Module 8 of 12

Shortest paths: BFS, 0-1 BFS, Dijkstra

Choose the algorithm from the weights — BFS for equal edges, 0-1 BFS with a deque, Dijkstra with a priority_queue and lazy deletion — and write it right under a clock.

Shortest-path problems are the most common graph problems in contests, and most of them are lost on the choice of algorithm or on one line of Dijkstra. This module makes the choice a table — equal weights take BFS, 0/1 weights take 0-1 BFS, non-negative weights take Dijkstra, negative ones take Bellman–Ford — and then drills the contest Dijkstra until it is automatic: a `priority_queue` of `{distance, vertex}` with `greater<>`, lazy deletion with `d > dist[u]`, `long long` sums.

The lesson animates Dijkstra's frontier on six vertices — the heap's pairs, each vertex settling, the stale pairs thrown away — and plays a matching game on which algorithm a graph needs.

The exercises are a grid maze with BFS and a 0/1-weighted graph with a deque. The timed checkpoint asks for Dijkstra's distances to every vertex at weights up to 10^9, and the number of shortest paths modulo 10^9 + 7.

CONTENTS

1	Shortest paths — BFS, 0-1 BFS and Dijkstra	16 min
✓	Checkpoint — Shortest paths	45 min

Exercises in this module compile on Clang 18 · C++20. Reference solutions and the answer key follow each lesson.

LESSON 1 · 16 MIN

Shortest paths — BFS, 0-1 BFS and Dijkstra

Pick the algorithm from the **edge weights**, never from habit:

Weights	Algorithm	Cost
all equal (grid moves)	BFS, `queue`	O(V + E)
only 0 or 1	0-1 BFS, `deque`	O(V + E)
non-negative	Dijkstra, `priority_queue`	O((V + E) log V)
some negative	Bellman–Ford	O(V · E)

BFS and 0-1 BFS

- BFS pops vertices in order of edge count; set `dist[v]` when `v` is **first pushed**, and a cell is never queued twice.
- 0-1 BFS: a 0-edge `push_front`s the neighbour, a 1-edge `push_back`s it. The deque holds at most two distance values, in order, so its front is always a closest vertex.

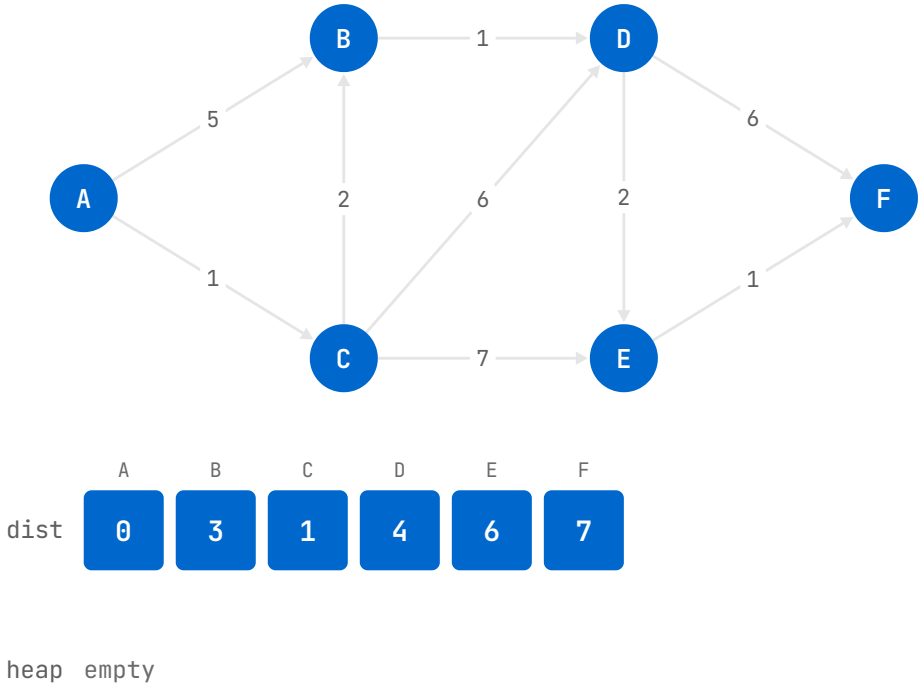
```
deque<int> dq{src};
dist[src] = 0;
while (!dq.empty()) {
    int u = dq.front(); dq.pop_front();
    for (auto [v, w] : adj[u])
        if (dist[u] + w < dist[v]) {
            dist[v] = dist[u] + w;
            w == 0 ? dq.push_front(v) : dq.push_back(v);
        }
}
```

