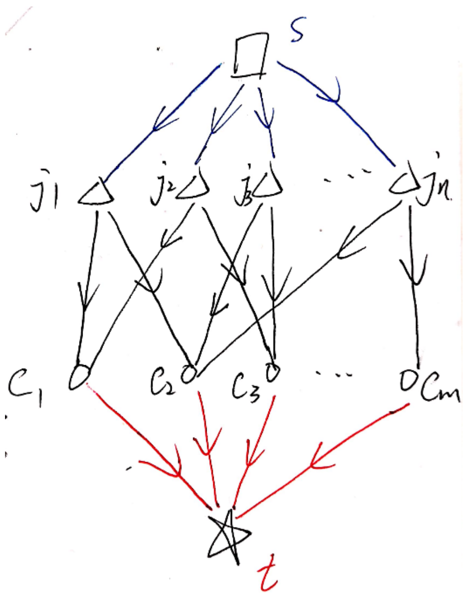
# Assignment 5

1. 假设有$n$项任务$j_1,j_2,\cdots,j_n\in J$和$m$台电脑$c_1,c_2,\cdots,c_m\in C$，我们用如下的有向图表示它们之间的关系：




如果$j_i$能分配到$c_k$上，则连一条边$j_i\rightarrow c_k$，所以每个$j_i$的出度为 2。并添加顶点$s$和$t$，从$s$到每个$j_i$连一条边，从每个$c_k$到$t$连一条边。设每条$s\rightarrow j_i$和每条$j_i\rightarrow c_k$的权重/最大 capacity 为 1。

先用求解最大流的算法求出每一个从$s$到$c_k$的最大流$f_k$，其中的最大值记为$R$，作为上界；设下界为$L$，其初值为 0，而当前的分配方案作为可行解$Ans$。进行如下的迭代：

令$M=(L+R)/2$，将每条$c_k\rightarrow t$的最大 capacity 配置为$M$，运行求解从$s$到$t$的最大流的算法，如果在该 capacity 的约束下能求解出最大流，则将$R$更新为$M$，并用当前解去更新可行解$Ans$，继续迭代；如果此时已不能求出满足条件的解，则将$L$更新为$M$，继续迭代；迭代的终止条件为$L\geq R$，此时保存的$Ans$就是最后一个可行解，即使得最大负载最小的分配方案。

伪代码如下：

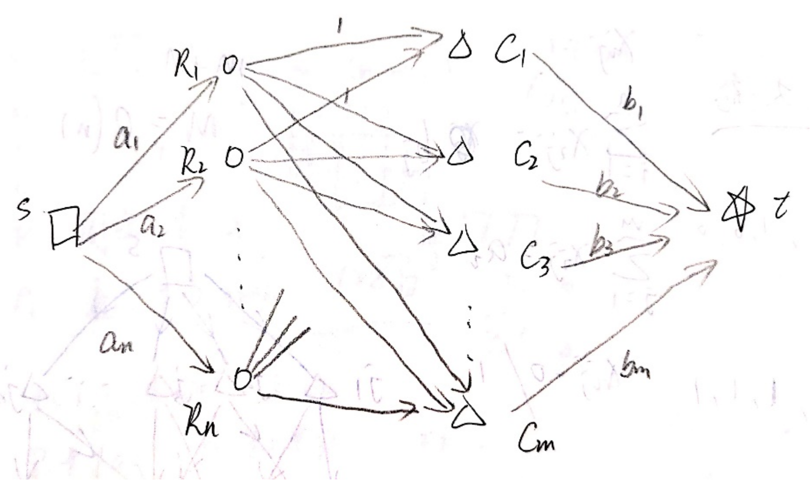
```
function load_balance(Graph G){
    Ans=max(EdmondsKarp(s,c[k],G)), in all k;
    R=f(Ans), L=0;
    while(L<R){
        M=(L+R)/2;
        for(i=1;i<=k;i++){
            C(c[i],t)=M;
        }
        Ans'=EdmondsKarp(s,t,G);
        if(f(Ans')<0){
            L=M;
        }else{
            R=M;
            Ans=Ans';
        }
    }
}
```

