}$$

Otherwise, impossible to meet all demands.

But this alone is **not sufficient** — the road network's capacities might limit flow even if total supply \geq total demand.

4. Algorithm

1. Construct G' as above.
2. Compute **max flow** F from s to t in G' .
3. If $F = \text{TotalDemand}$, then **yes**, we can meet all demands.

If $F < \text{TotalDemand}$, then **no**, we cannot meet all demands (bottleneck in road capacities or supply locations' access to demand nodes).

5. Example

Suppose:

- Supply nodes: A (supply 50), B (supply 30)
- Demand nodes: X (demand 40), Y (demand 30)
- Roads: A \rightarrow X cap 30, A \rightarrow Y cap 20, B \rightarrow X cap 10, B \rightarrow Y cap 25.

$\text{TotalSupply} = 80$, $\text{TotalDemand} = 70$.

Build network:

- $s \rightarrow A$: cap 50
- $s \rightarrow B$: cap 30
- Existing roads with given capacities

- $X \rightarrow t$: cap 40
- $Y \rightarrow t$: cap 30

Compute max flow: We can check if it reaches 70.

6. Why total supply vs total demand matters

- If total supply $<$ total demand \rightarrow immediately "no".
- If total supply \geq total demand, we still need to check max flow = total demand.
- If we just want to know if we can meet demands, compare max flow to total demand.
- If total supply $>$ total demand, the extra supply is irrelevant — we only need to send total demand amount from s to t .

Final answer:

Use max flow with super-source connected to supplies (cap = supply), demands connected to super-sink (cap = demand), roads as edges with their capacities. Compare max flow value to total demand.

QUICK SORT

01. When randomized quicksort runs, how many calls are made to the random number generator in the (i) worst case and (ii) best case? Give your answer in terms of **OO** notation.

1. Understanding the problem

Randomized quicksort chooses a *random* pivot at each recursive call to a subarray of length ≥ 1 .

The number of calls to the random number generator equals the number of times we pick a pivot.

If the array has (n) elements, initially we pick 1 pivot for the whole array, then for the resulting two subarrays (if length ≥ 1) we pick again, and so on.

2. How many pivot selections happen?

Let $C(n)$ = number of random pivot selections for an array of size n .

- Base case: $C(0) = 0$, $C(1) = 1$ (for 1 element, we still pick it as pivot, though no partitioning is needed beyond that).
- Recurrence:

In randomized quicksort, we pick 1 pivot for current array, partition into two parts of sizes k and $n-1-k$ (where k is the rank of pivot after partitioning, $0 \leq k \leq n-1$), then recurse:

$$[C(n) = 1 + C(k) + C(n-1-k)]$$

Averaged over random choices of pivot, but for **worst case / best case** we can think of the recursion structure determined by pivot choice.

3. Best case

Best case for *number of random calls* is the same as the **fewest possible recursive calls**? No — we always call the RNG once for every node in the recursion tree (every subarray that exists in the recursion).

But the number of such nodes depends on the *shape* of the recursion tree.

- In **best-case partitioning**, pivot is always the median \rightarrow splits into two equal (up to ± 1) subarrays.

Then the recursion tree is balanced, of height $\Theta(\log n)$, and total nodes $\approx (2n - 1)$ in a balanced binary tree? Let's check:

Actually:

In the recursion tree for quicksort:

- Root processes n elements
- Each node corresponds to one RNG call (pivot selection)
- Total nodes = total times we call RNG.

For perfectly balanced split:

Level 0: 1 node

Level 1: 2 nodes

Level 2: 4 nodes

... until subarray size = 1.

Number of elements processed: n at root, then $\approx n/2, n/4, n/8$, etc.

But total number of nodes = ?

Let $N(m)$ = number of nodes to sort m elements in balanced case:

$$N(n) = 1 + N(\lfloor (n-1)/2 \rfloor) + N(\lceil (n-1)/2 \rceil)$$

This is like the number of nodes in a binary tree with (n) leaves (since each leaf is a base case of size 0 or 1? Actually base case size 1: $C(1)=1$ node, no children).

Let's think simpler:

Every element except the last in a certain traversal becomes a pivot exactly once? Not exactly — every element is chosen as pivot exactly once over the entire execution in the best case? Actually no, because pivots are removed from subarrays after partitioning.

Better: In best case, every element is a pivot exactly once. Why?

In the recursion tree, each node corresponds to one pivot. There are (n) pivots total because each of the (n) distinct items will be chosen as pivot at some node.

Check:

- Initially, choose 1 pivot, it goes to its sorted position.
- Partition into left and right, neither contains the pivot.
- Recurse on left and right, each element will eventually be a pivot in some call.
- So total RNG calls = (n).

Let's verify with small (n):

$n=1$: picks 1 pivot $\rightarrow 1$ call.

$n=2$: pick 1 pivot (say first element), partition, 1 subarray of size 1, pick 1 pivot there \rightarrow total 2 calls.

$n=3$: pick 1 pivot, partition into sizes 1 and 1, each requires 1 pivot pick \rightarrow total 3 calls.

Indeed, regardless of balance, total RNG calls = n in every case? Wait, that seems important.

4. Key observation

In randomized quicksort, every element is chosen as pivot **exactly once** over the entire process.

Reason:

- When an element is chosen as pivot, it is placed in its final sorted position and never moved again, and excluded from later recursive calls.
- Therefore, the number of times we call the random number generator = number of pivots chosen = n .

This is **independent** of best or worst case!

So even in worst-case scenario (pivot is smallest/largest each time), we still pick n pivots total.

5. Check with worst-case example

Worst case: pivot is smallest each time.

Array size n : pick 1 pivot $\rightarrow 1$ call,

then recurse on right partition size $n-1$: pick 1 pivot $\rightarrow 1$ call,
... until size 1.

Total = n calls.

Best case: pivot is median each time.

Still, every element becomes pivot once $\rightarrow n$ calls.

6. Conclusion

Number of calls to RNG = n in **all cases**.

Thus:

- (i) Worst case: $O(n)$
- (ii) Best case: $O(n)$

More precisely, it's exactly n (for $n \geq 1$).

02. If the partition procedure in quicksort divides the array into $n/3$ and $2n/3$ during each recursion of quicksort, express it as a recurrence relation and derive the running time.

1. Problem restatement

Partition splits array into sizes $n/3$ and $2n/3$ **each time** (approximately).

We need:

1. Recurrence relation for running time $T(n)$.
2. Solve it for asymptotic running time.

2. Recurrence relation

At each call:

- Time to partition array of size n : $O(n)$ (linear time partitioning).
- Then recurse on two subarrays of sizes $\approx n/3$ and $2n/3$.

So:

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$$

3. Solving the recurrence

We can use the **recursion tree** method.

- **Root cost:** cn
- **Level 1:**

Left child size $n/3$, cost $c.n/3$

Right child size $2n/3$, cost $c.2n/3$

Sum = $c.n/3 + c.2n/3 = c.n$ (same as root!)

• Level 2:

From $n/3$: children sizes $n/9$ and $2n/9 \rightarrow$ costs $c.n/9 + c.2n/9 = c.n/3$

From $2n/3$: children sizes $2n/9$ and $4n/9 \rightarrow$ costs $c.2n/9 + c.4n/9 = c.2n/3$

Sum level 2: $c.n/3 + c.2n/3 = c.n$

So **every level** has total cost $O(n)$.

4. Height of the recursion tree

The longest path follows the larger subproblem size $2n/3$.

Height h :

$$\begin{aligned} n \cdot \left(\frac{2}{3}\right)^h &= 1 \\ \left(\frac{2}{3}\right)^h &= \frac{1}{n} \\ h &= \log_{3/2} n \end{aligned}$$

5. Total cost

Number of levels = $O(\log n)$ (since $\log_{3/2} n$ is $\Theta(\log n)$, each level cost $O(n)$).

So: $T(n) = O(n \log n)$

03. What is the running time of QUICKSORT when all elements of array A have the same value?

If **all elements in the array are equal**, then:

- Each partition step compares every element with the pivot — so each call takes $\Theta(n)$ time.
- But because all elements are equal, one side of the partition becomes **empty**, and the other side has $n-1$ elements.

Thus, the recurrence is:

$$T(n) = T(n-1) + \Theta(n)$$

Solving this gives:

$$T(n) = \Theta(n^2)$$

✓ Final answer:

When all elements of the array have the same value, **Quicksort runs in $\Theta(n^2)$ time**.

04. Show that the running time of QUICKSORT is , $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

First state the (implicit) pivot rule: assume QUICKSORT always picks the **first** (**or last**) element of the subarray as pivot (the usual deterministic version used to show worst-case). With the array in **strictly decreasing** order and all elements distinct, that pivot will always be the **largest** element of the

subarray, so every partition is maximally unbalanced: one side is empty and the other has size one less.

Recurrence for ($n \geq 2$): $T(n) = T(n-1) + n$

because Partition does $\Theta(n)$ work about $(n-1)$ comparisons and then we recurse on $(n-1)$ elements.

Unroll the recurrence (or sum comparisons directly). Total comparisons \approx

$$(n-1) + (n-2) + \dots + 1 = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \Theta(n^2).$$

Therefore, $T(n) = \Theta(n^2)$

when the array is sorted in decreasing order (with the described pivot rule).

CLOSEST PAIR PROBLEM

01. Define the closest pair problem.

The **Closest Pair Problem** is a fundamental computational geometry problem. It can be defined as follows:

Given a set of n points in a d -dimensional space, find the two distinct points that are closest to each other. "Closest" is defined by the minimum Euclidean distance between any pair of points.

In simpler terms, it's the task of finding the two points that are nearest neighbors in a given collection.

Key Components

1. **Input:** A set P of n points: $\{p_1, p_2, \dots, p_n\}$.
2. **Output:** A pair of distinct points (p_i, p_j) from the set P , such that the distance between them is the smallest among all possible pairs.
3. **Distance Metric:** The most common metric is the **Euclidean distance**. For two points $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ in a 2D plane, the distance is: $\text{distance}(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

However, the problem can be defined with other metrics like Manhattan distance.

02. Describe the closest pair algorithm. Derive the time complexity of the closest pair algorithm.

Algorithm Overview

1. **Preprocessing:** Sort the list of points by their x-coordinate. This sorted list is used throughout the algorithm. Also, create a list of points sorted by their y-coordinate.
2. **Divide:**
 - a. If the number of points n is small (e.g., 3 or less), use the brute-force method to find the closest pair.
 - b. Otherwise, find a vertical line L that splits the set of points into two nearly equal halves, Left and Right.
3. **Conquer:** Recursively find the closest pair in the Left half and the closest pair in the Right half. Let the minimum distances found be δ_{left} and δ_{right} .

4. Combine:

- a. Let $\delta = \min(\delta_{\text{left}}, \delta_{\text{right}})$.
- b. The real challenge is that the overall closest pair might have one point in Left and one in Right. We only need to consider points within a distance δ of the dividing line L . This region is called the "strip".
- c. For each point p in the strip, you don't need to check it against all other points in the strip. Crucially, you only need to check it against the next few points (at most 7) in the strip when the points are sorted by their y-coordinate. This is the key insight.

Time Complexity Derivation

Let $T(n)$ be the time complexity of the algorithm for n points.

Step 1: The Divide Step

Finding the median x-coordinate to split the set can be done in $O(1)$ time if the list pre-sorted by x is used. Creating the sorted-by-y lists for the left and right halves from the main sorted-by-y list can be done in $O(n)$ time by a simple linear scan.

Step 2: The Conquer Step

We make two recursive calls on halves of size approximately $n/2$. So, this contributes $2T(n/2)$ to the recurrence.

Step 3: The Combine Step (The Critical Part)

This step involves finding the closest "split pair" (one point in left, one in right).

1. Building the "strip" of points within δ of the line L can be done in $O(n)$ time by scanning the list sorted by y-coordinate.
2. For each point p in this strip, we check it against the next few points (a constant number, $c=7$ or 15 depending on the proof) in the strip's y-sorted list. Since we do this for all n points in the worst case (the strip could contain all points), this check takes $O(c * n) = O(n)$ time.

Therefore, the entire combine step runs in $O(n)$ time.

Step 4: The Recurrence Relation

Combining all the parts, we get the following recurrence relation: $T(n) = 2T(n/2) + O(n)$

Step 5: Solving the Recurrence

We can solve this using the **Master Theorem**.

The Master Theorem applies to recurrences of the form: $T(n) = aT(n/b) + f(n)$

In our case:

- $a = 2$ (number of subproblems)
- $b = 2$ (factor by which the problem size is reduced)
- $f(n) = O(n)$ (cost of dividing and combining)

Now, we compare $f(n)$ with $n^{(\log_b a)}$:

- $n^{(\log_b a)} = n^{(\log_2 2)} = n^1 = n$
- $f(n) = O(n)$

This is **Case 2** of the Master Theorem, where $f(n) = \Theta(n^{(\log_b a)})$. The solution for this case is: $T(n) = \Theta(n^{(\log_b a)} * \log n) = \Theta(n * \log n)$

Step 6: Accounting for Preprocessing

We must not forget the initial step: sorting the points by x and y coordinates. This preprocessing takes $O(n \log n)$ time.

Thus, the total time complexity of the entire algorithm is: $O(n \log n) + \Theta(n \log n) = \Theta(n \log n)$

03. Describe Randomized closest pair algorithm. Derive the time complexity of the Randomized closest pair algorithm.

Algorithm Overview

The algorithm builds the solution incrementally, maintaining the closest pair as it adds points one by one in a random order. It uses a data structure to efficiently find points near a newly added point.

1. **Randomization:** Randomly permute the input points. Let's call this sequence p_1, p_2, \dots, p_n .
2. **Initialization:** Start with the first two points. Compute their distance and set this as the initial δ (the current minimum distance). Initialize a data structure D (like a grid) to store the points for efficient neighborhood queries.
3. **Incremental Update:** For each new point p_i (from $i = 3$ to n):
a. **Query:** Use D to find all points that are within a distance δ of p_i . (These are the only candidates that could form a closer pair with p_i).
b. **Update:** Compute the distance between p_i and each of these candidate points. If a distance smaller than δ is found, update δ and the closest pair.
c. **Insert:** Insert the new point p_i into the data structure D.

The key to efficiency is that the data structure D allows for very fast insertion and very fast "range queries" for points within a square of side 2δ .

Choosing the Data Structure: A Grid

A common and effective choice for D is a **hashing-based grid**.

- The entire plane is divided into a grid of squares, each with side length δ .
- Each point falls into one grid cell. The cell of a point (x, y) can be found by $(\text{floor}(x/\delta), \text{floor}(y/\delta))$.
- We maintain a hash table where the keys are cell coordinates (i, j) and the values are linked lists of points in that cell.

Operations:

- **Insert(p):** Compute its cell (i, j) and add it to the linked list in that hash bucket. $O(1)$ time.

- **Query(p, δ):** To find all points within distance δ of point p , we only need to check points in its own cell and the 8 surrounding cells. This is because any point further away than these 9 cells must be more than distance δ from p .

Time complexity analysis:

1. **Random Order:** The points are ordered randomly, which helps in evenly distributing the points for processing.
2. **Initialization:** The minimum distance δ is initialized based on the distance between the first two points.
3. **Dictionary Operations:** A dictionary (or hash table) is used to store points in sub-squares of a grid defined by $\delta/2$.
4. **Lookup and Distance Computation:** For each point, the algorithm looks up nearby sub-squares and computes distances to points in those squares.
5. **Updating Closest Pair:** If a closer pair is found, the algorithm updates δ and reinitializes the dictionary.

Time Complexity Breakdown

1. **Initialization:** Sorting the points randomly takes $O(n)$ time.
 2. **Distance Computations:** For each point, a constant number of distance computations is performed, leading to $O(n)$ distance calculations overall.
 3. **Lookup Operations:** Each point requires a constant number of lookup operations in the dictionary, resulting in $O(n)$ lookup operations.
 4. **Dictionary Operations:** The algorithm performs $O(n)$ operations for creating and updating dictionaries.
- Total Time Complexity= **$O(n)$**

04. In Randomized closed pair algorithm , how randomization is used ?

In the Randomized Incremental closest pair algorithm, **randomization is used to ensure a favorable expected input order**, which guarantees the algorithm's $O(n \log n)$ expected running time. It's a protective measure against worst-case scenarios. Let's break down exactly how it's used and why it's necessary.

The Core Idea: "Randomization for Oblivious Adversary"

The algorithm works by processing points one by one. Its performance depends critically on *the order in which points are processed*. Randomization ensures that no matter what the original input is, the processing order is random, which leads to good *expected* performance.

How Randomization is Applied

1. **Step 0: Preprocessing**
 - a. **Action:** Before the algorithm proper begins, it takes the input set of n points and generates a **random permutation** of them.
 - b. **Result:** The points are processed in this random order: $p_1, p_2, p_3, \dots, p_n$.

Why This Random Order is So Important

The entire analysis hinges on this random order. Let's see its impact on the two most expensive parts of the algorithm.

1. Controlling the Number of Distance Computations

The algorithm maintains the current closest pair distance δ . When a new point p_i is added, it only checks points within a distance δ of p_i .

- **Worst-Case without Randomization:** An adversary could give you points in an order where each new point is just barely outside the current δ but forms a new, slightly closer pair with a point that is very far away in the data structure. This could force the algorithm to check many points for every insertion, leading to $O(n^2)$ behavior.
- **With Randomization (Backward Analysis):** Because the order is random, we can use a powerful technique called **backward analysis**. We ask: "For the set of the first i points $S_i = \{p_1, p_2, \dots, p_i\}$, which is a random set, what is the probability that the i -th point is part of the closest pair?"
 - In a random set, every point is equally likely to be part of the closest pair. The probability that a specific point p_i is one of the two points in the closest pair of S_i is at most $2/i$.
 - This logic extends to the "neighbors" of p_i in the data structure. It can be proven that the **expected number of points** the algorithm needs to check when inserting p_i is **constant** ($O(1)$).
 - **Conclusion:** Over all n points, the total expected number of distance computations is $O(n)$.

2. Controlling the Cost of Grid Rebuilds

The data structure (a grid) depends on the current δ . When a new closest pair is found, δ decreases, and the entire grid must be rebuilt with a new cell size. This is an $O(i)$ operation when processing the i -th point.

- **Worst-Case without Randomization:** An adversary could order the points so that δ is updated with *every single insertion*. This would mean rebuilds at $i=2, 3, 4, \dots, n$. The total cost would be $O(2 + 3 + 4 + \dots + n) = O(n^2)$.
- **With Randomization:** We again use backward analysis. We ask: "For the random set S_i , what is the probability that the closest pair in S_i is different from the closest pair in S_{i-1} ?" In other words, what is the probability that inserting the i -th point causes δ to change?
 - For δ to change, the new point p_i must form a new closest pair with some other point in S_i . The two points forming the new closest pair must be p_i and one of the previous $i-1$ points.
 - In a random set of i points, the probability that a *specific pair* is the closest pair is roughly $1 / (\text{number of pairs}) = 2/(i(i-1))$.
 - There are $(i-1)$ pairs that involve p_i . So, the probability that *any* pair involving p_i is the closest pair in S_i is at most $(i-1) * [2/(i(i-1))] = 2/i$.
 - **Conclusion:** The probability of an expensive grid rebuild when inserting the i -th point is only $\sim 2/i$.

Now, let's sum the expected cost of all rebuilds: Expected Cost = $\sum (\text{Cost of rebuild at step } i) * (\text{Probability of rebuild at step } i) = \sum [O(i) * (O(1)/i)]$ for i from 1 to n = $\sum O(1)$ for i from 1 to n = $O(n)$

5. Define Universal hash function and explain how is it used in randomized closest pair algorithm.

Core Definition

A family of hash functions H that map from a universe U to a set of buckets $\{0, 1, \dots, m-1\}$ is called **universal** if:

For any two distinct keys $x, y \in U$, the probability of collision over a randomly chosen hash function $h \in H$ is at most $1/m$.

$$\Pr_{h \in H} [h(x) = h(y)] \leq \frac{1}{m}$$

Mathematically:

Key Implications

1. **Uniform Distribution:** The hash values of any two distinct keys are essentially independent. Choosing a random h from H gives behavior similar to a completely random function.
2. **Good Average Performance:** For a hash table with n elements, the expected length of any chain is n/m (the load factor).
3. **Deterministic but Random-Looking:** Once a specific h is chosen from the family, the function is deterministic. But from the perspective of an adversary who doesn't know which h was chosen, the behavior appears random.

Simple Example Construction

A classic universal family for integers is:

$$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod m$$

where:

- p is a prime larger than the universe size
- $a \in \{1, 2, \dots, p-1\}$
- $b \in \{0, 1, \dots, p-1\}$
- m is the number of buckets

How Universal Hashing is Used in Randomized Closest Pair

In the randomized incremental closest pair algorithm, universal hashing is used to implement the **dynamic grid data structure** efficiently.

The Grid Data Structure Revisited

Recall the algorithm:

1. Process points in random order
2. Maintain current minimum distance δ
3. Use a grid where each cell has side length δ
4. For a new point, only check points in its cell and 8 neighboring cells

The Hashing Implementation

Instead of using a 2D array (which would be infeasible since the grid coordinates depend on δ and can be very large), we use a **hash table** where the keys are **cell coordinates**.

Cell Identification: For a point $p = (x, y)$, its grid cell is:

$$(i, j) = \left(\lfloor \frac{x}{\delta} \rfloor, \lfloor \frac{y}{\delta} \rfloor \right)$$

Hash Function: We need a hash function that maps cell coordinates (i, j) to bucket indices.

Why Universal Hashing is Crucial

1. Dynamic Nature of the Grid

- The grid cell size **changes** when we find a closer pair
- Each time δ changes, we must **rebuild** the entire grid with the new cell size
- This means all points get reassigned to new cells based on the new δ
- We need a hash function that works well for these new, unpredictable cell coordinates

2. Avoiding Worst-Case Behavior

If we used a fixed hash function, an adversary could:

- Analyze the hash function
- Craft input points that all hash to the same bucket
- Force the algorithm to check many points for each insertion
- Degenerate the algorithm to $O(n^2)$ time

3. The Solution: Universal Hashing

At each grid rebuild:

1. **Randomly choose** a new hash function h from a universal family H
2. Use this h to hash all cell coordinates (i, j) into the hash table

Specific Hash Function for 2D Grid Cells

A common universal hash function for 2D integer coordinates (i, j) is:

$$h_{a,b,c}(i, j) = ((a \cdot i + b \cdot j + c) \mod p) \mod m$$

where:

- p is a large prime
- $a, b \in \{1, 2, \dots, p-1\}$ are randomly chosen coefficients
- $c \in \{0, 1, \dots, p-1\}$ is a random offset
- m is the hash table size (typically $O(n)$)

06. In randomized closest pair algorithm, how is the notion of randomization used?

This question expects a direct, technical explanation of the steps.

Answer: In the randomized incremental closest pair algorithm, randomization is used in one specific, critical step at the beginning:

1. **Input Permutation:** The algorithm takes the input set of n points and generates a **random permutation**. The points are then processed in this random order: $p_1, p_2, p_3, \dots, p_n$.

This single step is fundamental to the entire algorithm's efficiency. The subsequent steps rely on this random order:

- The algorithm maintains the current closest pair and its distance δ .
- It uses a grid data structure, where each cell has side length δ , to store processed points.
- When a new point p_i is added, it only checks for closer pairs with points in its own grid cell and the 8 surrounding cells.
- If a closer pair is found, δ is updated, and the grid is rebuilt with the new cell size.

The random order guarantees that when we use **backward analysis**, the probability that the i -th point causes an expensive operation (like a grid rebuild or many distance checks) is low. Specifically, the probability that inserting p_i forces a rebuild of the grid is at most $2/i$. This inverse relationship between cost and probability ensures the overall expected running time is $O(n \log n)$.

08. In the combined step of divide and conquer closest pair algorithm, explain why the algorithm needs only to check the distance between each point and the next eight points in the sorted sequence of Y to obtain the closest pair.

1. The Setup for the "Combine" Step

- **Divide:** We have recursively found the closest pair in the left half and the right half. Let the minimum distance found be $\delta = \min(\delta_{\text{left}}, \delta_{\text{right}})$.
- **The Challenge:** The overall closest pair might be a **split pair**—one point in the left half and one in the right half, with a distance $< \delta$.
- **The Strip:** We only need to consider points within a vertical strip of width 2 δ centered on the dividing line L . Any split pair closer than δ must have both points inside this strip.
- **Sorted List Y':** We have a list Y' of all points within this strip, sorted by their **y-coordinate**.

2. The Key Geometric Argument

The algorithm iterates through each point p in the sorted list Y' . For each p , it checks the distance to other points in Y' that are **ahead** of it (to avoid duplicate checks). The claim is: we only need to check the next **7 points** after p in Y' .

Why is this true?

Let's consider a single point p from the left side of the strip.

1. **The $\delta \times 2\delta$ Rectangle:** Draw a rectangle of height 2δ and width δ directly above point p . Why this size?
 - Width δ : Any point to the left or right of p by more than δ would already be further than δ away in the x-direction alone.
 - Height 2δ : We are looking for points whose **total Euclidean distance** from p is less than δ . We will consider the area above p ; the area below is handled when those points become p in the iteration.

2. How many points can be inside this rectangle?

- a. The crucial observation is that **no two points can be closer than δ within the left half or within the right half** (because δ is the minimum distance from the recursive calls).
- b. This rectangle has a width of δ and a height of 2δ . Its diagonal length is $\sqrt{(\delta^2 + (2\delta)^2)} = \sqrt{5\delta^2} = \delta\sqrt{5} \approx 2.236\delta$, which is greater than δ . However, the powerful constraint is the minimum distance δ between points.

3. Packing the Rectangle with Points

- a. Let's divide this $\delta \times 2\delta$ rectangle into **8 equal squares**, each of size $\delta/2$ by $\delta/2$.

- b. What is the maximum distance between any two points inside one of these small squares? It's the diagonal: $\sqrt{((\delta/2)^2 + (\delta/2)^2)} = \sqrt{(\delta^2/2)} = \delta/\sqrt{2} \approx 0.707\delta$.
- c. Since this distance is **less than δ** , each of these 8 small squares can contain **at most one point**. If a small square contained two points, they would be closer than δ , contradicting the fact that δ is the minimum distance from the left and right halves.

3. The Final Count: Why 7 or 8 Points?

- The point p itself is in the left half. The 8 small squares are all to the right of p (they are within the δ width of the rectangle).
- Therefore, **any point that is a candidate to form a split pair with p and is within a vertical distance of δ must lie in one of these 8 squares**.
- This means there can be **at most 7 other points** (one in each of the 8 squares, excluding the square that might be occupied by p itself, but since p is on the left, it's not in these squares) within a vertical distance of δ from p that we need to check.

Important Nuance: The algorithm checks the next **8 points** in the Y' list because:

- The list Y' is sorted by y-coordinate.
- If you have more than 7 points ahead of p in the list that are within a vertical distance of δ , then by the **pigeonhole principle**, at least two of them would have to fall into the same one of the 8 small squares, which is impossible.
- Therefore, at most 7 points following p in the Y' list can be within the critical $\delta \times 2\delta$ rectangle. Checking 8 points is a safe upper bound to ensure we don't miss any.

09. How hash table used in randomized closest pair. Prove that expected number of insert operation in the hash table in the randomized closest pair algorithm is $O(n)$.

Part 1: How a Hash Table is Used

In the randomized incremental closest pair algorithm, a hash table is used to implement the **dynamic grid data structure** that allows for efficient range queries.

The Problem

We need to quickly find all points within distance δ (the current minimum distance) of a new point p . A naive approach would check all previous points, which is $O(n)$ per insertion, leading to $O(n^2)$ total time.

The Solution: A Grid-Based Hash Table

1. **Grid Creation:** Superimpose a grid over the plane where each cell is a square of side length δ .
2. **Cell Identification:** For any point $p = (x, y)$, we can determine its grid cell coordinates as: $(cell_x, cell_y) = (\lfloor x/\delta \rfloor, \lfloor y/\delta \rfloor)$
3. **Hash Table Mapping:** Use a hash table where the **key** is the cell coordinate ($cell_x, cell_y$) and the **value** is a linked list of all points currently in that cell.
4. **Operations:**
 - a. **Insert(p):** Compute p 's cell coordinates, hash them, and add p to the linked list in that bucket.

- b. **Query(p, δ):** To find all points within distance δ of p , we only need to check points in p 's cell and the 8 surrounding cells. This is because any point outside this 3×3 block of cells must be more than distance δ away from p .

Why Use a Hash Table?

- The grid coordinates can be very large (depending on point coordinates and δ).
- A 2D array would require impractically large memory.
- A hash table provides **$O(1)$ expected time** for insertions and lookups while using **$O(n)$ space**.

2nd portion khata

10. Modify the closest pair algorithm such that Y array is not pre-sorted yet the running time complexity remains $O(n\log n)$.

The Standard Algorithm's Bottleneck

In the standard divide-and-conquer closest pair algorithm:

1. **Preprocessing:** Sort all points by x-coordinate (X) and also by y-coordinate (Y). ← This is what we want to avoid.
2. **Divide:** Recursively split the set into left and right halves based on x-coordinate.
3. **Conquer:** Find the closest pair in each half.
4. **Combine:** Check for a closer "split pair" that has one point in each half. This step uses the pre-sorted Y array to efficiently build the strip of points near the vertical dividing line, sorted by y-coordinate.

The key insight is that the **combine step needs the points in the strip to be sorted by y-coordinate**. The standard algorithm ensures this by having a globally pre-sorted Y array and then merging the y-sorted lists from the left and right halves in $O(n)$ time.

The Modification: Merge-Sort Style Sorting

We can eliminate the pre-sorting of Y by **integrating the sorting by y-coordinate into the recursion itself**. This approach is similar to how Merge Sort works.

Modified Algorithm

Input: A list P of n points. **Output:** The closest pair distance and the list of points sorted by y-coordinate.