Dijkstra, the contest way

```
vector<ll> dist(n + 1, LLONG_MAX);
priority_queue<pair<ll,int>, vector<pair<ll,int>>, greater<>> pq;
dist[src] = 0;
pq.push({0, src});
while (!pq.empty()) {
    auto [d, u] = pq.top(); pq.pop();
    if (d > dist[u]) continue;              // stale pair: lazy deletion
    for (auto [v, w] : adj[u])
        if (d + w < dist[v]) {
            dist[v] = d + w;
            pq.push({dist[v], v});
        }
}
```

- `greater<>` turns the max-heap into a min-heap; the distance goes **first** in the pair so it decides the order.
- The heap cannot lower a key, so an improvement pushes a new pair; the old one is skipped later by `d > dist[u]`.
- Path sums are `long long`: a path of 10^5 edges of 10^9 is 10^14.

Dijkstra's frontier: a heap of (distance, vertex) pairs



1. dist[A] = 0 and every other distance is infinity. The heap holds one pair, (0, A): distance first, so the smallest distance is always on top.
2. Pop (0, A): A is settled at 0. Relax its edges: B ∞ → 5, C ∞ → 1.
3. Pop (1, C): C is settled at 1, since no path through a later vertex can be shorter. Relax its edges: B 5 → 3, D ∞ → 7, E ∞ → 8.
4. Pop (3, B): B is settled at 3, since no path through a later vertex can be shorter. Relax its edges: D 7 → 4.
5. Pop (4, D): D is settled at 4, since no path through a later vertex can be shorter. Relax its edges: E 8 → 6, F ∞ → 10.
6. Pop (5, B): stale, because dist[B] is already 3. Lazy deletion just skips it — the check d > dist[v] is one line.
7. Pop (6, E): E is settled at 6, since no path through a later vertex can be shorter. Relax its edges: F 10 → 7.
8. Pop (7, D): stale again — dist[D] is 4. Skipped.
9. Pop (7, F): settled at 7. None of its edges improves anything.
10. Every pair left is stale: (8, E) against dist 6, (10, F) against dist 7. Each pop is skipped in O(log V).
11. The heap is empty. Final distances from A: A 0, B 3, C 1, D 4, E 6, F 7. Each vertex settled once; each edge relaxed at most once: O((V + E) log V).

Pitfalls

- Printing `LLONG_MAX` for an unreachable vertex — map it to `-1` (or what the statement asks).
- `pq.push({v, dist[v]})` — the vertex first sorts by vertex number, not distance.
- Forgetting the stale check: still correct, but a vertex's edges are scanned many times — TLE on dense graphs.

More practice

- Network Delay Time
- Shortest Path in Binary Matrix
- Path With Minimum Effort
- Minimum Obstacle Removal to Reach Corner
- Number of Ways to Arrive at Destination

EXERCISE 1

Escape the grid

A grid of `R` rows and `C` columns holds `.` (open), `#` (wall), exactly one `S` and exactly one `T`. Each move goes one cell up, down, left or right, onto any cell that is not a wall. Print the fewest moves from `S` to `T`, or `-1` if `T` cannot be reached.

