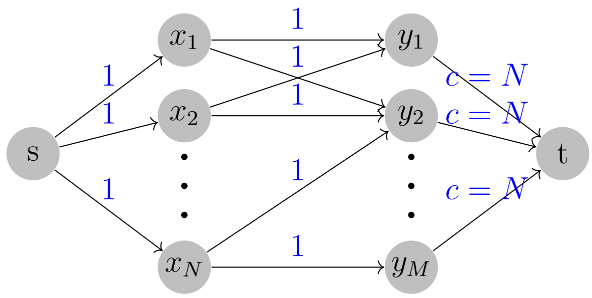
Assignment 5 of Algorithm

Zuyao Chen 201728008629002

1. Load balance

- **Description:**
  Let $x_i, i = 1, 2, ..., N; y_k, k = 1, 2, ..., M$ represent the nodes of jobs and computers respectively, add two nodes s and t, construct a initial network like this



  we can find a flow of maximum value $val(f)$ using method like FORD-FULKERSON , if $val(f)$ is exactly the number of jobs N, then we reduce each capacity c from y_i to t, if we still get $val(f) = N$, we can continue to reduce the capacity c; otherwise, we should increase the capacity.If we can not reduce the capacity c to maintain $val(f) = N$, then we get the max load. Binary search can be used to adjust the capacity.

- **Correctness:**
  Since we must maintain the maximum flow value $val(f)$ equals to the number of jobs N, the max load is no larger than the capacity c, reduce the capacity c until $val(f)$ not equals N,then the capacity c is the minimum max load.

- **Complexity:**
  - FORD-FULKERSON costs $O((3N + M)^2(N + M + 2))$ time using Edmonds-karp
  - binary search averagely costs $O(\log N)$ time

  In summary , it costs $O(\log N(3N + M)^2(N + M + 2))$ time

Algorithm 1 load balance

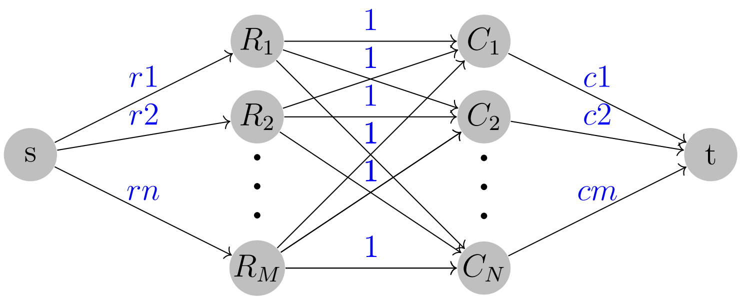
Input: the number of jobs and two computer ID for each job

Output: minimum of the max load

```
function LOAD-BALANCE
    add two nodes s and t,construct the initial network G
    set each capacity c with N
    left = 0, right = N
    max_load = N, pre_load = N
    while left <= right do
        pre_load = max_load
        max_load = left + (right-left)/2
        set the capacity c with max_load
        max_flow = FORD-FULKERSON(s,t)
        if max_flow = N then
            right = max_load - 1
        else
            left = max_load + 1
        end if
        if left >= right and max_flow ≠ N then
            max_load = pre_load
        end if
    end while
    return max_load
end function
```

2. Matrix

- **Description:**
  construct the network



in the network, the capacity of the edge from s to R_i represents the sum of ith row, the capacity of the edge from C_i to t represents the sum of ith column, the capacity of the edge from R_i to C_k is 1. Solve the network as a maximum flow problem, if the maximum flow value equals to the sum of all matrix elements then $Matrix[i][j] = f(R_i \to C_j)$ otherwise no feasible solution.

Algorithm 2 Matrix

Input: the sum of each row and column
Output: a matrix that satisfys the conditions

function MATRIX
  construct a network as shown above
  find the maximum flow value $val(f)$
  if $val(f) = \sum_i r_i$ **then**
    $Matrix[i][j] = f(R_i \to C_j)$
  else
    return “no feasible solution”
  end if
  return $Matrix$
end function

- **Correctness:**
  the problem can be formulated as a Circulation problem.
- **Complexity:**
  the flow value is at most equal to MN, so time complexity is $O(M^2N^2)$.

3. Problem Reduction

- **Description:**

  I. construct a graph network with each item of the matrix $M[i][j]$ as a vertex $v_{i,j}$, add edges from $v_{i,j}$ to $v_{i+1,j}$ and $v_{i,j+1}$.Set the capacity $C(u \to w) = 1$ and cost $W(u \to v_{i,j}) = M[i][j]$, let $v_{1,1} = s, v_{m,n} = t$.