1. **Base Case:** If $n \leq 3$, compute the closest pair using a brute-force method. Also, sort these few points by y-coordinate and return this sorted list along with the closest pair distance.
2. **Divide:**
 - a. Sort the points by x-coordinate (we still need this, but only for the current call to find the median). For efficiency, this should be done once at the very beginning, or we can use the median of medians for linear-time splitting, but typically we just do one global sort by x at the start.
 - b. Split P into two halves, L (left) and R (right), of approximately equal size by a vertical line $x = x_{mid}$.
3. **Conquer:**
 - a. Recursively solve the problem for L. Let the result be δ_L (min distance in left half) and Y_L (points in left half sorted by y-coordinate).
 - b. Recursively solve the problem for R. Let the result be δ_R (min distance in right half) and Y_R (points in right half sorted by y-coordinate).

4. Combine:

- a. Let $\delta = \min(\delta_L, \delta_R)$.
- b. **Merge:** Merge Y_L and Y_R into a single list Y sorted by y-coordinate. This takes $O(n)$ time.
- c. **Build Strip:** Scan through the merged list Y and collect all points that are within δ horizontal distance of the vertical line x_{mid} into a list S (the strip). Since Y is sorted by y , S will automatically be sorted by y . This takes $O(n)$ time.
- d. **Check Split Pairs:** For each point in S , check its distance to the next 7 points in S (as in the standard algorithm). This takes $O(n)$ time.
- e. Return the updated δ (if a closer split pair was found) and the merged list Y .

Why the Running Time Remains $O(n \log n)$

Let's analyze the time complexity with recurrence relations.

Let $T(n)$ be the time complexity for n points.

- **Divide:** Finding the median and splitting can be done in $O(n)$ time.
- **Conquer:** Two recursive calls on halves of size $n/2$: $2T(n/2)$
- **Combine:**
 - Merging Y_L and Y_R : $O(n)$
 - Building the strip S : $O(n)$
 - Checking points in the strip: $O(n)$
 - **Total Combine:** $O(n)$

This gives us the recurrence relation: $T(n) = 2T(n/2) + O(n)$

This is the same recurrence as Merge Sort, and it solves to $T(n) = O(n \log n)$.

Space Complexity Improvement

- **Standard Algorithm:** Requires $O(n)$ extra space for the pre-sorted Y array.
- **Modified Algorithm:** No additional global arrays. The sorting by y is done locally during the recursion and the merged arrays are built and then discarded. The space complexity becomes $O(\log n)$ for the recursion stack if implemented carefully, or $O(n)$ if we consider the storage for the recursive Y arrays (though this storage is temporary and can be managed).

1. Write a pseudocode for the naïve string matching algorithm to find the first occurrence of a pattern P of length m in a text T of length n. What is the worst case time complexity of this algorithm.

Khata+

Worst-Case Time Complexity

- For each of the $n - m + 1$ possible alignments, up to m comparisons are made.
- Worst case = $O((n - m + 1) * m) = O(nm)$

👉 Example of worst case:

Text = "aaaaaaaaaa..." (all a's)

Pattern = "aaaab"

4. Compute the prefix function π for the pattern 'ababab' when the alphabet set is ababbab.

Step 1: Recall definition

The **prefix function** $\pi[i]$ = length of the longest **proper prefix** of the substring $P[0...i]$ which is also a suffix of it.

- Proper prefix ≠ whole string itself.

Pattern = abababab

Length = 8

Step 2: Compute table

i	$P[i]$	Substring $P[0...i]$	Longest prefix = suffix	$\pi[i]$
0	a	a	-	0
1	b	ab	-	0
2	a	aba	a	1
3	b	abab	ab	2
4	a	ababa	aba	3
5	b	ababab	abab	4
6	a	abababa	ababa	5
7	b	abababab	ababab	6

Final Prefix Function (π)

$\pi = [0, 0, 1, 2, 3, 4, 5, 6]$

👉 The given alphabet set {a, b} doesn't affect the computation, since prefix function depends only on the pattern itself.

5. Draw a state transition diagram for a string matching automaton for the pattern 'abababababababab' over the alphabet {a,b}. Compute the prefix function for the same pattern.

Sure — concise answer.

Pattern: abababababababab (length $m=19$)

Prefix function π (indices 0..18):

[0,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,0]

State-transition (automaton) table for states 0..19 over alphabet {a,b} (row = current state, entries = on 'a' →, on 'b' →):

0: a→1	b→0
1: a→1	b→2
2: a→3	b→0
3: a→1	b→4
4: a→5	b→0
5: a→1	b→6
6: a→7	b→0
7: a→1	b→8
8: a→9	b→0
9: a→1	b→10
10: a→11	b→0
11: a→1	b→12
12: a→13	b→0
13: a→1	b→14
14: a→15	b→0
15: a→1	b→16
16: a→17	b→0
17: a→1	b→18

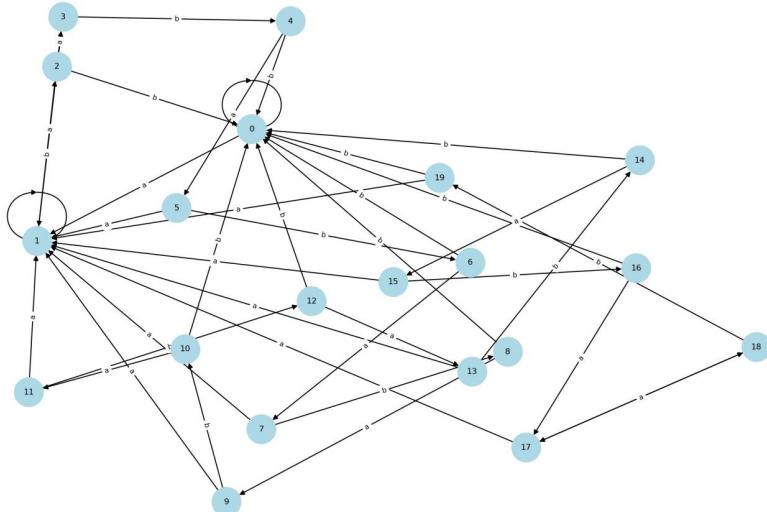
18: a→17 b→19

19: a→1 b→0 (final state = 19)

Transition diagram: ababababababababb

- Each circle = a state (0 → 19).
- Edges labeled a or b = transitions.
- State 19 is the final accepting state (pattern matched).

String Matching Automaton for pattern: 'ababababababababb'



6. Suppose all characters in pattern p are different. Show how to accelerate the naïve-string matching algorithm to run in $O(n)$ on a n -character text T .

If all characters in pattern p are distinct, then we can skip more characters after a mismatch.

- Naïve algorithm normally shifts the pattern by 1 position.
- But if all characters in p are unique, then **after a mismatch at shift s , we can safely skip ahead by the number of matched characters + 1** (because no prefix of p can overlap with its suffix).

Pseudocode

AcceleratedNaive(T, P):

$n \leftarrow \text{length}(T)$

$m \leftarrow \text{length}(P)$

$s \leftarrow 0$

while $s \leq n - m$:

$j \leftarrow 0$

 while $j < m$ and $T[s + j] = P[j]$:

$j \leftarrow j + 1$

 if $j = m$:

 return s // match found

$s \leftarrow s + \max(1, j + 1)$ // jump more than 1

return -1

Complexity

- Each character in text is compared at most once.
- Overall time = $O(n)$.

👉 This is essentially the **improved naive algorithm** (like Horspool's shift idea but simpler since pattern has all distinct characters).

Okay 👈 Let's take a **small example**.

Text $T = "xabcda\textcolor{red}{bc}"$ ($n=8$)

Pattern $P = "abc"$ ($m=3$, all characters different).

Steps (Accelerated Naïve Matching)

- $s=0$: Compare "xab" with "abc" → mismatch at first char.
- Shift by $\max(1, j+1) = 1$.
- $s=1$: Compare "abc" with "abc" → full match ✅ → return 1.

👉 Found at index 1 in $O(n)$ time.

7. Suppose we allow a pattern P to contain occurrences of a gap character '<>' that can match an arbitrary string of characters (one of zero length). For example, the pattern ab<>ba<>c occurs in the text 'cacbccbacbacab' as 'cab<>ba<>cab'. Give a polynomial time algorithm to determine whether such pattern P occurs in a given text and analyze the running time complexity of your algorithm.

Got it! This is essentially **pattern matching with wildcards (gaps)**, where "<>" can match any substring (including empty).

Algorithm (Polynomial Time)

1. Split the pattern P into segments separated by "<>".
 - a. Example: ab<>ba<>c \rightarrow segments = [ab, ba, c].
2. Scan the text T left to right:
 - a. For each segment, find its occurrence in T in order, allowing arbitrary characters between them.
 - b. Each segment can be found using any linear-time string matching algorithm (e.g., KMP).
3. If all segments appear in order, then P occurs in T .

Pseudocode

```
GapStringMatch(T, P):
    Segments <- split(P, "<>")
    pos <- 0
    for seg in Segments:
        idx <- find(T, seg, start=pos) // e.g. KMP or simple search
        if idx == -1:
            return False
        pos <- idx + len(seg)
    return True
```

Complexity Analysis

- Let k =number of segments.
- Each find can be done in $O(n + m_i)$ using KMP, where m_i is length of segment i .
- Total complexity:

$$O(k \cdot n) \text{ (polynomial, linear if } k \text{ is small)}$$

Example:

Pattern = ab<>ba<>c

Text = cacbccbacbacab

- Segments: [ab, ba, c]
- Found in order as "cab ... ba ... cab" \rightarrow Match exists.

8. Find the KMP prefix function for the string 'aabbaabacab'.

Let's compute the KMP prefix function (π array) for the string:

Pattern = aabbbaabacab (length = 12).

We maintain:

- $\pi[i]$ = length of longest proper prefix of $P[0..i]$ that is also a suffix.
- Variable k = current longest prefix-suffix length.

i	P[i]	Substring (P[0..i])	$\pi[i]$
0	a	a	0
1	a	aa	1
2	b	aab	0
3	b	aabb	0
4	b	aabb	0
5	a	aabbba	1
6	a	aabbbaa	2
7	b	aabbbaab	3
8	a	aabbbaaba	4
9	c	aabbbaabac	0
10	a	aabbbaabaca	1
11	b	aabbbaabacab	2

Final Prefix Function

$\pi=[0,1,0,0,0,1,2,3,4,0,1,2]$

10. How is string matching automaton computation difference from the computation of prefix function in KMP algorithm?

Difference between Automaton vs KMP Prefix Function

1. KMP Prefix Function (π array)

- Stores only the **longest proper prefix-suffix lengths** for each prefix of pattern.
- Used to decide how far to "jump back" when mismatch occurs.
- Space: $O(m)$, Time: $O(m)$ to compute.

2. String Matching Automaton

- Precomputes **full transition table** $\delta(\text{state}, \text{character})$ for every state (0..m) and every alphabet symbol.
- Gives *direct next state* after reading a character (no backtracking needed).
- Space: $O(m \cdot |\Sigma|)$, Time: $O(m \cdot |\Sigma|)$ to build.

Key Point

- **Prefix function** = compact information (only needed for mismatch handling).
- **Automaton** = full deterministic machine (explicit transitions for all characters).

👉 In practice:

- KMP (π array) is **more space-efficient**.
- Automaton is **faster at runtime** (direct transitions, but bigger preprocessing).

Perfect 👍 Let's take the small pattern:

Pattern = "abab"

Alphabet = {a, b}

Length = 4

1. KMP Prefix Function (π array)

We compute longest prefix = suffix for each prefix:

- $\pi[0] = 0$ (a)
- $\pi[1] = 0$ (ab)
- $\pi[2] = 1$ (aba, prefix "a")
- $\pi[3] = 2$ (abab, prefix "ab")

✓ So:

$\pi=[0,0,1,2]$

2. String Matching Automaton

We build $\delta(q, a) \rightarrow$ next state.

State q	on a	on b
0	1	0
1	1	2
2	3	0
3	1	4
4	—	— (final)

✓ Final state = 4 (pattern matched).

11. Prove that, for any x and character a, we have $\sigma(xa) \leq \sigma(x)+1$

Alright 👍 This is a property of the **string matching automaton** (σ = transition function).

Recall

- $\sigma(x)$ = length of the longest prefix of the pattern P that is also a suffix of string x .
- For any string x and character a , $\sigma(xa)$ is the length of the longest prefix of P that is a suffix of xa .

Proof

1. Let $q = \sigma(x)$.

$\Rightarrow P[1..q]$ is the longest prefix of P that is a suffix of x .

2. Now append character a to form xa .

- a. The new match length can increase **by at most 1**, because we only added **one character**.
- b. If $P[q+1]=a$, then the match extends:

$$\sigma(xa) = q + 1 = \sigma(x) + 1$$

3. If $P[q+1] \neq a$, then the match cannot extend, and $\sigma(xa) \leq q$.

Therefore

$$\sigma(xa) \leq \sigma(x) + 1$$

✓ Intuition: Appending one character can at best increase the matched prefix by **one**, never more.

12. Prove that, for any string x and character a, if $q = \sigma(x)$, then $(xa) = \sigma(Pqa)$.

Recall

- $\sigma(x)$ = length of the longest prefix of pattern P that is also a suffix of x .
- If $q = \sigma(x)$, then $P[1..q]$ is the longest prefix of P matching the suffix of x .
- We want to show:

$$\sigma(xa) = \sigma(Pqa)$$

where $Pq = P[1..q]$.

Proof

1. By definition of $q = \sigma(x)$:
 - a. Suffix of x of length $q = P[1..q]$.
 - b. No longer prefix of P matches a suffix of x .
2. Append character a :
 - a. Now, the suffix of xa may extend the match.
 - b. Which prefix of P can be suffix of xa ? It must also be a suffix of Pqa (since x 's longest suffix already equals Pq).

3. Thus, the length of the longest prefix of P that is suffix of xa = the length of the longest prefix of P that is suffix of $P_q a$.

Therefore

$$\sigma(xa) = \sigma(P_q a)$$

Intuition: To know how the automaton extends on xa , it is enough to know the state $q = \sigma(x)$. That's why the automaton can be defined with only $m+1$ states instead of infinitely many strings.

13. If φ is the final-state function of a string matching automaton for a given pattern p and $T[1..n]$ is an input text for automaton, then $\varphi(T_i) = \sigma(T_i)$.

Recall

- $\sigma(x)$: length of the longest prefix of pattern P that is also a suffix of string x .
- In a **string matching automaton**:
 - States are numbered $0, 1, 2, \dots, m = |P|$.
 - $\delta(q, a)$ is the transition function.
 - The **final-state function** $\varphi(x) =$ state reached after feeding string x to the automaton, starting at state 0.

Proof

1. **Base Case:** Start with empty string.
 - a. $\varphi(\epsilon) = 0$.
 - b. Holds true.
2. **Inductive Hypothesis:** Assume for some string x

$$\varphi(x) = \sigma(x)$$
3. **Inductive Step:** Append a character a .

- a. By automaton definition:

$$\varphi(xa) = \delta(\varphi(x), a)$$

- b. By induction, $\varphi(x) = \sigma(x)$. So:
c. $\varphi(xa) = \delta(\sigma(x), a)$
d. From the property we proved earlier:

$$\sigma(xa) = \sigma(P\sigma(x)a) = \delta(\sigma(x), a)$$

- e. Thus:

$$\varphi(xa) = \sigma(xa)$$

3. By induction, holds for all prefixes $T_i = T[1..i]$.

Conclusion

$$\varphi(T_i) = \sigma(T_i) \text{ for all } i = 1, \dots, n$$

Decision Tree

1. Define decision tree. Construct the decision tree for the following Boolean functions: (i) A XOR B (ii) A B U C.

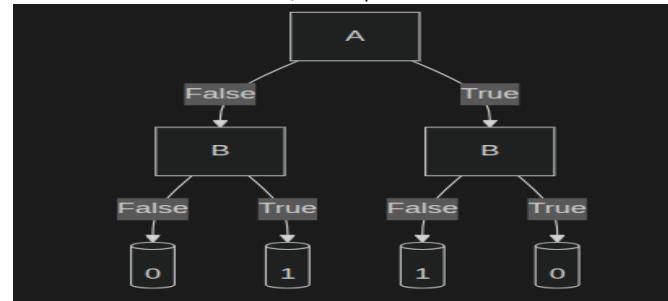
Definition of a Decision Tree

A **Decision Tree** is a graphical model used for representing a Boolean function. It is a rooted tree where:

1. **Internal Nodes:** Represent input variables (attributes). They test the value of a single variable and branch out based on the outcome (typically True or False).
2. **Edges (Branches):** Represent the outcome of the test on the parent node. For a Boolean variable, there are two edges: one for the True case and one for the False case.
3. **Leaf Nodes (Terminal Nodes):** Represent the final output or decision of the function (e.g., True or False, 1 or 0).

The path from the root to a leaf corresponds to a specific set of conditions that lead to the output value in that leaf.

- If A is True (the right branch):
 - We must then test B.
 - If B is False, the output is 1.
 - If B is True, the output is 0.



Explanation: This tree cannot be simplified further. Unlike the AND or OR functions, you cannot determine the output of XOR by looking at just one variable; you must always know the values of both A and B.

Constructing the Decision Trees

(i) Decision Tree for A XOR B

The **XOR** (Exclusive OR) function, A XOR B, is True if and only if the two inputs are different.

Truth Table:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Construction: We start by testing variable A.

- If A is False (the left branch):
 - We must then test B.
 - If B is False, the output is 0.
 - If B is True, the output is 1.

(ii) Decision Tree for A · (B + C)

First, let's clarify the function. The notation A B U C is a mix of Boolean algebra and set theory. Interpreting it as a Boolean function:

- A B typically means A AND B ($A \cdot B$).
- U means OR (+).

So, the function is $A \cdot B + C$ (A AND B OR C).

Truth Table:

A	B	C	$A \cdot B + C$
0	0	0	0

0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Construction & Optimization: A key feature of decision trees is that they can be optimized to avoid unnecessary tests. A good strategy is to test the most "decisive" variable first.

Here, C is a good candidate because if C is True, the entire expression is True regardless of A or B.

1. Test C first:

- a. If C is True (right branch), the output is immediately 1. We don't need to check A or B.
- b. If C is False (left branch), the function simplifies to $A \cdot B$. We now need to test A.

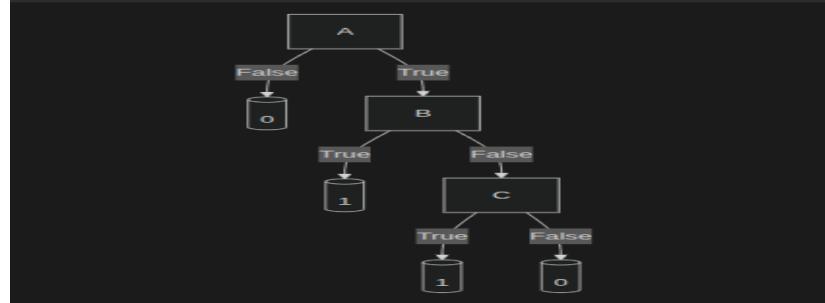
2. Test A (on the C=False branch):

- a. If A is False, the output is 0 (because $A \cdot B$ is false if A is false).
- b. If A is True, the function simplifies to B. We now need to test B.

3. Test B (on the A=True, C=False branch):

- a. If B is False, the output is 0.
- b. If B is True, the output is 1.

Decision Tree Diagram:



Explanation: This optimized tree shows that you don't always need to check all variables. If C is True, you get the answer immediately. This is a more efficient representation of the Boolean function than a full tree that tests all variables on every path.

2. What are the different terminating conditions for decision trees? Explain your answer.

The terminating conditions can be broadly categorized into conditions that lead to a **"Pure Leaf"** and those that lead to a **"Impure but Stopped Leaf"**.

Here are the primary terminating conditions:

1. The Node is Pure (All Data Belongs to One Class)

This is the most straightforward and ideal stopping condition.

- **Explanation:** When all the data points in a current node have the same value for the target variable, there is no further uncertainty to resolve. Splitting this node would be pointless.
- **Example:** In a node with 50 samples, if all 50 are classified as "Spam", the node is declared a "Spam" leaf.
- **Result:** The splitting process stops for this branch.

2. No Information Gain from Any Possible Split

This is a core condition based on the very purpose of a decision tree.

- **Explanation:** The algorithm searches for the split that maximizes "Information Gain" (or minimizes "Impurity" like Gini Index or Entropy). If no split can be found that improves the purity of the child nodes compared to the parent node, splitting is terminated.
- **Why it happens:** This can occur when the features available are not powerful enough to separate the classes any further, or when the data is inherently noisy.
- **Result:** The node becomes a leaf, and its class is set to the majority class of the samples in that node.

3. The Tree Has Reached a Pre-defined Maximum Depth

This is a common **regularization** technique to prevent overfitting.

- Explanation:** You can set a hyperparameter `max_depth` that limits how deep the tree can grow. A node that reaches this depth will not be split further, even if it's impure and good splits are available.
- Analogy:** It's like limiting the number of "why?" questions you can ask to avoid getting into overly specific details.
- Result:** The node becomes a leaf.

4. The Number of Samples in a Node is Below a Minimum Threshold

This prevents the tree from learning from too few samples, which are often just noise.

- Explanation:** You can set a hyperparameter like `min_samples_split`. If the number of samples in a node is less than this threshold, the node will not be split.
- Purpose:** This stops the tree from creating highly specific rules that model the random noise in the training data rather than the underlying pattern.
- Result:** The node becomes a leaf.

5. The Number of Samples in a Resulting Child Node Would Be Too Small

Similar to the previous condition, but it applies to the *result* of a potential split.

- Explanation:** You can set a hyperparameter like `min_samples_leaf`. The algorithm will only consider a split if each resulting child node would have at least this many samples. If no split can meet this criterion, the node is not split.
- Purpose:** Ensures that the leaf nodes have a sufficient number of samples to make a statistically reliable prediction.

6. All Features Have Been Used on the Path from the Root

(More relevant for specific implementations or when features are non-repeating)

- Explanation:** In some implementations, a feature is only used once per branch/path. If a node has no new features left to use for splitting, it must terminate.
- Note:** This is less common in modern implementations (like scikit-learn) for non-categorical data, as they can use the same numeric feature multiple times with different split thresholds.

Summary and Analogy

Think of it like this: You're trying to classify animals.

- Pure Node:** You have a group of only "Cats". You stop. ✓
- No Information Gain:** You have a mixed group of "Dogs" and "Cats". The only feature left to ask about is "Number of Legs". All animals have 4 legs, so this question doesn't help you separate them. You stop and go with the majority. ☹
- Maximum Depth:** You decided you would only ask a maximum of 3 questions. You've asked your 3rd question, so you stop. ↘
- Minimum Samples in Node:** You have a group of only 2 animals. This is too few to draw a reliable conclusion, so you stop. ↪
- Minimum Samples in Leaf:** A potential question would split the group into one large group and one tiny group with just 1 animal. You reject this split. Since no other good splits exist, you stop. 🎉

3. Build and explain the decision tree up to 2 levels for the following transactions table. What is the information gain of A₁ and A₂ relative to the transactions in the table?

3. (a) Build and explain the decision tree upto 2 levels for the following transactions :

Tid	Refund	Material Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Married	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- (b) What is the information gain of A₁ and A₂ relative to the transactions in the table?

Instance	1	2	3	4	5	6	7	8	9
A ₁	T	T	F	F	F	F	F	T	F
A ₂	T	T	F	F	T	T	F	F	T
Target Class	+	+	-	+	-	-	-	+	-

Deepseek(Decision Tree PYQ(SKS))

4. Define entropy in the context of decision trees.

In the context of decision trees and data science, **Entropy** is a metric used to measure the level of **impurity** or **uncertainty** in a dataset (or a subset of a dataset). The core idea is to quantify how "mixed" the classes are within a group of data points. A node in a decision tree is **pure** if it contains only data points belonging to a single class. In this case, the entropy is **zero**, and there is no uncertainty. Conversely, a node is **impure** if it contains a random mixture of points from different classes, leading to high entropy.

The Mathematical Formula

For a given dataset (or node) S with c total classes, the entropy E(S) is calculated as:

$$E(S) = - \sum_{i=1}^c p_i \log_2(p_i)$$

Where:

- p_i is the proportion (probability) of data points in S that belong to class i.
- log₂ is the base-2 logarithm, which gives the result in **bits**, the standard unit of information.

Key Interpretation of the Formula

- Pure Node (Entropy = 0):** This occurs when one p_i = 1 and all others are 0.
 - Example: A node with 10 "Yes" and 0 "No". p_{yes} = 1, p_{no} = 0.
 - Calculation: E(S) = -[1.log₂(1) + 0.log₂(0)] = -[1.0 + 0] = 0
 - (Note: By convention, 0.log(0) is taken as 0).
- Maximum Impurity (Entropy = 1, for binary classification):** This occurs when the classes are perfectly mixed.
 - Example: A node with 5 "Yes" and 5 "No". p_{yes} = 0.5, p_{no} = 0.5.

- b. Calculation: $E(S) = -[0.5.\log_2(0.5) + 0.5.\log_2(0.5)] = -[0.5 \cdot (-1) + 0.5 \cdot (-1)] = -[-1] = 1$
3. **Intermediate Impurity (0 < Entropy < 1):** This occurs when there is a mix of classes, but one is more frequent than the other.
- Example: A node with 7 "Yes" and 3 "No". $p_{\text{yes}} = 0.7$, $p_{\text{no}} = 0.3$.
 - Calculation: $E(S) = -[0.7.\log_2(0.7) + 0.3.\log_2(0.3)] \approx -[0.7 \cdot (-0.5146) + 0.3 \cdot (-1.737)] \approx -[-0.3602 - 0.5211] \approx 0.881$

Why is Entropy Used in Decision Trees?

The primary goal of a decision tree is to split the data into purer and purer subsets. Entropy is the measure used to achieve this through a concept called **Information Gain (IG)**.

- Information Gain** is calculated as the difference between the entropy of the parent node and the weighted average entropy of the child nodes after a split.

$$IG(S, A) = E(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} E(S_v)$$

- feature we are splitting on, and (S_v) is the subset of data for each value (v) of that feature.
- The Splitting Criterion:** The decision tree algorithm evaluates all possible features and split points, and it chooses the one that provides the **highest Information Gain**. A high Information Gain means the split has effectively reduced uncertainty the most.

Analogy

Imagine a bag of marbles.

- Low Entropy (Pure):** A bag with only red marbles. You are certain you will pick a red marble. Entropy = 0.
- High Entropy (Impure):** A bag with an equal number of red and blue marbles. You are very uncertain about which color you will pick. Entropy = 1 (maximum for binary).
- Medium Entropy:** A bag with 8 red and 2 blue marbles. You are fairly certain you'll pick a red marble, but there's still some uncertainty. Entropy is between 0 and 1.

5. Discuss how the depth of a decision tree affects the trade-off between bias and variance.

Core Concepts: Bias and Variance

First, let's quickly define these terms in the context of model prediction error:

- Bias:** The error due to overly simplistic assumptions in the model. A high-bias model fails to capture the underlying patterns in the data (underfitting). Think of it as being consistently wrong.
- Variance:** The error due to excessive sensitivity to small fluctuations in the training data. A high-variance model learns the noise in the training data as if it were a true pattern, leading to poor performance on new data (overfitting). Think of it as being wildly inconsistent.

The total error of a model is often decomposed into **Bias² + Variance + Irreducible Error**.

The Effect of Decision Tree Depth

The **depth** of a tree determines how many sequential "questions" it can ask before making a prediction. This directly controls its complexity.

1. Shallow Trees (Low Depth)

- Complexity: Low**
 - The model is too simple. It can only make broad, general rules and fails to capture important patterns and interactions in the data. For example, a very shallow tree might only split on one feature, missing all other relevant relationships.
- Variance: Low**
 - Because the model is simple and not very sensitive to the specific training set, different training sets will result in very similar simple trees. It is stable but inaccurate.
- Result: Underfitting**
 - The model performs poorly on both the training data and the test data.

2. Deep Trees (High Depth)

- Complexity: High**
- Bias: Low**
 - The model is highly complex. It can create very specific rules, following the intricate patterns in the training data. It can often achieve perfect or near-perfect accuracy on the training set.
- Variance: High**
 - The model becomes extremely sensitive to the training data. It starts to learn the noise and outliers as if they were true concepts. A slightly different training set could produce a completely different tree with different rules at the deep levels.
- Result: Overfitting**
 - The model performs exceptionally well on the training data but poorly on the unseen test data because it has "memorized" the training set instead of "learning" the generalizable concepts.

3. optimally Tuned Depth

- Complexity: Moderate**
- Bias: Moderate/Low**
 - The model is complex enough to capture the true, underlying patterns in the data.
- Variance: Moderate/Low**
 - By limiting the depth, we prevent the tree from becoming so specific that it learns the noise. The model remains relatively stable across different datasets.
- Result: Good Generalization**

- This is the sweet spot. The model performs well on both the training data and, more importantly, on the unseen test data. The total error ($\text{Bias}^2 + \text{Variance}$) is minimized.

How to find the optimal depth in practice:

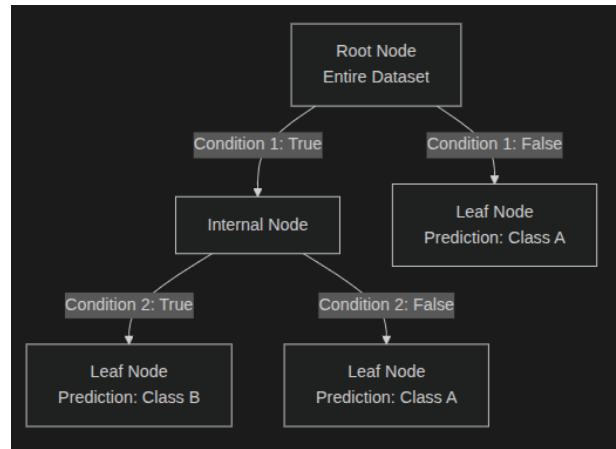
1. **Use a Validation Set:** Train trees of different depths and evaluate their performance on a validation set (data not seen during training).
2. **Cross-Validation:** Use k-fold cross-validation to get a robust estimate of performance for each depth.
3. **Observe Learning Curves:** Plot the training and validation scores against the tree depth.
 - a. As depth increases, the **training score** will keep improving.
 - b. The **validation score** will first improve and then start to deteriorate.
 - c. The optimal depth is around the point where the validation score is maximized, just before it starts to fall due to overfitting.