Input: `R C`, then `R` lines of `C` characters (`1 ≤ R, C ≤ 1000`). **Output:** one integer.

```
4 5
S..#.
.#...
...#T
.#...
```

prints

```
6
```

INPUT 1

```
4 5
S..#.
.#...
...#T
.#...
```

OUTPUT 1

```
6
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int R, C;
    cin >> R >> C;
    vector<string> g(R);
    for (auto& row : g) cin >> row;
    int sr = 0, sc = 0, tr = 0, tc = 0;
    for (int r = 0; r < R; ++r)
        for (int c = 0; c < C; ++c) {
            if (g[r][c] == 'S') sr = r, sc = c;
            if (g[r][c] == 'T') tr = r, tc = c;
        }
    vector<vector<int>> dist(R, vector<int>(C, -1));
    queue<pair<int, int>> q;
    dist[sr][sc] = 0;
    q.push({sr, sc});
    const int dr[] = {1, -1, 0, 0}, dc[] = {0, 0, 1, -1};
    while (!q.empty()) {
        auto [r, c] = q.front();
        q.pop();
        for (int k = 0; k < 4; ++k) {
            int nr = r + dr[k], nc = c + dc[k];
            if (nr < 0 || nc < 0 || nr >= R || nc >= C || g[nr][nc] == '#' || dist[r
            dist[nr][nc] = dist[r][c] + 1;
            q.push({nr, nc});
        }
    }
    cout << dist[tr][tc] << '\n';
    return 0;
}
```

EXERCISE 2

Free corridors, paid doors

A directed graph has vertices `1` to `n` and `m` edges, each of weight `0` or `1`. Print the shortest distance from vertex `1` to every vertex, `-1` for one that cannot be reached.

Input: `n m`, then `m` lines `u v w` with `w` in `{0, 1}` (`1 ≤ n ≤ 2·10^5`, `0 ≤ m ≤ 2·10^5`). **Output:** `n` integers on one line, separated by spaces.

```
5 6
1 2 1
1 3 0
3 2 0
2 4 1
3 4 1
4 5 0
```

prints

```
0 0 0 1 1
```

INPUT 1

```
5 6
1 2 1
1 3 0
3 2 0
2 4 1
3 4 1
4 5 0
```

OUTPUT 1

```
0 0 0 1 1
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int, int>>> adj(n + 1);
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
    }
    const int INF = INT_MAX;
    vector<int> dist(n + 1, INF);
    deque<int> dq{1};
    dist[1] = 0;
    while (!dq.empty()) {
        int u = dq.front();
        dq.pop_front();
        for (auto [v, w] : adj[u])
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                if (w == 0) dq.push_front(v);
                else dq.push_back(v);
            }
    }
    for (int v = 1; v <= n; ++v) cout << (dist[v] == INF ? -1 : dist[v]) << (v == n
    return 0;
}
```

QUIZ

1. Every edge weighs 7. Which algorithm is the cheapest correct choice for shortest paths?

A. BFS, then multiply edge counts by 7

B. Dijkstra

C. Bellman–Ford

D. 0-1 BFS

2. In `priority_queue<pair<ll,int>, vector<pair<ll,int>>, greater<>>`, what does `greater<>` do?

A. Makes `top()` the smallest pair

B. Makes `top()` the largest pair

C. Removes duplicates

D. Sorts by the second member

3. Dijkstra pops `(9, v)` but `dist[v]` is already `6`. What should happen?

A. Relax v's edges with 9

B. Skip it: the pair is stale

C. Set dist[v] = 9

D. Stop the algorithm

4. In 0-1 BFS, where does a neighbour reached by a weight-0 edge go?

A. The back of the deque

B. The front of the deque

C. A separate priority_queue

D. Nowhere until the deque empties

ANSWER KEY · SHORTEST PATHS — BFS, 0-1 BFS AND DIJKSTRA

1. A With all weights equal, the fewest edges is the cheapest route, so BFS's edge counts times 7 are the distances in O(V + E). Dijkstra is correct but pays a log factor; Bellman–Ford is far slower; 0-1 BFS needs 0/1 weights.

