Листинг 1.

```
BeginPackage["FlowSolver`"]
```

```
readGraph[file_,dir_]:=Module[{
    fn=FileNameJoin[{dir,file}],
    stream,imod,umod,u,b
    },
    stream=OpenRead[fn];
    imod=Read[stream,{Word,Number}][[2]];
    umod=Read[stream,{Word,Number}][[2]];
    u=#[[1]]•→#[[2]]&/@ReadList[stream,Expression,umod];
    b=ConstantArray[0,imod];
    (b[[Read[StringToStream[StringTake[#1,{5,-3}]],Number]]]=#2)&
    @@@ReadList[stream,{Word,Number},imod];
{Graph[u,VertexSize->Medium, VertexLabels->{x_:>Placed[{x,Style[{{
 {{
   {-b[[x]]},
   {"↑"}
  }, b[[x]]<0},
 {{
   {b[[x]]},
   {"↓"}
  }, b[[x]]>0},
 {"", True}
}//TableForm ,Medium]},{Center,Above}]},
VertexLabelStyle->Directive[Black,Italic,24],
EdgeShapeFunction-> GraphElementData[{"Arrow"}],
GraphLayout->"CircularEmbedding"],b}
]
```

```
buildTree::usage="build a tree based on graph g with a specified root"

getDepth1[x_,depth_]:={{
 {depth[[x]], x>0},
 {-1, True}
}
```

```
edge::usage="ret edge from i to j with specified direction"
edge[i_,j_,dir_]:={{
 {i•→j, dir<0},
 {j•→i, dir>0},
 {i•-•j, dir==0}
}
```

```
buildTree[g_?GraphQ,root_]:=Module[{
ng,
    rt={},
    tmp,
    pred=ConstantArray[0,VertexCount[g]+1],
    dir=ConstantArray[0,VertexCount[g]+1],
    depth=ConstantArray[0,VertexCount[g]+1],
    d=ConstantArray[VertexCount[g]+1,VertexCount[g]+1],
    curD=0,lastVis=0,edgeN=ConstantArray[0,VertexCount[g]+1],
    lroot=VertexCount[g]+1
    },
    ng=EdgeAdd[VertexAdd[g,lroot],(lroot•↔#)&/@root];
    BreadthFirstScan[UndirectedGraph[ng],lroot,{
    "PrevisitVertex"-> ((If[#==0,,depth[[#]]=getDepth1[pred[[#]],depth]+1;])&),
    "FrontierEdge"-> ((pred[[Last[#]]]=First[#];
    dir[[Last[#]]]={{
 {1, MemberQ[EdgeList[ng],First[#]•↔Last[#]]},
 {-1, True}
};
tmp=Position[EdgeList[ng],
edge[First[#],Last[#],-dir[[Last[#]]]]];
edgeN[[Last[#]]]=If[Length[tmp]==0,0,tmp[[1,1]]];
AppendTo[rt,First[#]•↔Last[#]];)&)}];
DepthFirstScan[UndirectedGraph[Graph[rt]],lroot,{"PrevisitVertex"->( (If[lastVis==0,1
    {{pred,dir,depth,d,lroot,edgeN,Graph[rt,VertexSize->Medium, VertexLabels->Placed[
    (*{pred,dir,depth,d,root,edgeN}*)
]
```

```
treeQ::usage="tests if the structure t is a tree"
treeQ[t_List]:=Length[t]==7&&Length[t[[1]]]==Length[t[[2]]]&&Length[t[[1]]]==Length
```

```
pred[t_]:=t[[1]]
dir[t_]:=t[[2]]
depth[t_]:=t[[3]]
getDepth[v_,t_?treeQ]:=getDepth1[v,depth[t]]
d[t_]:=t[[4]]
root[t_]:=t[[5]]
reverceD[t_]:=Module[{
rd=ConstantArray[0,t//dir//Length]
},rd[[t//root]]=NestWhile[(rd[[d[t][[#]]]]=#;d[t][[#]])&,
t//root,(d[t][[#]]!=(t//root))&];rd]
tableForm[t_]:=TableForm[t[[1;;4]],
TableHeadings->{{"pred","dir","depth","d"},t//pred//Length//Range}]
uNb[g_?GraphQ,t_]:=EdgeList[g,τ_•↔ρ_/;pred[t][[τ]]!=ρ&&pred[t][[ρ]]!=τ]
```

```
path[t_,v_]:=NestList[pred[t][[#]]&,v,getDepth[v,t]]
alignDepth=Compile[{{vert1,_Integer},
{vert2,_Integer},{pred,_Integer,1},{depth,_Integer,1}},
NestWhile[pred[[#]]&,vert1,getDepth1[#,depth]>getDepth1[vert2,depth]&]
];
```

```
lcmHelper=Compile[{{vert1,_Integer},{vert2,_Integer},{pred,_Integer,1},{depth,_Intege
Module[{v1=vert1,v2=vert2},
v1=alignDepth[v1,v2,pred,depth];