5. Describe the basic structure of a decision tree and how it makes predictions.

The Basic Structure of a Decision Tree

A decision tree is a flowchart-like structure that breaks down a dataset into smaller and smaller subsets while simultaneously developing the tree. It is composed of three main types of nodes:

1. **Root Node:** This is the topmost node in the tree. It represents the entire dataset, which is then split into two or more homogeneous sets (child nodes) based on the most significant feature.
2. **Internal Nodes (Decision Nodes):** These nodes represent a decision (or a test) on a specific feature. Each internal node splits the data further into smaller subsets. The branches emanating from an internal node represent the outcome of the test (e.g., "Yes"/"No", "True"/"False", or a value range).
3. **Leaf Nodes (Terminal Nodes):** These are the final nodes at the ends of the branches. They represent the final **prediction** (or outcome) and do not split any further. In classification, a leaf node holds a class label (e.g., "spam" or "not spam"). In regression, it holds a continuous numerical value (e.g., a house price of \$350,000).



How a Decision Tree Makes Predictions

Making a prediction for a new, unseen data point is a straightforward process of "routing" the data point down the tree from the root to a leaf.

The Process:

1. **Start at the Root Node:** Begin with the root node and evaluate the condition (the feature test) on the new data point.
2. **Follow the Branch:** Based on the outcome of the test (e.g., "Is Age > 30?" → Yes or No), follow the corresponding branch to the next node.
3. **Repeat:** If the next node is an internal node, repeat steps 1 and 2. Evaluate the new condition for the data point at that node and follow the appropriate branch.
4. **Reach a Leaf Node:** Continue this process until you reach a leaf node.
5. **Make the Prediction:** The value (class label or numerical value) stored in that leaf node is the final prediction for the data point.

A Concrete Example: Classifying an Email

Let's imagine a simple decision tree built to classify emails as "Spam" or "Not Spam."

The Tree Structure:

- **Root Node:** Tests: Does the email contain the word "Winner!"?
 - **Branch: Yes -> Internal Node:** Tests: Does the sender's address contain a known domain (e.g., @gmail.com)?
 - **Branch: Yes -> Leaf Node:** Prediction: Not Spam
 - **Branch: No -> Leaf Node:** Prediction: Spam
 - **Branch: No -> Leaf Node:** Prediction: Not Spam

Making a Prediction for a New Email:

Suppose we receive a new email with the text "You are a winner!" from the address prize@random-lottery.com.

- Root Node:** The email *does* contain the word "Winner!". So, we follow the "Yes" branch.
- Internal Node:** We check the sender's domain. It is "[@random-lottery.com](#)", which is not a known, trusted domain. So, we follow the "No" branch.
- Leaf Node:** We arrive at a leaf node with the prediction **Spam**.

The decision tree has successfully classified the new email as spam.

6.Explain how information gain is calculated and its role in decision tree splitting.

Information Gain (IG) is a metric used to decide which feature to split on at a given node in a decision tree. It measures how much "information" a feature gives us about the target class.

The core idea is simple: **We want to choose the feature that reduces uncertainty the most.** A high information gain means the split creates child nodes that are purer (less disordered) than the parent node.

How Information Gain is Calculated

The calculation of Information Gain is based on the concept of **Entropy**, which measures impurity or disorder. The formula for Information Gain is:

$$\text{Information Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \cdot \text{Entropy}(S_v)$$

Where:

- S is the original dataset (the parent node).
- A is the specific feature we are considering to split on.
- Values(A) are all the possible values for feature A.
- S_v is the subset of S where feature A has the value v.
- |S_v| is the number of samples in S_v.
- |S| is the total number of samples in S.
- |S_v|/|S| is the weight (proportion) of samples that have value v for feature A.

Let's go through the steps in detail.

Step 1: Calculate the Entropy of the Parent Node (Entropy(S))

First, we need to know how impure our starting point is. Entropy for a dataset (S with c classes) is calculated as:

$$\text{Entropy}(S) = - \sum_{i=1}^c p_i \log_2(p_i)$$

where p_i is the proportion of samples in class i.

Example: If a node has 10 samples: 6 "Yes" and 4 "No". [$p_{\text{yes}} = 6/10 = 0.6$, $p_{\text{no}} = 4/10 = 0.4$] [$\text{Entropy}(S) = - (0.6 * \log_2(0.6)) + 0.4 * \log_2(0.4) \approx 0.971$]

Step 2: Calculate the Weighted Average Entropy After the Split

Now, for each candidate feature A, we imagine splitting the data.

1. We split the parent node S into subsets S_v based on the values of feature A.
2. We calculate the entropy for each of these child nodes S_v.
3. We take the average of these child entropies, weighted by the number of samples in each child node.

This gives us the expected entropy *after* the split.

$$\text{Weighted Avg. Entropy} = \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \cdot \text{Entropy}(S_v)$$

Step 3: Calculate Information Gain

Finally, we subtract this weighted average entropy from the original parent entropy.

$$IG = \text{Entropy}(\text{Before Split}) - \text{Weighted Avg. Entropy}(\text{After Split})$$

A high IG means the split was very effective at reducing uncertainty. An IG of 0 means the split did nothing to improve purity.

The Role of Information Gain in Decision Tree Splitting

Information Gain plays the **starring role** in the tree-building algorithm (like ID3). Its function is:

1. **The Splitting Criterion:** At every node, the algorithm calculates the Information Gain for **every possible feature** (and for continuous features, every possible split point).
2. **Feature Selection:** The algorithm **selects the feature with the highest Information Gain** to split the node.
3. **Recursion:** This process is repeated recursively for each of the resulting child nodes until a stopping condition is met (e.g., a node becomes pure, maximum depth is reached, or IG is too low).

Why is this effective? By always choosing the split that maximizes IG, the tree prioritizes the features that give the most "bang for the buck" in terms of reducing uncertainty and creating pure nodes. This leads to a simpler, more efficient, and generally more accurate tree.

7.What are the different terminating conditions during the generation of a decision tree?

Here are the primary terminating conditions, categorized logically:

1. Node Purity Conditions

These conditions stop splitting when a node is sufficiently "pure" (homogeneous).

- **The Node is Perfectly Pure:** This is the most straightforward condition.
 - **Condition:** All samples in the node belong to the same class.

- **Why Stop?** There is no uncertainty left to resolve. Any further split would be unnecessary.
- **Example:** A node with 50 samples, all classified as "Spam".
- **The Node Reaches a Minimum Impurity Threshold:**
 - **Condition:** The impurity (Gini Index or Entropy) of the node falls below a pre-defined threshold.
 - **Why Stop?** The node is considered "pure enough," and further splitting is unlikely to yield significant improvement, helping to avoid overfitting to noise.

2. Data Availability Conditions

These conditions stop splitting when there isn't enough data to make a reliable, generalizable split.

- **Number of Samples in the Node is Too Small to Split:**
 - **Condition:** The number of samples in the node is less than a pre-set `min_samples_split` parameter (e.g., 5).
 - **Why Stop?** Splitting a very small number of samples can lead to leaf nodes that model random noise in the training data rather than the true underlying pattern.
- **A Split Would Create a Child Node That is Too Small:**
 - **Condition:** The best possible split would result in at least one child node with fewer samples than the `min_samples_leaf` parameter (e.g., 3).
 - **Why Stop?** This ensures that the leaf nodes have a sufficient number of samples to make a statistically reliable prediction. It prevents the tree from creating highly specific rules based on just one or two data points.

3. Tree Complexity Conditions

These are direct regularization techniques to limit the overall size of the tree.

- **The Tree Reaches the Maximum Depth:**
 - **Condition:** The node is at a depth equal to the pre-defined `max_depth`.
 - **Why Stop?** This is one of the most effective ways to control overfitting. It limits the number of questions the tree can ask, forcing it to learn broader, more generalizable patterns.
- **The Number of Leaf Nodes Reaches a Limit:**
 - **Condition:** The total number of leaf nodes in the tree reaches `max_leaf_nodes`.
 - **Why Stop?** This provides an alternative way to control the tree's complexity by directly limiting its final size, encouraging the algorithm to find the most important splits first.

4. Split Usefulness Conditions

These conditions stop splitting when no worthwhile split can be found.

- **No Significant Information Gain:**
 - **Condition:** The Information Gain (or decrease in Gini Impurity) from the best possible split is below a certain threshold (`min_impurity_decrease`).
 - **Why Stop?** If no split provides a meaningful reduction in impurity, splitting is pointless. This prevents the tree from making splits that offer negligible predictive benefit, which is often a sign of fitting noise.

- **No Valid Splits Remain:**

- **Condition:** All features have identical values for the samples in the node, or there are no features left to split on (though this is rare in most implementations).
- **Why Stop?** There is no feature that can be used to further separate the samples.

Practical Implication: The Bias-Variance Trade-off

Tuning these termination criteria is essentially tuning the **bias-variance trade-off**:

- **Loose Conditions** (e.g., high `max_depth`, low `min_samples_leaf`): Allow the tree to grow deep and complex. This **lowers bias** but **increases variance**, leading to **overfitting**.
- **Strict Conditions** (e.g., low `max_depth`, high `min_samples_leaf`): Restrict the tree's growth. This **lowers variance** but **increases bias**, leading to **underfitting**.

In practice, these parameters are often treated as hyperparameters and optimized using techniques like cross-validation to find the right balance for a given dataset.

- 8. Consider the dataset below to learn a decision tree that predicts if people pass machine learning (Yes or No), based on their previous GPA (High, Medium, or Low), and whether or not they studied.**

(For this problem, you can write your answers using \log_2 , but it may be helpful to note that $\log_2 3 \approx 1.6$)

GPA	Studied	Passed
L	F	F
L	T	T
M	F	F
M	T	T
H	F	T
H	T	T

- (a) What is the entropy $H(\text{Passed})$?
(b) What is the entropy $H(\text{Passed} | \text{GPA})$?
(c) What is the entropy $H(\text{Passed} | \text{Studied})$?
(d) Draw the entire decision tree that would be learnt for this dataset.

Deepseek(Decision Tree PYQ(SKS))

Second Generation Cellular Networks –2G :

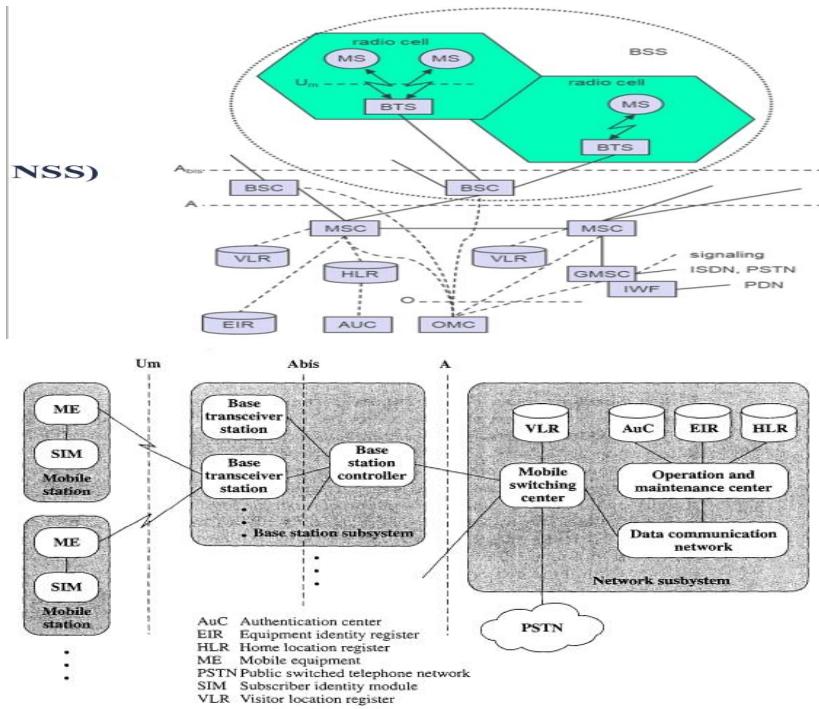


Figure 10.13 Overall GSM Architecture

• Complete GSM Network Architecture (2G)

- The full system can be divided into the three subsystems from the PDF, plus the added components you mentioned.

- **1. Mobile Station (MS):** A Mobile Station (MS) in a GSM system includes all the hardware and software required for communication with the network. It consists of the mobile device itself and a **Subscriber Identity Module (SIM)**, which stores user-specific data such as the **International Mobile Subscriber Identity (IMSI)**, **Authentication Key (Ki)**, **Personal Identification Number (PIN)**, **PIN Unblocking Key (PUK)**, and details of subscribed services. The mobile device is identified by the **International Mobile Equipment Identity (IMEI)**, while the user is identified through the SIM, which enables personalized services like authentication and billing. Without the SIM card, only emergency calls are possible.

The Mobile Station temporarily stores data such as the **Cipher Key (Kc)** and location information consisting of the **Temporary Mobile Subscriber Identity (TMSI)** and **Location Area Identification (LAI)** while connected to the GSM network. Typical transmit power is up to **2 watts** for GSM operating in the 900 MHz band and **1 watt** for GSM operating in the 1800 MHz band, due to smaller cell sizes.

Modern Mobile Stations offer various interfaces such as displays, loudspeakers, microphones, Bluetooth, and computer modems. They also include additional features such as cameras, fingerprint sensors, calendars, address books, games, and internet browsers. A Mobile Station may exist as a mobile phone, a **Personal Digital Assistant (PDA)** with phone capability, or be integrated into vehicles or tracking systems for applications such as container monitoring.

- **Mobile Equipment (ME):** The physical phone device.
- **Subscriber Identity Module (SIM):** Smart card containing the user's identity (IMSI), authentication key (Ki), and other data.
- **2. Base Station Subsystem (BSS):** A GSM network comprises many BSSs, each controlled by a base station controller (BSC). The BSS performs all functions necessary to maintain radio connections to an MS, coding/decoding of voice, and rate adaptation to/from the wireless network part. Besides a BSC, the BSS contains several BTSs.
- **Base Transceiver Station (BTS):** A BTS comprises all radio equipment, i.e., antennas, signal processing, amplifiers necessary for radio transmission. A BTS can form a radio cell or, using sectorized antennas, several cells (see section 2.8), and is connected to MS via the Um interface (ISDN U interface for mobile use), and to the BSC via the Abis interface. The Um interface contains all the mechanisms necessary for wireless transmission (TDMA, FDMA etc.) and will be discussed in more detail below. The Abis interface consists of 16 or 64 kbit/s connections. A GSM cell can measure between some 100 m and 35 km depending on the environment (buildings, open space, mountains etc.) but also expected traffic.

- **Base Station Controller (BSC):** The BSC basically manages the BTSs. It reserves radio frequencies, handles the handover from one BTS to another within the BSS, and performs paging of the MS. The BSC also multiplexes the radio channels onto the fixed network connections at the A interface.
- **3. Network Switching Subsystem (NSS) - The "Core Network"**
- **Mobile Switching Center (MSC):** The central switch for call routing and mobility management. It is the link between the cellular network and external fixed-line networks.
- **Gateway MSC (GMSC):** A specific MSC that routes calls *into* the GSM network from external networks (like PSTN).
- **Home Location Register (HLR):** The HLR is the most important database in a GSM system as it stores all user-relevant information. This comprises static information, such as the mobile subscriber ISDN number (MSISDN), subscribed services (e.g., call forwarding, roaming restrictions, GPRS), and the international mobile subscriber identity (IMSI). Dynamic information is also needed, e.g., the current location area (LA) of the MS, the mobile subscriber roaming number (MSRN), the current VLR and MSC. As soon as an MS leaves its current LA, the information in the HLR is updated. This information is necessary to localize a user in the worldwide GSM network. All these user-specific information elements only exist once for each user in a single HLR, which also supports charging and accounting. The parameters will be explained in more detail in section 4.1.5. HLRs can manage data for several million customers and contain highly specialized data bases which must fulfill certain real-time requirements to answer requests within certain time-bounds.
- **Visitor Location Register (VLR):** One important, temporary piece of information is the location of the subscriber. The location is determined by the VLR into which the subscriber is entered. The visitor location register maintains information about subscribers that are currently physically in the region covered by the switching center. It records whether or not the subscriber is active and other parameters associated with the subscriber. For a call coming to the subscriber, the system uses the telephone number associated with the subscriber to identify the home switching center of the subscriber. This switching center can find in its HLR the switching center in which the subscriber is currently physically located. For a call coming from the subscriber, the VLR is used to initiate the call. Even if the subscriber is in the area covered by its home switching center, it is also represented in the switching center's VLR, for consistency.
- **Authentication center database (AuC):** This database is used for authentication activities of the system; for example, it holds the authentication and encryption keys for all the subscribers in both the home and visitor location registers. The center controls access to user data as well as being used for authentication when a subscriber joins a network. GSM transmission is encrypted, so it is private. A stream cipher, A5, is used to encrypt the transmission from subscriber to base transceiver.

However, the conversation is in the clear in the landline network. Another cipher, A3, is used for authentication.

- **Equipment identity register database (EIR):** The EIR keeps track of the type of equipment that exists at the mobile station. It also plays a role in security (e.g., blocking calls from stolen mobile stations and preventing use of the network by stations that have not been approved).
- **4. Operation and Support Subsystem (OSS)**
- **Operation and Maintenance Center (OMC):** This is the crucial component you noted. The OMC is the centralized management system for the entire network. It is used for:
 - **Network Monitoring:** Monitoring the performance and status of BTS, BSC, and MSC.
 - **Fault Management:** Detecting, logging, and diagnosing network faults.
 - **Configuration Management:** Software updates and hardware configuration for network elements.
 - **Security Management:** Managing access controls and security logs.

Interconnection with External Networks

- The GSM network does not operate in isolation. The MSC connects to various external networks, which you also correctly identified:
- **PSTN (Public Switched Telephone Network):** The traditional circuit-switched telephone network used for landline calls. The MSC allows mobile subscribers to call landline numbers and vice-versa.
- **ISDN (Integrated Services Digital Network):** A digital telephone network that can carry voice, data, and video over digital lines. GSM was designed to interoperate with ISDN, hence the use of ISDN-based signaling protocols (SS7) in its core.
- **PDN (Public Data Network):** This is a general term for a network providing data services to the public. The most common example is the **Internet**. In a GSM context, data calls from a mobile phone could be routed to a PDN via special interworking functions (IWF) at the MSC.

Pic(SK folder 2G.PNG)

MDR (Maximum Data Rate) Formula

Formula: $MDR = B \cdot \log(1 + SNR)$

What it represents:

- **MDR:** Maximum Data Rate (theoretical capacity of a channel)
- **B:** Bandwidth (in Hz) - the radio spectrum allocated
- **SNR:** Signal-to-Noise Ratio - measures signal quality
- **log:** Logarithm function (usually base 2 for bits)

Function in GSM:

- Determines channel capacity for each physical channel (200 kHz carrier + time slot)
- Explains why GSM has specific data rates:
 - TCH/Full Rate: 22.8 kbps
 - TCH/Half Rate: 11.4 kbps
- Helps in network planning to ensure sufficient SNR for quality service

Example:

If a GSM channel has 200 kHz bandwidth and good SNR, this formula calculates the maximum possible data rate that can be achieved.

_REUSE DISTANCE FORMULA

Formula: $D^2 = 3R^2N$

- D: Reuse Distance (distance between cells using the same frequencies)
- R: Cell Radius
- N: Cluster Size (number of cells in a frequency reuse pattern)

Function in GSM:

- Prevents interference: Ensures cells using same frequencies are far enough apart
- Determines frequency planning: Helps design the hexagonal cell pattern
- Relates to capacity vs. quality trade-off:
 - Small N (e.g., N=3) → More capacity but higher interference
 - Large N (e.g., N=7) → Less capacity but better signal quality

Practical Example:

If GSM uses 200 kHz channels and Cluster Size N=7:

$$\begin{aligned}D^2 &= 3R^2 \times 7 = 21R^2 \\D &= \sqrt{21} \times R \approx 4.58R\end{aligned}$$

- Same frequencies reused every 4.58 cell radii apart
- This ensures acceptable interference levels while maximizing capacity

CALL SETUP IN 2G GSM:

The diagram on Page 12 shows three main call scenarios:

1. Mobile Terminated Call (Incoming Call to GSM Subscriber)

Path: PSTN → GMSC → HLR → VLR → MSC → BSC → MS

Step-by-Step Process:

1. Step 1: Calling party dials a GSM subscriber's number from PSTN
2. Step 2: Call is forwarded to the Gateway MSC (GMSC)
3. Step 3: GMSC signals call setup to HLR to locate the subscriber

4. Step 4-5: HLR connects with current VLR to get subscriber location
5. Step 6-7: Forward call to responsible MSC serving the subscriber
6. Step 8-9: Get current status of Mobile Station (available/busy)
7. Step 10-11: Paging of MS through BSC/BTS
8. Step 12-13: MS answers the paging request
9. Step 14-15: Security checks (authentication & encryption)
10. Step 16-17: Set up the connection and start ringing

2. Mobile Originated Call (Outgoing Call from GSM Subscriber)

Path: MS → BSC → MSC → External Network

Step-by-Step Process:

1. Step 1-2: MS sends connection request via RACH (Random Access Channel)
2. Step 3-4: Network performs security check (authentication)
3. Step 5-8: Check available resources (free circuits)
4. Step 9-10: Set up the call to the destination

KEY COMPONENTS INVOLVED

From PSTN Side:

- **PSTN:** Public Switched Telephone Network (landline network)
- **GMSC:** Gateway MSC (entry point to GSM network)

From GSM Core Network:

- **HLR:** Home Location Register (subscriber's master database)
- **VLR:** Visitor Location Register (subscriber's current location)
- **MSC:** Mobile Switching Center (call routing brain)

From GSM Access Network:

- **BSC:** Base Station Controller
- **BTS:** Base Transceiver Station
- **MS:** Mobile Station (phone + SIM)

SECURITY FEATURES HIGHLIGHTED

- **Authentication:** Step 14-15 in Mobile Terminated Call
- **Encryption:** Established during security checks
- **SIM-based security:** Uses Ki key from SIM card

CALL SETUP TIME

The entire process typically takes **2-5 seconds** from dialing to ringing, depending on network conditions.

VISUAL FLOW SUMMARY

Mobile Terminated Call:

PSTN → GMSC → HLR → VLR → MSC → BSC → MS

(Query) (Locate) (Page) (Connect)

Mobile Originated Call:

MS → BSC → MSC → External Network
(Request) (Auth) (Connect)

GSM Authentication Process:

Purpose:

- Verify subscriber identity
- Prevent unauthorized network access
- Protect against SIM cloning and fraud

Authentication Flow

Components Involved:

- **SIM Card:** In the Mobile Station
- **Authentication Center (AuC):** In the network
- **Mobile Switching Center (MSC):** Coordinates the process

Step-by-Step Process

Step 1: Challenge Generation

Network (AuC) → Mobile Station (SIM)

- AuC generates a **128-bit RAND** (Random Challenge Number)
- RAND is sent to the mobile station via MSC

Step 2: Response Calculation (SIM Side)

SIM Card Calculation: RAND (128-bit) + Ki (128-bit) → A3 Algorithm → SRES* (32-bit)

- **Ki:** Individual subscriber authentication key (stored in SIM)
- **A3 Algorithm:** Authentication algorithm
- **SRES*:** Signed Response (calculated by SIM)

Step 3: Response Calculation (Network Side)

AuC Calculation: RAND (128-bit) + Ki (128-bit) → A3 Algorithm → SRES (32-bit)

- AuC performs the **same calculation** using its stored Ki
- Generates expected response **SRES**

Step 4: Verification

MSC Comparison:

SRES* (from SIM) =? SRES (from AuC)

- Mobile sends SRES* back to network
- MSC compares SRES* with expected SRES
- **If match:** Authentication successful → Call continues
- **If no match:** Authentication failed → Call rejected

Key Security Elements

Ki (Subscriber Authentication Key)

- **128-bit secret key**
- **Stored in two places only:**
 - SIM card (physically secure)
 - Authentication Center (secure network database)
- **Never transmitted** over the air

RAND (Random Challenge)

- 128-bit random number
- Changes with each authentication attempt
- Prevents replay attacks

SRES (Signed Response)

- 32-bit response to challenge
- Proves knowledge of Ki without revealing it

Encryption Setup (After Authentication)

Once authenticated, the same RAND + Ki combination is used to generate the **encryption key (Kc)**:

RAND (128-bit) + Ki (128-bit) → A8 Algorithm → Kc (64-bit)

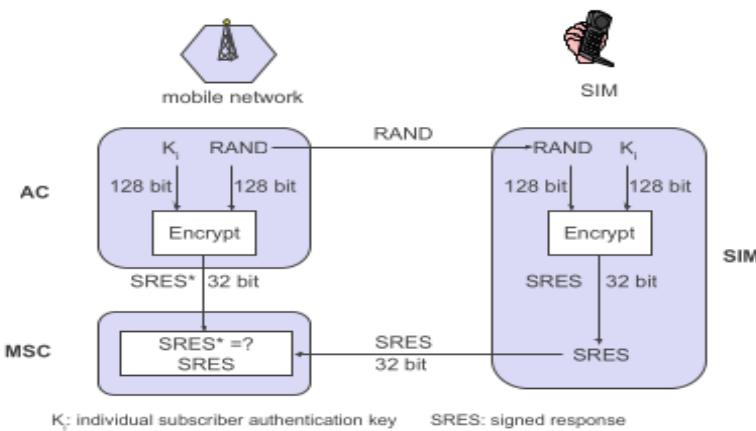
- **Kc:** Cipher Key for voice/data encryption
- Used with A5 algorithm for over-the-air encryption

Security Benefits

1. **Mutual Authentication:** Network authenticates subscriber, and subscriber trusts authenticated network
2. **No Key Transmission:** Ki never leaves SIM or AuC
3. **Dynamic Session Keys:** New Kc generated for each session
4. **Protection Against Cloning:** Without physical SIM, cannot clone subscriber identity

Limitations

- **One-way authentication:** Primarily network authenticates subscriber
- **Encryption not mandatory:** Some networks don't enable A5 encryption
- **64-bit Kc:** Considered weak by modern standards



GSM Physical Channel:

GSM uses a combination of **FDMA (Frequency Division Multiple Access)** and **TDMA (Time Division Multiple Access)** to create multiple physical channels.

GSM 900 Frequency Bands

Frequency Allocation:

- UPLINK: 890-915 MHz (Mobile to Base Station)
- DOWNLINK: 935-960 MHz (Base Station to Mobile)
- Total Bandwidth: $2 \times 25 \text{ MHz} = 50 \text{ MHz}$

Carrier Distribution:

Total Carriers: 125 carriers (each 200 kHz)

Usable Carriers: 124 carriers (1 carrier used as Guard Band)

Band Arrangement: 890.2 MHz, 890.4 MHz, ..., 914.8 MHz (Uplink)
935.2 MHz, 935.4 MHz, ..., 959.8 MHz (Downlink)

TDMA Structure

Time Slot Organization:

- Each Carrier = 200 kHz bandwidth
- Each Carrier divided into = 8 Time Slots (TS)
- Time Slot Duration = 0.577 ms
- Frame Duration (8 time slots) = 4.615 ms

Channel Capacity Calculation

Total Physical Channels: $124 \text{ carriers} \times 8 \text{ time slots} = 992 \text{ full-duplex channels}$

What is a Physical Channel?

A Physical Channel is defined by:

- Frequency (which 200 kHz carrier)
- Time Slot (which of the 8 time slots)
- Direction (Uplink/Downlink)

Example: Carrier #12, Time Slot #3 = One physical channel

Guard Channel

Purpose:

- 1 carrier (200 kHz) is reserved as a **Guard Band**
- Prevents interference between GSM and adjacent frequency bands
- Located at band edges for spectrum isolation

Duplex Operation

Frequency Division Duplex (FDD):

- Uplink and Downlink use different frequencies
- 45 MHz separation between uplink and downlink
- Example:
 - Mobile transmits on 890.2 MHz
 - Base Station transmits on 935.2 MHz
 - Same time slot, different frequencies

System Capacity Example

For a typical GSM 900 network:

- 124 carriers available
- 8 users per carrier simultaneously
- Total: 992 simultaneous conversations
- With frequency reuse (N=7): ~142 conversations per cell

Key Advantages of This Design

1. **Spectrum Efficiency:** 8 users share same frequency
2. **Cost Effective:** Less hardware required per user
3. **Flexible Capacity:** Can allocate multiple time slots for data
4. **Interference Reduction:** Careful frequency planning minimizes co-channel interference

Practical Implications

- Each physical channel can carry:
 - Voice call (TCH)
 - Data connection
 - Signaling information
- **Dynamic allocation:** Network can assign time slots based on demand
- **Half-rate channels:** Can double capacity by putting 2 users in same time slot

GSM Frame Structure Hierarchy:

GSM uses a complex **nested frame structure** to organize time slots, signaling, and data transmission.