2. A The default `less` gives a max-heap; `greater` reverses it into a min-heap, so `top()` is the pair with the smallest distance (ties broken by vertex). Pairs still compare first member first.

3. B A better pair `(6, v)` was pushed later and already processed. `(9, v)` is a leftover of lazy deletion; `if (d > dist[u]) continue;` skips it.

4. B It has the same distance as the vertex just popped, which is the smallest in the deque, so it belongs at the front. A weight-1 neighbour goes to the back.

CHECKPOINT · 45 MIN

Checkpoint — Shortest paths

A 45-minute timed checkpoint on module 8: choosing BFS, 0-1 BFS or Dijkstra from the weights, and writing Dijkstra correctly under a clock.

How it works. Ten questions and two programs, in 45 minutes from when you start the timer. 70% on the questions and both programs accepted clears the module.

Before you start, make sure you can answer these from memory:

- Which algorithm fits equal weights, 0/1 weights, non-negative weights, negative weights.
- What `greater<>` does to a `priority_queue`, and why the distance goes first in the pair.
- What the line `if (d > dist[u]) continue;` skips, and why it is safe.
- Why path sums are `long long`, and how an unreachable vertex is printed.

The programs are Dijkstra's distances from vertex 1 to every vertex with weights up to 10^9, and the number of shortest paths to vertex n modulo 10^9 + 7.

EXERCISE 1

Dijkstra to every vertex

A directed graph has vertices `1` to `n` and `m` weighted edges. Print the shortest distance from vertex `1` to every vertex, `-1` for one that cannot be reached.

Input: `n m`, then `m` lines `u v w` (`1 ≤ n ≤ 10^5`, `0 ≤ m ≤ 2·10^5`, `1 ≤ w ≤ 10^9`). **Output:** `n` integers on one line, separated by spaces.

```
5 6
1 2 4
1 3 1
3 2 2
2 4 5
3 4 8
4 5 3
```

prints

```
0 3 1 8 11
```

INPUT 1

```
5 6
1 2 4
1 3 1
3 2 2
2 4 5
3 4 8
4 5 3
```

OUTPUT 1

```
0 3 1 8 11
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int, ll>>> adj(n + 1);
    for (int i = 0; i < m; ++i) {
        int u, v;
        ll w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
    }
    vector<ll> dist(n + 1, LLONG_MAX);
    priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<>> pq;
    dist[1] = 0;
    pq.push({0, 1});
    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adj[u])
            if (d + w < dist[v]) {
                dist[v] = d + w;
                pq.push({dist[v], v});
            }
    }
    for (int v = 1; v <= n; ++v) cout << (dist[v] == LLONG_MAX ? -1 : dist[v]) << (v
    return 0;
}
```

EXERCISE 2

How many shortest routes

A directed graph has vertices `1` to `n` and `m` weighted edges. Print the length of the shortest path from `1` to `n` and the number of different shortest paths modulo `10^9 + 7` (two paths differ when their edge sequences differ), separated by a space. Print `-1` alone if `n` cannot be reached.

Input: `n m`, then `m` lines `u v w` (`1 ≤ n ≤ 10^5`, `0 ≤ m ≤ 2·10^5`, `1 ≤ w ≤ 10^9`). **Output:** one line.

```
4 5
1 2 1
1 3 1
2 4 1
3 4 1
1 4 3
```

prints

```
2 2
```

INPUT 1

```
4 5
1 2 1
1 3 1
2 4 1
3 4 1
1 4 3
```

OUTPUT 1

```
2 2
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll MOD = 1'000'000'007;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int, ll>>> adj(n + 1);
    for (int i = 0; i < m; ++i) {
        int u, v;
        ll w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
    }
    vector<ll> dist(n + 1, LLONG_MAX), ways(n + 1, 0);
    priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<>> pq;
    dist[1] = 0;
    ways[1] = 1;
    pq.push({0, 1});
    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (d + w < dist[v]) {
                dist[v] = d + w;
                ways[v] = ways[u];
                pq.push({dist[v], v});
            } else if (d + w == dist[v]) {
                ways[v] = (ways[v] + ways[u]) % MOD;
            }
        }
    }
    if (dist[n] == LLONG_MAX) cout << -1 << '\n';
    else cout << dist[n] << ' ' << ways[n] << '\n';
    return 0;
}
```