正确性：我们要求从$s \to j_i$的 capacity 为 1，所以对于确定的$j_i$，其实际流向的$c_k$肯定只会有一个，所以按照最大流算法求出的分配都是可行的分配。另一方面，我们上述迭代的过程相当于进行一次二分搜索，找到最大流的上界$M$的最小值，即为最大负载的最小值。

时间复杂度：初始时查找最大值，每次调用花费$O(n^3)$时间，一共调用$m$次；按照二分法，迭代次数最多为$O(\log M) = O(\log n)$，每次迭代调用最大流算法，花费$O((m+n)^3)$时间，于是算法的复杂度为$O(mn^3 + (m+n)^3 \log n)$

空间复杂度：存储图 G，如果采用链表存储，花费$O(m+n)$的空间

2. 考虑一个有$n$行、$m$列的矩阵$X$，记第$i$行、第$j$列的元素为$x_{ij}$，假设第$i$行所有元素之和为$\sum_{j=1}^{m} x_{ij} = a_i, 1 \le i \le n$，第$j$列所有元素之和为$\sum_{i=1}^{n} x_{ij} = b_j, 1 \le j \le m$，构造如下的有向图$G$：




每个顶点$R_i, 1 \le i \le n$向每个$C_j, 1 \le j \le m$连一条有向边，流量限制为 1，添加顶点$s$，对$\forall i\ (1 \le i \le n)$，连有向边$s \to R_i$，其流量限制为$a_i$；添加顶点$t$，对$\forall j\ (1 \le j \le m)$，连有向边$C_j \to t$，其流量限制为$b_j$。

然后在上述限制下，求解从$s$到$t$的最大流，如果能求出，验证解中每个$R_i$的实际流量是否都是$a_i$，每个$C_j$的实际流量是否都是$b_j$，如果能求出，且验证通过，则从该解能得到矩阵$M$，即当$R_i \to C_j$的实际流量为 1 时，在矩阵$M$中$x_{ij} = 1$，否则$x_{ij} = 0$。

伪代码如下：

```
function matrix_search(Graph G){
    (f,Ans)=FordFulkerson(s,t,G);
    if(f<0){
        return NULL;
    }
    for(i=1;i<=n;i++){
        for(j=1;j<=m;j++){
            matrix[i][j]=capacity(R[i],C[j]);
        }
    }
    check(R[i]==a[i]);
    check(C[j]==b[j]);
    return matrix;
}
```

正确性：按上述算法求出的解中，假设从$R_i$到$C_j$的实际流量为$y_{ij}$（$y_{ij}=0,1$），由于每个$R_i$和每个$C_j$都是中间节点，且最后的验证通过，于是有$a_i=\sum_{j=1}^{m}y_{ij}, b_j=\sum_{i=1}^{n}y_{ij}$，求出的$y_{ij}$即对应了矩阵$M$的元素$x_{ij}$

时间复杂度：调用 Ford-Fulkerson 算法，复杂度为$O\big((mn+m+n)\cdot(a_1+a_2+\cdots+a_n)\big)=O(m^2n^2)$

空间复杂度：存储矩阵$M$，花费$O(mn)$空间

3. 设矩阵$A=(a_{ij})_{m\times n}$。构造图$G$，其前$mn$个节点为$b_{ij}$（$1\le i\le m, 1\le j\le n$），记$s=b_{11}, t=b_{mn}$，为每个$b_{ij}$中，非$s$和$t$的节点添加一个新的节点$c_{ij}$（$1\le i\le m, i\le j\le n, 1<i\cdot j<mn$）。连边$s\to b_{12}, s\to b_{21}, b_{m-1,n}\to t, b_{m,n-1}\to t, b_{ij}\to c_{ij}, c_{ij}\to b_{i,j+1}, c_{ij}\to b_{i+1,j}$，设任意一条流向$b_{ij}$的边的费用为$a_{ij}$，任意一条流向$c_{ij}$的边的费用为 0，而每条边的流量限制都是 1。我们将流量的目标设为 2，对图$G$运行最小费用流算法（Practical Problem3 in Lec8），可以求解出从$s$到$t$的最小费用，该费用同时也是途径原来矩阵$A$的最小费用。