Complete Frame Hierarchy

1. HyperFrame (Largest Unit)

Duration: ≈ 3 hours, 24 minutes

Structure: 2048 SuperFrames

Purpose: Highest-level timing and encryption synchronization

2. SuperFrame

Duration: 6.12 seconds

Types:

- 51-Multiframe (6.12 sec) - Used for Traffic Channels
- 26-Multiframe (6.12 sec) - Used for Control Channels

3. Multiframe

Two Types:

A) 51-Multiframe (Traffic Channels)

Duration: 6.12 seconds

Frames: 51 TDMA frames

Used for: TCH (Traffic Channels) + associated control channels

B) 26-Multiframe (Control Channels)

Duration: 6.12 seconds

Frames: 26 TDMA frames

Used for: BCCH, CCCH, SDCCH (Broadcast and Common Control Channels)

4. TDMA Frame(TDMA Format): TDMA GSM uses a complex hierarchy of TDMA frames to define logical channels (Figure 10.14). Fundamentally, each 200-kHz frequency band is divided into 8 logical channels defined by the repetitive occurrence of time slots. At the lowest level is the time slot, also called a burst period, which has a duration of 15/26ms, or approximately 0.577 ms. With a bit rate of 270.833 kbps, each time slot has a length of 156.25 bits. The time slot includes the following fields: (Tail bits, Encrypted bits, Stealing bit, Training sequence, Guard bits[discussed below])

Duration: 4.615 ms

Structure: 8 time slots (0-7)

Carries: 8 simultaneous conversations on one frequency

5. Time Slot (Burst)

Duration: 0.577 ms

Bits: 156.25 bits

Structure: [Tail | Data | Training | Data | Tail | Guard]

Time Slot Detailed Structure

Burst Composition:

| 3 bits | 57 bits | 1 bit | 26 bits | 1 bit | 57 bits | 3 bits | 8.25 bits |
| Tail | Data | Steam | Training | Steam | Data | Tail | Guard |

Field Descriptions:

- **Tail Bits (3+3):** Allow synchronization of transmissions from mobile units located at

different distances from the base station, as explained subsequently. (Signal start and end of burst)

- **Data Bits (57+57):** Data is encrypted in blocks by conventional encryption of 114 plaintext bits into 114 ciphertext bits; the encrypted bits are then placed in two 57-bit fields in the time slot. (Encrypted user data or signaling)

- **Stealing Flags (1+1):** Used to indicate whether this block contains data or is "stolen" for urgent control signaling. (Indicates if data is replaced by signaling (FACCH))

- **Training Sequence (26):** Used to adapt the parameters of the receiver to the current path propagation characteristics and to select the strongest signal in case of multipath propagation. The training sequence is a known bit pattern that differs for different adjacent cells. It enables the mobile units and base stations to determine that the received signal is from the correct transmitter and not a strong interfering transmitter. In addition, the training sequence is used for multipath equalization, which is used to extract the desired signal from unwanted reflections. By determining how the known training sequence is modified by multipath fading, the rest of the signal is processed to compensate for these effects. (Known pattern for channel estimation and equalization)

- **Guard Period (8.25):** Prevents overlap between consecutive time slots

Timing Calculations

Key Duration Calculations:

1 HyperFrame = $2048 \times 6.12 \text{ sec} = 12,533.76 \text{ sec} \approx 3 \text{ h } 28\text{m } 53.76\text{s}$

1 SuperFrame = $51 \times 4.615 \text{ ms} \times 26/51 = 6.12 \text{ sec}$

1 TDMA Frame = $8 \times 0.577 \text{ ms} = 4.615 \text{ ms}$

1 Time Slot = $156.25 \times (1/270.833 \text{ kHz}) = 0.577 \text{ ms}$

Bit Rate Calculations:

Gross Bit Rate = $156.25 \text{ bits} / 0.577 \text{ ms} \approx 270.833 \text{ kbps}$

Net User Bit Rate (TCH/FS) = 22.8 kbps (after coding & overhead)

Functional Purpose

Why This Complex Structure?

1. **Multiple Channel Types:** Accommodates both traffic and control channels
2. **Synchronization:** Training sequences help overcome multipath propagation
3. **Efficiency:** Guard periods prevent inter-slot interference
4. **Flexibility:** Different multiframe types for different purposes
5. **Security:** Hyperframe timing used for encryption synchronization

Practical Operation

For a Voice Call:

- User assigned to specific **carrier frequency** and **time slot**
- **Every 4.615 ms**, the phone transmits/receives in its assigned slot
- **Training sequence** helps equalizer combat signal reflections
- **Stealing flags** allow urgent signaling without dropping call

For Signaling:

- Control channels use **26-multiframe** structure
- Dedicated time slots for broadcast, paging, and access grant
- SDCCH handles call setup while TCH handles voice traffic

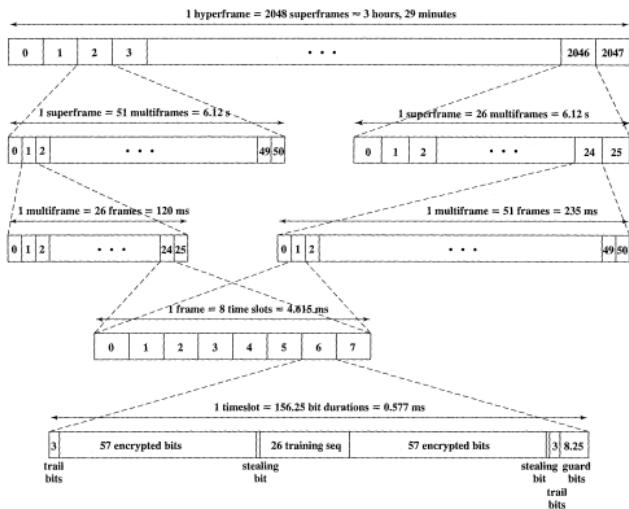


Figure 10.14 GSM Frame Format

GSM Logical Channels - Traffic Channels (TCH):

Overview:

Logical channels are **virtual channels** mapped onto physical channels (carrier + time slot) to carry different types of information.

Traffic Channel Types

1. TCH/F (Full Rate Traffic Channel)

Data Rate: 22.8 kbps

Structure: Uses 24 frames out of 26-frame multiframe

Usage: One voice conversation per time slot

2. TCH/H (Half Rate Traffic Channel)

Data Rate: 11.4 kbps

Structure: Shares one time slot between two users

Usage: Two voice conversations per time slot (doubles capacity)

Frame Structure for Traffic Channels

Multiframe Structure:

Duration: 120 ms

Frames: 26 TDMA frames (numbered 0-25)

Cycle: Repeats every 120 ms

Frame Allocation in 26-Multiframe:

Frames 0-11: User Data

Frame 12: Signaling (SACCH)

Frames 13-24: User Data

Frame 25: Idle (used for measurements)

Data Rate Calculations

TCH/F (Full Rate) Calculation:

Bits per frame: 2×57 bits = 114 bits

Frames for data: 24 frames per 120 ms

Total bits: 114 bits \times 24 frames = 2736 bits per 120 ms

Data Rate: 2736 bits / 0.12 sec = 22.8 kbps

Effective Voice Rate: 13 kbps (after channel coding & FEC)

TCH/H (Half Rate) Calculation:

Bits per frame: 2×57 bits = 114 bits

Shared between 2 users: 57 bits per user per frame

Frames for data: 24 frames per 120 ms

Total bits: 57 bits \times 24 frames = 1368 bits per 120 ms

Data Rate: 1368 bits / 0.12 sec = 11.4 kbps

Time Slot Structure Details

Time Slot Composition:

| 3 | 57 | 1 | 26 | 1 | 57 | 3 | 8.25 | bits

| T | Data | F | Training | F | Data | T | Guard |

- T (3+3):** Tail bits for synchronization
- Data (57+57):** Encrypted user data
- F (1+1):** Stealing flags for FACCH
- Training (26):** Channel estimation sequence
- Guard (8.25):** Prevents time slot overlap

Multiframe Operation

For TCH/F (Full Rate):

- FN 12:** Used for **SACCH** (Slow Associated Control Channel)
- FN 25:** **Idle frame** – used for neighbor cell measurements
- Remaining 24 frames:** Carry user voice/data

For TCH/H (Half Rate):

- User 1:** Uses FN 12 for signaling, FN 25 idle

- **User 2:** Uses FN 25 for signaling, FN 12 idle
- **Both users** share the same time slot alternately

Capacity vs Quality Trade-off

TCH/F Advantages:

- Better voice quality (13 kbps codec)
- More robust against errors
- Standard voice quality

TCH/H Advantages:

- Doubles network capacity
- Efficient for low-traffic areas
- Acceptable quality for most users

Voice Codec Information

- **Full Rate Codec:** 13 kbps speech coding
- **Half Rate Codec:** 5.6 kbps speech coding
- **Channel Coding:** Adds redundancy for error protection
- **Total Rate:** 22.8 kbps including coding overhead

Practical Implementation

Network Planning:

- Operators can mix TCH/F and TCH/H based on traffic patterns
- Urban areas: Mostly TCH/F for better quality
- Rural areas: More TCH/H for extended coverage

Dynamic Allocation:

- Network can switch between TCH/F and TCH/H based on load
- During congestion: May force TCH/H to serve more users
- Quality priority: Assign TCH/F to premium subscribers

GSM Logical Channels - Control Channels:

Control channels manage **signaling, synchronization, and network coordination** rather than carrying user voice/data.

Control Channel Hierarchy

LOGICAL CHANNELS

- **TRAFFIC** (User Data)
- **CONTROL** (Signaling)
 - **BROADCAST** (Downlink only)
 - **COMMON CONTROL** (Uplink & Downlink)
 - **DEDICATED CONTROL** (Uplink & Downlink)

BROADCAST CONTROL CHANNELS (Downlink Only)

1. FCCH (Frequency Correction Channel)

- **Function:** Provides frequency synchronization reference

- **Content:** All "0" bits creating pure sine wave
- **Purpose:** Mobile stations adjust their frequency to base station

2. SCH (Synchronization Channel)

- **Function:** Provides timing synchronization
- **Content:** Base Station Identity Code (BSIC) and frame number
- **Purpose:** Mobile stations synchronize their TDMA timing

3. BCCH (Broadcast Control Channel)

- **Function:** Broadcasts general network information
- **Content:** Cell identity, network identity, frequency lists
- **Purpose:** Mobiles learn about network configuration

COMMON CONTROL CHANNELS

4. PCH (Paging Channel)

- **Direction:** Downlink only
- **Function:** Alerts mobile about incoming calls/SMS
- **Usage:** Network pages mobile when call arrives

5. RACH (Random Access Channel)

- **Direction:** Uplink only
- **Function:** Mobile requests dedicated channel
- **Characteristic:** Slotted ALOHA protocol, collision-prone

6. AGCH (Access Grant Channel)

- **Direction:** Downlink only
- **Function:** Grants channel assignment after RACH request
- **Usage:** Assigns SDCCH or TCH to mobile

DEDICATED CONTROL CHANNELS(DCCH) (Bidirectional)

7. SDCCH (Standalone Dedicated Control Channel)

- **Function:** Handles call setup, location updates, SMS
- **Usage:** Used before traffic channel assignment
- **Capacity:** Can handle multiple simultaneous signaling procedures

8. SACCH (Slow Associated Control Channel)

- **Function:** Carries continuous control information during calls
- **Content:** Signal quality measurements, timing advance
- **Timing:** One SACCH frame per 26-multiframe (every 120ms)

9. FACCH (Fast Associated Control Channel)

- **Function:** Handles urgent signaling during active calls
- **Method:** "Steals" voice frames (uses stealing flags)
- **Usage:** Handovers, urgent commands during conversation

Multiframe Organization

51-Multiframe Structure (Used for Control Channels):

Duration: $3060/13 \approx 235.38$ ms

Frames: 51 TDMA frames

Used for: SDCCH, SACCH, BCCH, etc.

Mapping: Complex pattern across multiple frames

Key Allocation:

- **TS 0:** Typically used for broadcast and common control channels
- **SDCCH/8:** 8 SDCCH subchannels can share one physical channel
- **Cyclic Pattern:** Channels repeat in specific multiframe patterns

Channel Usage Flow

Mobile Originated Call Example:

1. **RACH:** Mobile requests access
2. **AGCH:** Network grants SDCCH
3. **SDCCH:** Call setup, authentication, encryption
4. **FACCH:** Traffic channel assignment
5. **TCH+SACCH:** Voice conversation with continuous control

Mobile Terminated Call Example:

1. **PCH:** Mobile paged for incoming call
2. **RACH:** Mobile responds
3. **AGCH:** Network grants SDCCH
4. **SDCCH:** Call setup procedures
5. **TCH+SACCH:** Active call with measurements

Capacity and Configuration

Typical TS0 Configuration:

FCCH → SCH → BCCH → CCCH (PCH+AGCH) → SDCCH

- Repeated in 51-multiframe structure
- Provides essential services to all mobiles in cell

SDCCH Capacity:

- One physical channel can support 8 SDCCH subchannels
- Each SDCCH can handle multiple simultaneous procedures
- Critical for handling SMS, location updates, and call setup

Functional Importance

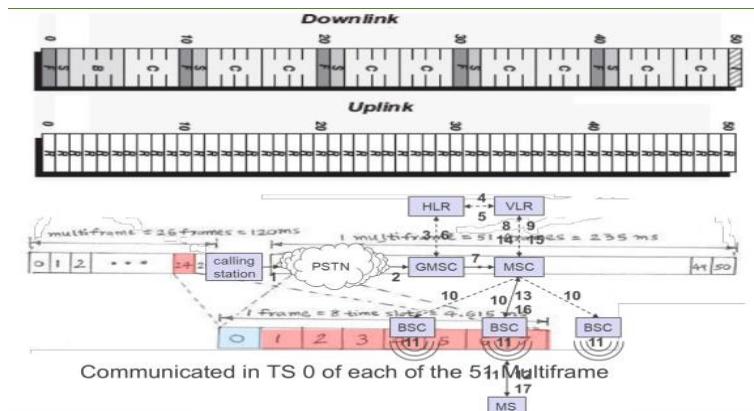
Network Efficiency:

- **Broadcast channels** serve all mobiles simultaneously
- **Common control** handles random access efficiently
- **Dedicated control** provides reliable signaling for active sessions

Mobility Management:

- **SACCH** enables continuous measurements for handovers
- **FACCH** provides fast response for urgent commands
- **BCCH** informs mobiles about neighbor cells

GSM Logical Channels in Time Slot 0:



Key Concept:

Time Slot 0 of each carrier is typically reserved for **control channels** and follows a **51-multiframe structure** for broadcasting essential system information.

51-Multiframe Structure for Control Channels

Duration & Composition:

Multiframe Duration: 235.38 ms (3060/13 ms)

Total Frames: 51 TDMA frames

Cycle: Repeats continuously in Time Slot 0

Channel Allocation Pattern in 51-Multiframe

Frame-by-Frame Allocation:

Broadcast Channels (First 5 Frames):

- Frame 0: FCCH (Frequency Correction Channel)
- Frame 1: SCH (Synchronization Channel)
- Frames 2-4: BCCH (Broadcast Control Channel)

Common Control Channels (Next Frames):

- Frames 5-50: CCCH (Common Control Channels)
 - PCH (Paging Channel) - Downlink
 - AGCH (Access Grant Channel) - Downlink
 - RACH (Random Access Channel) - Uplink (response frames)

SDCCH & SACCH Allocation:

- **SDCCH:** Spread across specific frames in the 51-multiframe

- **SACCH:** Associated with each SDCCH for continuous control

⌚ Functional Purpose of This Allocation

Why Time Slot 0?

- **Centralized Control:** All mobiles in the cell monitor TS0
- **Predictable Structure:** Mobiles know exactly where to find system information
- **Efficient Resource Use:** Dedicated slot prevents control/traffic interference

Why 51-Multiframe?

- **Accommodates Multiple Channels:** Fits FCCH, SCH, BCCH, CCCH, SDCCH, SACCH
- **Regular Repetition:** Ensures continuous availability of critical information
- **Balanced Capacity:** Provides enough paging, access, and signaling capacity

📱 Mobile Station Behavior

Initial Cell Selection:

1. **Scan for FCCH:** Find frequency correction burst
2. **Sync with SCH:** Get timing and BSIC
3. **Read BCCH:** Learn cell parameters and neighbor info
4. **Monitor PCH:** Listen for paging messages

Call Setup:

1. **Send RACH:** Request access in appropriate frame
2. **Listen to AGCH:** Wait for channel assignment
3. **Switch to SDCCH:** Move to dedicated signaling channel

🌐 Capacity Considerations

Paging Capacity:

- Multiple PCH frames in the 51-multiframe
- Determines how many subscribers can be paged simultaneously
- Affects call setup time and network responsiveness

Access Grant Capacity:

- AGCH frames determine how many channel assignments can be made
- Critical for handling simultaneous call originations

SDCCH Capacity:

- Number of SDCCH subchannels determines concurrent signaling procedures
- Affects call setup success rate and SMS handling

🔑 Network Planning Implications

Optimization Areas:

- **Paging Group Calculation:** Determines how often each mobile listens to paging
- **SDCCH Dimensioning:** Ensures enough signaling capacity for expected traffic
- **BCCH Power Setting:** Typically transmitted at higher power for better coverage

Troubleshooting:

- **SDCCH Congestion:** Causes call setup failures
- **PCH Overload:** Results in missed incoming calls
- **RACH Collisions:** Increase call setup delays

SECOND-GENERATION CDMA:

Code division multiple access (CDMA) is a spread spectrum based technique for multiplexing, introduced in Section 7.4, that provides an alternative to TDMA for second-generation cellular networks. We begin this section with an overview of the advantages of the CDMA approach and then look at the most widely used scheme, IS-95.

Code Division Multiple Access

CDMA for cellular systems can be described as follows. As with FDMA, each cell is allocated a frequency bandwidth, which is split into two parts, half for reverse (mobile unit to base station) and half for forward (base station to mobile unit). For full duplex communication, a mobile unit uses both reverse and forward channels. Transmission is in the form of direct-sequence spread spectrum (DS-SS), which uses a chipping code to increase the data rate of the transmission, resulting in an increased signal bandwidth. Multiple access is provided by assigning orthogonal chipping codes to multiple users, so that the receiver can recover the transmission of an individual unit from multiple transmissions.

CDMA has a number of advantages for a cellular network:

- **Frequency diversity:** Because the transmission is spread out over a larger bandwidth, frequency-dependent transmission impairments, such as noise bursts and selective fading, have less effect on the signal.
- **Multipath resistance:** In addition to the ability of DS-SS to overcome multipath fading by frequency diversity, the chipping codes used for CDMA not only exhibit low cross correlation but also low autocorrelation. Therefore, a version of the signal that is delayed by more than one chip interval does not interfere with the dominant signal as much as in other multipath environments.
- **Privacy:** Because spread spectrum is obtained by the use of noiselike signals, where each user has a unique code, privacy is inherent.
- **Graceful degradation:** With FDMA or TDMA, a fixed number of users can access the system simultaneously. However, with CDMA, as more users access the system simultaneously, the noise level and hence the error rate increases; only gradually does the system degrade to the point of an unacceptable error rate.

A number of drawbacks of CDMA cellular should also be mentioned:

- **Self-jamming:** Unless all of the mobile users are perfectly synchronized, the arriving transmissions from multiple users will not be perfectly aligned on chip boundaries. Thus the spreading sequences of the different users are not

orthogonal and there is some level of cross correlation. This is distinct from either TDMA or FDMA, in which for reasonable time or frequency guard-bands, respectively, the received signals are orthogonal or nearly so. LOA / SECOND-GENERATION CDMA

299

- **Near-far problem:** Signals closer to the receiver are received with less attenuation than signals farther away. Given the lack of complete orthogonality, the transmissions from the more remote mobile units may be more difficult to recover. Thus, power control techniques are very important in a CDMA system.
- **Soft handoff:** As is discussed subsequently, a smooth handoff from one cell to the next requires that the mobile unit acquires the new cell before it relinquishes the old. This is referred to as a soft handoff and is more complex than the hard handoff used in FDMA and TDMA schemes.

SOFT HANDOFF AND HARD HANDOFF:

Core Concept: What is a Handoff?

A handoff (or handover) is the process of transferring an ongoing call or data session from one cell (base station) to another as a user moves through the network.

1. Hard Handoff (HHO) - "Break-Before-Make"

This is the type of handoff used in **2G GSM** networks.

- **Principle: "Break-Before-Make":** The Mobile Station (MS) is disconnected from the current Base Station (BTS) *before* it establishes a connection with the new one.
- **How it works in GSM:**
 - The MS continuously measures the signal strength of its serving BTS and neighboring BTSSs, reporting this to the Base Station Controller (BSC) via the SACCH.
 - The BSC (and sometimes the MSC) makes the handoff decision based on these reports.
 - The network commands the MS to switch to a new frequency and time slot assigned to the new BTS.
 - The MS **breaks the radio link** with the old BTS.
 - The MS **tunes its radio** to the new frequency and synchronizes with the new BTS.
 - The MS **makes a new connection** with the new BTS and sends a handoff complete message.
- **Key Characteristic:** There is a brief, momentary interruption in the voice path during the switch. This interruption is typically so short (a few tens of milliseconds) that it is imperceptible to the human ear.

Why GSM uses Hard Handoff: GSM uses a combination of FDMA and TDMA. Each cell uses different carrier frequencies. To move to a new cell, the MS must physically retune its radio to a new frequency, which necessitates breaking the connection with the old cell first.

2. Soft Handoff (SHO) - "Make-Before-Break"

This is the type of handoff used in **2G CDMA-based systems** (like IS-95, also known as cdmaOne) and is a key feature of 3G CDMA systems (like UMTS).

- **Principle: "Make-Before-Break":** The Mobile Station establishes a connection with the new Base Station *before* breaking the connection with the old one.
- **How it works in CDMA:**
 - Since all cells in a CDMA network use the *same frequency*, the MS can communicate with multiple BTSSs simultaneously using different "codes."
 - The MS, through its multiple "rake receivers," can listen to and be in contact with up to 2 or 3 BTSSs at the same time. This set of BTSSs is called the **Active Set**.
 - The network combines the signals from these multiple BTSSs (a process called macro-diversity) at the BSC to select the best possible signal.
 - As the MS moves, BTSSs are added to and removed from the Active Set seamlessly.
- **Key Characteristic:** There is **no momentary interruption** in the voice path. The transition is seamless, which provides a smoother voice quality.

Comparison Table: Hard Handoff vs. Soft Handoff in 2G

Feature	Hard Handoff (Used in 2G GSM)	Soft Handoff (Used in 2G CDMA)
Principle	Break-Before-Make	Make-Before-Break
Connection	Connected to only ONE BTS at a time.	Connected to MULTIPLE BTSSs simultaneously.
Interruption	Brief, momentary interruption during the switch.	No interruption; seamless transition.
Complexity	Less complex, implemented at the BSC/MSC.	More complex, requires multiple rake receivers in the MS and combining at the BSC.
Underlying Tech	FDMA/TDMA (Different frequencies per cell).	CDMA (Same frequency for all cells).
Call Quality	Generally good, but a "ping-pong" effect can occur if the handoff decision is poor.	Superior and smoother, especially at cell boundaries.

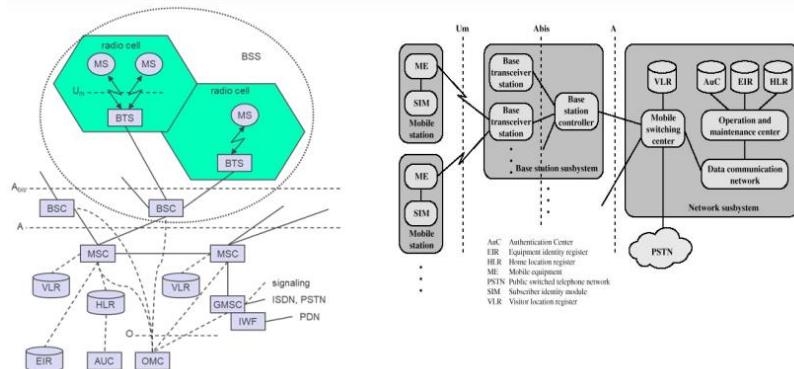
Resource Usage	Efficient. Resources are used in only one cell at a time.	Inefficient ("Wasteful"). The same call uses resources in multiple cells, reducing overall capacity.
"Ping-Pong" Effect	More prone to it if the MS is on the cell edge and the signal fluctuates.	Less prone, as the MS is connected to multiple cells, providing a more stable link.

Conclusion

To answer your question directly:

- 2nd Generation GSM networks use **Hard Handoff**.
- 2nd Generation CDMA-based networks (IS-95/cdmaOne) use **Soft Handoff**.

3G & 4G



Introduction to 2G

The Second Generation (2G) of cellular networks marked a revolutionary shift from the first generation (1G) by moving from **analog** to **digital** transmission. This fundamental change unlocked significant improvements in:

- **Voice Quality:** Digital signals are less susceptible to noise and interference.
- **Capacity:** Digital compression techniques allowed more calls to be carried on the same amount of radio spectrum.
- **Security:** Digital encryption made eavesdropping on calls much more difficult.
- **Data Services:** For the first time, basic data services like SMS (Short Message Service) became possible.

1. Core Network Architecture of 2G (GSM)

The provided PDF outlines a classic 2G architecture, primarily based on the **GSM (Global System for Mobile communications)** standard. The network can be divided into several key subsystems:

A. Mobile Station (MS)

This is the user's handset, which consists of two parts:

- **Mobile Equipment (ME):** The physical phone device itself.
- **Subscriber Identity Module (SIM):** A removable smart card that contains the user's identity, subscription information, and phonebook. The SIM card is what allows a user to switch phones easily.

B. Base Station Subsystem (BSS)

This subsystem manages the radio link with the Mobile Station. It comprises:

- **Base Transceiver Station (BTS):** The "cell tower" or radio equipment that communicates directly with the mobile phones over the Um interface (the air interface).
- **Base Station Controller (BSC):** Manages one or more BTSs. It handles radio resource allocation, handovers between BTSs, and controls transmission power.

C. Network Switching Subsystem (NSS) - The "Core Network"

This is the heart of the 2G network, managing calls, switching, and subscriber functions.

- **Mobile Switching Center (MSC):** The central telephone switch for the cellular network. It handles call setup, routing, and termination to other phones or fixed networks (like PSTN and ISDN).
- **Home Location Register (HLR):** A central database that contains permanent subscriber information (e.g., phone number, subscription profile, current location).
- **Visitor Location Register (VLR):** A temporary database that holds information about subscribers currently physically present in the geographical area controlled by a particular MSC.
- **Authentication Center (AUC):** Works with the HLR to authenticate the SIM card and provide encryption keys to ensure communication is secure.
- **Equipment Identity Register (EIR):** A database that tracks mobile equipment by their IMEI number to prevent calls from stolen or unauthorized devices.
- **Gateway MSC (GMSC):** A specific MSC that routes calls to and from external networks (like the public phone network).

2. Multiple Access Technologies in 2G

The PDF highlights the transition from analog to digital and the key multiple access methods used to share the radio spectrum.

- **1G - FDMA (Frequency Division Multiple Access):** The first generation used an analog, FDMA-based system. It divided the frequency band into channels, and each call used one dedicated channel for its entire duration. This was inefficient.
- **2G - TDMA (Time Division Multiple Access):** This was the core digital technology for GSM. TDMA divides a single radio frequency channel into **time slots**. Multiple

users take turns using the channel, transmitting in their assigned time slot. This allows a single frequency to serve multiple simultaneous calls.

- **GSM (Global System for Mobile communications)** is the most successful 2G standard using TDMA.
- **GPRS (General Packet Radio Service)**, often called **2.5G**, was an upgrade to GSM that introduced **packet-switching** for data, enabling "always-on" internet connectivity (like early WAP browsing and MMS).
- **EDGE (Enhanced Data rates for GSM Evolution)**, sometimes called 2.75G, improved data rates further.

2G - CDMA (Code Division Multiple Access): An alternative path to GSM, primarily used in the Americas. CDMA allows all users to transmit over the entire frequency spectrum simultaneously. Users are separated by a unique code, and the receiver uses this code to pick out the desired signal from the background noise.

IS-95 (cdmaOne) was the main 2G CDMA standard.

Of course. This is an excellent question because it points to a fundamental conceptual leap that happened between 3G and 4G. The discussion of OFDM in a presentation about 3G and 4G is crucial, as **OFDM is the bedrock technology upon which 4G/LTE is built**.

Step 1: The Problem with Single Carrier Systems (like 2G and 3G)

Imagine data as a stream of cars traveling down a highway (the radio frequency channel).

- In a **single carrier system** (like **GSM or WCDMA**), all these cars travel in a single, very fast lane. The speed of each car is incredibly high to move all the data.

The Problem: Multipath Fading Radio signals bounce off buildings, hills, and other objects. This means the receiver gets multiple copies of the signal, each delayed by a tiny amount. For a very fast data stream (a single, high-speed car), these delayed copies can interfere with the main signal, causing "Intersymbol Interference" (ISI) – the data symbols blur into each other and become corrupted. This requires complex equalizers to fix.

Analogy: It's like trying to listen to someone who is shouting a very fast sentence in a large echoey hall. The echoes of the first word can interfere with you hearing the subsequent words.

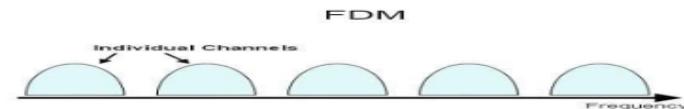
Step 2: The Old Solution: FDM (Frequency Division Multiplexing)

The intuitive solution to avoid interference is to keep channels separate.

- **FDM (Frequency Division Multiplexing)** assigns each user or data stream its own dedicated frequency channel, with **guard bands** in between them.
- **Guard Bands** are empty spaces in the frequency spectrum that prevent the channels from interfering with each other.

The Diagram Explained (Individual Channels): The "FDM (Individual channels)" part of your diagram would look like this in the frequency domain:

[Car 1] [Guard] [Car 2] [Guard] [Car 3]



Disadvantage: This is very inefficient! The guard bands waste precious radio spectrum.

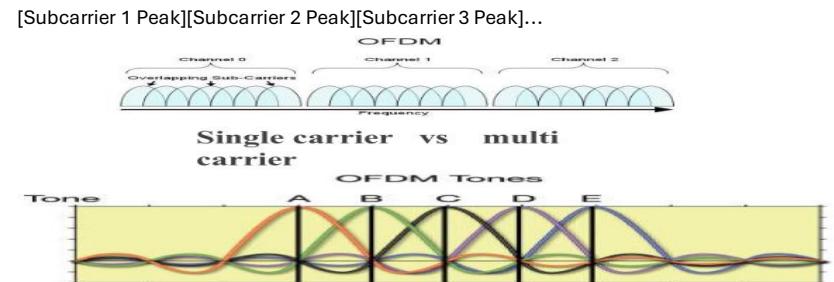
Step 3: The Brilliant Solution: OFDM (Orthogonal Frequency Division Multiplexing)

OFDM takes a completely different approach. Instead of one fast lane, it creates **many slow, parallel lanes**.

