Lab09-Network Flow

CS214-Algorithm and Complexity, Xiaofeng Gao & Lei Wang, Spring 2021.

* Name: Haoyi You Student ID: 519030910193 Email: yuri-you@sjtu.edu.cn

1. Consider there is a network consists n computers. For some pairs of computers, a wire i exists in the pair, which means these two computers can communicate with each other. When a signal passes through the wires, the noise in the signal will be amplified.If you know the magnification rate of noise $m_{i,j}$ of each wire (which must be greater than 1). Design an algorithm to find the route for each other computer to send signals to the computer v with the minimum total magnification rate of noise and analyze the time complexity.

   Solution. We define $n_{i,j} = ln(m_{i,j})$, thus if the route from u to v has the minimum noise $m = \prod_{e \in path} m_e$, the $ln(m) = \sum_{e \in path} n_e$ is also the minimum,thus we can change the problem to finding the shortest path problem, then we apply the Dijkstra algorithm.

   Algorithm 1: minimum noise

   Input: $G(V, E, M)$, a specific vertex v_0
   Output: Minimum noise M of V

   ```
   set V'
   for v ∈ V do
       if e = (v, v_0) ∈ E then
           d_{v,v_0} ← ln(m_{v,v_0})
       else
           d_{i,j} ← ∞
       V'.insert(v)
   while V' is not empty do
       find v' ∈ V' with d_{v_0,v'} is minimal
       V'.pop(v')
       for v ∈ V' do
           d_{v_0,v} ← min(d_{v_0,v'} + ln(m_{v,v'}), d_{v_0,v})
   for v ∈ V do
       M_v ← exp(d_{v_0,v})
   return M
   ```

   Time complexity: in while loop it recursive $|V|$ times, and in each recursion the for loop it cost $|V'|$.So the complexity is $\sum_{i=1}^{|V|} |V| - i = O(|V|^2)$.
   Space complexity: storing the edges and noise needs $O(|E|)$ space, and storing d, V', M cost $O(|V|)$ space, so the total complexity is $O(|E| + |V|)$ □

2. Suppose that we wish to maintain the transitive closure of a directed graph $G = (V, E)$ as we insert edges into E. That is, after each edge has been inserted, we want to update the transitive closure of the edges inserted so far. Assume that the graph G has no edges initially and that we represent the transitive closure as a boolean matrix.

   (a) Show how to update the transitive closure of a graph $G = (V, E)$ in $O(V^2)$ time when a new edge is added to G.

(b) Give an example of a graph G and an edge e such that $\Omega(V^2)$ time is required to update the transitive closure after the insertion of e into G, no matter what algorithm is used.

(c) Describe an efficient algorithm for updating the transitive closure as edges are inserted into the graph. For any sequence of m insertions, your algorithm should run in total time $\sum_{i=1}^{m} t_i = O(V^3)$, where t_i is the time to update the transitive closure upon inserting the ith edge. Prove that your algorithm attains this time bound.

Solution. (a)

Algorithm 2: update once transitive closure

Input: graph $G = (V, E)$,boolen matrix M,an edge $e = (v_i, v_j)$
Output: Modified M

```
for v1 ∈ V do
    for v2 ∈ V do
        M_{v1,v2} ← M_{v1,v2} | (M_{v1,vi} & M_{vj,v2})
```

(b) In the situation that the whole graph is dichotomous complete graph with two parts G_1, G_2, each part has $\frac{|V|}{2} vertexs$, choose $v_1 \in G_1, v_2 \in G_2$,add (v_1, v_2) into the graph, and $\forall v_i \in G_1, v_j \in G_2, M_{v_i,v_j}$ change to true from false, so we at least needs to modify $\frac{n^2}{4} = \Omega(n^2)$ datas.

(c) The algorithm is below.

Algorithm 3: Online updating transitive closure

Input: graph $G = (V, E)$,boolen matrix M,a sequence of edges $\{e_i\} = \{(u_i, v_i)\}$
Output: Modified M

```
for u ∈ V, v ∈ V do
    if u == v then
        M_{u,v} ←true
    else
        M_{u,v} ←false
\\ initialization
for e ∈ {e_i} do
    \\ first layer of loop
    for w ∈ V do
        \\ second layer of loop
        if M_{w,u_i} == true & M_{w,v_i} == false then
            for x ∈ V do
                \\ third layer of loop
                M_{w,x} ← M_{w,x} | M_{v_i,x}
    \\ This time the M is the result after adding e_i
```

Here we prove the time complexity of this algorithm is $O(n^3)$.
Let $\Phi(i)$ be defined as the number of false elements in boolen matrix M.Obviously, $\Phi(i) = n^2 - n = O(n^2)$.
In the algorithm, every time entering into the third layer of loop means $M_{w,v_i} == false$, so if $\Phi(i) == 0$, it will never enter into this loop. Assume t is the total time entering into this loop. The complexity of this algorithm is $O(n * t)$
Each time it enters into the loop, M_{w,v_i} will change to true from false, so at least $\Phi(i)$

will minus 1,thus $t \leq \Phi(i) = O(n^2)$.
So the total time of complexity is $O(n * t) = O(n^3)$.

