网络流

什么是网络流?

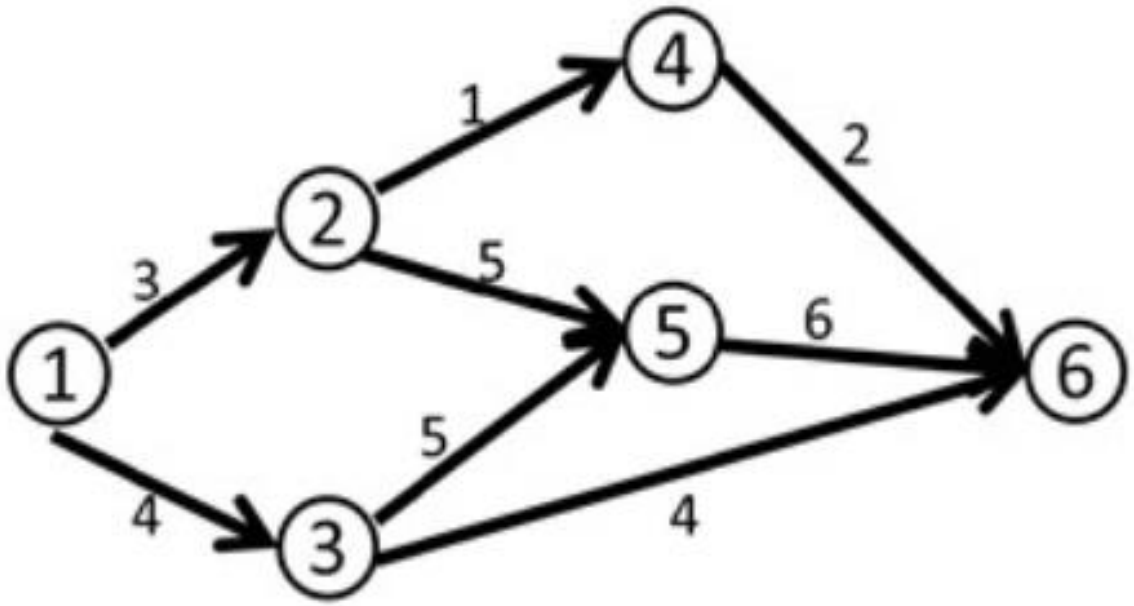
所谓网络或容量网络指的是一个连通的赋权有向图 D= (V、E、C)，其中V 是该图的顶点集，E是有向边(即弧)集，C是弧上的容量。此外顶点集中包括一个起点和一个终点。网络上的流就是由起点流向终点的可行流，这是定义在网络上的非负函数，它一方面受到容量的限制，另一方面除去起点和终点以外，在所有中途点要求保持流入量和流出量是平衡的(百度百科)

简单地说:

在一个有向图上选择一个源点，一个汇点，每一条边上都有一个流量上限（以下称为容量），即经过这条边的流量不能超过这个上界，同时，除源点和汇点外，所有点的入流和出流都相等，而源点只有流出的流，汇点只有汇入的流。这样的图叫做网络流

最大流

有一个工厂生产了一批货物，要运到很远外的一个仓库中，途中有n个城市，工厂与某些城市，城市与城市，某些城市与仓库均有道路相连，每条道路都有一个限制该种货物运输的最大承载量，问最终最多能有多少货物能运到仓库中？



