THE 2ND-SHORTEST PATH PROBLEM

Date: 2021-11-25

1.Introduction

We have learnt that there are several algorithms to find a shortest path between 2 vertices in a graph. Here we are to solve the 2nd-shortest path problem. The problem is defined as:

Given a positive-weighted undirected graph $G = (V, E)$, find a path between 2 vertices whose total length is the smallest among all paths except the shortest paths.

Specified in this project, the number of vertices $1 \le M \le 1000$, the number of edges $1 \le N \le 5000$, the weight of any edge $1 \le D_n \le 5000$. We are to find the length of the 2-nd shortest path from 1 to M and its route.

2.Algorithm Specification

a.Theories

Theorem: A 2nd-shortest path from 1 to M consists of a shortest path from 1 to V_s, an edge between V_s and V_t, and a shortest path from V_t to M.

Proof:

1. The 2nd-shortest path must include at least one edge(namely, V_s to V_t) which is not included in the shortest path, apparently. This path can be cut into the mentioned 3 parts, and the 1st and 3rd path must be the shortest path, or this will become a 3rd-or-above-shortest path.

2. So we can enumerate all possible edges V_s to V_t and consider the shortest paths from 1 to V_s and V_t to M. We construct a path from 1 to M using the 3 parts. Possibly it could form a shortest path (so it cannot be the different edge V_s to V_t); if this happens, we can easily eliminate it.

3. After enumerating all edges V_s to V_t in the graph, we can get all possible candidates of the 2nd-shortest path. And the real 2nd-shortest path will be the one who has the least length among these.

b.Algorithm Description (using Dijkstra algorithm)

Input: a positive-weighted undirected graph $G = (V, E)$

Output: The total length w; the path route of the 2nd-shortest path

Main Idea:

1. Use Dijkstra algorithm to find shortest paths from 1 to all vertexes and from M to all vertexes.

2. Enumerate the edge V_s to V_t, find the 2nd-shortest path according to the theorem above.

Data Structure: We use LIST to store the graph $G = (V, E)$, and use ARRAY to store path length and path route. To optimize Dijkstra algorithm, we also wrote a HEAP with *decrease_key* operation. However, these was not shown in the pseudo code, and you should read my C source code to get them.

Pseudo Code:

```
using Dijkstra to assign $dist[0][x] (1 \le x \le M)$ as the shortest distance from $1$ to $x$, and save the route as $route[0][x]$
using Dijkstra to assign $dist[1][x] (1 \le x \le M)$ as the shortest distance from $x$ to $M$, and save the route as $route[1][x]$
$tmplen = inf$
For every edge $e = (V_s, V_t)$ in $G = (V, E)$
  $w = dist[0][V_s] + dist[1][V_t] + length[e]$
  if $w != dist[0][M]$ and $w < templen$ then
    $tmplen = w$
    $tmproute = route[0][V_s] + e + route[1][V_t]$
return $(tmplen, tmproute)$
```

c. A scratch of the main program

Main Idea: The main program is only used to get inputs, give outputs, and call the algorithm process. Main frame of the algorithm is shown in (b).

Pseudo Code:

```
Input($M$,$N$)
For each $i$ from $1$ to $N$
  Input($S$,$T$,$W$)
  Addedge($S$,$T$,$W$)
  Addedge($T$,$S$,$W$)
```

```
    Output(RunAboveAlgorithm())
    EXIT
```

3.Testing Results

Some generated inputs are in the in1,...,in4.txt, and their outputs are in out1,...,out4.txt. These results on time are specifically defined on my personal computer.

Test Case	Description	Purpose	Running Time	Output Conditions
1	sample input	-	0.1s	Correct
2	only 1 edge from 1 to M and 1 from M to 1	backtracking	0.1s	Correct
3	Added 1 edge from test2	strict greater	0.1s	Correct
4	N=4, M=10,with self-loops	self-loop	0.1s	Correct
5	N=1000, M=5000	extreme conditions	0.1s	Correct

Besides these simple tests, you can also test the program in LibreOJ 1099 or Luogu P2865.