伪代码如下：

```
function matrix_cost(Matrix M){
    construct G by M;
    min=Minimum_Cost_Flow(G,s,t,2);
    return min;
}
```

正确性：在矩阵$A$中，我们要求出一条从$s$到$t$的只往右/下走的路径，和一条从$t$到$s$的只往左/上走的路径，使得两条路径不相交，且总的花费最小。相当于求两条从$s$到$t$的只往右/下走的不相交路径，使得总的花费最小。在上述算法中，由于流量目标为 2，所以最终有且仅有两条路径从$s$出发，最终到达$t$，同时，由于每条边的流量限制为 1，所以每个顶点至多经过一次，于是两条路径不相交。另一方面，从顶点$b_{ij}$到$b_{i,j+1}$或是$b_{i+1,j}$，中间都需要经过$c_{ij}$，两条路径只有一条能被选中，且其花费恰好是$a_{i,j+1}$或$a_{i+1,j}$，于是上述算法得到的花费和原问题的花费是一致的。

时间复杂度：和最大流相同，为$O(m^2n)$

空间复杂度：存储矩阵$A$，占用$O(mn)$空间

4. 改进 Ford-Fulkerson 算法，其中，选取从$s$到$t$的路径时不是任意选取，而是按照 Bellman-Ford 算法，选取费用最小的路径，最终得到的可行解能够保证最大流，同时是所有保证最大流的可行解中费用最小的解（反向边的费用是正向边的相反数）。

伪代码如下：

```
function Revised_FF(){
    initialize f(e)=0 for all e;
    while there is an s-t path in Gf do{
        choose an s-t path p in Gf by BellmanFord;
        f=AUGMENT(p,f);
    }
    return f;
}
```

正确性：首先，Ford-Fulkerson 算法中可以任意选路径，所以我们求出的可行解一定能保证最大流。另一方面，如果还有另一个可行解$Ans_1$也能够保证最大流，且不是按照上述算法得到的，那么我们可以得到它的

剩余图$H_f$，且$H_f$和上述算法中的剩余图$G_f$不同。我们按照 Ford-Fulkerson 算法一步一步选取，总能够“模拟出”$H_f$，所以不妨认为这另一个可行解就是用 Ford-Fulkerson 算法得到的。$Ans_1$在选取 bottleneck 的路径时，总有一些步，选取的路径不是当前开销最小的路径，考虑其中的倒数第一步，此时如果我们换为选取当前开销最小的路径，则最终得到的解$Ans_2$的质量不会比$Ans_1$差，同理，对$Ans_2$也做上述调整，最终能逐步调整至上述算法得到的解$Ans$，于是为最优的。

时间复杂度：Bellman-Ford 算法的复杂度为$O(m \cdot n)$，最多循环$O(m \cdot n)$次，于是该算法花费$O(m^2n^2)$时间

空间复杂度：使用链表存储图，占用$O(m+n)$空间

5. 我们根据矩阵$M$的元素来构造图$G$，首先添加顶点$g_{11}, g_{12}, \cdots, g_{1m}, g_{21}, g_{22}, \cdots, g_{2m}, \cdots, g_{n1}, g_{n2}, \cdots, g_{nm}$，并添加顶点$s$和$t$对于矩阵$M$中的每一对相邻元素$M_{i,j}$和$M_{x,y}$（其中$(x,y)=(i-1,j),(i,j-1),(i,j+1),(i+1,j)$），对应顶点为$g_{i,j}$和$g_{x,y}$，添加顶点$a_{i,j,x,y}$，连边$s \to a_{i,j,x,y}$，$a_{i,j,x,y} \to g_{i,j}$，$g_{i,j} \to t$，$a_{i,j,x,y} \to g_{x,y}$，$g_{x,y} \to t$，其负载限制分别为 2, 1, 1, 1, 1，花费分别为$0, M_{i,j}, 0, M_{x,y}, 0$，为每一对相邻元素配置好以后（如果边已经存在，则不连），我们沿着 Ford-Fulkerson 算法的思路来求解。我们的流量目标是，在保证每一条$s \to a_{i,j,x,y}$边上实际流量至少为 1（即所有$s \to a_{i,j,x,y}$至少被选中一次）的前提下，使得花费最低。一共有$m \cdot n$条从$s \to t$的路径，我们从中选取路径时，按照$a_{i,j,x,y}$下面连接点的花费递增的顺序来遍历所有$s \to a_{i,j,x,y}$，从$s \to a_{i,j,x,y}$出发，如果$a_{i,j,x,y}$下面连接的两个点都还没有被选中，则选择其中花费最小的一条路径，添加到解中，如果有点已经被选中过，则检查下一个$a_{i,j,x,y}$，直至遍历完，得到满足条件的解。

