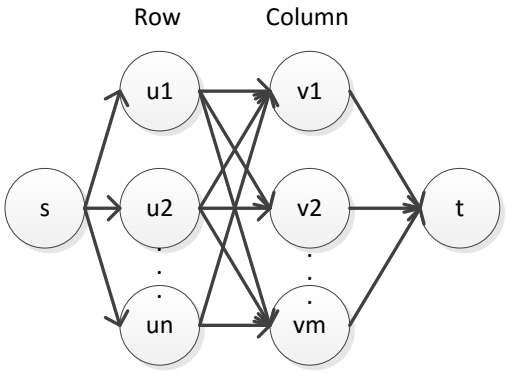
学号： 201618013229011
姓名： 李坚松

Assignment 5

Problem-2

Solution

We can construct the following network flow model to solve this problem:



Assume that the matrix size is n*m. We add edges from s to ui, where i=1,2,…,n; and we add edges from vi to t, where i=1,2,…,m; and we add edges for every two pair of ui and vj, where i=1,2,…,n, j=1,2,…,m. The capacity of s to ui is the row sum of the i-th row, the capacity of vj to t the column sum of the j-th column, the capacity of ui and vj is 1. So we can treat this problem as an max flow problem. We can take use of Ford-Fulkerson algorithm to solve it. If the max flow is sum of all the row sum, then we can construct the matrix by each flow. Otherwise, the matrix doesn't exist. If the matrix does exist, we can construct the matrix by the following method: If there is a flow from ui to vj, then the value in the i-th row and j-th column is 1, else the value in the i-th row and j-th column is 0. The pseudo-code is shown below:

Algorithm: FindMatrix(rowsum[], columnsum[])

Input: row sum and column sum

Output: the 0-1 matrix

```
Begin
    Add edges from s to each row node;
    Add edges from each column node to t;
    add edges for every two pair of row node and the column node;
    Set the capacity of s to ui is the row sum of the i-th row ;
    Set the capacity of vj to t the column sum of the j-th column;
    Set the capacity of ui and vj is 1;
    Using Ford-Fulkerson algorithm to get the max flow f;
    If f== sum of the row sum
```

```
    Matrix[i][j]=flow(ui->vj);
    Return Maxtrix;
  Else
    Return error.
End
```

Proof

By constructing the network, we can know that each flow in the network corresponds to a value in the matrix. So if the max flow is sum of all row sum, it means that the network satisfy the sum of every row and column, so the matrix does exist.

Time Complexity

The main time consumption is the construction of the matrix from the recorded max-flow, so the time complexity of the problem is O(m^2*n^2).

Problem-3

Solution

To determine whether G has a unique minimum st cut, we can use the following method: we perform Ford-Fulkerson algorithm on the network G, then we DFS on the residual graph forward along the not full edge, then we get the set of nodes marked by S, similarly, we perform DFS on the residual graph backward along the not full edge, then we get the set of nodes marked by T, if |S|+|T|=|V|, we can say that G has a unique minimum st cut. The pseudo-code is shown below:

```
Algorithm: UniqueCut( G, s, t )
Input: the network graph G
Output: whether exist a unique st cut
Begin
  Using Ford-Fulkerson algorithm on G to get the max flow f;
  DFS on the residual graph forward along the not full edge to get the set of
  nodes marked by S;
  DFS on the residual graph backward along the not full edge to get the set of
  nodes marked by T;
  If(|S|+|T|==|V|)
    Return true;
  Else
```

Return false.

End

Proof

After we perform Ford-Fulkerson algorithm on the network G, we can get the minimum cut by DFS the residual graph, we have two methods to DFS the residual graph: the first is to DFS forwards from s along the not full edge we get the set of nodes S, so the partition of all nodes is {S} and {V}-{S}; the second method is to DFS backwards from t along the not full edge we get the set of nodes T, so we get the partition of all nodes is {T} and {V}-{T}; now we can say if {V}-{S}={T}, that is to say |V|-|S|=|T|, G has unique minimum st cut, else it doesn't has unique minimum st cut.

Time Complexity

The main time consumption is to use Ford-Fulkerson algorithm on G, so the time complexity is the time complexity of Ford-Fulkerson algorithm, that is to say the time complexity is $O(|V|^2*|E|)$, where |V| is the number of nodes, and |E| is the number of edges.

Problem-4

Solution

We can construct the following network flow model to solve this problem. Assume the matrix is M[m][n], we treat M[1][1] as s, and M[m][n] as t, for each value in the matrix we add a node $c_{i,j}$ except s and t and each value in the matrix can be taken as a node $n_{i,j}$. Add edges from s to its right node and the node below it, e.g.:s->$n_{1,2}$;s->$n_{2,1}$; $n_{m-1,n}$->t;$n_{m,n-1}$->t; $n_{i,j}$->$c_{i,j}$; $c_{i,j}$->$n_{i,j+1}$; $c_{i,j}$->$n_{i+1,j}$; Then we set the cost u->$n_{i,j}$ is M[i][j], cost u->$c_{i,j}$ is 0 and the capacity of each edge is 1. Then the problem can be transformed into minimum cost problem. The pseudo-code is shown below:

Algorithm: MinimumCost(Matrix[m][n])

Input: the cost matrix M

Output: the minimum cost from top left point to the right bottom point and return to the top left point

Begin

treat M[1][1] as s, and M[m][n] as t;

For each value in the matrix add a node $c_{i,j}$ except s and t and each value in the matrix can be taken as a node $n_{i,j}$.