4.Analysis and Comments

a. Realization using Dijkstra (as what I wrote)

Time Complexity Analysis: $O((M+N)logM)$. Input processs cost $O(N)$. Two Dijkstra processes make $O(M+N)$ operations on the HEAP, which cost $O(logM)$. Get the 2nd-shortest path costs $O(N)$. Output costs $O(M)$. Adding up being $O((M+N)logM)$. And, if we use FIBONACCI HEAP the time complexity can be reduced to $O(MlogM)$ (but not realized in the source code).

Space Complexity Analysis: $O(M+N)$. To store the edges, we need $O(N)$ space. To save distances and routes, we need $O(M)$ space. The size of the HEAP wouldn't be more than $O(M)$. As a result, the total space complexity is $O(M)+2O(N)=O(M+N)$.

Note: The actual time complexity is heavily depended on the shortest path algorithm chosen. Like if we use Bellman-Ford, the time complexity will become $O(MN)$, which is also acceptable.

Appendix

The realizations of the algorithm are put in pro.c.

The following codes are for comparation with the pseudo codes. You may better read the original code (to get some comments).

Note: the program use stdin and stdout.

a. Heap Realization

```c
typedef struct{int id,key;}pair;
struct heaptype{
    pair data[2*MAXM+5];
    int idpos[2*MAXM+5];
}h;
int top(){return h.data[1].id;}
void decrease_key(int id,int key){
    pair x;
    x.id=id,x.key=key;
    int t=h.idpos[id],i=t;
    while(h.data[i>>1].key>key){
        h.data[i]=h.data[i>>1];
        h.idpos[h.data[i].id]=i;
        i>>=1;
    }
    h.data[i]=x;
    h.idpos[id]=i;
}
void pop(){
    h.data[1].key=inf;
    pair ret=h.data[1];
    int now=1;
    while(1){
        int nxt;
        if(h.data[now<<1].key){
            if(h.data[now<<1|1].key&&h.data[now<<1].key>h.data[now<<1|1].key)
            else nxt=now<<1;
        }else{
            h.data[now]=ret;
            h.idpos[ret.id]=now;
            break;
        }
        if(h.data[nxt].key<ret.key){
            h.data[now]=h.data[nxt];
            h.idpos[h.data[nxt].id]=now;
            now=nxt;
        }else{
            h.data[now]=ret;
            h.idpos[ret.id]=now;
            break;
        }
    }
}
```

b. Dijkstra Algorithm

```
void dijkstra(int x,int lr){
    for(int i=1;i<=m;++i){
        vis[i]=0;
        dist[lr][i]=inf;
        pair xx;
        xx.id=i,xx.key=inf;
        h.data[i]=xx;
        h.idpos[i]=i;
    }
    dist[lr][x]=0;
    decrease_key(x,0);
    while(!vis[top()]){
        int now=top();
        pop();
        vis[now]=1;
        for(edge *p=head[now];p;p=p->next){
            if(dist[lr][now]+p->weight<dist[lr][p->target]){
                dist[lr][p->target]=dist[lr][now]+p->weight;
                decrease_key(p->target,dist[lr][p->target]);
                from[lr][p->target]=now;
            }
        }
    }
}
```

c. The Main Program

```
int main(){
    scanf("%d%d",&m,&n);
    for(int i=0;i<n;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        addedge(x,y,z);
        addedge(y,x,z);
    }
    dijkstra(1,0);
    dijkstra(m,1);
    int minc=inf,rems,remt;
    for(int i=1;i<=m;++i){
        for(edge* p=head[i];p;p=p->next){
            int d=dist[0][i]+p->weight+dist[1][p->target];
            if(d!=dist[0][m]&&minc>d){
                minc=d;
                rems=i,remt=p->target;
            }
        }
    }
    printf("%d ",minc);
    printpath(rems,remt);
}
```

Declaration

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