□

3. An $n \times n$ grid is an undirected graph consisting of n rows and n columns of vertices, as shown in Figure 26.11. We denote the vertex in the ith row and the jth column by (i, j). All vertices in a grid have exactly four neighbors, except for the boundary vertices, which are the points (i, j) for which $i = 1, i = n, j = 1$, or $j = n$. Given $m \leqslant n^2$ starting points $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ in the grid, the escape problem is to determine whether or not there are m vertex-disjoint paths from the starting points to any m different points on the boundary such that every vertex in V is included in at most one of the m paths. For example, the grid in Figure 1(a) has an escape, but the grid in 1(b) does not.

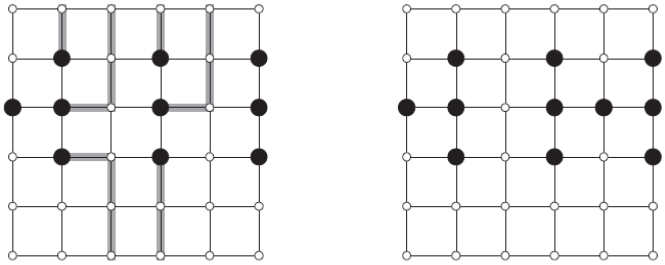

图 1: Grids for the escape problem. Starting points are black, and other grid vertices are white. (a) A grid with an escape, shown by shaded paths. (b) A grid with no escape.

(a) Consider a flow network in which vertices, as well as edges, have capacities. That is, the total positive flow entering any given vertex is subject to a capacity constraint. Show that determining the maximum flow in a network with edge and vertex capacities can be reduced to an ordinary maximum-flow problem on a flow network of comparable size. That is, the sizes of the two graph are in the same order of magnitude.

(b) Describe an efficient algorithm to solve the escape problem, and analyze its running time.

Solution.

Assume the graph $G = (V, E, M, N)$ with M is the capacity of V and N is the capacity of E. Firstly,we split each point v into 2 points v_{in}, v_{out} that if a directed edge ends at v, we change it to end at v_{in}. Similarly, if a edge starts from v, we change it to start from $vout$. The capacity of the edge do not change.
Secondly,for each two splited points $v_{in}, vout$,we add an edge starting from v_{in} to v_{out} with the capacity of the node v.
Thus, the new graph G' has $2|V|$ nodes and $|V| + |E|$ edges, each edge has a capacity from M or N, so the sizes of two graph are in the same order of magnitude.

The algorithm is below,similar to the algorithm above,change the problem to network flow:

Algorithm 4: Escape

```
Input: Size N,starting points array A={(x, y)}
Output: boolen Bool
vertex source, sink
V.insert(source)
V.insert(sink)
for v ∈ E do
    if v ∈ A then
        for u ∈ V and (v, u) ∈ E do
            E.delete((v, u))
            E.insert(source, u)
        V.pop(v)
    else
        if v in boundary then
            for u ∈ V and (u, v) ∈ E do
                E.delete((u, v))
                E.insert(u, sink)
            V.pop(v)
        else
            vertex v_in, v_out V.insert(v_in)
            V.insert(v_out)
            for u ∈ V and (u, v) ∈ E do
                E.delete((u, v))
                E.insert(u, v_in)
            for u ∈ V and (v, u) ∈ E do
                E.delete((v, u))
                E.insert(v_out, u)
            E.insert(v_in, v_out V.delete(v)
initialize G with all edge capacity be 1
max ← Max-flow(G)
if max == sizeof(A) then
    return True
else
    return False
```

Change the graph the same to (a). And merge all the starting points into source and merge all the boundary points(except starting points) into sink. Then the graph has only one sink and source, thus it can apply the Max-flow algorithm mentioned in the class.
Since the merge time complexity is $O(N^4)$ and the max-flow complexity is $O(|V||E|C)$ with $C == 1, |V| = O(N^2), |E| = O(N^2)$ so the complexity is $O(n^4)$. □

Remark: Please include your .pdf, .tex files for uploading with standard file names.