Dijkstra Sequence

May 4th 2024

Chapter 1: Introduction

In this project aims at providing us an example to show that the great power of greedy algorithms when we apply them to some certain complicated problems. In this project, our job is simply checking whether the sequences are built in Dijkstra order.

Chapter 2: Algorithm Specification

1) Sketch of Main Program

The whole program is constructed with the following parts:

- Input and Output.
- Initialize the array vert, which is used as the min distance from the source.
- check update function, used to update the min distance.
- check min function, used to check if the sequence sort is in Dijkstra sort.
- **Data Structure:** We use array to store the minimum distance and whether the node is visited. And we store the graph in a adjacent matrix, which is easier to update the distance.

2) Algorithm Pseudo-Code

Algorithm 1: Check Update Algorithm

```
Data: index, Nv, vert[MAXN][2], graph[MAXN][MAXN]
Result: Used to update minimum distance of every node besides index
initialization;
for i ← 1 to Nv do
    if index and i is connected then
        if the distance from index to i is smaller the original distance of
          i then
            change the minimum distance of i
        end
    end
end
```

Algorithm 2: Check Minimum Algorithm

```
Data: check-point, Nv, vert[MAXN][2], graph[MAXN][MAXN]
Result: used to check whether the check-point is in Dijkstra Order
initialization;
min ← ∞
for i ← 1 to Nv do
    if we have never been to node i and the distance from source to node
     i is smaller than min then
        min ← the distance from source to node i
    end
end
if the distance from source to check-point equals min and check-point is
 never visited then
    mark that check-point is visited check-update the nodes besides
     check-point
    return 1
end
return 0
```

3) Description of Algorithms

- input and input-seq function are used to scan the data in a more elegant way
- check_update function is used to form the minimum distance, based on the nodes that you have visited.
- check_min function is the main function to check if the sequence is the Dijkstra sequence. The method is to check is the node every time the smallest and never visited before, which is from Dijkstra algorithm.
- In a word, this program is kind of test program, for the data has pointed the node you need to visit, and your task is to find whether the point is wrong.

Chapter 3: Testing Results

Table 1: Testing Results

Sample	1	2	3	4
Output	Yes	Yes	Yes	No
Output	Yes	Yes	Yes	No
Output	Yes	Yes	No	Yes
Output	No	No	NULL	NULL

In this section, we choose 4 samples which represent some of the strict situations, and you can find them in the **sample.txt** file to check them in detail.

- sample1 and sample2 are provided by the original problem.
- sample3 presents the situation when we have the least nodes.
- sample4 shows that the program works well under unweighted edge.

Chapter 4: Analysis and Comments

1) Time Complexity

From the examples above, I believe that we have already known that how powerful the Dijkstra algorithm is in calculating the shortest path, which enables us to do the work with the time complexity:

$$T(N) = O((V + E)logV)$$

which means we only need to go over all the vertices and edges for one time, and in the meantime, check the vertices we have never visited before. (This is just the average time complexity, not the worst one, which is $T(N) = O(V^2)$)

2) Space Complexity

the space complexity seems to just involve the basic data we need, the status of vertices and the adjacent matrix, which lead to a basic space complexity:

$$T(N) = O(N)$$

For the data that is not too big, the space complexity is enough.

3) Some Potential Improvement

The bfs can still be improved, such as using layers to mark the visited vertices, which can make it search for less times, but still cannot reduce the time complexity.

Appendix: Source Code

```
#include <stdio.h>
#include <limits.h>
//put global variable here
#define MAXN 1000
#define IFN INT_MAX
int graph[MAXN][MAXN];
//
void input(int Ne, int graph[MAXN][MAXN])
{
```

```
    for(int i = 1; i <= Ne; i++)
    {
        int a, b, weight;
        scanf("%d%d%d", &a, &b, &weight);
        graph[a][b] = weight;
        graph[b][a] = weight;
    }
}//input every edge and its weight

void input_seq(int K,int Nv, int seq[100][MAXN])
{
    for(int i = 0; i < K; ++i)
        for(int j = 0; j < Nv;++j)
            scanf("%d", &seq[i][j]);
}//input the sequence that you have to test

void check_update(int index, int Nv, int vert[MAXN][2], int
↪  graph[MAXN][MAXN])
{
    for(int i = 1; i <= Nv; ++i)
    {
        if(graph[index][i] && vert[i][0] == 0)
        {
            if(vert[index][1] + graph[index][i] < vert[i][1])
            {
                vert[i][1] = vert[index][1] +
                ↪  graph[index][i];//update the minimum distance
                ↪  if the neighbour node provides a smaller one
            }
        }
    }
}
int check_min(int check_point, int Nv, int vert[MAXN][2], int
↪  graph[MAXN][MAXN] )
{
    int min = IFN;
    for(int i = 1; i <= Nv; ++i)
    {
        if(!vert[i][0] && vert[i][1] < min)
        {
            min = vert[i][1];
        }//find the minimum distance among the unvisited vertex.
    }
    if(vert[check_point][1] == min && !vert[check_point][0])
    {
        vert[check_point][0] = 1;//mark it visited
```

```
        check_update(check_point, Nv, vert, graph);//update the
        ↪  minimum distance
        return 1;
    }
    return 0;
}
int main()
{
    int seq[100][MAXN];
    int Ne,Nv;
    int K;

    scanf("%d%d", &Nv, &Ne);
    input( Ne, graph);

    scanf("%d", &K);
    input_seq(K, Nv, seq);

    for(int i = 0; i < K; ++i)
    {
        // for every seq
        int vert[MAXN][2];
        int flag = 0;//used to record whether the sequence is
        ↪  Dijkstra sequence
        for(int j = 1; j <= Nv; ++j)
        {
            vert[j][0] = 0;
            vert[j][1] = IFN;
        }//initialize the vert array
        for(int j = 0; j < Nv;++j)
        {
            if(j == 0)
            {
                vert[seq[i][j]][0] = 1;
                vert[seq[i][j]][1] = 0;
                check_update(seq[i][j], Nv, vert, graph);//update
                ↪  the node besides the source.
            }
            else
            {
                if(check_min(seq[i][j], Nv, vert, graph) == 0)
                {
                    ++flag;//if flag == 0, it means we found no
                    ↪  answer .
                }
            }
```

```
        }
        if(flag)
        {
            printf("No\n");
        }
        else
        {
            printf("Yes\n");
        }
    }
}
```

Declaration

I hereby declare that all the work done in this project titled "Dijkstra Sequence" is of my independent effort.