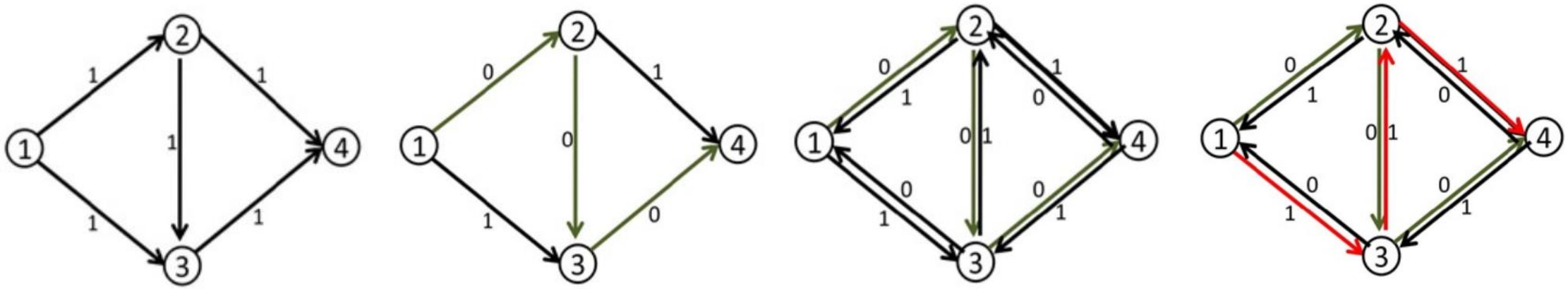
为了方便之后的解释 我们规范一些叫法

工厂 == 源点
仓库 == 汇点
最大承载量 == 容量
已经从这条路上运过的货物 == 流量
还能运多少货物 == 残量

什么是增广路?

找到一条从源点到汇点的路径，使得路径上任意一条边的残量>0（注意是小于而不是小于等于，这意味着这条边还可以分配流量），这条路径便称为增广路

~~我们不断的找增广路，更新残量，直到找不出增广路，此时我们就找到了最大流~~



1.找到一条增广路

2.找到这条路径上最小的残量记为flow

3.将这条路径上的每一条有向边u->v的残量减去flow，同时对于起反向边v->u的残量加上flow (后悔的机会)