- It splits the high-speed data stream into **N** much slower, parallel streams.
- Each of these slow streams is then used to modulate a separate radio wave, called a **subcarrier**.
- These subcarriers are packed extremely close together.

The Key Innovation: Orthogonality The magic that makes this work is the **orthogonal** placement of the subcarriers. "Orthogonal" means the peaks of one subcarrier line up with the zeros of all the others. This allows them to overlap perfectly without interfering, eliminating the need for guard bands.

The Diagram Explained (n channels overlapping subcarriers): The "OFDM" part of your diagram would look like this:



Even though they overlap, at the exact moment you sample Subcarrier 1's peak, all other subcarriers are at zero, so they contribute nothing. This is what the equation and the orthogonality condition in your PDF are proving.

Step 4: Deep Dive into the OFDM Equation and Diagram

Let's connect the equation to the "OFDM Tones" diagram.

The Equation:

For N subcarriers, the transmitted signal (baseband) is

$$s(t) = \sum_{k=0}^{N-1} X_k e^{j2\pi f_k t}, \quad 0 \leq t < T$$

where

- X_k : complex data symbol (QPSK, QAM) on the k^{th} subcarrier
- $f_k = f_0 + k\Delta f$: frequency of subcarrier
- $\Delta f = \frac{1}{T}$: subcarrier spacing ensures **orthogonality**

The orthogonality between subcarriers k and l in the OFDM signal

$$s(t) = \sum_{k=0}^{N-1} X_k e^{j2\pi k \Delta f t}$$

comes from the subcarrier spacing choice $\Delta f = \frac{1}{T}$.

Two subcarriers k and l are orthogonal if

$$\int_0^T e^{j2\pi k \Delta f t} e^{-j2\pi l \Delta f t} dt = 0 \quad \text{for } k \neq l$$

Substituting $\Delta f = \frac{1}{T}$:

$$\int_0^T e^{j2\pi(k-l)t/T} dt = 0, \quad \text{for } k \neq l$$

Hence, the OFDM signal equation ensures orthogonality because subcarrier frequencies are spaced by exactly $1/T$.

The "OFDM Tones" Diagram Explained: The diagram showing "Time" on one axis and "Tone" on the other is a bit abstract. It's best interpreted as a **time-frequency grid**:

- **Vertical Axis (Tone):** Represents the different subcarriers (frequencies), f_0, f_1, f_2, \dots
- **Horizontal Axis (Time):** Shows the symbol duration T .
- **The Boxes (A, B, C, D, E):** Each box represents one data symbol X_k being transmitted on one subcarrier for the duration of one OFDM symbol T .

So, in a given time period T , you are simultaneously transmitting symbols A, B, C, D, and E, each on its own orthogonal subcarrier.

Step 5: Why OFDM was a Game-Changer for 4G/LTE

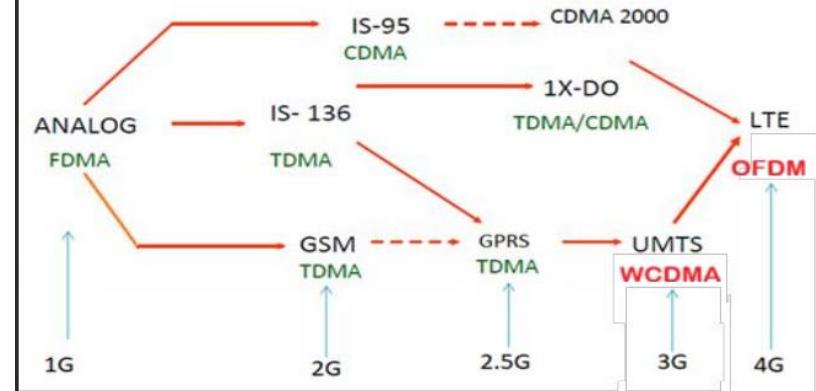
1. **Robustness to Multipath:** Since the data rate per subcarrier is low, the symbol duration T is long. A delayed echo only affects a small portion (the beginning) of the long OFDM symbol. A simple "guard interval" (a copied-and-pasted section from the end of the symbol placed at the beginning) can be used to absorb the echo, making the system very robust. This is far simpler than the complex equalizers needed in 3G.
2. **Spectral Efficiency:** By eliminating guard bands between subcarriers, OFDM uses the available spectrum much more efficiently than FDM.

3. **Flexibility:** It's easy to allocate different numbers of subcarriers to different users (this is the basis for LTE's scheduling) and to avoid subcarriers in noisy parts of the spectrum.

Summary

Feature	Single Carrier (2G/3G)	FDM	OFDM (4G/5G)
Core Idea	One fast data stream	Multiple separate channels	Many slow, parallel streams
Subcarriers	One carrier	Multiple, with guard bands	Multiple, overlapping & orthogonal
Efficiency	Low (complex equalizers needed)	Low (wasted spectrum)	High
Robustness to Multipath	Poor	Good	Excellent

In conclusion, the move from the single-carrier and CDMA-based systems of 3G to the **OFDM-based system of 4G/LTE** was a fundamental architectural shift. It was this shift that enabled the massive increases in data speed, spectral efficiency, and network stability that we now associate with 4G and which continue to form the foundation of 5G.



Step 1: The Common Ancestor - 1G (Analog FDMA)

- Technology: ANALOG FDMA (Frequency Division Multiple Access)

- **Concept:** The very first generation of cellular technology. It was entirely analog. The available radio spectrum was divided into multiple frequency channels, and each call occupied one entire channel for its duration.
- **Limitations:**
 - **No Security:** Calls could be easily eavesdropped on with simple radio scanners.
 - **Poor Voice Quality:** Susceptible to static and noise.
 - **Inefficient:** One channel per user, even during silence.
 - **No Data Services:** Primarily for voice only.

This common starting point is the root from which all digital paths emerged.

Step 2: The Great Digital Schism - The Three Paths of 2G

The core innovation of 2G was the shift from **Analog to Digital**. This improved voice quality, capacity, and security (via encryption). However, different regions and companies standardized on different digital access methods, leading to three main branches.

Branch 1: The GSM Path (Europe & Global Dominance)

- **Technology:** **GSM (Global System for Mobile communications) using TDMA**
- **Concept:** This path used **TDMA (Time Division Multiple Access)**. A frequency channel is divided into time slots. Several users take turns using the channel, transmitting in their assigned time slot. This was more efficient than pure FDMA.
- **Impact:** GSM became the most widely adopted 2G standard in the world, forming the basis for a global ecosystem.

Branch 2: The IS-136 Path (North America)

- **Technology:** **IS-136 (D-AMPS) using TDMA**
- **Concept:** This was essentially the North American counterpart to GSM, also using a TDMA-based technology. It competed directly with GSM and IS-95 in markets like the United States.
- **Impact:** While successful for a time, it was ultimately overtaken by the global momentum of GSM and the technical advantages of CDMA.

Branch 3: The IS-95 Path (A Different Philosophy)

- **Technology:** **IS-95 (cdmaOne) using CDMA**
- **Concept:** This path took a radically different approach with **CDMA (Code Division Multiple Access)**. Instead of dividing by time or frequency, it allows all users to transmit simultaneously over the entire frequency band. Users are separated by unique codes.
- **Advantages:** Offered better capacity and call quality than early TDMA systems.

Step 3: The Data Revolution - The Birth of 2.5G

The next challenge was to add data to these voice-centric networks. The key innovation was the introduction of **packet-switching**, enabling "always-on" internet connectivity.

- **The Converging Step:** The diagram correctly shows that **both the IS-136 and GSM paths evolved into GPRS**.

- **GPRS (General Packet Radio Service):** This was a packet-switched overlay on top of the existing TDMA-based networks (both GSM and IS-136). It didn't require a new air interface, just upgrades to the core network.
- **Why it's "2.5G":** It was not a new generation, but a major evolutionary step that enabled the first widespread mobile internet services (like WAP browsing and email).

Note: The IS-95 (CDMA) path had its own parallel data evolution (like CDMA2000 1xRTT), which also offered packet data, but it's not shown merging with GPRS in the diagram, which is accurate as they were separate technological streams.

Step 4: The Road to 3G - High-Speed Data

The goal of 3G was to provide high-speed, seamless mobile data for video calling, multimedia, and the real internet.

Branch 1: The GSM/IS-136 Path → UMTS

- **Technology:** **UMTS (Universal Mobile Telecommunications System) using WCDMA**
- **Concept:** To achieve the high data rates required for 3G, the GSM world abandoned TDMA and adopted a **wideband** version of CDMA (**W-CDMA**). This was a completely new radio interface, requiring new infrastructure, but it allowed a smooth transition from the widely deployed GSM core network.

Branch 2: The IS-95 Path → CDMA2000

- **Technology:** **CDMA2000 (1xEV-DO)**
- **Concept:** This was the natural evolution for the IS-95 camp. It enhanced their existing CDMA technology to support high-speed data. CDMA2000 1xEV-DO (Evolution-Data Optimized) used a hybrid approach, dedicating a channel solely for high-speed packet data.

The "1xEV-DO" in the Diagram:

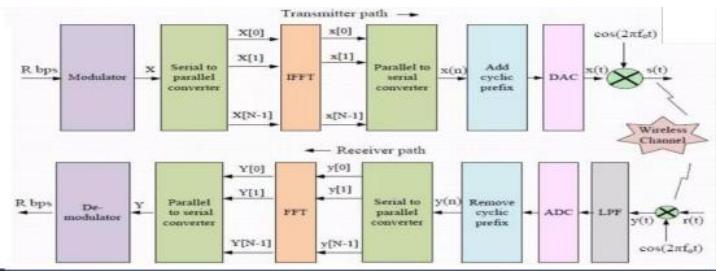
This likely refers to **1xEV-DO**. It's shown as a separate box from CDMA2000, which is technically precise. "CDMA2000" often refers to the voice and data standard, while "1xEV-DO" was the data-only optimization that delivered the highest speeds on this path.

Step 5: The Grand Unification - 4G LTE (OFDM)

By the time the industry looked towards 4G, the limitations of CDMA (both W-CDMA and CDMA2000) for very high data rates were becoming clear. The industry needed a more efficient and scalable technology.

- **The Great Convergence:** The diagram perfectly illustrates how **all three historical paths finally merged into a single technology: LTE (Long Term Evolution)**.
- **The Unifying Technology:** **OFDM (Orthogonal Frequency Division Multiplexing)**
 - As discussed previously, OFDM, with its multiple, parallel, low-rate subcarriers, proved far more efficient and robust for high-speed data in challenging radio environments than the single-carrier, spread-spectrum techniques of CDMA.
- **The "All-IP" Core:** LTE wasn't just a new radio technology. It also introduced a simplified, flat, "**All-IP**" core network. This meant that even voice was treated as

data packets (VoLTE - Voice over LTE), finally retiring the old circuit-switched core of 2G and 3G.



The main purpose of this system is to implement **OFDM (Orthogonal Frequency Division Multiplexing)** — a technique used in modern wireless communication (like Wi-Fi, LTE, 5G) to transmit data efficiently and reliably.

Overview: The Big Idea of OFDM

The core principle of OFDM is to combat the problem of **multipath fading** and **Inter-Symbol Interference (ISI)**. It does this by:

1. Converting a **high-speed serial data stream** into many **low-speed parallel streams**.
2. Transmitting each of these slow streams on a separate, orthogonal subcarrier.

This makes the system incredibly robust against signal echoes and delays.

Part 1: The Transmitter (Top Half of the Diagram)

Step 1: Modulation & Serial-to-Parallel (S/P) Conversion

- **Modulator:** The incoming binary data (at a rate of R bps) is first converted into complex-valued data symbols (e.g., using QPSK or 16-QAM). Each symbol (e.g., $X[0], X[1]$) represents a point on a constellation diagram.
- **Serial to Parallel Converter:** This is a crucial step. Instead of sending these symbols one after another, they are grouped into a **block of N symbols**. This single, fast stream becomes **N parallel, slow streams**.

Step 2: The Heart of OFDM - The IFFT

- **IFFT (Inverse Fast Fourier Transform):** This block is the magic behind practical OFDM. The block of N complex symbols ($X[0]...X[N-1]$) is in the frequency domain (each symbol is assigned to a specific subcarrier).
- The IFFT performs a mathematical transformation that converts this **frequency-domain block into a time-domain signal**. The output is a sequence of N time-domain samples ($x[0], x[1], \dots, x[N-1]$), which is the summation of all N orthogonal subcarriers, each modulated with its own data symbol.

Step 3: Parallel-to-Serial (P/S) Conversion & Cyclic Prefix

- **Parallel to Serial Converter:** The parallel time-domain samples are converted back into a single serial stream, $x(n)$.
- **Add Cyclic Prefix (CP):** This is a **guard interval** critical for combating multipath.
 - The last few samples of the OFDM symbol are copied and placed at its beginning.
 - **Purpose:** The CP acts as a buffer. If a delayed echo from a previous symbol spills into the current symbol, it only affects the CP, not the actual data part of the symbol. As long as the delay is shorter than the CP duration, **orthogonality between subcarriers is preserved**, and Inter-Symbol Interference is eliminated.

Step 4: Digital-to-Analog and Upconversion

- **DAC (Digital-to-Analog Converter):** Converts the digital signal $x(t)$ into an analog waveform.
- **Mixer:** The baseband analog signal is "upconverted" by multiplying it with a high-frequency carrier wave ($\cos(2\pi f_c t)$), shifting it to the desired radio frequency for transmission.
- The signal is then sent through the **Wireless Channel**, where it suffers from effects like noise, attenuation, and multipath.

Part 2: The Receiver (Bottom Half of the Diagram)

The receiver essentially reverses the transmitter's process.

Step 1: Downconversion and Analog-to-Digital

- **Mixer:** The received RF signal $r(t)$ is "downconverted" back to baseband by multiplying it with the same carrier frequency.
- **LPF (Low-Pass Filter):** Removes high-frequency components generated by the mixing process.
- **ADC (Analog-to-Digital Converter):** Samples the analog baseband signal to convert it back into a digital stream, $y(n)$.

Step 2: Removing the Guard Interval and S/P Conversion

- **Remove Cyclic Prefix:** The receiver discards the cyclic prefix samples. By doing this, it removes the part of the signal that contains ISI from the previous symbol.
- **Serial to Parallel Converter:** The remaining "clean" samples of the OFDM symbol are grouped into a parallel block of N samples ($y[0]...y[N-1]$).

Step 3: The Core Recovery - The FFT

- **FFT (Fast Fourier Transform):** This block performs the reverse operation of the IFFT. It takes the time-domain samples and transforms them back into the **frequency domain**. This process effectively extracts the original data symbols from each of the N orthogonal subcarriers.

Step 4: Parallel-to-Serial Conversion and Demodulation

- **Parallel to Serial Converter:** The recovered parallel symbols are converted back into a serial stream.

- Demodulator:** The complex symbols are mapped back to their corresponding binary bits (e.g., a QPSK symbol becomes 2 bits), reconstructing the original data stream at R bps.

This entire process makes OFDM exceptionally resilient in challenging radio environments, which is why it was chosen as the foundation for all modern high-speed wireless communication systems like Wi-Fi, 4G LTE, and 5G.

Comparison Table: CDMA2000 vs. WCDMA (UMTS)

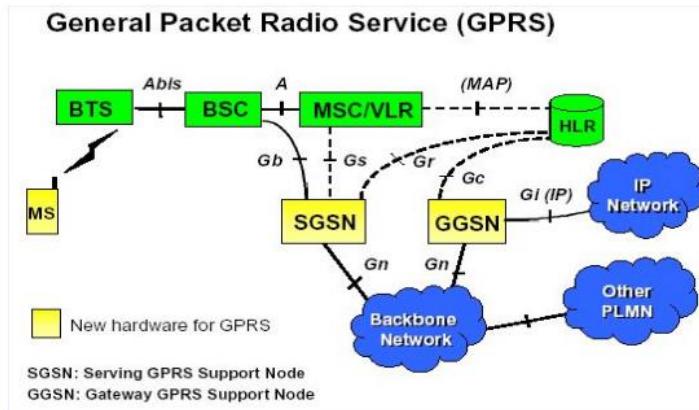
Feature	CDMA2000 (IS-95 Evolution)	WCDMA (UMTS)	Explanation of the Difference
Generation	2G / 3G (often called 2.5G for 1x, 3G for 1xEV-DO)	3G	CDMA2000 was an evolutionary path, with early versions (1x) considered a 2.5G enhancement, while WCDMA was designed from the start as a true 3G standard.
Bandwidth	1.25 MHz (Narrowband)	5 MHz (Wideband)	This is the core difference. WCDMA uses a wider radio channel, which is why it's called Wideband CDMA . This wider channel is the primary reason it can achieve higher data rates.
Data Rate	Up to 153 kbps (CDMA2000 1x) Up to 2.4 Mbps (1xEV-DO)	Up to 2 Mbps (Release 99) Up to 14 Mbps (HSDPA)	The wider bandwidth of WCDMA allows for a higher potential data speed. With subsequent upgrades like HSDPA, the WCDMA path pulled significantly ahead in peak data rates.
Control Architecture	BSC (Base Station Controller)	RNC (Radio Network Controller)	While both serve a similar purpose (managing multiple base stations), the RNC is a more advanced node designed for the complex packet-switching and higher speeds of the 3G UMTS architecture. The BSC is a holdover from the 2G architecture.

##Single Carrier vs. Multi-Carrier

Feature	Single Carrier	Multi-Carrier (e.g., OFDM)
Basic Concept	Transmits all data using one single radio frequency at a very high symbol rate.	Splits the data stream into many (N) parallel streams , each modulating a separate, slower subcarrier .
Symbol Rate	High. All data is sent sequentially in one stream.	Low. Data is divided, so each subcarrier has a much lower symbol rate.
Susceptibility to Multipath Delay Spread	High. A long delay spread causes significant Intersymbol Interference (ISI) , where echoes of one symbol corrupt the next.	Low. The long symbol duration (T) makes the system less sensitive to delays. A Guard Interval easily absorbs echoes.
Equalizer Complexity	High. Requires a complex, adaptive equalizer at the receiver to counteract the ISI in a frequency-selective channel.	Low/Simple. The use of a Guard Interval (Cyclic Prefix) removes the need for a complex time-domain equalizer.
Power Efficiency	High. The signal has a low Peak-to-Average Power Ratio (PAPR) , allowing power amplifiers to operate efficiently.	Low. The superposition of many subcarriers creates a noise-like signal with a very high PAPR , requiring inefficient, linear power amplifiers.
Spectral Efficiency	Lower in multipath environments due to the need for complex equalization and guard times.	Higher. Subcarriers are packed closely together (orthogonally) without guard bands , using spectrum very efficiently.
Implementation Complexity	Simpler in concept, but the receiver is complex due to the equalizer.	More complex in concept (requires IFFT/FFT operations), but simplifies receiver design.
Flexibility	Low. The entire bandwidth is allocated as one block.	High. Easy to allocate different numbers of subcarriers to different users (as in LTE/5G) and avoid noisy frequencies.

Primary Use Cases	<ul style="list-style-type: none"> • 2G (GSM) • 3G (WCDMA, CDMA2000) • Satellite communications 	<ul style="list-style-type: none"> • 4G (LTE), 5G (NR) • Wi-Fi (802.11a/g/n/ac/ax) • DSL Internet (ADSL)
--------------------------	--	--

##General Packet Radio Network (GPRS) : 2.5G



GPRS Network Architecture Breakdown

The diagram shows how the new GPRS packet-core was overlaid onto the existing 2G GSM network.

1. The Existing 2G GSM Radio Access Network (RAN)

This part of the network handles the radio connection and is shared for both voice (circuit-switched) and data (packet-switched) traffic.

- **MS (Mobile Station):** The user's mobile phone or data terminal.
- **BTS (Base Transceiver Station):** The cell tower that the MS connects to over the air.
- **BSC (Base Station Controller):** Manages multiple BTSSs. It handles radio resource allocation and is a key point where voice and data traffic are split.

2. The New GPRS Core Network (The "2.5G" Innovation)

This is the new packet-switched core that was added to enable mobile internet.

- **SGSN (Serving GPRS Support Node)**
 - **Function:** The heart of the GPRS routing and mobility management. It is responsible for:

- **Packet Delivery:** Sending and receiving data packets to/from the MS within its service area.
- **Mobility Management:** Tracking the location of the MS (e.g., which routing area it is in).
- **Authentication & Billing:** Authenticating the user and collecting data usage records.
- **Connection:** It connects to the BSC via the **Gb interface**, which carries the packetized data traffic.

• GGSN (Gateway GPRS Support Node)

- **Function:** Acts as the gateway between the mobile GPRS network and external data networks (like the Internet). It is the "router" of the GPRS network.
- **Key Tasks:**
 - **IP Address Assignment:** It assigns an IP address to the MS (this is the device's address on the internet).
 - **Routing:** Routes packets from the external world to the correct SGSN (and thus to the correct MS), and vice-versa.
 - **Firewall:** Provides a basic level of security between the mobile network and the public internet.
- **Connection:** It connects to the SGSN via the **Gn interface** (an IP-based tunnel within the carrier's network) and to the external IP network via the **Gi interface**.

3. Interfaces and External Connections

- **Gb Interface:** Connects the BSC to the SGSN, carrying the user's data packets.
- **Gn Interface:** The internal IP-based tunnel between the SGSN and GGSN, used to carry user data within the core network.
- **Gi Interface:** The gateway to the outside world. This is where the GPRS network connects to the public **Internet** or a corporate intranet.
- **Gp Interface:** Similar to Gn, but used for connecting to GGSNs in **other PLMNs (Public Land Mobile Networks)**, enabling roaming data services.
- **Gc Interface:** Connects the GGSN to the **HLR**. This allows the GGSN to get subscriber information, which is necessary for certain advanced services.

4. Integration with the Legacy GSM Core

- **HLR (Home Location Register):** The central database that contains subscriber information and service profiles. Both the circuit-switched (MSC) and packet-switched (SGSN/GGSN) networks query the HLR to authenticate users and know what services they are allowed to access. The connection uses the **MAP (Mobile Application Part)** protocol.

Summary: How Data Flows in This GPRS Diagram

1. Your phone (MS) connects to a BTS.
2. The BSC routes your data session to the **SGSN**.

3. The SGSN checks with the HLR to see if you are a valid subscriber.
4. The SGSN then tunnels your data through the core network to the **GGSN**.
5. The **GGSN** acts as your router, giving you an IP address and connecting you to the **Internet**.
6. All your internet traffic (web browsing, email) flows through this path: **MS → BTS → BSC → SGSN → GGSN → Internet**.

This architecture was revolutionary because it introduced the efficient, "always-on" packet-switched model that is the foundation of all modern mobile data, from 3G to 4G and 5G. The **SGSN** and **GGSN** from GPRS evolved directly into the **S-GW (Serving Gateway)** and **P-GW (PDN Gateway)** in 4G LTE, as shown in your original PDF.

[Universal Mobile Telecommunications Standards \(UMTS\) :3G](#)

What is UMTS?

UMTS (Universal Mobile Telecommunications System) is the third-generation (3G) mobile cellular technology that succeeded GSM/GPRS (2G/2.5G). It was designed to provide higher data speeds and support a wide range of services, including voice, video calling, mobile internet, and multimedia messaging.

The Core Innovation: A New Radio Access Technology

The most significant shift from 2G to 3G was the change in the radio access technology.

- **2G (GSM)** used TDMA/FDMA
- **3G (UMTS)** uses W-CDMA (Wideband Code Division Multiple Access)

Key Differences of W-CDMA:

- **Wider Bandwidth:** Uses a **5 MHz** wide channel compared to the 200 kHz channels in GSM. This "wideband" characteristic is where it gets its name and is the primary reason for its higher data rates.
- **Code Division Multiple Access:** Instead of dividing users by time slots (TDMA) or frequency (FDMA), W-CDMA allows all users to transmit simultaneously over the entire frequency band. Users are separated by unique codes, providing better capacity and resistance to interference.

UMTS Network Architecture

The UMTS architecture introduced a new radio access network while evolving the core network from GPRS.

1. UMTS Radio Access Network (UTRAN)

This replaced the old GSM Base Station Subsystem (BSS).

- **Node B (pronounced "Node Bee"):** This is the 3G equivalent of the GSM BTS (Base Transceiver Station). It is the physical unit that communicates directly with User Equipment (UE) over the air interface.
- **RNC (Radio Network Controller):** This is the 3G equivalent of the GSM BSC (Base Station Controller). It controls multiple Node Bs. The RNC is more intelligent than the BSC, handling functions like soft handovers (a key feature of CDMA where a

phone can be connected to multiple Node Bs simultaneously), and managing radio resources.

2. UMTS Core Network (CN)

The core network was an evolution of the GPRS core, supporting both **Circuit-Switched (CS)** and **Packet-Switched (PS)** domains.

Circuit-Switched Domain (for Voice & SMS):

- **MSC (Mobile Switching Center):** The same as in 2G, it handles traditional voice calls and SMS.
- **VLR (Visitor Location Register):** Works with the MSC as in 2G.

Packet-Switched Domain (for Data):

- **SGSN (Serving GPRS Support Node):** Carried forward from GPRS, but upgraded for 3G. It handles packet routing, mobility management, and authentication for the packet-switched domain.
- **GGSN (Gateway GPRS Support Node):** Also carried forward from GPRS. It acts as the gateway to external packet data networks (like the Internet).

Common Databases:

- **HLR (Home Location Register):** The central subscriber database.
- **AUC (Authentication Center):** Provides authentication and encryption parameters.

Key Features and Capabilities of UMTS

1. **Higher Data Rates:**
 - a. **Release 99:** Offered theoretical peak data rates of **2 Mbps** (though typical user speeds were 384 kbps).
 - b. **HSDPA (High-Speed Downlink Packet Access - 3.5G):** Enhanced the downlink, boosting speeds to a theoretical **14 Mbps**.
 - c. **HSUPA (High-Speed Uplink Packet Access - 3.75G):** Enhanced the uplink for faster uploads.
2. **Support for Advanced Services:**
 - a. **Video Calling & Conferencing**
 - b. **Mobile TV** and video streaming
 - c. **High-Speed Internet Access** (web browsing, email with attachments)
 - d. **Location-Based Services**
 - e. **Online Gaming**
3. **Improved Voice Capacity and Quality:** W-CDMA provided more efficient use of spectrum and better voice quality.
4. **Global Roaming:** UMTS was designed as a global standard, allowing for seamless international roaming between 3G networks.

UMTS in the Evolutionary Path

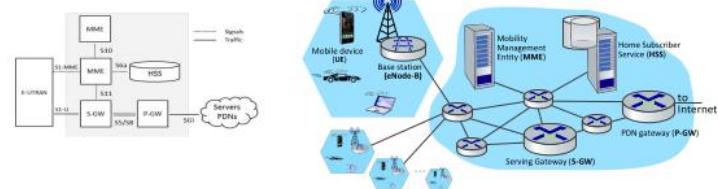
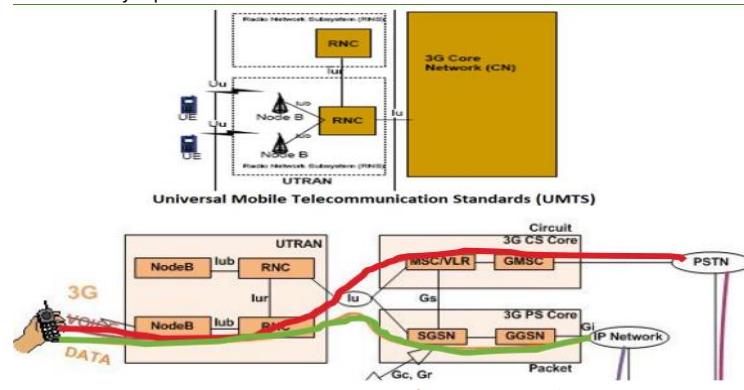
As shown in your final architecture diagram, UMTS is the direct successor to the GSM/GPRS path.

- **From 2.5G (GPRS) to 3G (UMTS):** The core network concept of **SGSN/GGSN** was retained and enhanced. The major change was in the **Radio Access Network**, which moved from **TDMA (BSC/BTS)** to **W-CDMA (RNC/Node B)**.
- **The "Two Data Paths":** As mentioned on Page 7 of your PDF, "3G and earlier maintains two data paths."
 - **Circuit-Switched Path (Voice[8 kbps and SMS/MMS):** UE → Node B → RNC → MSC → PSTN
 - **Packet-Switched Path (Data[All IP data]):** UE → Node B → RNC → SGSN → GGSN → Internet

This dual-domain architecture was a hallmark of 3G, bridging the old world of circuit-switched voice and the new world of packet-switched data.

Summary

UMTS (3G) was the technology that truly launched the mobile internet era for consumers. By introducing a high-speed, wideband radio interface (W-CDMA) and building upon the packet-switched core of GPRS, it enabled the rich multimedia services and "anytime, anywhere" internet access that we now take for granted. It served as the crucial bridge between the voice-centric world of 2G and the all-IP, data-centric world of **4G LTE**, which would eventually replace both the circuit-switched core and the W-CDMA air interface.



What is LTE?

Long Term Evolution (LTE) is the standard for high-speed wireless communication for mobile devices and data terminals. It is the technology that succeeded 3G UMTS and is officially designated as **4G**. The primary goals of LTE were to increase the speed and capacity of mobile networks using a new, simplified network architecture and a revolutionary radio access technology.

The Core Innovations of LTE

LTE introduced two fundamental shifts from its 3G predecessors:

1. **A New "All-IP" Core Network:** Everything, including voice, is treated as data packets.
2. **A New Radio Access Technology:** A move from W-CDMA to **OFDMA**.

1. The "All-IP" Network Architecture (The LTE Core Network - EPC)

As shown on Page 9 of your PDF, LTE introduced the **Evolved Packet Core (EPC)**, which is a purely **packet-switched** network. This was a radical departure from 2G/3G, which maintained separate paths for circuit-switched voice and packet-switched data.

