گزارش کامل برای کد مکان‌یابی شعبه جدید استارباکس:

1. مقدمه:

هدف این گزارش، مکان‌یابی یک شعبه جدید از استارباکس در یکی از شهرهای ایران است. برای انتخاب مکان بهینه، باید از معیارهای فاصله از مکان‌های مهم موجود در شهر، نظیر مترو و پمپ بنزین، استفاده کنیم. همچنین برای بهره‌برداری از اطلاعات بهتر، میزان تقاضا برای قهوه و هزینه احداث شعبه در هر مکان مهم نیز در نظر گرفته می‌شود.

2. الگوریتم:

برای حل این مسئله، از الگوریتم دایکسترا استفاده می‌کنیم. الگوریتم دایکسترا یک الگوریتم کوتاه‌ترین مسیر در گراف‌ها است. در اینجا، هر نقطه به معنای یک گره در گراف می‌باشد و فاصله بین هر دو نقطه معیاری برای یال بین دو گره متصل است.

3. تابع euclidean_distance:

این تابع با گرفتن دو نقطه به عنوان ورودی، فاصله اقلیدسی بین آن‌ها را محاسبه می‌کند. فرمول فاصله اقلیدسی بین دو نقطه P1(x1, y1) و P2(x2, y2) به صورت زیر است:

distance = sqrt((x1 - x2)^2 + (y1 - y2)^2)

تابع برای محاسبه فاصله بین نقطه‌ها با توجه به مختصات آن‌ها کاربرد دارد.

4. تابع find_optimal_location:

این تابع با گرفتن مختصات شعبه جدید و لیستی از نقاط مهم در شهر، بهترین نقطه برای احداث شعبه جدید را پیدا می‌کند. این تابع از الگوریتم دایکسترا برای محاسبه کوتاه‌ترین مسیر بین نقطه شعبه جدید و نقاط مهم استفاده می‌کند. در این تابع، ابتدا لیستی از فاصله‌ها به اندازه نقاط مهم تعریف می‌شود و با استفاده از الگوریتم دایکسترا فاصله‌های کوتاه‌ترین مسیر بین همه نقاط مهم و شعبه جدید محاسبه می‌شود. سپس برای هر نقطه مهم، مجموع فاصله‌ها به آن نقطه تقسیم بر وزن نقطه‌ها محاسبه می‌شود و نقطه‌ای که کمترین مقدار دارد، به عنوان بهترین مکان برای احداث شعبه جدید انتخاب می‌شود.

5. نتیجه‌گیری:

با اجرای این کد، بهترین مکان برای احداث شعبه جدید از استارباکس در شهر به دست می‌آید. با استفاده از الگوریتم دایکسترا و محاسبه فاصله‌های کوتاه‌ترین مسیر بین نقاط مهم و شعبه جدید، مکانی انتخاب می‌شود که فاصله‌اش از نقاط مهم بهینه‌تر است و همچنین میزان تقاضا و هزینه احداث در آن مکان نیز در نظر گرفته شده است. این روش به شرکت استارباکس کمک می‌کند تا شعبه جدید را در موقعیت بهینه‌تری احداث کند و از سود حداکثری بهره‌مند شود.

توضیحات برای هر خط کد به شرح زیر است:

```
#include <iostream>
```

```
#include <vector>
#include <queue>
#include <cmath>

using namespace std;

struct Point {
    double x, y;
    int weight; // وزن نقطه (1 برای شعبه‌های استارباکس، 2 برای مترو، و 3 برای پمپ بنزین)

    Point(double x, double y, int weight) : x(x), y(y), weight(weight) {}
};
```

این قسمت از کد، نوع ساختار داده‌ای Point را تعریف می‌کند. این ساختار شامل مختصات یک نقطه (x و y) و وزن آن برای محاسبه مسافت‌ها است.

```
// محاسبه فاصله اقلیدسی بین دو نقطه
double euclidean_distance(const Point& p1, const Point& p2) {
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return sqrt(dx * dx + dy * dy);
}
```