4.重复上述过程，直到找不出增广路，此时我们就找到了最大流

增广路定理(Augmenting Path Theorem): 网络达到最大流当且仅当残留网络中没有增广路

用层次图的增广的最大流算法（Dinic）

从宏观来讲，该算法就是不停地用BFS构造层次图，然后用阻塞流来增广。

“层次图”和“阻塞流”是Dinic算法的关键字。

什么叫层次图呢？

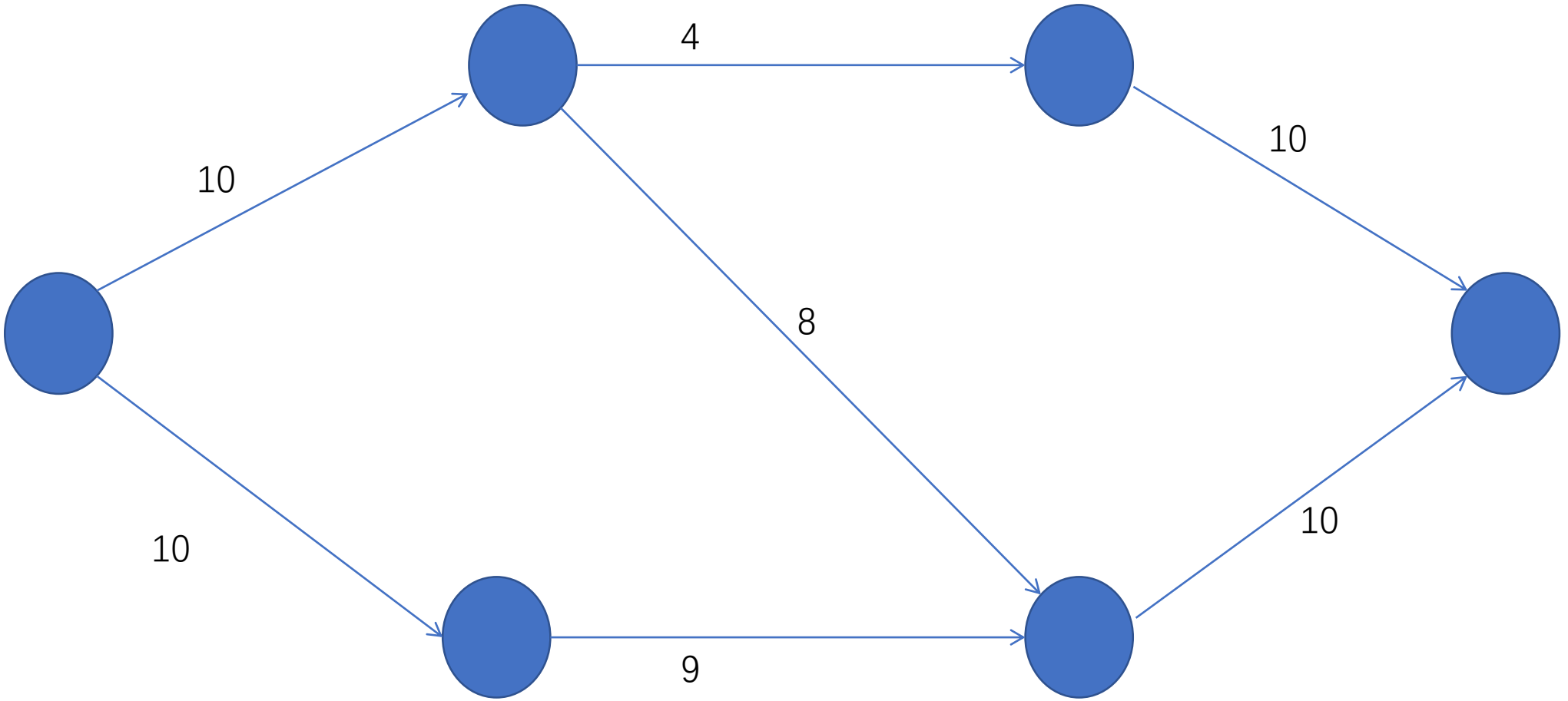
在**残量网络**中，起点到结点U的距离为dist（u），我们把dist（u）看做结点U的“层次”。只保留每个点出发到下一层次的弧（dist（v）=dist（u）+1），得到的图就是层次图。

