1.Các đặc điểm chính của hàng đợi ưu tiên:
Min-heap:
priority_queue< int, vector< int>, greater<int>> pq;

Max-heap:
priority_queue<int> pq; // max-heap

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   priority_queue<int> pq; // max-heap

   // Thêm phần tử
   pq.push(5);
   pq.push(1);
   pq.push(3);

   // Lấy phần tử ưu tiên cao nhất
   while (!pq.empty()) {
      cout << pq.top() << " "; // In ra phần tử có ưu tiên cao nhất
      pq.pop(); // Xóa phần tử đó khỏi hàng đợi
   }
   return 0;
}
```
Kết quả: 5 3 1

Thuật toán Dijkstra là một trong những thuật toán phổ biến nhất để tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh khác trong một đồ thị có trọng số không âm.
4 6
Tên con đường:
A B C D
Con đường:
A - B: 3
A - C: 2
A - D: 1
B - C: 1
B - D: 4
C - D: 1

```
#include<bits/stdc++.h>
using namespace std;
void dijkstra(int source, const vector<vector<pair<int, int>>>& a, const vector<string>& d) {
    int n = a.size();
    vector<int> dist(n, numeric_limits<int>::max());
    vector<bool> visited(n, false);

    dist[source] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, source});

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        if (visited[u]) continue;
        visited[u] = true;

        for (auto& neighbor : a[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;

            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }
    // In KQ:khoảng cách từ nguồn đến tất cả các đỉnh

    for (int i = source+1; i < n; ++i) {
        cout << d[source] << " - " << d[i] << ": ";
        if (dist[i] == numeric_limits<int>::max()) {
            cout << "INF" << endl;
        } else {
            cout << dist[i] << endl;
        }
    }
}
int main() {
    int n, m; // số đỉnh và số cạnh
    cin >> n>>m;

    vector<string> d(n); // tên các đỉnh
    vector<vector<pair<int, int>>> a(n); // danh sách kề
    // Nhập tên các đỉnh
```

```
    for (int i = 0; i < n; ++i) {
        cin >> d[i];
    }

    // Nhập các cạnh
    for (int i = 0; i < m; ++i) {
        string u, v; // Tên đỉnh
        int weight;
        cin >> u >> v >> weight; // Nhập tên đỉnh đầu, đỉnh cuối và trọng số
        // Tìm chỉ số của các đỉnh
        int uvt = find(d.begin(), d.end(), u) - d.begin();
        int vvt = find(d.begin(), d.end(), v) - d.begin();
        // Lưu cạnh vào danh sách kề
        a[uvt].emplace_back(vvt, weight);
        a[vvt].emplace_back(uvt, weight); // Nếu đồ thị vô hướng
    }

    // Tìm đường đi ngắn nhất từ đỉnh đầu tiên (chỉ số 0)
    dijkstra(0, a, d);

    return 0;
}
```

Bài toán 1: Đường đi ngắn nhất trong một thành phố

Bạn có một đồ thị đại diện cho một thành phố, trong đó các đỉnh là các điểm giao nhau và các cạnh là các con đường với trọng số là khoảng cách. Bạn cần tìm đường đi ngắn nhất từ một điểm giao nhau (đỉnh nguồn) đến tất cả các điểm khác.

Bài toán 2: Mạng máy tính

Trong một mạng máy tính, các máy tính được kết nối với nhau thông qua các đường truyền có trọng số (băng thông). Bạn cần tìm đường đi ngắn nhất từ một máy tính đến tất cả các máy tính khác để tối ưu hóa lưu lượng mạng.

Bài toán 3: Bản đồ giao thông

Bạn có một bản đồ giao thông với các thành phố và đường đi giữa chúng. Mỗi đường đi có thời gian di chuyển khác nhau. Bạn cần tìm thời gian di chuyển ngắn nhất từ một thành phố đến tất cả các thành phố khác.