  II. apply minimum cost algorithm (Klen algorithm) with flow $v = 2$, the sum of cost is minimum.

Algorithm 3 MinCost

Input: Matrix M_{ij} with number that means the costs when walk through a certain point
Output: A flow f that contains two path from s to t without intersection

function MINCOST(M)
  construct a network G as discribed above
  f = Ford-Fulkerson(G)
  while G_f has a negatibe circle C **do**
    b = bottleneck(C)
    $\hat{f}$ is the unit flow of C
    $f = f + b\hat{f}$
  end while
  return f
end function

- **Correctness:**

- the capacity of edges equals to 1,thus one certain vertex can be passed at most once and the value of a single path from s to t can only be 1
- there exist two edges starting from s, and the capacity of each edge is 1,so the maximum flow value is 2.According to the conclusion that the value of flow of a single path is 1, there are two different paths from s to t without intersection
- a path from s to t with a intersection to right or bottom is equivalent to a path from t to s with a direction to top or left. Hence, walking through one of the two path in the flow f in reverse direction, we can get a single path from the top left point to the right bottom point and then return to the top left point with the minimal cost

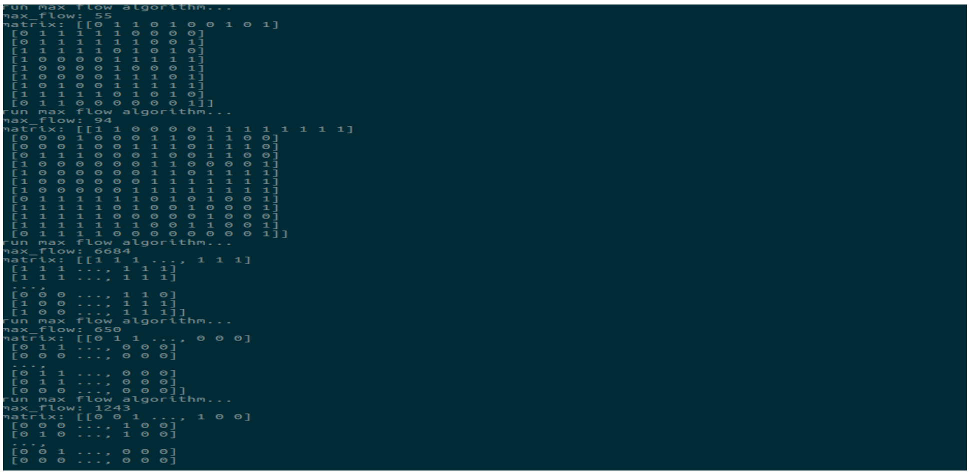
• **Complexity:**

- there are $2mn - m$ edges in the graph and $C = 1$, so the Ford-Fulkerson costs $O(mn)$ time
- there are mn vertices in the graph, the time complexity of Bellman-Ford algorithm is $O(m^2n^2)$
- the number of possible negative circle is at most $O(mn)$

Hence, the total time complexity is at most $O(m^3n^3)$.

4. Ford Fulkerson

5. Push relabel



```
run max flow algorithm...
max_flow: 55
matrix: [[0 1 1 0 1 0 0 1 0 1]
 [0 1 1 1 1 1 0 0 0 0]
 [0 1 1 1 1 1 1 0 0 1]
 [1 1 1 1 1 0 1 0 1 0]
 [1 0 0 0 0 1 1 1 1 1]
 [1 0 0 0 0 1 0 0 0 1]
 [1 0 0 0 0 1 1 1 0 1]
 [1 0 1 0 0 1 1 1 1 1]
 [1 1 1 1 1 0 1 0 1 0]
 [0 1 1 0 0 0 0 0 0 1]]
run max flow algorithm...
max_flow: 94
matrix: [[1 1 0 0 0 0 1 1 1 1 1 1 1 1]
 [0 0 0 1 0 0 0 1 1 0 1 1 0 0]
 [0 0 0 1 0 0 1 1 1 0 1 1 1 0]
 [0 1 1 1 0 0 0 1 0 0 1 1 0 0]
 [1 0 0 0 0 0 0 1 1 0 0 0 0 1]
 [1 0 0 0 0 0 0 1 1 0 1 1 1 1]
 [1 0 0 0 0 0 0 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 1 1 1 1 1 1 1 1]
 [0 1 1 1 1 1 1 0 1 0 1 0 0 1]
 [1 1 1 1 1 0 1 0 0 1 0 0 0 1]
 [1 1 1 1 1 0 0 0 0 0 1 0 0 0]
 [1 1 1 1 1 1 1 0 0 1 1 0 0 1]
 [0 1 1 1 1 0 0 0 0 0 0 0 0 1]]
run max flow algorithm...
max_flow: 6684
matrix: [[1 1 1 ..., 1 1 1]
 [1 1 1 ..., 1 1 1]
 [1 1 1 ..., 1 1 1]
 ...,
 [0 0 0 ..., 1 1 0]
 [1 0 0 ..., 1 1 1]
 [1 0 0 ..., 1 1 1]]
run max flow algorithm...
max_flow: 650
matrix: [[0 1 1 ..., 0 0 0]
 [0 1 1 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 ...,
 [0 1 1 ..., 0 0 0]
 [0 1 1 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
run max flow algorithm...
max_flow: 1243
matrix: [[0 0 1 ..., 1 0 0]
 [0 0 0 ..., 1 0 0]
 [0 1 0 ..., 1 0 0]
 ...,
 [0 0 1 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
```

the test result on 'problem2.data' is showed above, the correctness of algorithm can be validated by broute fource. It is worth noting that using push relabel insead of Ford Fulkerson is wise since time complexity.