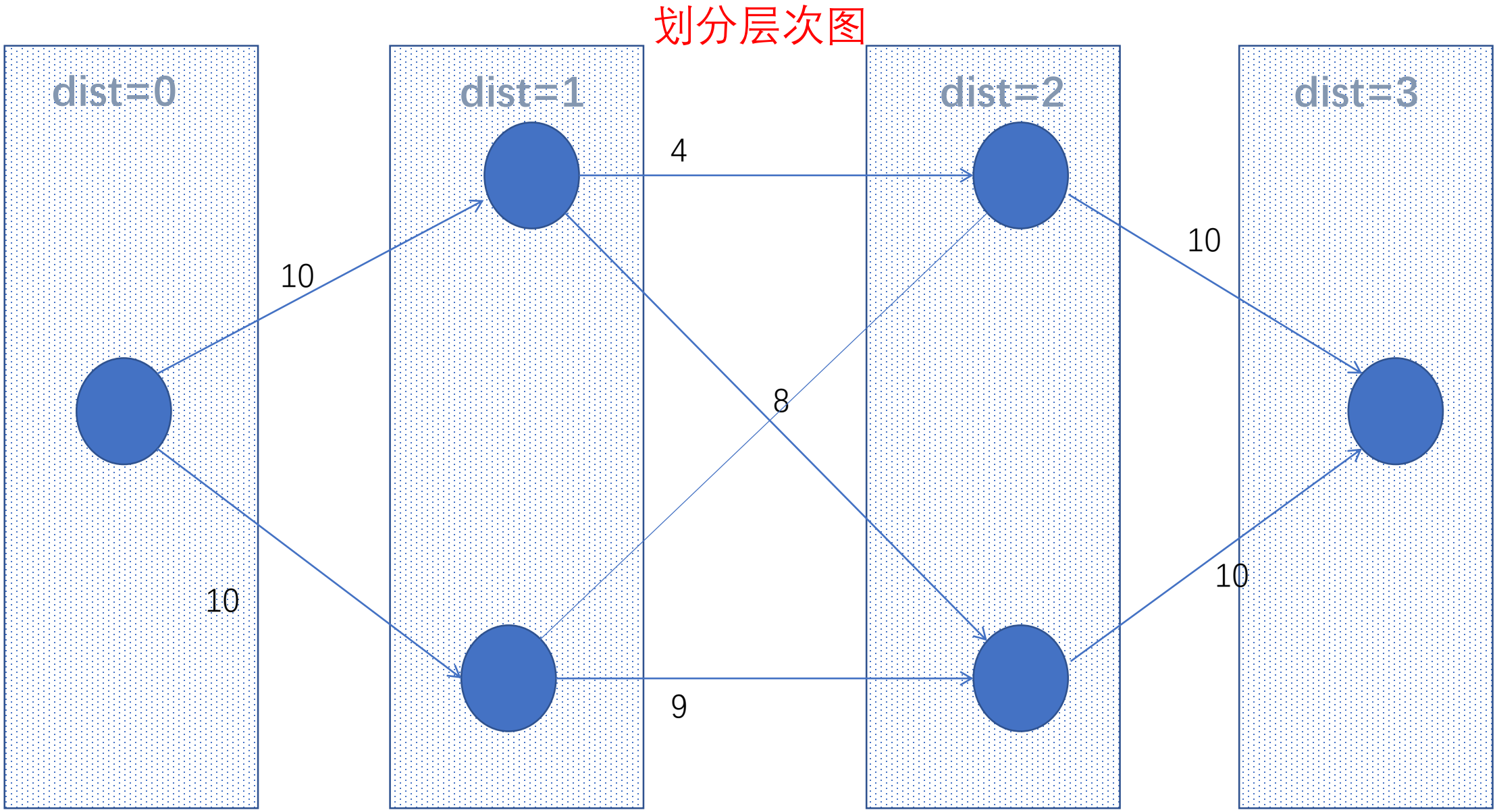
于是我们从起点出发每一次的路径一定都是：
起点->层次1->层次2->……->终点。