CHECKPOINT QUESTIONS

1. Which weights make Dijkstra's algorithm give wrong answers?

A. Weights of 0

B. Very large weights

C. Negative weights

D. Equal weights

2. What is the time complexity of the contest Dijkstra with a binary heap and lazy deletion?

A. O(V + E)

B. O((V + E) log V)

C. O(V · E)

D. O(V^2 log V)

3. `pq.push({v, dist[v]})` with `greater<>` — what goes wrong?

A. Nothing

B. The heap orders by vertex number, so vertices settle in the wrong order

C. It does not compile

D. It only fails when weights are 0

4. How many pairs can the lazy-deletion heap hold at most?

A. V

B. About E (one push per successful relaxation, plus the source)

C. V^2

D. log V

5. A grid maze, moves cost 1, 10^6 cells. Which is the best choice?

A. BFS with a queue

B. Dijkstra with a priority_queue

C. Bellman–Ford

D. DFS

6. In BFS on a grid, when should a cell's distance be set?

A. When it is pushed

B. When it is popped

C. After the BFS ends

D. Only for the target

7. What does 0-1 BFS keep true about its deque?

A. It is sorted by distance and holds at most two distinct values

B. It holds every vertex exactly once

C. Its back is always the closest vertex

D. It is empty after the first pop

8. A path has up to 10^5 edges of weight up to 10^9. Which type holds `dist`?

A. int

B. long long

C. float

D. short

9. You need shortest distances from many sources to one target in a directed graph. What is the standard trick?

A. Run Dijkstra from the target on the reversed graph

B. Run Dijkstra from every source

C. Use BFS

D. Sort the edges

10. When counting shortest paths with Dijkstra, relaxing `u → v` gives `d + w == dist[v]`. What do you do?

A. ways[v] = ways[u]

B. ways[v] += ways[u]

C. Push (dist[v], v) again

D. Nothing

ANSWER KEY · CHECKPOINT — SHORTEST PATHS

1. C Dijkstra settles the closest vertex for good; a negative edge found later could make a settled distance shorter. Zero, equal and large non-negative weights are all fine (large ones need long long).

2. B Each edge can push one pair, so the heap sees O(E) pushes and pops at O(log E) = O(log V) each, plus O(V + E) for the scans.

3. B Pairs compare their first member first. With the vertex first, the smallest vertex number pops first, not the closest vertex, and the settled distances can be wrong.

4. B A push happens only when an edge improves a distance, and each edge is relaxed once from its settled tail, so at most E + 1 pairs.

5. A Equal weights: BFS is O(cells) and gives distances directly. Dijkstra adds a log factor; DFS does not find shortest paths at all.

6. A Setting it when pushed marks the cell as seen, so it never enters the queue twice. Setting it on pop lets the same cell be queued many times.

7. A Front pushes have distance d and back pushes d + 1, so the deque is ordered with values d and d + 1 only — its front is always a closest vertex. A vertex can be pushed more than once.

8. B The sum can reach 10^14, past int's 2.1 × 10^9; float is inexact. long long holds up to about 9.2 × 10^18.

9. A Reversing every edge turns "from v to target" into "from target to v", so one Dijkstra gives every source's distance.

10. B Another shortest route into v has been found through u, so its count adds to v's. Replacing is only for a strictly shorter distance, and no push is needed because the distance did not change.

CodeKairo · Competitive Programming study plan · Module 8: Shortest paths: BFS, 0-1 BFS, Dijkstra. Solve the exercises online to track your progress and earn XP.