Key Elements of the EPC:

- **eNodeB (Evolved Node B):** This is the 4G base station. A key innovation is that the eNodeB **absorbed the functions of the 3G RNC**. This creates a "flat" architecture, reducing latency and simplifying the network. The eNodeB handles all radio-related functions directly.
- **MME (Mobility Management Entity):** This is the control plane's brain. It handles signaling, including user authentication, tracking, and mobility management (handovers between eNodeBs), and establishing data paths. It does not handle the user's data traffic itself.
- **S-GW (Serving Gateway):** This is the data plane's router. It is the point of connection for the eNodeB and routes and forwards user data packets. It acts as the local anchor point during handovers between eNodeBs. It evolved from the **SGSN** of 2.5G/3G.
- **P-GW (PDN Gateway):** This is the gateway to the outside world. It connects the LTE network to external Packet Data Networks (PDNs), like the Internet. It assigns the user's IP address, performs policy enforcement, and charging. It is the direct evolution of the **GGSN**.

- **HSS (Home Subscriber Server):** This is the combined and enhanced version of the HLR and AUC from previous generations. It is a central database that contains user subscription and authentication information.

The "All-IP" Consequence: Since the circuit-switched core is gone, voice calls in LTE are handled as data packets using **VoLTE (Voice over LTE)**, which provides higher quality voice and faster call setup times.

2. The New Radio Access Technology: OFDMA and SC-FDMA

LTE abandoned the CDMA technology of 3G in favor of **OFDM-based** access schemes.

- **Downlink (eNodeB to UE): Uses OFDMA (Orthogonal Frequency Division Multiple Access)**
 - This is the multi-carrier technology we discussed earlier. The available spectrum is divided into many narrowband, orthogonal subcarriers.
 - **OFDMA** allows the base station to dynamically assign different groups of these subcarriers to different users *simultaneously*.
 - **Benefits:** Highly efficient in combating multipath fading, provides high spectral efficiency, and enables flexible scheduling.
- **Uplink (UE to eNodeB): Uses SC-FDMA (Single Carrier Frequency Division Multiple Access)**
 - SC-FDMA is a variant of OFDMA that has a lower **Peak-to-Average Power Ratio (PAPR)**.
 - **Why it's used:** A lower PAPR is more power-efficient, which is critical for mobile devices (UEs) to conserve battery life while still leveraging the benefits of orthogonal subcarriers.

Key Features and Capabilities of LTE

1. **High Data Rates:**
 - Theoretical peak downlink speeds can exceed 100 Mbps, with real-world speeds typically ranging from 5-50 Mbps, a significant jump from 3G.
2. **Low Latency:**
 - The flat, all-IP architecture significantly reduces round-trip time, making applications like online gaming and video conferencing much smoother.
3. **Simplified Architecture:**
 - By eliminating the RNC and integrating its functions into the eNodeB, the network becomes simpler, more robust, and cheaper to operate.
4. **Spectrum Flexibility:**
 - LTE can be deployed in various frequency bands and bandwidths (e.g., 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 20 MHz), giving operators great flexibility.

LTE in the Evolutionary Path

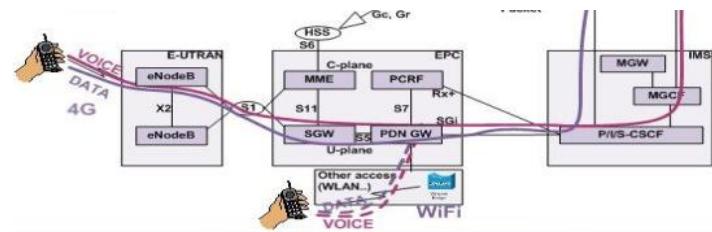
As your final architecture diagram correctly shows, LTE was the **grand unification point** where all previous technological paths (GSM/UMTS and CDMA2000) converged.

- **From 3G (UMTS) to 4G (LTE):** This was a **complete network overhaul**, not just an upgrade.
 - **Radio Access:** W-CDMA → **OFDMA/SC-FDMA**
 - **Core Network:** Circuit-Switched + Packet-Switched Core → **All-IP Evolved Packet Core (EPC)**
 - **Control Node:** RNC → **Its functions integrated into the eNodeB**

Summary

LTE (4G) was a revolutionary leap in mobile communications. By adopting an **all-IP, flat architecture** and a highly efficient **OFDM-based air interface**, it provided the necessary foundation for the modern mobile broadband experience. It enabled the widespread adoption of bandwidth-intensive applications like HD video streaming, mobile gaming, and advanced web services, setting the stage for the connected world we live in today and the further advancements of **5G**, which builds directly upon LTE's core principles.

4G LTE Network Architecture (Evolved Packet System):



4G LTE Network Architecture (Evolved Packet System)

The 4G LTE architecture represents a complete revolution from 3G, featuring a simplified, flat, all-IP network. It consists of two main parts:

1. **E-UTRAN (Evolved Universal Terrestrial Radio Access Network)** - The Radio Access Network
 2. **EPC (Evolved Packet Core)** - The Core Network
- Together, they form the **EPS (Evolved Packet System)**.
- ### 1. E-UTRAN: The Radio Access Network
- This is the wireless part of the network that connects user devices to the core network.
- **UE (User Equipment):** The user's mobile device (phone, tablet, modem).
 - **eNodeB (Evolved Node B):** The 4G base station.
 - **Key Innovation:** The eNodeB absorbs the functions of the 3G **RNC (Radio Network Controller)**. This creates a "flat" architecture, reducing latency and simplifying the network.
 - **Responsibilities:**
 - Radio Resource Management (RRM)
 - Header Compression

- Security (Encryption) on the air interface
- Connection Mobility Control
- Scheduling and dynamic resource allocation

Interface:

- **LTE-Uu:** The air interface between the UE and the eNodeB.

2. EPC (Evolved Packet Core): The "All-IP" Core Network

This is the heart of the 4G network, handling all data and control signaling. It is a purely **packet-switched** network.

Core Network Elements:

A. Control Plane Entities (The "Brains")

- **MME (Mobility Management Entity):**
 - **Function:** The central controller for the access network. It handles all control plane signaling.
 - **Key Tasks:**
 - User Authentication and Authorization (works with HSS)
 - NAS (Non-Access Stratum) Signaling
 - Tracking Area Management
 - Idle Mode UE tracking and paging
 - Selecting the S-GW and P-GW
- **HSS (Home Subscriber Server):**
 - **Function:** The central user database. It is the evolved version of the GSM HLR and AUC.
 - **Stores:** User identities, security information (authentication keys), and service profiles.

B. User Plane Entities (The "Muscle")

- **S-GW (Serving Gateway):**
 - **Function:** The data plane's router and anchor point. It is the point of contact for the eNodeB.
 - **Key Tasks:**
 - Routing and forwarding user data packets.
 - Acting as the local mobility anchor during handovers *between eNodeBs*.
 - Lawful interception.
 - **Evolution:** It replaces the user plane functions of the 2.5G/3G **SGSN**.
- **P-GW (PDN Gateway):**
 - **Function:** The gateway between the LTE network and the external world (the Internet).
 - **Key Tasks:**
 - Assigning the UE its IP address (DHCP function).
 - Connecting to external Packet Data Networks (PDNs) like the Internet or corporate networks.
 - Policy enforcement, charging, and per-user packet filtering.

- **Evolution:** It is the direct successor to the 2.5G/3G **GGSN**.

C. Interconnections (Key Interfaces)

- **S1 Interface:** Connects the E-UTRAN to the EPC.
 - **S1-MME:** For control signaling (between eNodeB and MME).
 - **S1-U:** For user data traffic (between eNodeB and S-GW).
- **S6a Interface:** Connects the MME to the HSS for subscriber authentication and authorization.
- **S11 Interface:** Connects the MME to the S-GW for tunnel management and handover signaling.
- **S5/S8 Interface:** Connects the S-GW to the P-GW. S8 is used when roaming (between S-GW in visited network and P-GW in home network).

How a Data Session Works in LTE:

1. **Attach Procedure:** The UE connects to the nearest eNodeB.
2. **Authentication:** The eNodeB contacts the MME, which authenticates the user with the HSS.
3. **Session Establishment:** The MME selects an S-GW and P-GW, establishing a data path (bearer).
4. **IP Assignment:** The P-GW assigns an IP address to the UE.
5. **Data Flow:** User data flows: **UE ↔ eNodeB ↔ S-GW ↔ P-GW ↔ Internet**

Key Revolutionary Aspects of 4G LTE:

1. **All-IP Network:** Everything (including voice as VoLTE) is treated as data packets.
2. **Flat Architecture:** Elimination of the RNC reduces latency and cost.
3. **Simplified Core:** Separation of Control Plane (MME) and User Plane (S-GW/P-GW) allows for independent scaling.
4. **New Radio Technology:** Uses OFDMA (Downlink) and SC-FDMA (Uplink) instead of W-CDMA, providing higher spectral efficiency and better performance.

This architecture is the foundation that enables the high-speed, low-latency experience of true 4G networks.

1. What is the justification for using hexagonal cells in cellular networks?

Of course. The use of hexagonal cells in cellular network design is a foundational concept, and the justification is primarily based on **geometry, efficiency, and simplification for modeling**.

While perfect hexagons don't exist in the real world due to terrain and signal variations, they are the ideal theoretical shape for planning. Here's a breakdown of the justification:

1. Geometric Efficiency: Optimal Coverage with Minimal Gaps

The primary goal is to cover a large area with contiguous circles of signal coverage (cells). However, circles themselves are inefficient for tiling a plane.

- **Circles Leave Gaps:** When you pack circles together, there are unavoidable uncovered gaps between them.
- **Squares and Triangles Have Issues:** Polygons that can tile a plane perfectly include squares, triangles, and hexagons.
 - **Squares:** The distance from the center of a square to its farthest point (the corner) is significantly longer than the distance to the middle of a side. This creates "hot" and "cold" spots for signal strength, making interference management difficult.
 - **Triangles:** They have very short distances to the vertices, leading to even more pronounced coverage inconsistencies.
- **Hexagons are the "Goldilocks" Shape:** A hexagon is the closest geometric shape to a circle that can tessellate (tile a plane) without gaps. It provides the most uniform coverage, with the distance from the center to any point on the perimeter being more consistent than with a square or triangle.

Visual Comparison: Imagine covering a floor with tiles. Hexagons fit together more seamlessly for coverage purposes than any other regular polygon, mimicking a circle's uniformity while leaving no gaps.

2. Frequency Reuse and Interference Management

This is the most critical practical reason. The same set of radio frequencies must be reused across different cells to serve millions of users with a limited radio spectrum.

- **The Co-channel Interference Problem:** If two adjacent cells used the same frequencies, they would cause severe interference for each other. Therefore, cells using the same set of frequencies (a "cluster") must be separated by a sufficient distance.

- **Hexagons Enable Systematic Planning:** The hexagonal grid allows for a clear, mathematical calculation of the **frequency reuse distance**.

- The reuse distance is determined by the formula: $(D = R \sqrt{3N})$, where (R) is the cell radius and (N) is the number of cells in a cluster (which can only be specific values like 3, 4, 7, 12, etc., based on the hexagonal geometry: $(N = i^2 + ij + j^2)$).
- This predictable structure is crucial for network engineers to design clusters that minimize co-channel interference effectively and reliably.

3. Simplified Modeling and Capacity Planning

Hexagons make the complex math of network capacity and expansion much more manageable.

- **Predictable Growth:** The hexagonal pattern allows for straightforward **cell splitting**. When a cell becomes too busy (e.g., in a city center), it can be systematically split into smaller hexagons to increase capacity. The geometry of this process is clean and predictable.
- **Handoff Analysis:** The points where a user's call is transferred from one cell to another (handoff) occur along the edges of the hexagons. The uniform shape simplifies the modeling and analysis of handoff rates and locations.

The Real-World Caveat

It's crucial to remember that **hexagons are a theoretical model**. In reality, cell coverage is an irregular, blob-like shape due to:

- **Terrain:** Hills, valleys, and buildings block and reflect signals.
- **Man-made Structures:** Skyscrapers and tunnels create shadows and corridors.
- **Antenna Type and Placement:** The direction and power of the antenna define the coverage area.

Despite this, the hexagonal cell remains the indispensable starting point for all cellular network design, providing the clean, efficient, and scalable framework upon which real-world, messy coverage is built and managed.

2. Does an ad hoc network suffer any disadvantage for lacking infrastructure compared to a cellular network ?

Absolutely. Yes, the lack of infrastructure in an ad hoc network is the source of its core disadvantages when compared to a cellular network. This trade-off

is fundamental: you gain spontaneity and independence but sacrifice performance, scalability, and management.

Here are the key disadvantages of an ad-hoc network due to its lack of infrastructure, point by point:

1. Limited Range and Connectivity

- **Cellular:** Base stations have high power and tall antennas, providing coverage over miles.
- **Ad Hoc:** Devices have low-power radios. Communication is typically short-range (e.g., Bluetooth: 10m, Wi-Fi Direct: ~100m). To reach a node outside this range, data must hop through intermediate nodes (multi-hop routing), which introduces complexity and delay.

2. Dynamic and Unstable Topology

- **Cellular:** The network topology is fixed and stable. Base stations don't move.
- **Ad Hoc:** Nodes are often mobile. Connections are constantly being formed and broken as nodes move in and out of range. This requires the routing protocol to continuously rediscover paths, consuming bandwidth and processing power, and leading to potential packet loss.

3. Higher Latency and Lower Throughput

- **Cellular:** A direct, dedicated link (or shared channel) exists between the device and the base station.
- **Ad Hoc:** Multi-hop routing means data takes a longer, less direct path. Each "hop" adds processing delay. Furthermore, wireless interference from neighboring nodes and the overhead of routing protocols themselves significantly reduce the effective data throughput available to each user.

4. Severe Scalability Issues

- **Cellular:** The infrastructure is designed for scalability. Adding a new user has a minimal impact on the core network.
- **Ad Hoc:** As more nodes join, the network becomes congested. The control traffic (e.g., "hello" messages, route discovery broadcasts) grows exponentially, consuming a large portion of the available bandwidth. The network can quickly become unstable and inefficient with even a moderate number of nodes.

5. Higher Power Consumption for End Devices

- **Cellular:** The phone mainly communicates with one fixed base station. The infrastructure handles the complex tasks of routing and long-distance backhaul, allowing the phone to conserve power.
- **Ad Hoc:** Each node acts as both a host and a router. It must constantly process routing information and forward packets for its neighbors. This CPU and radio activity drains the battery much faster, which is a critical concern for mobile devices.

6. Security Vulnerabilities

- **Cellular:** Has a centralized trust model (the operator) and a protected, wired backbone network.
- **Ad Hoc:** The lack of a central authority makes key management and trust establishment very difficult. The network is vulnerable to:
 - **Eavesdropping:** Wireless medium is open.
 - **Spoofing:** A malicious node can easily join.
 - **Wormhole & Blackhole Attacks:** Malicious nodes can disrupt routing.
 - **No Centralized Monitoring:** There is no single point to detect intrusions or monitor traffic.

4. Explain how to improve wireless system capacity by reusing frequencies?

Since the radio spectrum is a limited and expensive resource, we cannot assign a unique frequency to every user. Instead, we reuse the same frequencies across a geographic area.

Here's a detailed explanation of how it works.

The Core Principle: The Cellular Concept

The entire system is built on the **cellular concept**:

1. **Divide a Large Area into Small Cells:** A large coverage area (like a city) is divided into smaller, contiguous areas called **cells**. Each cell is served by a Base Station (a tower with antennas).
2. **Assign a Subset of Frequencies to Each Cell:** The total available radio spectrum is divided into sets of frequencies, called **channel sets**.
3. **Reuse Frequencies in Non-Adjacent Cells:** The same channel set is reused in multiple cells across the network, but with a key constraint: **cells that use the same frequencies must be separated by a sufficient distance to avoid interference**.

This interference is called **Co-channel Interference**, and managing it is the key to effective frequency reuse.

The Key Mechanism: The Reuse Pattern (Cluster)

How to Determine the Reuse Distance (D)

The distance between the centers of two cells using the same channel set is the **Reuse Distance (D)**. It is calculated using the cluster size and the cell radius (R).

Formula: $(D = R \sqrt{3N})$

Where: (D) = Reuse Distance, (R) = Radius of a cell, (N) = Cluster Size (Reuse Factor)

The Rule: A smaller N (e.g., $N=3$) means the same frequencies are reused more aggressively, which increases capacity. However, it also reduces D , bringing co-channel cells closer together and increasing interference.

Therefore, choosing N is a trade-off between **Capacity** and **Interference**.

How to Improve Capacity: Techniques Based on Reuse

There are several strategies to increase capacity by making the reuse of frequencies more efficient.

1. Cell Splitting

This is the most direct method.

- **Concept:** You split a large, congested cell (e.g., one covering a large radius, R) into several smaller cells (with a smaller radius, r).

Effect:

- More cells are created in the same geographic area.
- Because the cluster pattern N remains the same, the reuse distance D for the smaller cells becomes smaller.
- This means the same frequencies can be reused *more times* within the same city.
- **Result:** Dramatic increase in capacity.

Analogy: Imagine a large library with one busy librarian (the macrocell). To serve more people, you divide the library into smaller sections, each with its own librarian (a microcell). You can now help many more people simultaneously.

2. Sectoring

This technique reduces interference, which allows for the use of smaller, more aggressive cluster sizes (like $N=3$ instead of $N=7$).

- **Concept:** Replace the single omnidirectional antenna at a cell site with multiple directional antennas. Each antenna covers a specific portion of the cell, called a **sector**.

Effect:

- A typical 120-degree sectoring divides a cell into 3 sectors.
- The base station now uses different channel sets for each sector.
- This focuses transmission and reception, significantly reducing the number of co-channel interferers. Instead of 6 interfering cells (in a hexagon model), there might only be 2.

- **Result:** With interference reduced, you can safely deploy a system with a smaller cluster size N , thereby reusing frequencies more often and increasing capacity.

*Estimate the path loss of a 900 MHz signal if the height of the transmitter and receiver are 40m and 5m, respectively. The receiver is 10 km away from the transmitter.

Ans:Deepseek

*What is the advantage of a hexagonal structure over a rectangular one for a cell in a cellular system?

Of course. This is a foundational concept in cellular network design. The advantage of a hexagonal structure over a rectangular one is primarily due to **geometry and efficiency**. Here's a breakdown of the key advantages:

1. Geometric Efficiency: Approximating a Circle

- **Hexagon:** A hexagon is the closest regular polygon to a circle that can tessellate (tile a plane) without gaps or overlaps. The radio coverage of a base station is essentially circular.
- **Rectangle:** A rectangle has corners that are significantly farther from the center than the edges. This creates "hot" and "cold" spots—excellent coverage along the edges and poor coverage in the corners, leading to inefficient and uneven coverage.

Result: Hexagons provide the most **uniform coverage**, minimizing dead zones and ensuring a more consistent signal strength throughout the cell.

2. Simplified and Predictable Frequency Reuse

This is the most critical advantage for system capacity.

- **Hexagon:** The geometry allows for a clear, systematic, and predictable calculation of the **co-channel reuse distance (D)**. The relationship is given by $(D = R\sqrt{3N})$, where (R) is the cell radius and (N) is the cluster size (e.g., 4, 7, 12). This predictability is vital for placing co-channel cells at the exact distance needed to manage interference.
- **Rectangle:** The reuse pattern is less intuitive and can lead to uneven distances between co-channel cells. This makes it difficult to consistently maintain the required Signal-to-Interference Ratio (SIR) across the entire network.

Result: Hexagons enable a **structured and mathematical approach** to frequency planning, which is essential for maximizing capacity while controlling interference.

3. Lower Complexity in Handoff Analysis

- **Hexagon:** In a hexagonal grid, a mobile user moving from one cell to another crosses a well-defined, linear cell boundary. This simplifies the modeling and analysis of handoff rates and locations.
- **Rectangle:** The boundaries are more complex, and a user can potentially be in a "corner" region where they are nearly equidistant from multiple base stations, complicating the handoff decision.

Result: Hexagons lead to **more reliable and predictable handoffs**.

4. Fewer Co-channel Interferers in the First Tier

- **Hexagon:** In a perfectly laid-out hexagonal grid, the number of first-tier co-channel cells (the closest ones using the same frequency) is always **six**, regardless of the cluster size (N). This simplifies interference calculations.
- **Rectangle:** Depending on the specific tiling, a rectangular grid can have a different number of first-tier interferers (e.g., four or eight), making the interference environment less predictable.

Result: A consistent number of interferers (**six**) simplifies the SIR derivation and network design.

Visual Comparison

Feature	Hexagonal Cell	Rectangular Cell
Coverage	Uniform. Consistent distance from center to edge.	Non-uniform. Poor coverage in corners.
Tessellation	Perfect, no gaps.	Perfect, no gaps.
Frequency Reuse	Predictable & Systematic. Clear formula for reuse distance.	Less Predictable. Inconsistent reuse distances.
Handoff Analysis	Simpler. Well-defined, linear boundaries.	More Complex. Ambiguous corner regions.
Co-channel Interferers	Consistently 6 in the first tier.	Can vary (e.g., 4 or 8).

Conclusion: Why the Hexagon Wins

While a rectangular grid can also tile a plane, the **hexagon provides the optimal compromise** between the ideal circular coverage pattern and the practical need for a simple, predictable, and efficient tessellating shape for system-wide planning.

Its geometric properties directly enable the core principles of cellular technology:

1. **Efficient use of the radio spectrum** through predictable frequency reuse.
2. **Controlled interference** through a fixed co-channel reuse distance.
3. **Seamless mobility** through simplified handoff analysis.

For these reasons, the hexagon remains the universal standard for modeling and designing cellular networks, even though real-world cell coverage is an irregular "blob" due to terrain and obstacles. The hexagonal model provides the essential theoretical framework.

7. Show that the raw data rate of GSM Traffic channel TCH-F is 22.8 kbps.

1. GSM full-rate vocoder produces a 260-bit speech frame every 20 ms ($\rightarrow 13 \text{ kb/s payload}$).
2. At the physical layer (L1) that 260-bit frame is channel-coded into a 456-bit L1 frame (FEC + parity/interleaving).
3. Raw channel rate = $(456 \text{ bits}/20\text{ms}) = 456/0.02\text{bps} = 22,800\text{bps} = 22.8 \text{ kbps}$.

Hence TCH-F raw data rate = **22.8 kbps**.

8. Discuss the role of power control in mitigating the near-far problem in GSM cellular systems.

Understanding the Near-Far Problem

First, it's crucial to define the issue:

The **Near-Far Problem** is a classic issue in wireless communication systems where a receiver (e.g., a Base Station - BS) tries to listen to a weak, distant transmitter (Mobile Station - MS) while being overwhelmed by a strong, nearby transmitter using the same frequency.

- **Scenario:** Imagine a BS with two connected mobile phones:
 - **MS A (The "Far" User):** Located at the cell edge, its signal is weak due to path loss.
 - **MS B (The "Near" User):** Located very close to the BS, its signal is very strong.
- **Problem:** If both MS A and MS B transmit at the same power and on the same frequency (in a different time slot in GSM's TDMA), the powerful signal from the "near" user (MS B) can:
 - **Desensitize the Receiver:** Overload the BS's receiver circuitry, making it "deaf" to the weak signal from MS A.
 - **Cause Adjacent Channel Interference (ACI):** Even if they are on different, but adjacent, frequencies, the strong signal can "bleed over" into the adjacent channel due to imperfect receiver filters, drowning out the desired weak signal.

In essence, the near-far problem can **block access for cell-edge users and drastically reduce the overall system capacity**.

The Role of Power Control in Mitigating this Problem

Power control is the primary mechanism used to combat the near-far problem. Its role can be broken down into three key areas:

1. Equalizing the Received Power at the Base Station (Uplink Power Control)

This is the most direct application.

- **How it works:** The BS continuously measures the signal strength (RXLEV) and quality (RXQUAL) from each mobile. Based on these measurements, it sends power control commands to each MS, instructing it to **increase or decrease its transmit power**.
- **The Goal:** The objective is for *all* signals, from both "near" and "far" users, to arrive at the BS with just enough power to maintain a good connection **and no more**.
- **Result:** The "near" user (MS B) is commanded to transmit at a very low power, while the "far" user (MS A) is commanded to transmit at high power. This equalizes the received power levels at the BS, preventing the strong signal from the near user from masking the weak signal from the far user.

2. Reducing Co-channel and Adjacent Channel Interference

By lowering the transmit power of mobiles, especially those close to the BS, power control reduces the overall interference in the system.

- **Co-channel Interference:** A mobile station's signal doesn't just reach its own BS; it can also cause interference in a *distant co-channel cell* using the same frequency. By reducing power, this distant interference is minimized, improving the Signal-to-Interference Ratio (SIR) in those cells and allowing for tighter frequency reuse (smaller cluster size N), which increases network capacity.
- **Adjacent Channel Interference (ACI):** Lower transmit power directly reduces the amount of power that can "leak" into adjacent frequency channels, which is a primary cause of the near-far effect in the frequency domain.

3. Conserving Battery Life in Mobile Stations

This is a significant secondary benefit that contributes to the system's efficiency.

- **How it works:** A mobile phone close to the BS does not need to transmit at full power. By commanding it to use the minimum necessary power, the phone's power amplifier can run at a lower level.
- **Result:** This dramatically **conserves battery life**, leading to longer talk and standby times for the user.

How Power Control is Implemented in GSM

GSM power control is a sophisticated, closed-loop process:

1. **Measurement:** The BS measures the received signal level (RXLEV) and bit error rate (RXQUAL) from the MS.
2. **Decision:** The BS compares these measurements against pre-defined thresholds set by the network operator.
3. **Command:** The BS sends a **Power Control Command** to the MS on the Slow Associated Control Channel (SACCH), which is paired with the traffic channel (TCH). This command tells the MS to change its power level by a certain step (e.g., 2 dB steps in GSM).
4. **Execution:** The MS adjusts its transmitter power accordingly.
5. **Repetition:** This process happens continuously, approximately twice every second.

Summary: The Impact of Power Control

In the context of mitigating the near-far problem, power control acts as a vital balancing mechanism. Its role is to:

- **Ensure Fairness:** It gives cell-edge users a fair chance to be heard by the network by boosting their power and suppressing the power of cell-center users.
- **Maximize Capacity:** By managing interference, it allows the network to support more users per cell and use the radio spectrum more efficiently.
- **Improve Quality:** It maintains a consistent signal quality for all users, regardless of their location within the cell.
- **Enhance Efficiency:** It prolongs the battery life of mobile handsets.

Without power control, a GSM network would be highly unstable, with cell-edge users experiencing frequent dropped calls and poor voice quality, while the overall system capacity would be severely limited.

9. Which TDMA frames of TCH-F are used to communicate signal strengths?

The straightforward answer is that no TCH-F (Traffic Channel - Full rate) frames are directly used to communicate signal strength measurements.

Here is the precise breakdown of why and how it actually works:

The Exact Point

Separation of Traffic and Control: In GSM, the Traffic Channel (TCH) is dedicated solely to carrying the user's encoded voice or data. The control signaling, which includes things like signal strength measurements, handover commands, and power control, is carried on separate, dedicated control channels.

The Correct Channels: SACCH The signal strength measurements (both from the Mobile Station (MS) to the Base Transceiver Station (BTS) and from the BTS to the MS) are communicated using the Slow Associated Control Channel (SACCH).

How it's associated: Every TCH (and every SDCCH) is permanently paired with an SACCH.

The Timing: The SACCH uses specific frames within the channel's multiframe structure. For a TCH, one SACCH frame is sent every 26-frame TCH multiframe. Specifically, frame number 12 (in the 0-25 numbering of the TCH/F multiframe) is dedicated to the SACCH.

12. Describe the similarity and differences of Wireless LAN and cellular networks. Similarities Between WLAN and Cellular Networks

At a fundamental level, both systems provide untethered, wireless connectivity and share some core concepts:

1. **Wireless Access:** Both use radio waves to connect user devices (laptops, smartphones, etc.) to a network without physical cables.

2. **Use of Licensed and Unlicensed Spectrum:** While traditionally distinct, both now utilize various radio frequencies. Cellular primarily uses licensed spectrum, while Wi-Fi uses unlicensed bands (2.4 GHz, 5 GHz, 6 GHz). However, technologies like LTE-LAA and CBRS allow cellular to use unlicensed spectrum, and Wi-Fi can be used for offloading cellular data.
3. **Network Architecture:** Both have a structured architecture involving access points (WLAN) or base stations (Cellular) that manage communication with multiple user devices and connect to a central backbone network.
4. **Core Network Technologies:** Both often use IP (Internet Protocol) as the fundamental protocol for data routing in their core network backhaul.
5. **Mobility Support:** Both systems offer a degree of mobility, allowing users to move while connected. They manage this mobility through handoffs (or roaming) between access points or cell towers.

Differences Between WLAN and Cellular Networks

Despite their similarities, they are designed for different primary use cases, leading to significant differences. The table below summarizes the key distinctions.

Feature	Wireless LAN (Wi-Fi)	Cellular Network (e.g., 5G, LTE)
Primary Purpose & Design	Wireless extension of a local network. Designed for high-speed, low-cost data in limited areas.	Wide-area voice and data coverage. Designed for seamless mobility and broad geographic coverage.
Coverage Area	Short Range (typically < 100 meters indoors). Limited to homes, offices, campuses, hotspots.	Long Range (up to several kilometers per cell). Provides regional, national, and international coverage.
Governance & Spectrum	Uses Unlicensed Spectrum (free to use, but shared with others). Governed by IEEE 802.11 standards.	Uses Licensed Spectrum (exclusively owned by operators, reducing interference). Governed by 3GPP standards.
Infrastructure & Cost	Decentralized, user-deployed. Inexpensive Access Points and Routers. Cost is borne by the end-user or organization.	Centralized, operator-deployed. Expensive Macrocell towers, core network. Cost is borne by the service provider.