不难发现每一条这样的路径都是S->T的最短路

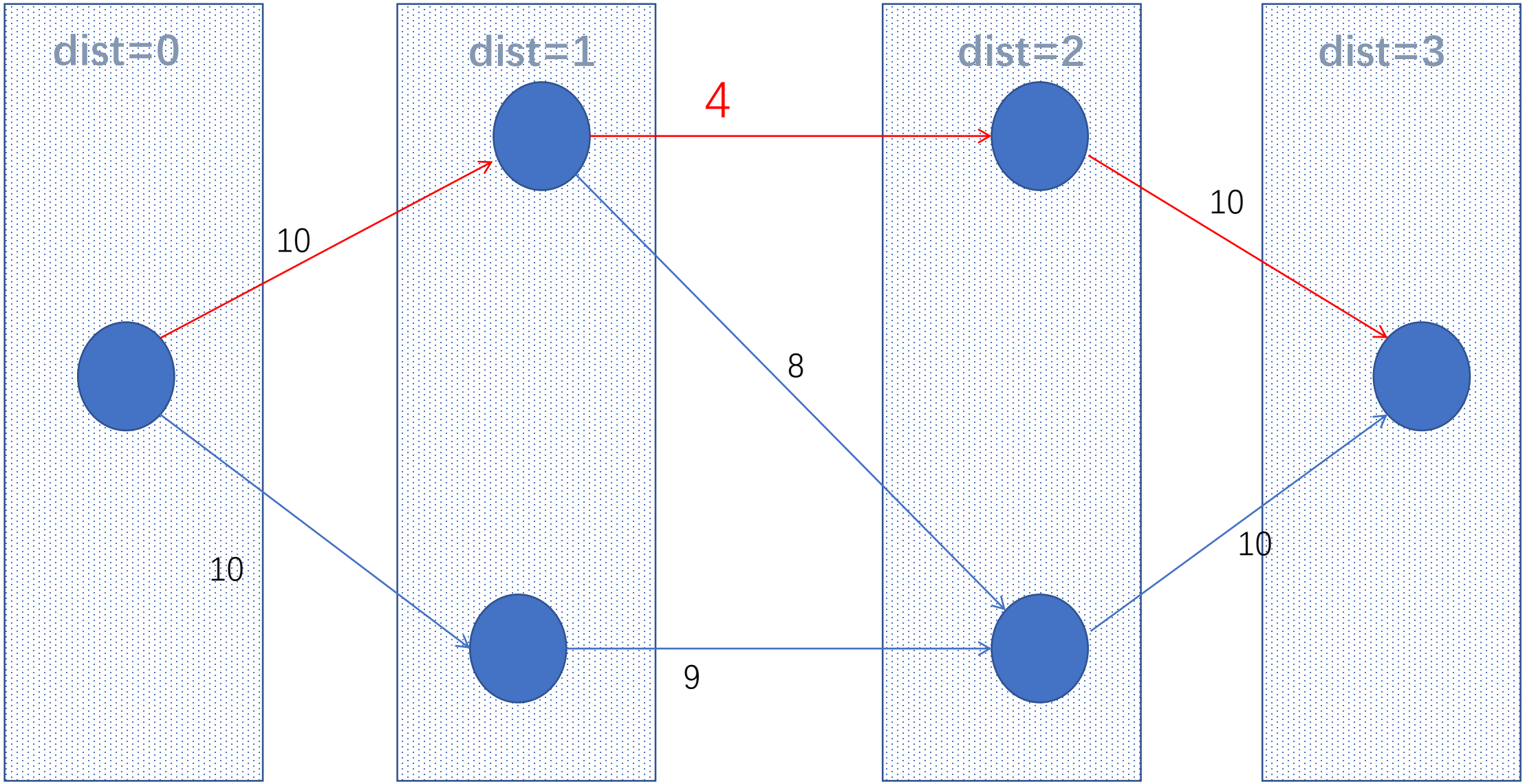
什么叫阻塞流呢？

