091M4041H - Algorithm Design and Analysis

Assignment 5 NF

2019 年 8 月 30 日

目录

1 Load Balance

You have some different computers and jobs. For each job, it can only be done on one of two specified computers. The load of a computer is the number of jobs which have been done on the computer. Give the number of jobs and two computer ID for each job. You task is to minimize the max load.

(hint: binary search)

1.1 Algorithm Description and Pseudo-code

假设有 m 个 job 和 n 个 computer，并且每个 job 可以选择 2 个 computer 中的一个完成工作，因此考虑建图如下：

1. 建立超级源点和超级汇点，超级源点和 m 个 job 相连并且由源点指向每个 job，边的容量为 1；
2. n 个 computer 与超级汇点相连接，并且每个 computer 指向汇点，边的容量记为 c，表示 n 台 computer 中的最大负载；
3. 每个 job 与其对应可选的两个 computer 相连，由 job 指向 computer。

采用二分搜索来猜测 c，若 t 能收到的流为 m，说明 c 偏大，可以继续缩小，直到 t 刚好不能收到 m。

Algorithm 1 Search the minimum of the max load on computers.

```
function LoadBalance
    for e doach job
        add edge (job_i, computer_a) and (job_i, computer_b) and capacity c_i = 1.
    end for
    add node s and t
    add edge (s, job_i) and capacity c_i = 1
    L = 1, R = m + 1
    add edge (computer_i, t) and capacity c_i = (L + R)/2
    while L < R do
        mid = (L + R) / 2
        change capacity c_i of edge (computer_i, t) to mid
        if max_flow(s,t) == m then
            R = mid
        else
            L = mid + 1
        end if
    end while
    return L
end function
```

1.2 Proof of Algorithm Correctness

每个 computer 到汇点 t 的容量上限为 c，使 computer 的最大负载不超过 c

使用二分搜索猜测 c 的值，因为 c 的取值范围为 [1,m] 所以每次缩减 L 和 R 两个边界缩小 c 的取值，若使用 (L + R)/2 得到最大流为 m，则将 R 取中值，表示实际最小 c 应该比中值小，同理若不能取得流 m，则实际最小 c 应该比中值大，因此通过控制 L 和 R 此算法必定收敛。

1.3 Complexity Analysis

由二分搜索，共需执行 log(m) 次最大流，若采用 Dinitz 算法计算最大流，其时间复杂度为 O(N^2M)，其中 N = m + n + 2, M = m + 2m + n，因此算法的时间复杂度为

$$T(n) = O((m+n)^3 log(m)) \tag{1}$$

2 Matrix

For a matrix filled with 0 and 1, you know the sum of every row and column. You are asked to give such a matrix which satisfys the conditions.

2.1 Algorithm Description and Pseudo-code

假设矩阵为 m 行 n 列，其中的点（n, m）看作一条有向边，因此考虑建图如下：

1. 构建超级源点和超级汇点，超级源点指向所有行节点 m，容量为对应行和；所有列节点 n 指向超级汇点，容量为对应列和

2. 将矩阵中的点 (m,n) 表示行节点 m 指向列节点 n，容量为 1

当进行最大流算法时，若源点 s 的每个出边和汇点 t 的每个入边都是满流的，则说明符合条件, 并且若行节点和列节点之间满流则说明其有值，对应值即为 1-剩余容量。

Algorithm 2 Search matrix which satisfys the conditions..

```
function SearchMatrix(matrix(m,n))
    s = 0
    t = m + n + 1
    for i from 1 to m do
        add_edge(0, i, row[i - 1])
        for j from m + 1 to m + n do
            add_edge(i, j, 1)
        end for
    end for
    for i from m + 1 to m + n do
        add_edge(i, t, col[i - m - 1])
    end for
    max_flow(m, n, s, t)
    for i from 1 to m do
        for j form 1 to n do
            matrix[i-1][j-1] = 1 - c_ij
        end for
    end for
    return matrix
end function
```

2.2 Proof of Algorithm Correctness

由源点的出边即限定了每一行的行和，由汇点的入边即限定了每一列的列和，中间的行节点到列节点容量即限定了矩阵中的节点取值，因此条件限制成立，若源点的出边和汇点的入边均满流，则原问题有可行解。

2.3 Complexity Analysis

时间复杂度即为最大流时间，若采用 Dinitz 算法最大流时间为 O(N^2M), 其中 N =m + n + 2, M = m + n + mn。因此算法的时间复杂度为:

$$T(n) = O(mn(m+n)^2) \tag{2}$$

3 Problem Reduction

There is a matrix with numbers which means the cost when you walk through this point. you are asked to walk through the matrix from the top left point to the right bottom point and then return to the top left point with the minimal cost. Note that when you walk from the top to the bottom you can just walk to the right or bottom point and when you return, you can just walk to the top or left point. And each point CAN NOT be walked through more than once.

3.1 Algorithm Description and Pseudo-code

假设矩阵 m 行 n 列，则考虑建图如下：

对于每一个矩阵内的节点，都将其转化为一条边 $(node1_{ij}, node2_{ij})$，其容量为 1，以限制每个节点仅经过 1 次，将矩阵最左上角的节点建设的边 $(node1_{00}, node2_{00})$ 的容量设为 2，以保证其出发和返回共经过两次，费用为 Matrix[i][j]。且将 $node1_{00}$ 设为 s 源点，同理，矩阵最右下角的节点建设的边 $(node1_{m-1n-1}, node2_{m-1n-1})$ 的容量设为 2，以保证其出发和返回共经过两次，且将 $node2_{m-1n-1}$ 设为 t 汇点，每个矩阵中的节点只向左上和右下节点移动，即每个节点的 $node2_{ij}$ 分别与 $node1_{i+1j}$ 和 $node1_{ij+1}$ 建立连边，且容量为 1，费用为 0，即将问题转化为求取最大流问题。

Algorithm 3 Search path in the matrix

function LoadBalance(Matrix)
    for e **do**ach (i,j) in Matrix
        if (**then**i = 0 and j = 0) or (i = m-1 and j = n-1)
            add edge $(node1_{ij}, node2_{ij})$, capacity c = 2, cost e = Matrix[i][j]
        else
            add edge $(node1_{ij}, node2_{ij})$, capacity c = 1, cost e = Matrix[i][j]
        end if
        if i **then** < m - 1
            add edge $(node2_{ij}, node1i + 1j)$, capacity c = 1, cost e = 0
        end if
        if j **then** < n - 1
            add edge $(node2_{ij}, node1ij + 1)$, capacity c = 1, cost e = 0
        end if
    end for
    solve_min_cost_flow(2)
end function

3.2 Proof of Algorithm Correctness

对于每一个矩阵内的节点，都将其转化为一条边其容量为 1，费用为矩阵节点值，以限制每个节点仅经过 1 次不重复。对于每个节点化成的边的末节点都与其下边和右边的节点所化边的头结点相连，其容量为 1，费用为 0，以限制每次只走右边和下边且不重复。对于源点和汇点边容量设为 2，以保证其往返两条流。

因此算法转化为最小费用流问题，且流位 2，算法必定收敛。

3.3 Complexity Analysis

若用 Bellman_Ford 求最短路,时间复杂度为 O(2VE), 如果采用二叉堆的 Dijkstra,时间复杂度为 O(2ElogV)。

$$T(n) = O(ElogV) \tag{3}$$