Add the following edges: s->$n_{1,2}$;s->$n_{2,1}$; $n_{m-1,n}$->t;$n_{m,n-1}$->t; $n_{i,j}$->$c_{i,j}$; $c_{i,j}$->$n_{i,j+1}$;

$c_{i,j}$->$n_{i+1,j}$;
Set the cost u->$n_{i,j}$ = M[i][j], cost u->$c_{i,j}$ = 0 and the capacity of each edge = 1;
Solve the problem with the minimum cost algorithm which v=2;
End

Proof

After we construct the network, we know that each node is visited by at most once, since the capacity of each edge is 1. There are 2 disjoint path from s to t, because v=2 and capacity is 1. One path is walk right and down and another path is the inverse of another flow. Since we use minimum cost to solve this problem, the cost is minimum.

Time Complexity

The main time consumption is to use minimum cost algorithm on the constructed network, so the time complexity is the time complexity of minimum cost algorithm, which is O(mnC).

Problem-7

Solution

The dual of the max flow is shown below:

$$\min \sum u_e y_e$$
$$s.t. \sum_{e \in p} y_e \geq 1, \textit{ for all path } p$$
$$y_e \geq 0$$

In the above equation, y_e is the edge, 0 means that we don't choose it, 1 means that we choose it, the objective function means the minimum st cut. The constraints mean for all the edges in the path p, at least one edge is in the cut.

Problem-9

Notice

The implement of this problem see the source file **push_relabel.cpp**.
Input: M and N, which means the matrix is M*N;
Next 2 line, the first line has M numbers, which indicate the sum of rows and the second line has N numbers, which mean the sum of columns.
Output: the 0-1 matrix if it really exists, else output “the matrix does not exist!”

Problem-10

Notice

The implement of this problem see the source file **min_cost_flow.cpp**.
INPUT: A directed graph G =< V,E >. Each edge e has a capacity c_e and a cost w_e. Two special points: source s and sink t. The input is in DIMACS format. e.g.:

```
c This is a simple example file to demonstrate the DIMACS
c input file format for minimum cost flow problems. The solution
c vector is [2,2,2,0,4] with cost at 14.
c
c Problem line (nodes, links)
p min 4 5
c
c Node descriptor lines (supply+ or demand-)
n 1 4
n 4 -4
c
c Arc descriptor lines (from, to, minflow, maxflow, cost)
a 1 2 0 4 2
a 1 3 0 2 2
a 2 3 0 2 1
a 2 4 0 3 3
a 3 4 0 5 1
c
c End of file
```

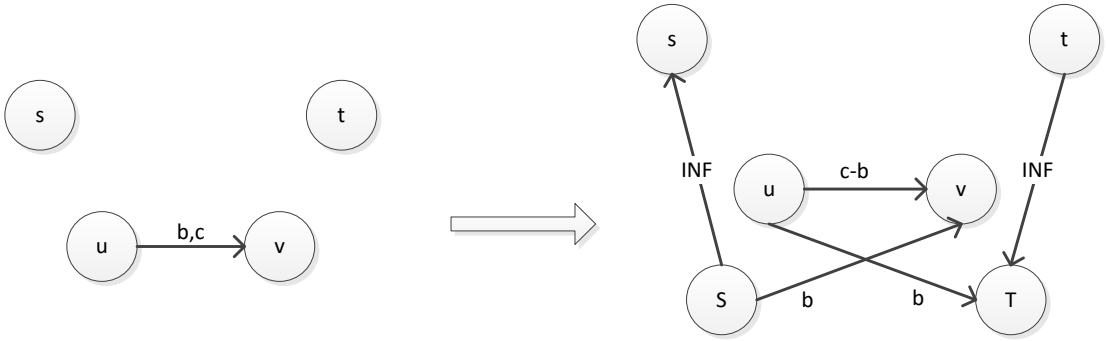
OUTPUT: the flow of each edge and the minimum cost.
For the DIMACS format, we see that each edge has minflow and maxflow, which means that the flow on each edge marked by f, then minflow <=f<= maxflow, in the implement, I simplify this problem by setting minflow=0. **If we do not simplify minflow to 0 for this**

problem, we can construct the following model:

Suppose the flow on the edge e is f(e), and its minflow is b(e), its maxflow is c(e), so we have b(e)<=f(e)<=c(e), suppose f'(e)=f(e)-b(e), so we have 0<=f'(e)<=c(e)-b(e), so the total in flow and total out flow for each edge is

$$\sum_{e\in\delta_-(v)} f'(e)+b(e)=\sum_{e\in\delta_+(v)} f'(e)+b(e)$$

This means that we have a source whose maximum out flow is $\sum_{e\in\delta_-(v)} b(e)$,and we have a sink whose maximum in flow is $\sum_{e\in\delta_+(v)} b(e)$ for each node, so we can add a new source node S and a new sink node T, for each edge e=(u,v), suppose c'(e)=c(e)-b(e), the capacity of each edge is c'(e), we add a link from S to v whose capacity is b(e), and we add a link from u to T whose capacity is b(e), and we add a link from S to the original source s whose capacity is INF, we add a link from t to T whose capacity is INF. So now we can solve this problem by min-cost flow algorithm. The constructed model is shown below:



If we get the maxflow from S to T is F', then the maxflow from s to t is $F=F'-\sum_{e\in E} b(e)$.