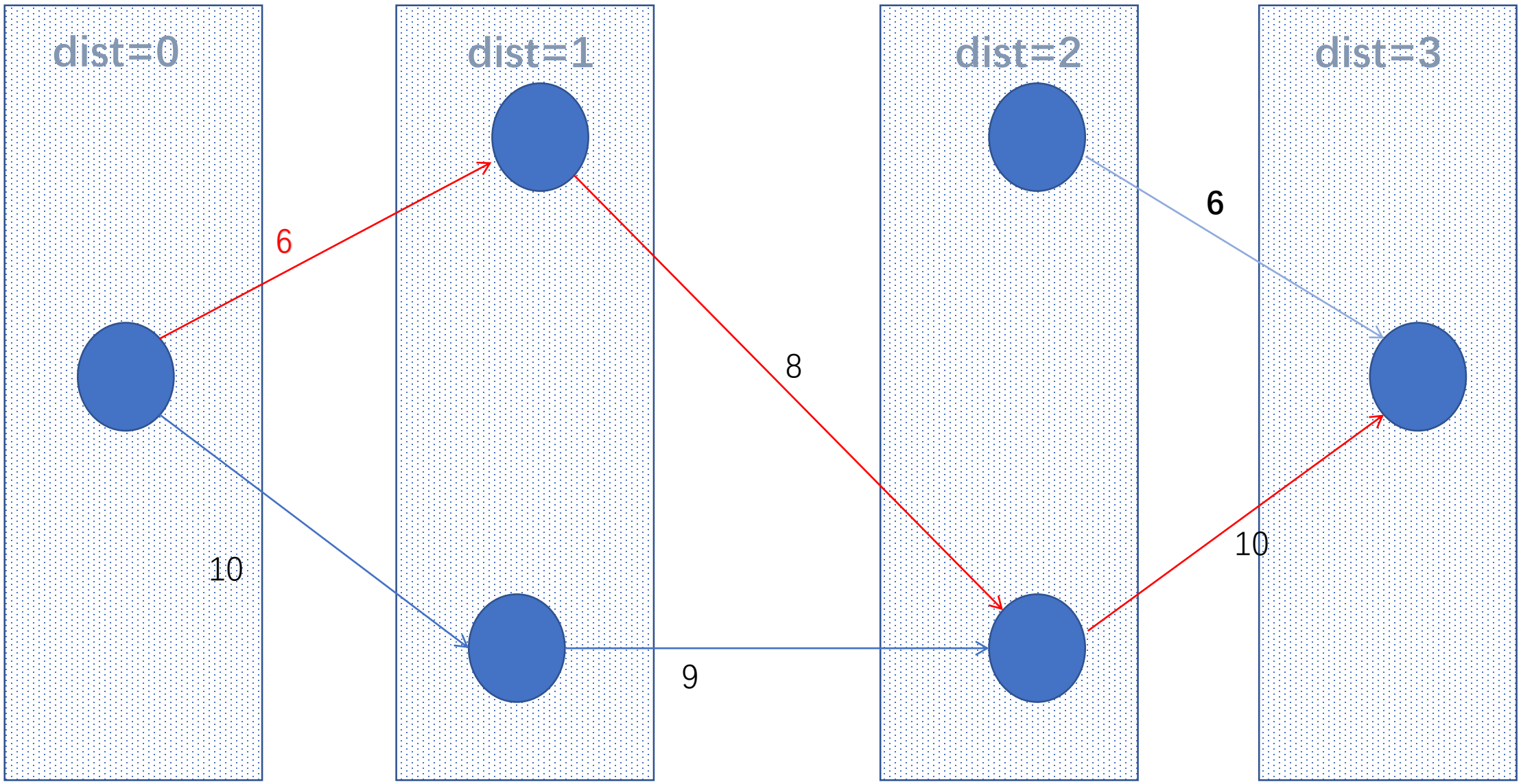
用文字描述就是不考虑反向弧时的“极大流”。只看文字很难看懂，但是配合图片就变得很好理解了。