Business Model	Free at point of use (after purchasing equipment). No per-bit billing.	Subscription-based. Users pay for data, voice, and SMS plans.
Quality of Service (QoS) & Reliability	" Best Effort " Service. Performance can degrade with more users/interference. Less reliable for real-time communication.	Guaranteed Service Level. Engineered for high reliability and availability. Supports strict QoS for voice and video.
Mobility & Seamless	Nomadic/Low Mobility. Handoffs between APs can be slow and may cause brief interruptions.	High Mobility & Seamless Handoff. Designed for handoffs at highway speeds without dropping calls.
Power Consumption	Higher power consumption in user devices, as high data rate is prioritized.	Optimized for lower power to maximize battery life in mobile devices.
Typical Data Rates	Very High Peak Speeds (can exceed 1 Gbps) but shared among all users in the coverage area.	Lower Peak Speeds per User (though 5G is competitive), but capacity is managed across the entire network.

Key Takeaways and The Blurring Line

- **Cellular** is like a **public utility**: It provides a reliable, seamless, and wide-area service for which you pay a monthly bill. Its core strength is coverage and mobility.
- **Wi-Fi** is like a **cordless phone for your internet**: It provides a very high-speed, cheap connection within a limited, private area like your home or office. Its core strength is cost-effective bandwidth in local areas.

The Convergence: The line between Wi-Fi and cellular is increasingly blurring:

- **Wi-Fi Calling/Cellular Offloading:** Smartphones automatically use Wi-Fi networks for voice calls and data to relieve congestion on cellular networks.
- **5G and Wi-Fi 6:** Both new generations use similar underlying technologies (e.g., OFDMA, MU-MIMO) to improve efficiency and performance.
- **Private Cellular Networks:** Companies are now using licensed (CBRS) and unlicensed spectrum to deploy their own local 4G/5G networks that behave like a more reliable, secure version of Wi-Fi for industrial applications.

In essence, they are complementary technologies. Wi-Fi dominates indoors and in fixed locations where cost and speed are paramount, while cellular provides the connective fabric that keeps us online everywhere else.

13. Suppose you are calling a GSM mobile, which is currently roaming, from a land phone. Describe the procedure in detail for connection establishment between your land phone and the mobile.

This procedure is known as **Mobile Terminated Call (MTC)** setup.

Pre-Condition: The Mobile is Roaming

This means the mobile station (MS) is located in a network different from its home network. It has already performed a "Location Update" procedure, so the visited network knows it is there, and the home network knows *where* it is.

Key Players Involved:

- **PSTN:** Public Switched Telephone Network (the landline network).
- **GMSC (Gateway MSC):** The MSC in the mobile's **home network** that acts as a gateway between the PSTN and the GSM network. It is the first point of contact from the fixed network.
- **HLR (Home Location Register):** The central database in the **home network** containing all subscriber data and their current location (i.e., the address of the current VLR).
- **VLR (Visitor Location Register):** The database in the **visited network** that temporarily stores subscription data for the roaming mobile.
- **MSC (Mobile Switching Center):** The switch in the **visited network** that the mobile is currently connected to. It handles call switching and mobility management.
- **BSC (Base Station Controller):** Manages multiple BTSs in the visited network.
- **BTS (Base Transceiver Station):** The cell tower in the visited network.
- **MS (Mobile Station):** The phone you are calling.

Step-by-Step Call Establishment Procedure

The entire process is visualized in the diagram from your PDF (Page 12, "Mobile Terminated Call") and can be broken down into the following phases:

Phase 1: Routing the Call to the Home Network's Gateway

1. **Call Initiation (Step 1):** You dial the mobile subscriber's number (MSISDN) from your landline phone.
2. **PSTN Routing (Step 2):** The PSTN, based on the dialed number, routes the call to the **Gateway MSC (GMSC)** in the mobile's **home network**. The PSTN does not know the mobile's current location; it only knows which network it belongs to.

Phase 2: Finding the Roaming Mobile's Current Location

3. **HLR Query (Step 3):** The GMSC does not know where the mobile is. It sends a signal message to the **HLR**, providing the subscriber's identity (MSISDN), and asks "How do I route this call?"
4. **VLR Request (Steps 4 & 5):** The HLR, which has stored the address of the **VLR** where the mobile is currently registered (from the last location update), sends a request to that VLR in the **visited network**. It requests a **Mobile Station Roaming Number (MSRN)**. The MSRN is a temporary, routable telephone number that acts like a "dummy" location pointer.
5. **Providing the MSRN (Steps 6 & 7):** The VLR assigns an MSRN from its pool and sends it back to the HLR. The HLR then forwards this MSRN to the requesting GMSC.

Phase 3: Routing the Call to the Visited Network

6. **Call Re-routing (Steps 8 & 9):** The GMSC now has a routable number (the MSRN). It uses this MSRN to re-route the incoming call through the PSTN (or dedicated lines) to the **specific MSC** in the **visited network** that is currently serving the mobile.

Phase 4: Locating and Paging the Mobile in the Visited Cell

7. **Call Arrival at Serving MSC (Step 10):** The call arrives at the serving MSC in the visited network. The MSC asks its associated VLR for the subscriber's data and the current Location Area (LA) where the mobile is located.
8. **Paging (Steps 11 & 12):** The MSC cannot know the exact cell. So, it sends a **paging request** through all the BSCs and BTSs in the mobile's last known **Location Area**. This is a broadcast message saying "Mobile X, you have a call."
9. **Paging Response (Step 13):** The mobile station (MS) hears its identity in the paging message. It responds through the nearest BTS and BSC to the MSC, saying "I am here."

Phase 5: Security, Alerting, and Connection

10. **Security Checks (Steps 14 & 15):** The MSC initiates the GSM security procedures:
 - a. **Authentication:** The network challenges the mobile to prove its identity using a secret key (Ki) stored on the SIM card.
 - b. **Ciphering:** Once authenticated, the network activates encryption on the radio link to prevent eavesdropping.
 - c. **IMEI Check:** The Equipment Identity Register (EIR) may be checked to see if the phone itself is reported as stolen.
11. **Call Alerting (Step 16):** After successful security checks, the MSC sends a "setup" message to the mobile. The mobile phone starts **ringing** (alerting the user).
12. **Connection Established (Step 17):** When the user answers the call, the mobile sends a "connect" message back to the MSC. The MSC completes the voice path through the BSC and BTS to the mobile and also back to the GMSC and the PSTN.

- 13. Conversation Begins:** The two-way voice communication is now established between your landline and the roaming GSM mobile.

Summary Flowchart

PSTN (Caller) -> GMSC (Home Network) -> HLR (Home Network) -> VLR (Visited Network)
-> MSC (Visited Network) -> BSC -> BTS -> MS (Callee)

14. What do you mean by a half rate and a full rate TCH in a GSM cellular network? Determine the capacity of half rate and full rate TCH channels.

In GSM, a Traffic Channel (TCH) carries the user's encoded voice. The terms "Full Rate" (TCH/F) and "Half Rate" (TCH/H) refer to two different speech codecs that trade off **voice quality for network capacity**.

- **Full Rate Traffic Channel (TCH/F):**

- Uses the original GSM speech codec.
- It requires one physical channel (one Time Slot per TDMA frame) to carry one voice call.
- **Voice Quality:** Standard, acceptable quality.
- **Capacity:** Carries **1 call** per time slot.

- **Half Rate Traffic Channel (TCH/H):**

- Uses a more advanced, efficient speech codec that compresses voice into roughly half the number of bits.
- It allows two voice calls to share the same physical channel (the same Time Slot).
- **Voice Quality:** Lower than Full Rate, as more compression introduces more artifacts.
- **Capacity:** Carries **2 calls** per time slot.

Analogy: Imagine a highway lane.

- **Full Rate** is like allowing only cars. One car per lane.
- **Half Rate** is like allowing motorcycles. Two motorcycles can fit in the same lane, doubling the number of vehicles, but the experience is less comfortable (lower quality).

Determining the Capacity of TCH/F and TCH/H

"Capacity" here can mean two things:

1. The **raw data rate** available for the encoded voice.
2. The **number of simultaneous calls** a cell can handle.

We will calculate both based on the GSM frame structure from your provided document.

Step 1: The Foundation - The 26-Multiframe for TCH

As per the document, a Traffic Channel is organized in a **26-Multiframe** that is **120 ms long**. It contains 26 TDMA frames, allocated as follows:

- **24 frames** for user traffic (TCH).
- **1 frame** for the Slow Associated Control Channel (SACCH).
- **1 frame** is idle.

Each TDMA frame contains **8 time slots**. Each time slot (a "Normal Burst") carries **2 x 57 = 114 data bits** (as shown in the burst structure: 3 | 57 | 1 | 26 | 1 | 57 | 3).

Step 2: Capacity of a Full Rate TCH (TCH/F)

- **A. Data Rate (Throughput):**

- Bits per 120 ms multiframe: 24 frames * 114 bits/frame = 2736 bits
- Data Rate: 2736 bits / 0.120 seconds = 22,800 bps or **22.8 kbps**.
- This is the **raw data rate**. After channel coding (FEC), the actual user data rate for the voice call itself is **13 kbps**.

- **B. Call Capacity (Number of Users):**

- In TCH/F, **one time slot carries one call**.
- Therefore, the number of simultaneous calls in a cell is equal to the number of traffic channels (time slots) configured at the BTS.

Step 3: Capacity of a Half Rate TCH (TCH/H)

- **A. Data Rate (Throughput) per User:**

- The physical channel is shared, but the data rate is calculated *per user*.
- The total bits in the multiframe are the same: 24 frames * 114 bits/frame = 2736 bits.
- These 2736 bits are now shared between **two users**.
- Data Rate **per user**: (2736 bits / 2) / 0.120 seconds = 11,400 bps or **11.4 kbps**.
- The actual user data rate for the voice call after channel coding is approximately **5.6 kbps**.

- **B. Call Capacity (Number of Users):**

- In TCH/H, **one time slot carries two calls**.
- Therefore, the number of simultaneous calls in a cell is **doubled** compared to a Full Rate configuration using the same number of time slots.

Summary and Comparison

Feature	Full Rate TCH (TCH/F)	Half Rate TCH (TCH/H)
Concept	One call per time slot.	Two calls per time slot.
Raw Data Rate	22.8 kbps	11.4 kbps (per user)
Effective Voice Codec Rate	~13 kbps	~5.6 kbps
Voice Quality	Good, Standard	Lower, More compressed
Call Capacity	1 call / time slot	2 calls / time slot
Use Case	Default mode, used when network capacity is sufficient.	Used during network congestion (e.g., in a busy hour at a stadium) to serve more users.

15. Discuss various dedicated control channels used in GSM.

Of course, in GSM, Dedicated Control Channels (DCCHs) are two-way channels used for signaling and control between the network and a *specific* mobile station, after a dedicated connection has been established.

They are crucial for maintaining an ongoing call or a temporary signaling connection.

There are three main types of Dedicated Control Channels, each serving a distinct and critical purpose.

The Three Dedicated Control Channels (DCCHs)

1. SDCCH (Stand-alone Dedicated Control Channel)

- Purpose:** This is the "signaling workhorse" used *before* a traffic channel (TCH) is assigned. It carries all signaling information when a call is *not* in progress.
- When is it used?**
 - During the **call setup** phase, for exchanging messages with the MSC/VLR.
 - For **Location Area Updates** (when the mobile moves to a new area).
 - For **Subscriber Authentication** and ciphering mode setting.
 - For **SMS** sending and receiving.
 - For **Supplementary Service** activation (e.g., call forwarding, etc.).
- Key Characteristic:** "Stand-alone" means it operates independently of a Traffic Channel. A mobile can be assigned an SDCCH without having a TCH, which is efficient for tasks like sending an SMS without making a call.
- Speed & Structure:** It is a slow channel, mapped onto a dedicated time slot in a 51-multiframe structure.

2. SACCH (Slow Associated Control Channel)

- Purpose:** This channel is *associated* with either a TCH or an SDCCH. It carries routine, non-urgent control information *during* an ongoing call or signaling connection.
- When is it used?**
 - Periodic Reporting:** It continuously sends **Measurement Reports** from the MS to the BSC about the signal strength and quality of the serving cell and its neighboring cells. This is vital for deciding on **handovers**.
 - Power Control:** The BSC uses the SACCH to send commands to the MS to adjust its transmission power.
 - Timing Advance:** The network uses it to send timing advance commands to the MS to synchronize transmission and avoid overlapping time slots.
- Key Characteristic:** "Slow" and "Associated." It is always paired with a TCH or SDCCH. In the 26-multiframe used for a TCH, one specific frame (frame number 12) is dedicated to the SACCH.
- Speed:** It is very slow, as it only gets one frame every 120 ms (in a 26-multiframe).

3. FACCH (Fast Associated Control Channel)

- Purpose:** This channel is used for *urgent* signaling that cannot wait for the slow SACCH, primarily during an active call. It is a "stealing" channel.
- When is it used?**
 - Handover Execution:** The most critical use. The command to switch to a new cell and the subsequent signaling are sent via the FACCH.
 - Urgent signaling during a call:** Such as a sudden call release.
- How it works (The "Stealing" Mechanism):** The FACCH does not have its own dedicated frames. Instead, it **"steals"** bits from the **Traffic Channel (TCH)**. When urgent signaling is needed, the system replaces a 20 ms segment of the user's speech data with FACCH data. The "stealing flags" within the normal burst are set to '1' to inform the receiver that this burst contains control information (FACCH) instead of voice data (TCH).
- Key Characteristic:** "Fast" because it can interrupt the voice traffic, providing a much higher data rate for signaling than the SACCH. This speed is essential for time-critical operations like handovers.

Summary Table

Feature	SDCCH (Stand-alone DCCH)	SACCH (Slow Associated DCCH)	FACCH (Fast Associated DCCH)
Primary Purpose	Call setup, location update, SMS.	Routine reporting & control during a call.	Urgent signaling, mainly handovers.
Association	Stand-alone (not tied to a TCH).	Associated with a TCH or SDCCH.	Associated with a TCH (by stealing it).
Speed	Slow.	Very Slow.	Fast (uses TCH capacity).
When Used	Before a call or without a call.	During a call (continuous).	During a call (only when urgent).
Key Functions	Authentication, call setup, SMS.	Measurement reports, power control, timing advance.	Handover commands, abrupt call release.
Physical Resource	Dedicated time slot in a 51-multiframe.	Fixed frame (e.g., FN 12) in a 26-multiframe.	"Steals" frames from the TCH.

The Big Picture: How They Work Together in a Call

Imagine a mobile user moving in a car while on a call:

- Call Initiation:** The mobile uses the **SDCCH** to communicate with the network for authentication and call setup.

2. **Call Active:** The network assigns a **TCH** for voice. Simultaneously, the **SACCH** becomes active.
3. **During the Call:** The mobile continuously sends measurement reports via the **SACCH** to the BSC.
4. **Handover Decision:** The BSC, based on the SACCH reports, decides a handover is necessary.
5. **Handover Execution:** The BSC sends the handover command to the mobile via the **FACCH**, which steals a segment of the voice channel to deliver the command quickly.
6. **Call Continues:** After the handover, the **SACCH** on the new cell resumes its routine reporting.

In conclusion, these three dedicated control channels form a sophisticated signaling system that manages mobility, maintains call quality, and ensures the seamless operation of the GSM network.

16. Describe the role of HLR and VLR in establishing a connection in GSM network.

The Home Location Register (HLR) and Visitor Location Register (VLR) are two critical databases in the GSM core network that work in tandem to track a subscriber's location and enable call routing and connectivity. Their roles are complementary and essential for establishing any connection.

Think of them as a dynamic phone book and a visitor's log.

1. Home Location Register (HLR) - The Central Master Database

The HLR is the **central, permanent database** in the subscriber's **home network**. There is typically one HLR per network operator.

- **What it stores (Permanent Data):**
 - **Subscriber Identity (IMSI):** The unique international identifier tied to the SIM card.
 - **Mobile Station ISDN Number (MSISDN):** The phone number (e.g., +91-9876543210).
 - **Subscriber Profile:** Service subscriptions (call waiting, call forwarding, roaming permissions, etc.).
 - **Authentication Key (Ki):** The secret key used to authenticate the subscriber.
 - **Current Location (Pointer):** This is the most crucial dynamic piece of information. The HLR does *not* store the exact cell location but **stores the address of the current VLR** where the subscriber is registered. If the subscriber is in their home network, this points to a home VLR. If roaming, it points to a VLR in the visited network.
- **Its Primary Role:** To answer the question, "**Where in the world is this subscriber right now?**"

2. Visitor Location Register (VLR) - The Temporary Local Log

The VLR is a **temporary, local database** associated with one or more Mobile Switching Centers (MSCs) in a specific geographical area (e.g., a city or state). It acts as a "visitor's log" for subscribers currently in that area.

- **What it stores (Temporary Data):**
 - A copy of the subscriber's profile from the HLR (for the duration of their stay).
 - The subscriber's IMSI and MSISDN.
 - The **Temporary Mobile Subscriber Identity (TMSI)**, a temporary, locally significant number used for security to avoid transmitting the IMSI over the air.
 - The subscriber's exact, current **Location Area Identity (LAI)**, which identifies a group of cells.
- **Its Primary Role:** To manage all subscribers currently in its area and answer the question, "**How do I find this specific subscriber within my local area?**"

How HLR and VLR Work Together to Establish a Connection

The entire process, especially for a **Mobile Terminated Call** (when someone calls the mobile), perfectly illustrates their roles. The sequence below follows the diagram from your PDF.

Phase 1: Location Update (The Prerequisite)

Before a call can be received, the mobile must register its location.

1. The mobile phone is powered on in a new city (roaming). It listens for the Broadcast Control Channel (BCCH) and detects it is in a new Location Area.
2. It contacts the new MSC/VLR in the visited network via a **Location Update Request**.
3. The new VLR receives the request and contacts the subscriber's **HLR** in the home network.
4. The HLR authenticates the subscriber, sends a copy of the subscriber's profile to the new VLR, and updates its own record to point to this new VLR.
5. The HLR instructs the old VLR (if one exists) to cancel its record of the subscriber.
6. The new VLR now has a temporary record for the subscriber and assigns a TMSI. The mobile is now "registered" and ready to receive calls.

Phase 2: Call Establishment (Mobile Terminated Call)

This is where the connection is established.

1. **Call Dialing:** A landline user dials the mobile's number (MSISDN). The Public Switched Telephone Network (PSTN) routes the call to the **Gateway MSC (GMSC)** in the mobile's **home network**.
2. **HLR Query (The "Where?"):** The GMSC doesn't know where the mobile is. It queries the **HLR**, asking for routing information.
3. **VLR Contact (The "How?"):** The HLR, which has stored the address of the current VLR, sends a request to that VLR for a **Mobile Station Roaming Number (MSRN)**. The MSRN is a temporary, routable phone number that acts like a "dummy" location pointer.
4. **Providing the Route:** The VLR generates an MSRN and sends it back to the HLR, which then forwards it to the requesting GMSC.
5. **Call Routing:** The GMSC now uses this MSRN to route the call through the PSTN to the **specific MSC** in the visited network.

6. **Locating the Subscriber:** The MSC asks its associated **VLR** for the subscriber's current Location Area.
7. **Paging and Connection:** The MSC sends a **paging message** through all the BTSSs in that Location Area. The mobile hears its identity and responds. After security checks (authentication via HLR/ Ki), the call is set up.

In essence, the **HLR finds the right city (VLR)**, and the **VLR finds the right neighborhood (Location Area)** to page the mobile and complete the connection. Without this sophisticated two-tiered system, seamless national and international roaming in GSM would be impossible.

17. What are the various logical channels used in a GSM network?

In a GSM network, Logical Channels are fundamental concepts that define what type of information is being carried over the physical radio resource (a specific frequency and time slot). They are a way of multiplexing different kinds of data (voice, signaling, synchronization) onto the same physical structure.

Think of a physical channel (a time slot) as a **pipe**. The logical channels are the **different types of fluid** (water, oil, data) that we send through that same pipe.

The logical channels in GSM are broadly divided into two main categories: **Traffic Channels (TCH)** and **Control Channels (CCH)**.

1. Traffic Channels (TCH)

These channels carry the user's encoded voice or data.

- **TCH/F (Full-Rate Traffic Channel):** Uses the entire capacity of one time slot. Carries one voice call at 22.8 kbps (raw rate, 13 kbps speech).
- **TCH/H (Half-Rate Traffic Channel):** Uses half the capacity, allowing two voice calls to share the same time slot. Carries voice at 11.4 kbps (raw rate).

2. Control Channels (CCH)

These channels carry signaling and control information necessary to manage the network.

They are further subdivided into three groups:

A. Broadcast Channels (BCCH) - Downlink Only

These are "one-to-many" channels transmitted by the Base Station to all mobiles in a cell. They provide the essential parameters needed to identify and access the network.

- **FCCH (Frequency Correction Channel):**
 - **Purpose:** Provides the mobile with a frequency reference for synchronization. It's essentially an unmodulated carrier.
 - **Function:** Allows the mobile to precisely tune to the base station's frequency.
- **SCH (Synchronisation Channel):**
 - **Purpose:** Provides the mobile with the timing synchronization information.
 - **Function:** Carries the Base Station Identity Code (BSIC) and the TDMA frame number, allowing the mobile to synchronize its timing with the network.
- **BCCH (Broadcast Control Channel):**
 - **Purpose:** Broadcasts general information about the cell.

- **Function:** Carries cell identity, frequency allocation, and the network identity. A mobile listens to the BCCH of neighboring cells to make measurements for handover.

B. Common Control Channels (CCCH) - Mainly for Access

These channels are used for initiating calls and for paging. They are shared by all mobiles in the cell.

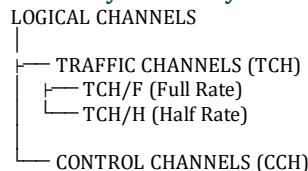
- **PCH (Paging Channel) - Downlink:**
 - **Purpose:** Alerts the mobile station of an incoming call or SMS.
 - **Function:** The network broadcasts the TMSI or IMSI of the mobile that has an incoming call.
- **RACH (Random Access Channel) - Uplink:**
 - **Purpose:** Used by the mobile to request a dedicated channel from the network.
 - **Function:** This is the mobile's "hand raise." It sends a short message to ask for an SDCCH when it wants to make a call, send an SMS, or respond to a page.
- **AGCH (Access Grant Channel) - Downlink:**
 - **Purpose:** The network's response to the mobile's RACH request.
 - **Function:** It assigns a dedicated channel (an SDCCH) to the mobile for further signaling.

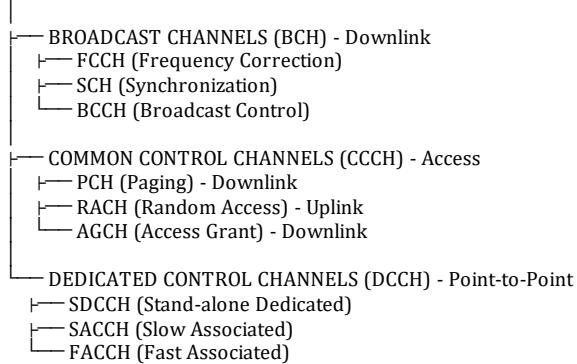
C. Dedicated Control Channels (DCCH) - For a Specific Mobile

These are two-way, point-to-point channels assigned to a single mobile for detailed signaling.

- **SDCCH (Stand-alone Dedicated Control Channel):**
 - **Purpose:** The "signaling workhorse" used *before* a traffic channel is assigned.
 - **Function:** Used for location area updates, subscriber authentication, call setup, and SMS transfer. It operates independently of a TCH.
- **SACCH (Slow Associated Control Channel):**
 - **Purpose:** Carries routine control information *during* an ongoing call or SDCCH connection.
 - **Function:** Used for continuous transmission of **measurement reports** (for handover decisions), power control, and timing advance commands.
- **FACCH (Fast Associated Control Channel):**
 - **Purpose:** Carries urgent signaling that cannot wait for the slow SACCH.
 - **Function:** Used for **handover execution** and sudden call release. It operates by "stealing" bits from the Traffic Channel (TCH), which is why it's "fast."

Summary Hierarchy





Practical Flow Example

1. A mobile is turned on. It listens to the **FCCH** and **SCH** to synchronize.
2. It then reads the **BCCH** to learn about the network and cell.
3. You make a call. The mobile sends a request on the **RACH**.
4. The network responds on the **AGCH**, assigning an **SDCCH**.
5. On the **SDCCH**, authentication and call setup happen.
6. The network assigns a **TCH** for voice. The **SACCH** is now active, sending measurement reports.
7. As you move, the network decides a handover is needed and sends the command via the **FACCH**.
8. The call ends, and all dedicated channels are released.

This structured use of logical channels is what allows GSM to efficiently manage radio resources, mobility, and call handling for millions of users.

***. Discuss various techniques to get rid of problems of Mobile IP regarding security and quality of service (Mobile IP is relevant as GSM networks interface with mobile IP for roaming and data services).**

Ans generated in Deepseek but maybe this question is related to 3/4g that is the reason the content not written here.

*. Discuss UMTS cellular architecture briefly.

NOT STUDIED YET(8,9)

01. Compare the handling of voice and data traffic in LTE vs. previous cellular generations.

3G vs 4G Architecture: Voice & Data Paths

3G Architecture (*Dual-Path System*)

In your diagram, 3G shows **two separate data paths**:

1. Circuit-Switched Path (Voice)

Mobile Device → Node B → RNC → MSC → PSTN (Phone Network)

- **Purpose:** Dedicated exclusively for voice calls and SMS
- **Technology:** Uses traditional circuit-switching (like a dedicated wire)
- **Characteristics:**
 - Guaranteed bandwidth (8-12 kbps per voice call)
 - Constant connection maintained throughout call
 - Inefficient for bursty data traffic

2. Packet-Switched Path (Data)

Mobile Device → Node B → RNC → SGSN → GGSN → Internet

- **Purpose:** For all IP data (internet browsing, email, apps)
- **Technology:** Uses packet-switching (like the internet)
- **Characteristics:**
 - Efficient for bursty data
 - Shared resources among multiple users
 - "Always-on" capability

Key Problem: This dual-architecture creates complexity and inefficiency. A phone on a voice call cannot use the high-speed data path simultaneously without special technology.

4G Architecture (*Single-Path "All-IP" System*)

In your diagram, 4G shows **only one unified path**:

Unified IP Path (Voice + Data)

Mobile Device (UE) → eNodeB → S-GW/P-GW → Internet
↓
MME/HSS (Control Only)

- **Purpose: Everything** - voice, data, video, SMS
- **Technology:** Pure packet-switching using IP protocols
- **Key Elements:**
 - **eNodeB:** Combined base station and controller (replaces Node B + RNC)
 - **S-GW (Serving Gateway):** Routes data packets (evolved from SGSN)
 - **P-GW (Packet Gateway):** Connects to internet (evolved from GGSN)
 - **MME (Mobility Management):** Handles signaling only (not the actual voice/data)

The Revolutionary Change: Voice over LTE (VoLTE)

How 4G Handles Voice Calls:

- Voice becomes just another IP data stream
- **VoLTE (Voice over LTE)** packages voice into IP packets

- No separate circuit-switched network needed
- Voice packets travel through the same path as web browsing, email, etc.

Key Benefits of 4G's "All-IP" Architecture

1. **Simpler Network:** Fewer network elements, lower cost
2. **Better Performance:** Lower latency for both voice and data
3. **Simultaneous Voice & Data:** No interruption of data during calls
4. **Higher Quality Voice:** HD Voice with better audio quality
5. **Faster Call Setup:** Calls connect in 1-2 seconds vs 5-10 seconds in 3G
6. **Efficiency:** Better use of radio spectrum

Summary

The diagram on Page 10 illustrates the **fundamental architectural revolution** from 3G to 4G:

- **3G = Dual Personality** (Circuit for voice + Packet for data)
- **4G = Single Personality** (Everything as IP packets)

This is why 4G LTE is described as an "**All-IP**" network - it treats everything, including your voice calls, as internet data, making the entire system more efficient, faster, and capable of supporting the massive data demands of modern smartphones.

02. Explain the differences between WCDMA (UMTS) and LTE in terms of architecture, data rates, and channel bandwidths

Executive Summary

WCDMA (Wideband Code Division Multiple Access) is the air interface for the **3G UMTS** network. LTE (Long-Term Evolution) is the technology for the **4G** network. The transition from UMTS to LTE was a major evolutionary step, moving from a circuit-switched architecture designed primarily for voice to an all-IP, packet-switched architecture designed for high-speed data.

1. Architecture

This is the most fundamental difference between the two technologies.

WCDMA (UMTS) Architecture

The UMTS architecture is an evolution of the 2G GSM/GPRS core network. It retains a clear separation between the **Circuit-Switched (CS) domain** for voice and the **Packet-Switched (PS) domain** for data.

- **Core Network:** Divided into two parts:
 - **Circuit-Switched Core:** Built around **MSC (Mobile Switching Center)** servers. This part handles traditional voice calls and SMS by establishing a dedicated circuit for the duration of the call.
 - **Packet-Switched Core:** Built around **SGSN (Serving GPRS Support Node)** and **GGSN (Gateway GPRS Support Node)**. This part handles internet data by routing data packets.
- **Radio Access Network (RAN):** Known as **UTRAN (UMTS Terrestrial Radio Access Network)**.
 - The key nodes are **Node B** (the base station) and the **RNC (Radio Network Controller)**.

- The RNC is a critical "intelligent" controller. It manages multiple Node Bs, handling radio resource management, handovers, and power control. This adds complexity and latency.

Key Takeaway: UMTS has a **hybrid architecture** with separate paths for voice and data, and a centralized, hierarchical RAN.

LTE Architecture

LTE was designed from the ground up as a **simplified, flat, all-IP architecture**. The circuit-switched core is completely eliminated.

- Core Network:** Called the **EPC (Evolved Packet Core)**. It is a single, unified packet-switched core for all services—voice, data, and video.
 - Key nodes include the **MME (Mobility Management Entity)** for control signaling, the **S-GW (Serving Gateway)** for the data path, and the **P-GW (Packet Data Network Gateway)** which connects to the internet.
- Radio Access Network:** Called **E-UTRAN (Evolved UTRAN)**. It is dramatically simplified.
 - It consists of only one node type: the **eNodeB (evolved Node B)**.
 - The eNodeB combines the functions of the old Node B and the RNC. This distributes intelligence to the cell site, reducing latency and making the network more efficient and scalable.

