Algorithm

孔静 2014K8009929022

December 6, 2016

1 Load Balance

1.1 代码

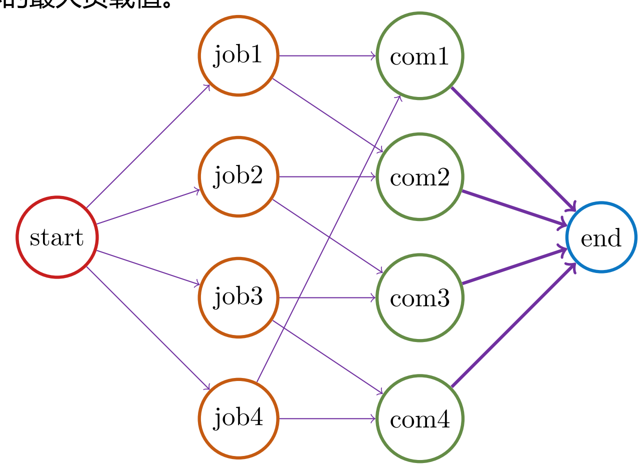
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define L 500
int capbility[L][L];
int backup[L][L] = {0};
int distance[L], current[L];
int a[L], b, c;
int Point, Edge, Head, Tail, MAX;
int min(int x, int y)
{
  return (x < y) ? x: y;
}
int BFS()
{
  int i,j;
  memset(distance, 0xff, sizeof(distance));
  distance[1] = 0;
  b = 0;
  c = 1;
  a[1] = 1;
  while(b < c)
  {
    j = a[++b];
    for (i = 1; i <= Point; i++)
    if (distance[i] < 0 && capbility[j][i] > 0)
    {
      distance[i] = distance[j] + 1;
      a[++c] = i;
    }
  }
  if(distance[Point] > 0)
    return 1;
  else
    return 0;
}
int dinic(int x,int low)
```

