**Алгоритм по шагам функции dijkstra()**

(функция реализующая алгоритм Дейкстрыс модификацией для учёта станций пересадок)

1. Начало алгоритма.
2. Объявление и инициализация переменных:

Вх:

int graph[V][V] – граф, хранящий расстояния между всеми вершинами.

int start – индекс города, из которого едет пользователь.

int end – индекс города, в который едет пользователь.

struct s1\* info – массив структур с информацией о всех городах.

struct s2\* infoWay – массив структур для хранения информации о поездке пользователя.

int\* num – указатель на переменную, хранящую информацию о количестве пересадок на маршруте.

Промеж:

int dist[V] – массив расстояний от стартовой вершины.

int visited[V] – массив посещенных вершин.

int prev[V] – массив предыдущих вершин на пути. int number = -1 – переменная для хранения кол-ва пересадок на маршруте.

int numb = 0 – переменная для записи нужных времён отбытия со станций пересадок и прибытия на станции пересадок.

int current = end – текущее положение начинается с конечного города поездки для прохождения по всем городам на маршруте.

int IndexTransfer – индекс города, где будет осуществлена ближайшая пересадка.

int nap – переменная для хранения направления поездки.

int ind = prev[current] – сохранение последовательности предыдущих вершин на пути.

int flag = 0 – город не является пересадкой.

Вых:

struct s2\* infoWay – массив структур с информацией о поездке пользователя.

1. Цикл от i = 0 до V инициализации переменных для каждого города.
2. Dist[i] = INT\_MAX – инициализация всех расстояний как “бесконечность”. Visited[i] = 0 – отметка всех вершин как не посещённых. Prev[i] = -1 – отсутствие предыдущих вершин на пути.
3. Конец цикла i.
4. Dist[start] = 0 – расстояние от начального города до самого себя равняется 0.
5. Цикл от i = 0 до V – 1 для поиска кратчайшего пути для всех вершин графа.
6. int u = min\_distance(dist, visited) – вызов функции для поиска вершины с наименьшей стоимостью среди вершин, ещё не включённых в кратчайший путь. Visited[u] = 1 – отметка выбранной вершины как посещённой.
7. Цикл от v = 0 до V для обновления расстояний до смежных вершин, учитывая станции пересадок.
8. Если вершина ещё не включена в кратчайший путь, есть ребро из выбранной вершины до смежной вершины и расстояние от источника до выбранной вершины плюс стоимость ребра меньше текущего расстояния до смежной вершины, то обновляем расстояние до смежной вершины: dist[v] = dist[u] + graph[u][v];  
   prev[v] = u;
9. Конец цикла v.
10. Конец цикла i.
11. Если расстояние до конечного города поездки равно “бесконечности”, то путь не найден.
12. Если начальный или конечный город является городом-пересадкой, то flag = 1.
13. Пока не пройдём всю последовательность предыдущих вершин на маршруте, выполняем…
14. Если линии данной станции и предыдущей на маршруте разные, то number = number + 1 и становимся на предыдущую вершину на маршруте (current = prev[current]).
15. Numb = number – сохраняем кол-во пересадок на маршруте для чтения времён прибытия на станцию пересадки и отбытия со станции пересадки.
16. Если number > 0, то…
17. Пока не пройдём всю последовательность предыдущих вершин на маршруте, выполняем…
18. Если линии данной станции и предыдущей на маршруте разные, то с помощью функции strcpy(infoWay[0].Peresadki[numb-1], info[prev[current]].City) копируем название города-пересадки с конца. Numb = numb – 1 – уменьшаем кол-во оставшихся пересадок.
19. Если numb == 0, то выходим из цикла, так как названия всех городов-пересадок уже скопированы.
20. Становимся на предыдущую вершину на маршруте.
21. С помощью функции IndexTransfer = find\_city\_index(info, infoWay[0].Peresadki[0]) получаем индекс ближайшей пересадки на маршруте.
22. С помощью функции nap = napravl(info, start, IndexTransfer, end, number) определяем направление поездки ( если nap = 1, то поездка от центра, если nap = 0, то к центру).
23. С помощью функции infoWay = GetTime(info, infoWay, start, end, nap, number) считываем нужные времена отбытия из начального города и прибытия в конечный город.
24. С помощью функции infoWay = GetTimeOfPeresadki(info, infoWay, nap, number) считываем нужные времена прибытия в город-пересадку и отбытия из него.
25. infoWay[0].Distance = dist[end] – сохраняем кратчайшее расстояние между начальным и конечным городами.
26. \*num = number – сохраняем кол-во пересадок для дальнейшего использования.
27. Возвращаем infoWay.
28. Конец алгоритма.