v2=alignDepth[v2,v1,pred,depth];

NestWhile[{pred[[#[[1]]]],pred[[#[[2]]]]}&,List[v1,v2],#[[1]]!=#[[2]]&][[1]]
],{{alignDepth[_,_,_,_],_Integer}}
];
lcm[t_,vert1_,vert2_]:=lcmHelper[vert1,vert2,pred[t],depth[t]]
pathLen[t_,v1_,v2_]:=getDepth[v1,t]+getDepth[v2,t]-2*getDepth[lcm[v1,v2],t]
subTree[t_,v_]:=NestWhileList[d[t][[#]]&,v,getDepth[d[[#]],t]>getDepth[v,t]&]
getLeafs[t_]:=Cases[Range[pred[t]//Length],x_/;getDepth[x,t]>=getDepth[d[t][[x]],t]
```

```
partSolve::usage=
"ret list of rulles with part solve for graph g
with outside flow b and base tree, left part symbols are x"
partSolve[g_?GraphQ,b_List,tree_,x_]:=Module[{xed,t=tree,rd,last=0},
rd=reverceD[t];
xed=ConstantArray[0,g//VertexCount];
last=NestWhile[(xed[[#]]+=-dir[t][[#]]×b[[#]];
xed[[pred[t][[#]]]]+=dir[t][[pred[t][[#]]]]×dir[t][[#]]×xed[[#]];
rd[[#]])&,rd[[t//root]],(pred[t][[#]]!=(t//root))&];
(xed[[#]]+=-dir[t][[#]]×b[[#]])&[last];

((Subscript[x, #]-> 0)&/@(uNb[g,t]))∪((Subscript[x, edge[#,pred[t][[#]],
dir[t][[#]]]]->xed[[#]])&/@((g//VertexList)~Complement~{t//root}))]
partSolve[g_?GraphQ,b_List,rootV_,x_]:=partSolve[g,b,buildTree[g,rootV],x]
```

```
Subscript[δ, τ_↦ρ_][g_?GraphQ,t_]:=Module[
{λ=lcm[t,τ,ρ],δ=ConstantArray[0,g//VertexCount]},
NestWhile[(δ[[#]]=dir[t][[#]];pred[t][[#]])&,τ,#!= λ&];
NestWhile[(δ[[#]]=-dir[t][[#]];pred[t][[#]])&,ρ,#!= λ&];
((Subscript[x, edge[#,pred[t][[#]],dir[t][[#]]]]->δ[[#]])&
/@(g//VertexList))~Join~{Subscript[x, τ↦ρ]-> 1,Subscript[x, _]->0}
]
```

```
δ2h=Compile[{{l,_Integer,1},{pr,_Integer,1},{dep,_Integer,1},
{direct,_Integer,1},{nums,_Integer,1},{n,_Integer}},
Module[{λ,δ=SparseArray[{},n],τ=l[[1]],ρ=l[[2]],j=l[[3]]},
λ=lcmHelper[τ,ρ,pr,dep];
NestWhile[(δ[[nums[[#]]]]=direct[[#]];pr[[#]])&,τ,#!= λ&];
NestWhile[(δ[[nums[[#]]]]=-direct[[#]];pr[[#]])&,ρ,#!= λ&];
δ[[j]]=1;
δ],Parallelization->False,  RuntimeAttributes -> {Listable}
]
```

```
δ2[{τ_,ρ_,j_},t_,n_]:=Module[{λ,δ=SparseArray[{},n]},
λ=lcm[t,τ,ρ];
NestWhile[(δ[[t[[6,#]]]]=dir[t][[#]];pred[t][[#]])&,τ,#!= λ&];
NestWhile[(δ[[t[[6,#]]]]=-dir[t][[#]];pred[t][[#]])&,ρ,#!= λ&];
δ[[j]]=1;
δ]
```

```
alignJ=Compile[{{j,_Integer},{l,_Integer,2},{ed,_Integer,1}},
NestWhile[(#+1)&,j,(l[[#]]!=ed)&]];
δ1[g_?GraphQ,t_]:=Module[
{λ,unb=uNb[g,t],unb1,tmp,τ,ρ,ed,j=1},

tmp={#[[1]],#[[2]]}&/@EdgeList[g];
unb1=Map[(ed=#;j=alignJ[j,tmp,{ed[[1]],ed[[2]]}];{#[[1]],#[[2]],j})&,unb];

δ2[#,t,g//EdgeCount]&/@unb1//SparseArray]
```

```
eqSystem[g_?GraphQ]:=Fold[ReplacePart[#1,{(#2//First)->#1[[#2//First]]-
Subscript[x, #2],(#2//Last)->#1[[#2//Last]]+Subscript[x, #2]}]&,
ConstantArray[0,g//VertexCount],g//EdgeList]
```

```
solveAll[g_?GraphQ,t_]:=Module[{xs=Subscript[x, #]&/@(g//EdgeList)},
    Cases[(MapThread[#1->#2&,{xs,Parallelize[ParallelMap[Subscript[x, #]&,
    uNb[g,t]].δMatr]+(ParallelMap[Subscript[x̃, #]&,(g//EdgeList)]/.ps)}]),
    Except[x_->x_]]]
```

```
setPred[t_,i_,val_]:=ReplacePart[t,{1,i}->val];
setDir[t_,i_,val_]:=ReplacePart[t,{2,i}->val];
setDepth[t_,i_,val_]:=ReplacePart[t,{3,i}->val];
setD[t_,i_,val_]:=ReplacePart[t,{4,i}->val];
```

```
EndPackage[]
```