Key Takeaway: LTE has a **flat, all-IP architecture** where even voice is treated as data (VoLTE), and the RAN is simplified by eliminating the RNC.

Architecture Comparison Table

Feature	WCDMA (UMTS)	LTE
Network Type	Hybrid Circuit & Packet Switched	All-IP, Packet-Switched Only
Core Network	Separate CS (MSC) & PS (SGSN/GGSN) domains	Unified EPC (MME, S-GW, P-GW)
Radio Access Network	UTRAN (Node B + RNC)	E-UTRAN (eNodeB only)
Voice Handling	Circuit-Switched Fallback (CSFB)	Voice over LTE (VoLTE) - treated as data

2. Data Rates

The move to LTE brought a massive leap in potential and practical data rates.

WCDMA (UMTS) Data Rates

- Theoretical Peak:** The original UMTS release (Rel 99) promised a peak downlink speed of **~2 Mbps**. With later enhancements like HSPA (High-Speed Packet Access) and HSPA+, these rates increased significantly.

- HSPA+ (Evolution of 3G):** Theoretical peak downlink speeds reached **~42 Mbps** (with 2x2 MIMO and 64-QAM) and even up to **~84 Mbps** with dual-carrier.

- Practical/Real-World Speeds:** Users typically experienced speeds between **1-10 Mbps** on a good HSPA+ network. Speeds were highly susceptible to network load and distance from the cell tower.

LTE Data Rates

- Theoretical Peak:** The initial LTE specification (3GPP Release 8) defined a peak downlink rate of **~300 Mbps** (with 4x4 MIMO in a 20 MHz channel).
- Practical/Real-World Speeds:** Early LTE deployments commonly delivered speeds between **10-50 Mbps**, a significant and noticeable improvement over 3G. With advancements like carrier aggregation (combining multiple channels), real-world speeds now routinely exceed **100 Mbps**.

Key Takeaway: LTE provides a **5x to 10x improvement** in practical user data rates compared to basic WCDMA/UMTS, and a clear advantage over even advanced 3G (HSPA+).

3. Channel Bandwidths

Channel bandwidth refers to the amount of radio spectrum allocated to a single carrier/cell.

WCDMA (UMTS) Channel Bandwidth

- Fixed and Wide:** A single WCDMA carrier uses a **fixed 5 MHz bandwidth**.
- How it Works:** It uses a **Code Division Multiple Access (CDMA)** scheme, where multiple users share the same 5 MHz channel simultaneously. They are separated by unique codes, not by frequency or time slots. This is why it's called *Wideband* CDMA.
- Scaling:** To increase capacity, network operators had to deploy additional 5 MHz carriers.

LTE Channel Bandwidth

- Flexible and Scalable:** LTE supports **multiple channel bandwidths**, which is a key factor in its flexibility and high data rates.
- Standard Bandwidths:** The standard supports 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, and 20 MHz.
- How it Works:** LTE uses **OFDMA (Orthogonal Frequency Division Multiple Access)** in the downlink. The channel is divided into many small, narrow sub-carriers. Users are allocated a group of these sub-carriers for a brief period.
- Scaling with Carrier Aggregation:** The most powerful feature is **Carrier Aggregation (CA)**, which allows an LTE-Advanced device to connect to multiple LTE carriers simultaneously (e.g., a 20 MHz + 20 MHz channel), effectively creating a **40 MHz** channel and dramatically boosting data rates.

Key Takeaway: While UMTS uses a fixed 5 MHz channel, LTE uses flexible bandwidths up to 20 MHz, and can combine them for even wider effective channels, directly enabling its higher speeds.

Data Rate & Bandwidth Comparison Table

Feature	WCDMA (UMTS)	LTE

Peak Downlink (Theoretical)	~2 Mbps (Rel 99), up to ~84 Mbps (HSPA+)	~300 Mbps (Rel 8), >1 Gbps with LTE-A
Practical User Speeds	1 - 10 Mbps (HSPA+ up to ~30 Mbps)	10 - 50+ Mbps (100+ Mbps with CA)
Channel Bandwidth	Fixed 5 MHz	Flexible: 1.4, 3, 5, 10, 15, 20 MHz
Bandwidth Scaling	Add more 5 MHz carriers	Carrier Aggregation (combining channels)
Air Interface / Access Method	CDMA	OFDMA (Downlink) / SC-FDMA (Uplink)

03. What is OFDM and why is it used in LTE systems?

What is OFDM?

OFDM stands for **Orthogonal Frequency Division Multiplexing**.

At its core, it's a method of transmitting large amounts of digital data over a radio wave by using multiple, closely spaced, orthogonal sub-carriers.

Let's break down the name:

- **Frequency Division Multiplexing (FDM):** This is the old concept of dividing the available radio spectrum into multiple frequency channels and sending different data streams over each channel. Think of it like a multi-lane highway, where each car (data stream) has its own lane (frequency).
- **Orthogonal (O):** This is the revolutionary part. "Orthogonal" means the peaks and troughs of the sub-carriers are perfectly aligned in a mathematical relationship. This allows the sub-carriers to be packed extremely close together *without interfering with each other*, even though they overlap.

A Simple Analogy: A Symphony Orchestra Imagine an orchestra (the transmitter) and a listener (the receiver).

- In a traditional system (like old FM radio), it would be like having one musician playing a single, complex melody very quickly. If an echo (signal reflection) occurs, the notes blur together.
- In an OFDM system, it's like the entire orchestra playing together. Each musician (sub-carrier) plays a single, simple, slow note. All the simple notes combine to create a complex piece of music. The listener's brain (the OFDM receiver) is perfectly tuned to pick out each individual instrument, even though the sound waves from all instruments are overlapping in the air.

Why is OFDM used in LTE?

OFDM was chosen as the downlink (from tower to device) technology for LTE specifically because it solves several critical problems that plagued older 3G technologies like WCDMA. Here are the key reasons:

1. Superior Resistance to Multipath Fading and Inter-Symbol Interference (ISI)

- **The Problem:** In a mobile environment, radio signals reflect off buildings, hills, and other objects. The receiver gets multiple copies of the same signal, each arriving at a slightly different time. This is called **multipath propagation**. In high-rate systems, these delayed copies cause the symbols (chunks of data) to overlap and interfere with each other, corrupting the data. This is Inter-Symbol Interference (ISI).
 - By transmitting many low-rate, parallel streams instead of one high-rate stream, the **symbol duration** (the length of time for each data chunk) becomes much longer.
 - A "Guard Interval" (or Cyclic Prefix) is added to each symbol. This is a copy of the end of the symbol placed at the beginning. As long as the multipath delay is shorter than this guard interval, the reflections from one symbol won't interfere with the next symbol. The receiver can simply ignore the corrupted part (the guard interval) and process the clean part.
- **The OFDM Solution:** OFDM turns this weakness into a strength.

2. High Spectral Efficiency

- **The Problem:** Spectrum is expensive and limited. Operators need to send as much data as possible in the bandwidth they own.
- **The OFDM Solution:** Because the sub-carriers are orthogonal and can overlap without interference, OFDM uses the available spectrum much more efficiently than older FDM techniques, where guard bands were needed to separate channels. This dense packing allows for very high data rates within a given channel bandwidth (e.g., 5 MHz, 10 MHz, 20 MHz).

3. Flexibility and Scalability

- **The Problem:** Operators around the world have different chunks of spectrum allocated to them—some large, some small, some contiguous, some fragmented.
- **The OFDM Solution:** LTE can easily adapt to different bandwidths by simply changing the number of sub-carriers used. A 1.4 MHz channel uses 72 sub-carriers, while a 20 MHz channel uses 1200 sub-carriers. This inherent flexibility was a major design goal for LTE.

4. Compatibility with Advanced Antenna Techniques (MIMO)

- **The Problem:** To increase data rates and capacity further, systems need to use Multiple-Input Multiple-Output (MIMO) technology, which uses multiple antennas at both the transmitter and receiver.
- **The OFDM Solution:** The simple, narrowband nature of OFDM sub-carriers makes it much easier to implement and process MIMO signals. MIMO algorithms can be applied independently to each individual sub-carrier, which is far simpler than applying them to a wide, fading channel as a whole. **OFDM and MIMO are a perfect match**, and their combination is a cornerstone of 4G and 5G performance.

5. Efficient Equalization

- **The Problem:** In a wide, single-carrier system (like WCDMA), the receiver needs a very complex "equalizer" to undo the distortion caused by the radio channel across the entire bandwidth.
- **The OFDM Solution:** Because each OFDM sub-carrier is very narrow, the radio channel appears "flat" across each one (i.e., it affects all frequencies within that tiny

sub-carrier equally). This makes the equalization process in the receiver incredibly simple—just a single-tap filter per sub-carrier.

A Note on Uplink: SC-FDMA

While OFDM is used for the **downlink** in LTE, a slight variation called **SC-FDMA (Single Carrier Frequency Division Multiple Access)** is used for the **uplink** (from device to tower).

Why the difference? OFDM has one major drawback: a high **Peak-to-Average Power Ratio (PAPR)**. This means the combined signal has very high power peaks, which requires a very linear (and inefficient) power amplifier in the transmitter. This drains smartphone batteries quickly.

SC-FDMA has a much lower PAPR, making it more power-efficient. This preserves battery life in mobile devices while retaining many of the benefits of OFDM, like multipath resistance and frequency-domain equalization.

In conclusion, OFDM is the fundamental physical layer technology that allows LTE to deliver high-speed, reliable, and scalable mobile broadband services by elegantly solving the key challenges of the wireless channel.

04. Describe the layered architecture and core network changes in LTE compared to previous generation cellular systems.

The transition to LTE (4G) involved a complete architectural overhaul, moving from a complex, hierarchical structure to a simplified, flat, all-IP architecture. This can be understood in two parts: the overall system architecture and the layered protocol architecture.

Part 1: Overall System Architecture Change – From "Hierarchical" to "Flat" Previous Generations (2G GSM & 3G UMTS)

The pre-4G architecture was a hybrid, built on the legacy of circuit-switched voice. The Core Network was split, and the Radio Access Network (RAN) had a centralized, intelligent controller.

A. Core Network (CN): A Split Personality

- **Circuit-Switched (CS) Domain:** Handled traditional voice calls and SMS. The key node was the **Mobile Switching Center (MSC)**. It established a dedicated, end-to-end circuit for the duration of a call.
- **Packet-Switched (PS) Domain:** Handled internet data. The key nodes were the **Serving GPRS Support Node (SGSN)** and the **Gateway GPRS Support Node (GGSN)**. This domain routed data packets.

B. Radio Access Network (RAN): The Intelligent Controller

- In 3G UMTS, this was called **UTRAN**.
- It consisted of:
 - **Node B:** The base station that handled the radio transmission/reception.
 - **RNC (Radio Network Controller):** A centralized, "intelligent" controller that managed multiple Node Bs. It handled critical functions like **Radio Resource Management (RRM)**, **handover decisions**, **power control**, and **security**

(**ciphering**). All data traffic had to pass through the RNC, creating a bottleneck and adding latency.

Key Takeaway: The pre-LTE architecture was a **hybrid, centralized, and complex** system with a clear separation between voice and data paths.

LTE (4G) Architecture: The "Flat IP" Revolution

LTE was designed from the ground up for high-speed packet data. The circuit-switched core was eliminated entirely in favor of an **All-IP Evolved Packet Core (EPC)**.

A. Core Network: The Evolved Packet Core (EPC) - A Unified IP Core The EPC is a single, unified architecture for all services—voice, data, and video. Everything is treated as packets.

- **MME (Mobility Management Entity):** The brain for *control-plane* functions. It handles signaling, subscriber authentication, tracking area updates, and bearer activation. **It does not handle user data traffic**.
- **S-GW (Serving Gateway):** The *user-plane* anchor. It routes and forwards user data packets. It acts as the local mobility anchor during handovers between eNodeBs.
- **P-GW (Packet Data Network Gateway):** The router to the outside world. It connects the LTE network to the internet. It also performs critical functions like **IP address allocation** for the UE and **policy enforcement** (e.g., charging, quality of service).

B. Radio Access Network: E-UTRAN - The Distributed Simplicity The RAN in LTE is called **E-UTRAN**. It is dramatically simplified by distributing the intelligence.

- It consists of only one node: the **eNodeB (evolved Node B)**.
- The eNodeB **incorporates the functions of the old Node B and the RNC**. Each eNodeB handles its own radio resource management, handover decisions, and security. This eliminates the RNC bottleneck.

Key Takeaway: The LTE architecture is a **flat, distributed, and all-IP** system. This reduces latency, increases scalability, and simplifies the network.

Part 2: Layered Protocol Architecture Change – User-Plane Simplification

The protocol stack on the user-plane (the path your data takes) was also significantly streamlined in LTE to reduce latency.

Pre-LTE (UMTS) User-Plane Stack (Iu-ps interface)

The path between the RNC and the SGSN was complex:

- **Multiple Layers:** Data > PDCP > RLC > MAC > PHY > (over the Iu-ps interface) > GTP-U > UDP > IP > ... and then a similar stack on the other side.
- **Tunneling:** Used **GTP-U (GPRS Tunneling Protocol)** to tunnel user data through the core network. This added overhead and processing delay.
- **Centralized Processing:** The RLC and PDCP layers, which handle segmentation/retransmission and header compression/security, were located in the RNC, far from the radio.

LTE User-Plane Stack (S1-U interface)

LTE features a much flatter and optimized protocol stack between the UE and the P-GW.

- **Simplified Path:** Data > PDCP > RLC > MAC > PHY
- **Distributed Intelligence:** The **PDCP and RLC layers are now located in the eNodeB**, not in a centralized controller. This allows for faster retransmissions and lower latency.

- **End-to-End Tunnel:** GTP-U is still used (on the S1-U interface between eNodeB and S-GW, and S5/S8 between S-GW and P-GW) to maintain the connection and enable mobility, but the overall path is shorter and more direct.

In conclusion, the layered and core network architecture of LTE was a clean-break design focused on performance. By flattening the RAN, unifying the core around IP, and distributing intelligence, LTE achieved the low latency and high data rates that truly enabled the mobile broadband revolution.

04. Explain how the LTE air interface differs from the UMTS air interface.

Here's a detailed explanation of how they differ.

Executive Summary

The core difference is the multiple access technique:

- UMTS uses **Wideband Code Division Multiple Access (W-CDMA)**.
- LTE uses **Orthogonal Frequency Division Multiple Access (OFDMA)** for the downlink and **Single-Carrier FDMA (SC-FDMA)** for the uplink.

This shift from a **code-based** system to a **frequency-and-time-based** system is the key to LTE's higher speed, lower latency, and greater efficiency.

Detailed Comparison

1. Fundamental Multiple Access Scheme

Feature	UMTS (W-CDMA)	LTE (OFDMA / SC-FDMA)
Core Principle	Code Division: All users transmit simultaneously over the entire 5 MHz bandwidth . Users are separated by unique channelization codes .	Frequency & Time Division: The bandwidth is divided into many narrow sub-carriers . Users are allocated specific time-frequency resources (groups of sub-carriers for short time slots).
Analogy	A crowded room where everyone speaks at once but in different languages (codes). A listener can focus on one language if they know it.	A multi-lane highway divided into many small, slow-moving conveyor belts (sub-carriers). Users are assigned specific belts for specific periods.

UMTS (W-CDMA):

- The "Wideband" refers to its use of a single, wide 5 MHz channel.
- It's a **spread-spectrum** technology. Each user's data is multiplied by a unique, high-rate code ("spreading") that spreads the signal across the full bandwidth.
- The receiver uses the same code to "de-spread" the signal and recover the original data. Other users' signals, with different codes, appear as low-level noise.
- **Key Challenge:** Requires very precise power control to ensure all signals arrive at the base station with similar power. Otherwise, a nearby user can "drown out" distant users (the "near-far problem").

LTE (OFDMA):

- Based on **OFDM (Orthogonal Frequency Division Multiplexing)**. The available bandwidth (e.g., 20 MHz) is divided into many narrowband **sub-carriers** (e.g., 1200 sub-carriers for 20 MHz).
- These sub-carriers are **orthogonal**, meaning they can be packed very closely without interference, leading to high spectral efficiency.
- **OFDMA** adds the multiple-access component by allowing different users to be assigned different groups of these sub-carriers for the duration of a **Transmission Time Interval (TTI)**, which is very short (1 ms).

2. Channel Bandwidth & Flexibility

Feature	UMTS (W-CDMA)	LTE (OFDMA / SC-FDMA)
Bandwidth	Fixed at 5 MHz per carrier.	Flexible: 1.4, 3, 5, 10, 15, 20 MHz.
Scaling	Capacity increased by adding more 5 MHz carriers.	Capacity increased by using wider bandwidths and Carrier Aggregation (combining multiple carriers).

- **UMTS:** Inflexible bandwidth was a major limitation. Deploying LTE in small slivers of spectrum (e.g., 3 MHz) was impossible with W-CDMA.
- **LTE:** The ability to scale the bandwidth was a core design goal. Carrier Aggregation, a key feature of LTE-Advanced, allows combining non-adjacent spectrum blocks, something very difficult with W-CDMA.

3. Handling of Multipath Interference and Delay Spread

- **UMTS (W-CDMA):** Uses a **RAKE Receiver**. The RAKE receiver has multiple "fingers" that try to combine the different delayed signal paths (multipath) to improve the signal. However, in challenging environments with high delay spread, it becomes complex and less effective, leading to Inter-Symbol Interference (ISI).
- **LTE (OFDMA):** Elegantly solves this by design.
 - **Long Symbol Duration:** Transmitting data slowly on many parallel sub-carriers makes each symbol longer. This makes it more resilient to delay spread.
 - **Cyclic Prefix (CP):** A guard interval is inserted before each symbol. This is a copy of the end of the symbol placed at the beginning. As long as the multipath delay is shorter than the CP, interference is eliminated. The receiver simply discards the CP.

4. Peak-to-Average Power Ratio (PAPR)

- **UMTS (W-CDMA):** Has a relatively moderate PAPR. This was efficient for the power amplifier in the mobile handset, preserving battery life.
- **LTE Downlink (OFDMA):** Has a **very high PAPR**. The combination of many independent sub-carriers can create large power peaks, requiring a very linear and

inefficient power amplifier at the base station (eNodeB). This is acceptable for base stations with ample power.

- **LTE Uplink (SC-FDMA):** To solve the high PAPR problem for handsets, LTE uses SC-FDMA for the uplink. It is sometimes called "DFT-spread OFDM." It has a similar structure to OFDMA but with an additional processing step that results in a **much lower PAPR**, similar to a single-carrier system. This is crucial for preserving smartphone battery life.

5. Channel Structure

- **UMTS:** Uses a complex set of dedicated physical channels (DPCH) and common channels.
- **LTE:** Simplified into a structure of **Shared Channels**.
 - **PDSCH (Physical Downlink Shared Channel):** Carries all user data in the downlink.
 - **PUSCH (Physical Uplink Shared Channel):** Carries all user data in the uplink.
 - By dynamically scheduling resources on these shared channels every 1 ms TTI, LTE achieves extremely efficient radio resource management and very low latency.

The shift from the W-CDMA air interface of UMTS to the OFDMA/SC-FDMA air interface of LTE was a revolutionary change, not an evolutionary one. It moved away from a philosophy of separating users by codes across a wide band to a philosophy of separating users in precise time and frequency slots across many narrow sub-carriers. This fundamental change is what enabled LTE to deliver on the promise of true high-speed, low-latency mobile broadband.

05. Describe the function and organization of eNodeB in the LTE radio access network.

Function of the eNodeB: The "All-in-One" Base Station

The eNodeB is not just a simple transmitter/receiver; it's an intelligent controller that consolidates the functions of the traditional 3G Node B and the Radio Network Controller (RNC) into a single entity. Its functions can be grouped into several key areas:

1. Radio Resource Management (RRM)

This is the core set of functions for managing the radio interface efficiently.

- **Dynamic Scheduling:** The eNodeB decides, every **1 millisecond (Transmission Time Interval - TTI)**, which user gets which time-frequency resources (Resource Blocks) on both the uplink and downlink. This is the most critical RRM function, ensuring optimal use of the shared channel.
- **Link Adaptation:** It continuously monitors channel conditions (e.g., based on CQI - Channel Quality Indicator from the UE) and dynamically selects the best **Modulation and Coding Scheme (MCS)**. For example, it switches between QPSK (robust but slow) and 64-QAM (fast but prone to errors) to maximize throughput.

- **Power Control:** Manages the transmission power of the User Equipment (UE) to ensure signals are received with just enough power, minimizing interference and saving battery life. (Note: Power control in LTE is less stringent than in WCDMA but is still crucial for the uplink).

2. Mobility Management

The eNodeB handles all handovers (handoffs) for connected mobiles.

- **Intra-LTE Handover:** Manages the process of a UE moving from one eNodeB to another (X2-handover) or within the same eNodeB (intra-eNodeB handover). It makes the handover decision, coordinates with the target eNodeB, and instructs the UE.
- **Connection Mobility Control:** Includes functions like cell re-selection for idle-mode users and load balancing between cells.

3. Security and Data Handling

- **Ciphering and Integrity Protection:** The eNodeB performs encryption and decryption of user data and control messages over the air interface. This security function was previously handled by the RNC in 3G.
- **Header Compression:** Using **ROHC (Robust Header Compression)**, the eNodeB compresses the IP headers of data packets (e.g., TCP/IP) to reduce overhead and improve efficiency over the radio link.

4. Interface Management

The eNodeB is the connection point for all interfaces in the RAN.

- **S1 Interface:** Connects the eNodeB to the Evolved Packet Core (EPC).
 - **S1-MME:** For control-plane signaling to the MME.
 - **S1-U:** For user-plane data traffic to the Serving Gateway (S-GW).
- **X2 Interface:** The direct interface between neighboring eNodeBs. This is a key innovation for a flat architecture, enabling:
 - **Fast Handovers:** Data can be forwarded directly between eNodeBs during a handover without going through the core network, reducing latency and packet loss.
 - **Interference Coordination:** eNodeBs can exchange messages to coordinate radio resources and minimize inter-cell interference.

5. Quality of Service (QoS) Enforcement

The eNodeB is responsible for enforcing the QoS policies set by the core network. It maps data flows to specific **Radio Bearers** and ensures that each bearer receives the appropriate priority, scheduling, and error rates (e.g., guaranteeing low latency for a VoIP call while allowing a file download to use best-effort resources).

Organization of the eNodeB: Logical and Physical Structure

The eNodeB can be understood in terms of its logical protocol stack and its physical deployment.

A. Logical Protocol Stack

The eNodeB implements the following layers of the radio protocol stack:

1. **Physical Layer (PHY):** Handles the actual signal transmission and reception. This includes:
 - a. OFDMA modulation for the downlink.
 - b. SC-FDMA modulation for the uplink.

- c. Coding/decoding, modulation/demodulation.
 - d. MIMO antenna processing.
2. **Medium Access Control (MAC) Layer:** Responsible for the critical **scheduling function**. It multiplexes data from different logical channels into transport channels for the PHY. It also handles Hybrid ARQ (HARQ) for fast error correction.
 3. **Radio Link Control (RLC) Layer:** Manages segmentation and reassembly of data packets to fit the PHY layer's transport blocks. It also provides reliable data transfer through ARQ (Automatic Repeat Request) for error recovery.
 4. **Packet Data Convergence Protocol (PDCP) Layer:** This layer performs two vital functions:
 - a. **Ciphering/Integrity Protection:** Secures the data.
 - b. **Header Compression (ROHC):** Improves efficiency.

The control-plane part of the eNodeB also implements the **RRC (Radio Resource Control)** protocol, which is responsible for broadcasting system information, managing the RRC connection state (Idle/Connected), and establishing the radio bearers.

B. Physical Deployment (Architecture)

The physical implementation of eNodeBs has evolved to be more flexible:

- **Traditional Macro Cell:** A single eNodeB cabinet serving a large cell area, typically mounted on a tower or rooftop.
- **Distributed eNodeB (D-RAN):** The base station is split into:
 - **Baseband Unit (BBU):** Houses the digital processing for the protocol stack (PHY and above).
 - **Remote Radio Head (RRH):** Located at the antenna site, handles analog radio functions (amplification, filtering, conversion). The BBU and RRH are connected via fiber optic cables (e.g., using CPRI protocol). This reduces power loss and allows for more flexible antenna placement.
- **Cloud RAN (C-RAN):** A further evolution where the BBU functions are virtualized and centralized in a data center, serving a large number of RRHs. This allows for better resource pooling and coordination (e.g., for CoMP - Coordinated Multi-Point).

In conclusion, the eNodeB is the workhorse of the LTE network. By consolidating the functions of the NodeB and RNC, it creates a flatter, more responsive architecture that is fundamentally optimized for high-speed, low-latency packet data. This distributed intelligence is a key reason for LTE's performance superiority over its 3G predecessors.

06. Suppose you are calling from a land phone, a 4G mobile which is currently in roaming. Describe the procedure in detail for the connection establishment between your land phone and the 4G mobile.

Here is a detailed, step-by-step procedure for a call from a landline to a roaming 4G mobile.

High-Level Overview

The call path essentially bridges two different worlds:

1. **The Calling Side:** The legacy **Circuit-Switched (CS) PSTN** (Landline).
2. **The Called Side:** The **Packet-Switched (PS) 4G LTE Network** (Mobile), which is roaming in a visited network.

Since the LTE network has no native circuit-switching, a special node called the **IMS (IP Multimedia Subsystem)** is used to handle the voice call, which is delivered as VoIP (Voice

over LTE or VoLTE). The key is finding the roaming subscriber and anchoring the call through their home network.

Step-by-Step Detailed Procedure

Phase 1: Landline Initiates the Call

1. **Call Dialing & PSTN Routing:** You dial the mobile subscriber's number (e.g., +1-234-555-7890, which includes the country and network operator code). Your landline switch routes the call through the PSTN based on this number.
2. **Call Reaches the Gateway:** The call eventually arrives at a **Gateway MSC (GMSC)** in the PSTN. The GMSC's job is to query the database that knows where the subscriber is currently located.

Phase 2: Locating the Roaming Subscriber

3. **GMSC Queries the HLR/HSS:** The GMSC uses the dialed number to identify the subscriber's **home network**. It sends a signaling message (using SS7/SIGTRAN protocol) to the **Home Location Register (HLR)** or, in 4G terms, the **Home Subscriber Server (HSS)** in the subscriber's home network.
4. **HSS Provides Routing Information:** The HSS/HLR is the master database containing the subscriber's profile, services, and, crucially, their current location. It knows the address of the **MSC Server** that is currently serving the subscriber for voice services. In a 4G context, this is not a standard MSC but a special node called the **IMS Access Transfer Gateway (IMS-ALG/ATCF)** which acts as the anchor point for the subscriber's voice calls.
5. **Routing the Call to the IMS:** The HSS replies to the GMSC with this routing number (a temporary number pointing to the IMS anchor). The GMSC now routes the call towards this IMS node in the subscriber's **home network**.

Phase 3: The Home IMS Core Takes Control

6. **Call Anchoring in IMS:** The call is received by the **IMS Core** in the home network. The key IMS nodes involved are:
 - a. **P-CSCF (Proxy-Call Session Control Function):** The first point of contact in the IMS (in this case, for the inbound leg from the PSTN, a different P-CSCF is used).
 - b. **I-CSCF (Interrog-CSCF):** Queries the HSS to find which **S-CSCF** is assigned to the subscriber.
 - c. **S-CSCF (Serving-CSCF):** The brain of the IMS. It authenticates the user and manages the call session. It knows the subscriber is registered and roaming.
7. **S-CSCF Identifies Roaming:** The S-CSCF checks the subscriber's profile and identifies that the subscriber is currently roaming in a **Visited Network**.

Phase 4: Establishing the Path to the Roaming Mobile

8. **S-CSCF Forks the Call to the Visited Network:** The S-CSCF routes the call towards the P-CSCF in the **visited (roaming) network**. This establishes a signaling path from the Home IMS to the device itself.
9. **Paging the Mobile:** The call signaling request (a SIP INVITE message) arrives at the P-CSCF in the visited network. The P-CSCF interacts with the **PCRF (Policy and Charging Rules Function)** and the **Visited EPC (S-GW, P-GW)**.

10. **Mobile is Alerted:** The network now needs to find the specific mobile, which is likely in a power-saving **Idle state**. The **MME** in the visited network triggers a **paging procedure** across all the base stations (eNodeBs) in the mobile's current tracking area.
11. **Mobile Responds:** The roaming 4G mobile hears its page and responds, establishing a radio connection and moving to the **Connected state**. It also establishes a dedicated **EPS Bearer** with the required **Quality of Service (QoS)** for voice (guaranteed bit rate, low latency).

Phase 5: Completing the End-to-End Connection

12. **Call Signaling to the Mobile:** The SIP INVITE message is delivered to the **IMS client** on the mobile phone over the newly established signaling bearer.
13. **Mobile Rings:** The mobile phone's IMS client processes the INVITE, checks its capabilities, and if acceptable, it triggers the ringtone. It sends back SIP "180 Ringing" messages back through the P-CSCF and S-CSCF to the GMSC and ultimately to your landline phone, so you hear the ringback tone.
14. **Call Answered:** When the roaming user answers the call, the mobile phone sends a SIP "200 OK" message.
15. **Media Path Established:** The "200 OK" contains the IP address and port number of the mobile phone for the voice media (RTP packets). The S-CSCF facilitates the exchange of these media parameters between the mobile and the media gateway connected to the PSTN.
16. **End-to-End VoIP Call:** The media gateway converts the incoming PSTN voice circuit (64 kbps PCM) into RTP/IP packets. A direct media path is now established:
 - a. **Landline <-> PSTN <-> Media Gateway in Home Network <-> Internet/IP Network <-> Visited EPC (P-GW, S-GW) <-> Visited eNodeB <-> Roaming 4G Mobile**

The conversation can now take place. The voice from the landline is converted to IP packets and streamed to the mobile, and vice-versa.

Call Flow Summary (Simplified)

4G.PNG