```
{
  int i, flow;
  if (x == Tail)
    return low;
  for(i = current[x]; i <= Point; i++)
  {
    current[x] = i;
    if (capbility[x][i] > 0 && distance[i] == distance[x] + 1)
    {
      flow = dinic(i, min(low, capbility[x][i]));
      if(flow)
      {
        capbility[x][i] -= flow;
        capbility[i][x] += flow;
        return flow;
      }
    }
  }
  return 0;
}
int main()
{
  int num1, num2, job, com1, com2, i, j;
  scanf("%d%d",&num1, &num2);//job number, computer number
  for(i = 1; i <= num1; i++)
  {
    scanf("%d%d%d", &job, &com1, &com2);
    backup[1][1 + job] = 1;
    backup[1 + job][1 + num1 + com1] = 1;
    backup[1 + job][1 + num1 + com2] = 1;
  }//data backup
  Head=1;Tail=2+num1+num2;Point=Tail;Edge=3*num1+num2;//set parameter
  int left, right, mid, add;
  left = 0;right = num1 + 1;
  mid = (left + right) / 2;
  while(left + 1 < right)//binary search
  {
    for(i = 1; i <= 1 + num1 + num2 + 1; i++)
      for(j = 1; j <= 1 + num1 + num2 + 1; j++)
        capbility[i][j] = backup[i][j];
    for(i = 1; i <= num2; i++)
      capbility[1 + num1 + i][1 + num1 + num2 + 1] = mid;
    MAX = 0;
    while(BFS())
    {
      memset(current, 0, sizeof(current));
      while(add = dinic(Head ,0x7fffffff))
      MAX += add;
    }//dinic
    if(MAX == num1)
      right = mid;
    else
      left = mid;
    mid = (left + right) / 2;
  }
  printf("%d\n", right);
}
```

1.2 证明

在算法研讨课代码基础上修改。通过二分猜测最小的最大负载值，然后通过网络流 dinic 算法，验证，这个值能否将每个任务都分配到计算机内。

如图，红色是源，黄色是任务，绿色是计算机，蓝色是汇，细紫线的容量都是 1，粗紫线的容量是 mid。源与任务全部相连，任务与它对应的 2 台计算机相连，所有计算机与汇相连。通过二分猜 mid，如果 MAX FLOW 等于任务数，就是从源出发的最大数量，那么就说明这个 mid 能满足条件，令 right=mid，反之令 left=mid，继续查找，直到 left+1==right，停止，输出 right，就是最小的最大负载值。



1.3 复杂度

算法复杂度是，设有 n 个任务，二分查找是 O(logn)，dinic+ 当前弧优化，点数 O(n)，边数 O(n)，算法复杂度是 O(n^3)。

综上算法复杂度 O($n^3 logn$)

2 Matrix

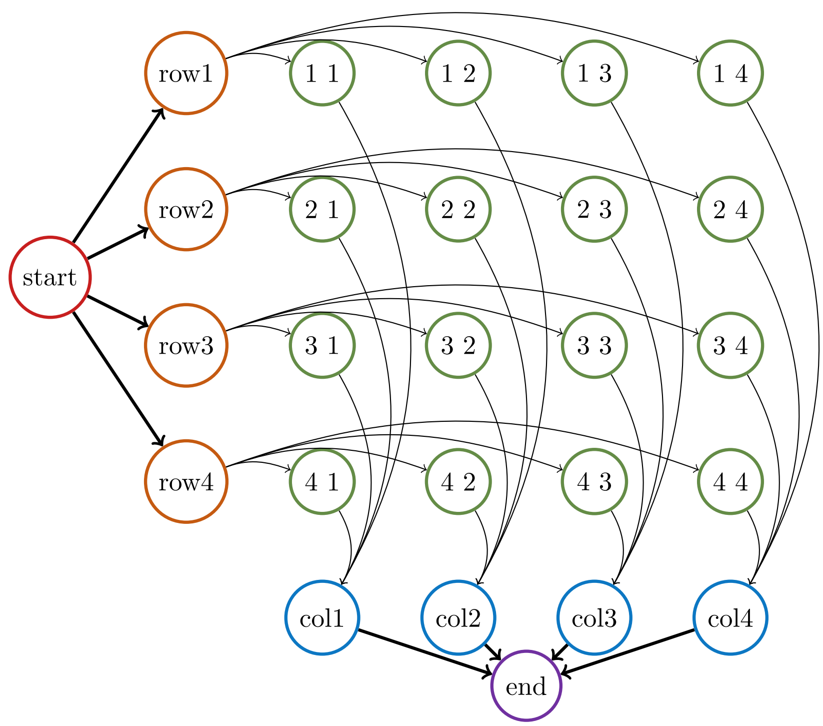
2.1 代码

Algorithm 1 Problem1

```
procedure Matrix(n)
    for i = 1 to n do
        for j = 1 to n do
            cap[1 + i][1 + j * n + i] = 1                          ▷ 与对应行容量 1
            cap[1 + j * n + i][1 + n + n * n + j] = 1              ▷ 与对应列容量 1
        end for
        cap[1][1 + i] = row[i]                                     ▷ i 行与源
        cap[1 + n + n * n + i][1 + n + n * n + n + 1] = col[i]     ▷ i 列与汇
    end for
    dinic(1, 1 + n + n * n + n + 1)
end procedure
```

2.2 证明

如图，红色是源，黄色是行，绿色是矩阵，蓝色是列，紫色是汇，细黑线容量都是 1，粗黑线容量是对应的行和、列和。源与行全相连，行与它对应的矩阵中的数相连，矩阵与相应的列相连，列全和汇相连。求解最大网络流即可。若所给数据正确，则最大网络流即行和之和，流经的点为 1，不流经的点为 0。



2.3 复杂度

算法复杂度是，设有 n 维矩阵，点数 $O(n^2)$，边数 $O(n^2)$。
综上算法复杂度 $O(n^6)$。

3 Unique Cut

3.1 代码

Algorithm 2 Problem1