伪代码如下：

```
function ChooseNumbers(Matrix M){
    construct Graph G by M;
    sort g[][] by cost increasingly, update their a[];
    for(a[i] in a[]){
        if(isVisited(a[i].left)==true or isVisited(a[i].right)==true){
            add path along a[i].left or a[i].right to Ans;(the true branch)
            continue;
        }
        if(a[i].left.cost>a[i].right.cost){
            add path along a[i].right to Ans;
        }else{
            add path along a[i].left to Ans;
        }
    }
    return Ans;
}
```

时间复杂度：一共有$O(m \cdot n)$个顶点，所以花费$O(m \cdot n)$的时间

空间复杂度：存储矩阵，占用$O(m \cdot n)$的空间，图中有$O(m \cdot n)$个顶点和边，占用$O(m \cdot n)$的空间

6. 根据输入的图$G$来构造图$H$，首先，在图$G$中的每个顶点都复制一份到图$H$中，添加顶点$s$和$t$，对于图$G$中的每一对顶点$u$和$v$，连接$s \to u$，其负载限制等于图$G$中顶点$u$的权重，连接$s \to v$，其负载限制等于图$G$中顶点$v$的权重，并在图$H$中，为$e=<u,v>$添加一个顶点，连接$u \to e$和$v \to e$，负载限制都为无穷大，即可以沿这两条边运送无限量的货物，并连接$e \to t$，其负载限制等于图$G$中边$e$的权重。然后在图$H$中运行 Edmonds-Karp 算法，求出最大流，则图$G$的最大权重等于$G$中所有边的权重之和减去该最大流。

伪代码如下：

```
function MaxWghtSubgraph(Graph G){
    construct Graph H by G;
    (f,Ans)=EdmondsKarp(G,s,t);
    sum=0;
    for(e in G.edge){
        sum+=e.weight;
    }
    return sum-f;
}
```

正确性：求图$H$的最大流相当于求最小割，即找一个划分$(S,\bar{S})$，使得$S$流出的货物最少，最小割$C=\Sigma$被割的$s\to u$上的负载限制+$\Sigma$被割的$e\to t$上的负载限制（这里的$s$对应原来$G$中的顶点，$e$对应原来$G$中的边），如果$s\to u$被割，则我们将$u$添加到子图中，如果$e\to t$被割，则不将$e$添加到子图中，同时，由于我们将$u\to e$和$v\to e$的负载限制都设为无穷大，所以当$G$的一条边被选中时，它的两个顶点也一定被选中。于是sum $-C=\Sigma$所有$e\to t$上的负载限制$-C=\Sigma$所有$e\to t$上的负载限制$-\Sigma$不选在子图中的边$e$对应的$e\to t$上的负载限制$-\Sigma$选在子图中的顶点$u$对应的$s\to u$上的负载限制$=\Sigma$选在子图中的边$e$对应的$e\to t$上的负载限制$-\Sigma$选在子图中的顶点$u$对应的$s\to u$上的负载限制=子图的权重，由于 sum 为定值，所以当割值C最小时，子图的权重最大，于是上述算法求出的确实是子图的最大权重。

时间复杂度：构造图$H$花费$O(m+n)=O(m)$的时间，在图$H$上运行 Edmonds-Karp 算法花费$O((m+n)^3)$的时间，于是上述算法的复杂度为$O(m^3)$

空间复杂度：存储图$H$，占用$O(m+n)$的空间