این تابع، فاصله اقلیدسی بین دو نقطه را محاسبه می‌کند. برای این منظور از فرمول فاصله اقلیدسی استفاده می‌کند که بر اساس فاصله افقی و عمودی بین دو نقطه محاسبه می‌شود.

```
// الگوریتم دایکسترا برای پیدا کردن بهترین نقطه
// الگوریتم دایکسترا برای پیدا کردن بهترین نقطه
int find_optimal_location(const Point& new_branch, const vector<Point>& important_points) {
    vector<double> distances(important_points.size(), INFINITY);
    priority_queue<pair<double, int>, vector<pair<double, int>>, greater<pair<double, int>>> pq;

    pq.push({0.0, 0}); // شروع از نقطه شعبه اولیه
```

تابع find_optimal_location نقطه بهترین مکان برای احداث شعبه جدید را پیدا می‌کند. این تابع از الگوریتم دایکسترا برای پیدا کردن کوتاه‌ترین مسیر بین نقاط مهم و نقطه شعبه جدید استفاده می‌کند. برای این منظور از یک صف اولویت برای محاسبه کوتاه‌ترین فاصله‌ها استفاده می‌کند.

```
  while (!pq.empty()) {
     int current_point = pq.top().second;
     double current_distance = pq.top().first;
     pq.pop();

     if (current_distance > distances[current_point]) {
        continue;
     }

     for (int i = 0; i < important_points.size(); ++i) {
        double dist = euclidean_distance(important_points[current_point], important_points[i]);
        double total_distance = current_distance + dist;

        if (total_distance < distances[i]) {
           distances[i] = total_distance;
           pq.push({total_distance, i});
        }
     }
  }
```

این بخش از تابع دایکسترا را اجرا می‌کند. در ابتدا، نقطه شعبه جدید به عنوان نقطه شروع در صف اولویت قرار می‌گیرد. سپس از بین نقاط مهم، نقطه با کمترین فاصله به نقطه شعبه جدید از صف حذف و فاصله‌های آن با سایر نقاط محاسبه می‌شوند و اگر فاصله‌ها کوتاه‌تر شوند، به صف اولویت اضافه می‌شوند.

```
    double min_weighted_distance = INFINITY;
    int optimal_location = -1;

    for (int i = 0; i < important_points.size(); ++i) {
        double weighted_distance = distances[i] / important_points[i].weight;
        if (weighted_distance < min_weighted_distance) {
            min_weighted_distance = weighted_distance;
            optimal_location = i;
        }
    }
    return optimal_location;
}
```

در این بخش، بهترین نقطه برای احداث شعبه جدید پیدا می‌شود. برای هر نقطه مهم، مجموع فاصله‌های آن تقسیم بر وزن نقطه‌ها محاسبه می‌شود و نقطه‌ای که کمترین مقدار دارد، به عنوان بهترین مکان انتخاب می‌شود.

```
int main() {
    vector<Point> important_points = {
        // مختصات نقاط پمپ بنزین
        Point(2, 4, 3),
        Point(5, 15, 3),
        Point(6, 7, 3),
        Point(10, 8, 3),
        // مختصات نقاط مترو
        Point(3.5, 5, 2),
        Point(4, 19, 2),
        Point(1.5, 11, 2),
        Point(11, 1.5, 2),
        Point(8, 1, 2),
        Point(12, 10, 2),
```

```
    // مختصات نقاط استارباکس
    Point(7, 5, 1),
    Point(8.5, 16, 1),
    Point(2.5, 9, 1),
    Point(3, 3, 1)
  };

  // مختصات شعبه جدید
  Point new_branch(9, 12, 1);

  int optimal_location = find_optimal_location(new_branch, important_points);
  cout << "مختصات بهترین نقطه: (" << important_points[optimal_location].x << ", " <<
important_points[optimal_location].y << ")" << endl;

  return 0;
}
```