```
procedure MIN-CUT(n)
    weight = Find(G,C)          ▷ dinic 算法，G 是图，C 是返回的最小割集合
    for i = 1 to m do                                        ▷ m 为 C 的边数
        G[C[i]] ++                   ▷ 将 C 中第 i 条线的 capacity 加 1
        if weight == Find(G) then
            return 0                                          ▷ 最小割不唯一
        end if
        G[C[i]] –
    end for
    return 1                                                   ▷ 最小割唯一
end procedure
```

3.2 证明

先用 dinic 算法，求得最小割的权值和，以及返回一个最小割的集合。然后对每一条变，分别加一，并算新图的最小割的权值和。

如果新的权值跟旧权值一样，那么说明有其他的最小割不涉及这一条边，所以最小割不唯一。

如果每一次新的权值都变化了，即 $+1$ 了，反证易知，没有其他的最小割，所以最小割唯一。

3.3 复杂度

算法复杂度是，设有 V 个点，E 条边。dinic 算法 $O(V^2E)$，验证循环 O(E)
综上算法复杂度 $O(V^2E^2)$。

4 Problem Reduction

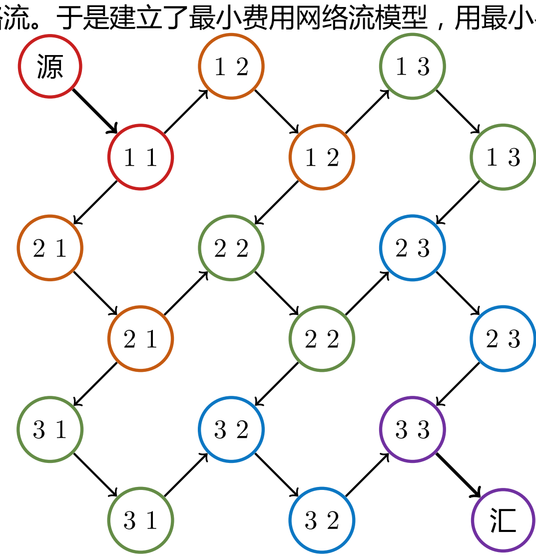
4.1 代码

Algorithm 3 Problem1

```
procedure MIN-CUT(n)
    capacity[M[1][1][1]][M[1][1][2]] = 2            ▷ M[i][j][k] 表示矩阵第 i 行 j 列
    capacity[M[n][n][1]][M[n][n][2]] = 2   ▷ k 是 1 表示进入点，2 表示离开点
    for i = 1 to n do
        for j = 1 to n do
            capacity[M[i][j][2]][M[i][j + 1][1]] = 1
            capacity[M[i][j][2]][M[i + 1][j][1]] = 1
            capacity[M[i][j][1]][M[i][j][2]] = 1
                ▷ 通过自己的进入点以及离开点，限制只经过一次，且流是 1
            weight[M[i][j][1]][M[i][j][2]] = M[i][j]                ▷ 赋值费用
        end for
    end for
    sum = MinCostFlow + M[1][1] + M[n][n]      ▷ 利用最小费用网络流算法
    return sum                                          ▷ 返回最小费用
end procedure
```

4.2 证明

如图，将每个矩阵上的点，拆成两个点，进入点和离开点，两点连线，限流 1，限制该矩阵点只被经过一次，带费用，即该点的数值，为了算最小费用。只有上方及左方的离开点才会接到该处的进入点。和源、汇相连的那条线的容量是 2，要算来回来回，回，其实可以视作是另一条来，所以此题化作流 2 的网络流。于是建立了最小费用网络流模型，用最小费用最大流算法求解即可。



4.3 复杂度

假设 n 维矩阵，那么点数 $O(n^2)$，边数是 $O(n^2)$，算法复杂度就是最小费用最大流算法的复杂度。

综上算法复杂度是 $O(n^6)$。