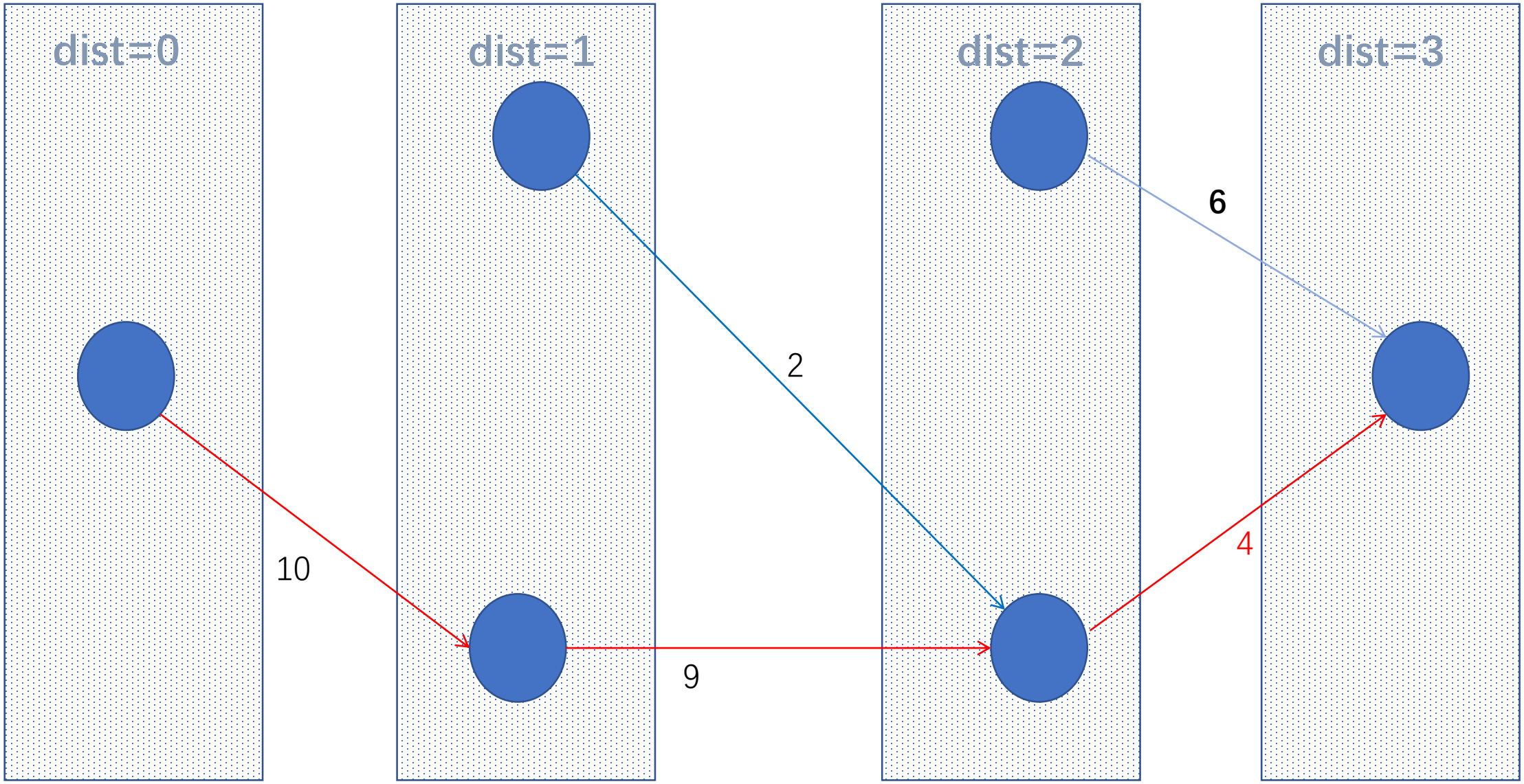

4
10
10
8
10
10
9


划分层次图
dist=0
dist=1
dist=2
dist=3
10
10
4
8
9
10
10

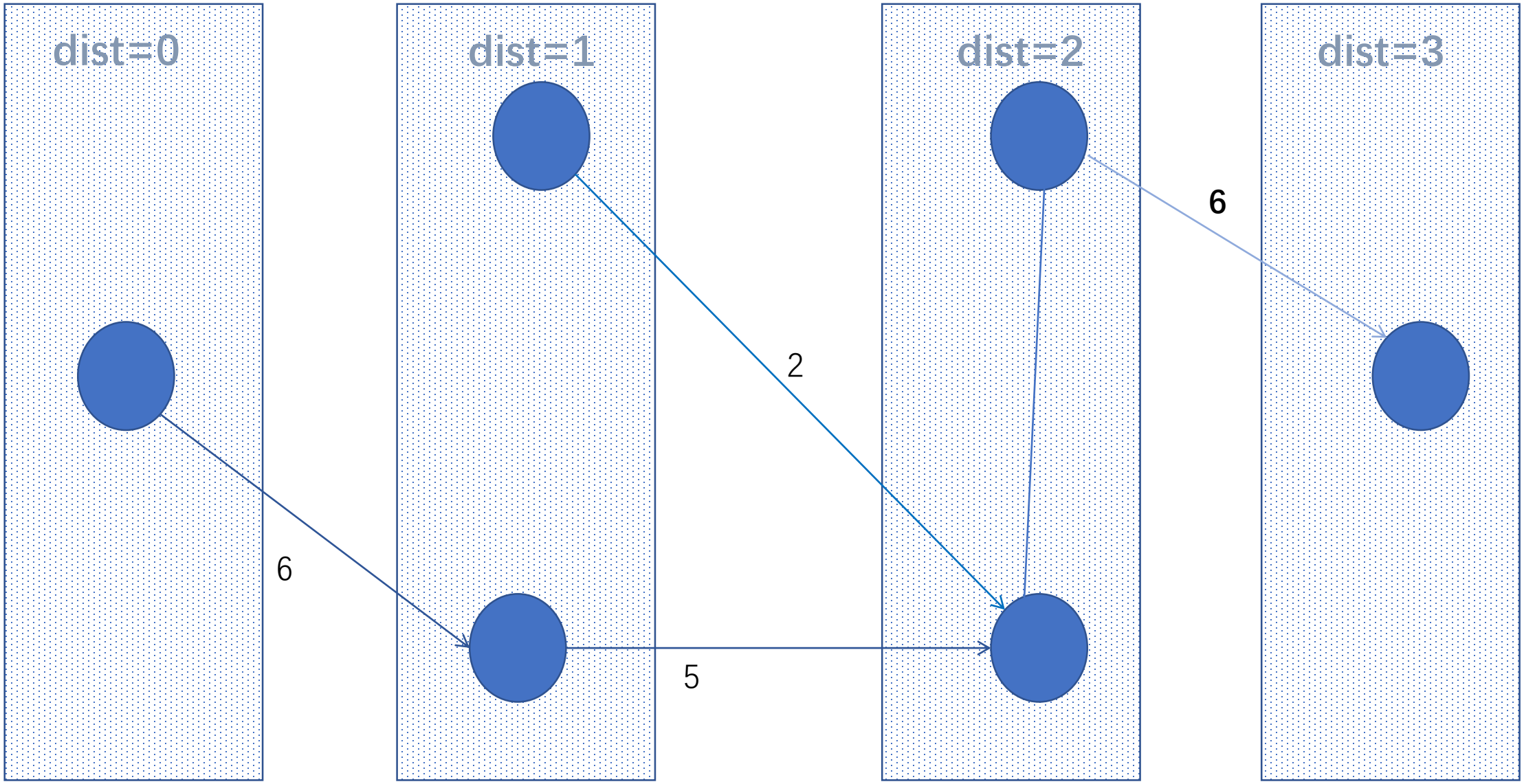
找到第一条阻塞流




第二次增广
dist=0
dist=1
dist=2
dist=3
6
10
8
9
6
10


第三次增广
dist=0
dist=1
dist=2
dist=3
10
2
6
9
4

完成（发现s-t不再联通）



```cpp
struct Edge{
    int from,to,cap,flow;
};

vector<Edge>edges;

inline void addedge(int from,int to,int cap){
    edges.push_back((Edge){from,to,cap,0});
    edges.push_back((Edge){to,from,0,0});
    int m = edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
}
```

边数组

划分层次图代码：

```
inline bool BFS(){
    memset(vis,0,sizeof(vis));
    queue<int>Q;
    Q.push(s);
    d[s] = 0;
    vis[s] = true;
    while(!Q.empty()){
        int x = Q.front();Q.pop();
        for(int i = 0;i < G[x].size();i++){
            Edge & e = edges[G[x][i]];
            if(!vis[e.to] && e.cap > e.flow){
                vis[e.to] = true;
                d[e.to] = d[x] + 1;
                Q.push(e.to);
            }
        }
    }
    return vis[t];
}
```

阻塞流增广代码:

```
15 int DFS(int x,int a){
16     if(x == t || a == 0) return a;
17     int flow = 0,f;
18     for(int& i = cur[x];i < G[x].size();i++){
19         Edge& e = edges[G[x][i]];
20         if(d[x] + 1 == d[e.to] && (f = DFS(e.to,min(a,e.cap-e.flow))) > 0){
21             e.flow += f;
22             edge[G[x][i]^1].flow -= f;
23             flow += f;
24             a -= f;
25             if(a == 0) break;
26         }
27     }
28     return flow;
29 }
```

主过程代码:

```
15  inline int Maxflow(int s,int t){
16      int flow = 0;
17      while(BFS()){
18          memset(cur,0,sizeof(cur));
19          flow += DFS(s,INF);
20      }
21      return flow;
```