# 实验四：最短路径算法

**一、实验目的**

（1）理解路径松驰技术的思想

（2）掌握迪杰斯特拉算法（Dijkstra）的基本思想

**二、实验内容**

实现单源最短路经的迪杰斯特拉算法（Dijkstra），要求：采用最小堆实现优先级队列。

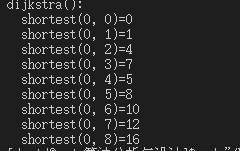
1. **迪杰斯特拉算法（Dijkstra）基本思想**

分成两个集合S，V，如果已经把这个数的最小距离找到就放进S集合里面，假设从一开始，算第一个点和其他点的最小值，遍历一节点的邻接表，找到和一节点最小的节点，放入S集合中，然后开始路径松弛，找到直接到和间接到哪一个近。

1. **实验过程**

遍历G.vexnum-1次；每次找出一个顶点的最短路径,先寻找直接到的最小路径，再用松弛计算比较间接到和直接到哪个小。

1. **实验结果**



using namespace std;

#include <iostream>

#define MAX 10

#define F 10086

typedef struct graph

{

int vexs[MAX] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 }; // 顶点集合

int vexnum = 9; // 顶点数

int matrix[MAX][MAX] = {{ 0, 1, 5, F, F, F, F, F, F },

{ 1, 0, 3, 7, 5, F, F, F, F },

{ 5, 3, 0, F, 1, 7, F, F, F },

{ F, 7, F, 0, 2, F, 3, F, F },

{ F, 5, 1, 2, 0, 3, 6, 9, F },

{ F, F, 7, F, 3, 0, F, 5, F },

{ F, F, F, 3, 6, F, 0, 2, 7 },

{ F, F, F, F, 9, 5, 2, 0, 4 },

{ F, F, F, F, F, F, 7, 4, 0 }};

} Graph;

void dijkstra(Graph G, int start);

int main()

{

Graph G;

dijkstra(G,1);

return 0;

}

/\*

\* Dijkstra最短路径。

\* 即，统计图(G)中"顶点vs"到其它各个顶点的最短路径。

\*

\* 参数说明：

\* G -- 图

\* start -- 起始顶点(start vertex)。即计算"顶点start"到其它顶点的最短路径。

\* prev -- 前驱顶点数组。即，prev[i]的值是"顶点start"到"顶点i"的最短路径所经历的全部顶点中，位于"顶点i"之前的那个顶点。

\* dist -- 长度数组。即，dist[i]是"顶点start"到"顶点i"的最短路径的长度。

\*/

void dijkstra(Graph G, int start)

{

int i, j, k;

int min;

int tmp = 0;

bool flag[MAX] = { false }; // flag[i]表示"顶点vs"到"顶点i"的最短路径是否已获取。TODO: 可用最小堆获取

int prev[MAX] = { start }; // prev 前驱，初始状态下均为到起始点

int dist[MAX] = { F }; // dist 到start点的最短距离

// 初始化

for (i = 0; i < G.vexnum; i++)

{

dist[i] = G.matrix[start][i];// 顶点i的最短路径为"顶点start"到"顶点i"的权。

}

// 对起始定点自身进行初始化

//flag[start] = true;

dist[start] = 0;

// 遍历G.vexnum-1次；每次找出一个顶点的最短路径。

for (i = 1; i < G.vexnum; i++)

{

// 寻找当前最小的路径；

// 即，在未获取最短路径的顶点中，找到离vs最近的顶点(k)。

min = F;

for (j = 0; j < G.vexnum; j++)

{

if (!flag[j] && dist[j] < min)

{

min = dist[j];

k = j; //标记距起点距离最小的点

}

}

// 标记"顶点k"为已经获取到最短路径

flag[k] = true;

// 修正当前最短路径和前驱顶点

// 即，当已经"顶点k的最短路径"之后，更新"未获取最短路径的顶点的最短路径和前驱顶点"。

for (j = 0; j < G.vexnum; j++)

{

//tmp = (G.matrix[k][j] == F ? F : (min + G.matrix[k][j])); // 防止溢出

tmp = min + G.matrix[k][j]; // 防止溢出

if (flag[j] == 0 && (tmp < dist[j]))

{

dist[j] = tmp;

prev[j] = k;

}

}

}

// 打印dijkstra最短路径的结果

printf("dijkstra(%c): \n", G.vexs[start]);

for (i = 0; i < G.vexnum; i++)

printf(" shortest(%d, %d)=%d\n", G.vexs[start], G.vexs[i], dist[i]);

}