Algolab: quick ref

Table of contents

1 misc

- keyboard, capslock

```
setxkbmap us
setxkbmap -option ctrl:nocaps


geany keybinding:
complete word, delete cur line, toggle line comment, find next/prev selection
```

- basrc-cgal...

```
gencgal_cmake_eclipse(){#~/.bashrc
  cgal_create_cmake_script
  echo 'set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")' >> CMakeLists.txt
  cmake -G "Eclipse CDT4 - Unix Makefiles" . }
```

- io config

```
ios_base::sync_with_stdio(false);
std::cout << std::setiosflags(std::ios::fixed);
```

```
std::cout << std::setprecision(0);
```

- rounding up/down

```
int round_up(const CGAL::Quotient<ET>& e) {
  double d = std::ceil(CGAL::to_double(e)); // #include <cmath>
  while(d < e) d++; while(d - 1 >= e) d--;  return d; }
int round_down(const CGAL::Quotient<ET>& e) {
  double d = std::floor(CGAL::to_double(e));
  while(d > e) d--; while(d + 1 <= e) d++;  return d; }
```

- def

```
#define forloop(i,lo,hi) for(int i = (lo); i <= (hi); ++i)
#define rep(i,N) forloop(i,0,(int)N-1)
```

2 STL

2.1 array

```
void f(int ** a){...} // function that takes 2d array
int **arr = new int *[n];
rep(i,n) arr[i]=new int[n]; f(arr);
rep(i,n) delete arr[i]; delete arr;
```

2.2 vector, queue, stack

```
#include <vector>/<queue>/<algorithm>
vector<int> v3(5,3); // v3 initialied as[3,3,3,3,3]
v.push_back(i);
for(vector<int>::iterator it=v1.begin(); it!=v1.end(); it++) cout << *it << " ";
sort(v1.begin(), v1.end()); // simple sort (ascending order)
reverse(v5.begin(), v5.end()); // reverse elements in v5
std::queue<int> q; vector<bool> visited(n,false);
q.push(s); visited[s]=true;// do a bfs
while(!q.empty()){const int u=q.top(); q.pop();
  for(v : out_vertices(u)) if(visited[v]==false)
    {visited[v]=true; q.push(v);} }
```

2.3 sort, pq

```
bool my_cmp(const pair<int, int> &lhs, const pair<int, int> &rhs)
  {return (lhs.first < rhs.first) || (lhs.first==rhs.first && lhs.second < rhs.second);}
sort(vp.begin(), vp.end(), my_cmp); // put the function name as 3rd argument
struct MyFooStruct { int x,y;
  MyFooStruct(int xx, int yy) {x = xx;y = yy;}
  bool operator < ( const MyFooStruct & other ) const
    {return (x<other.x) || (x==other.x && y<other.y);} };
vector<MyFooStruct> vs; sort(vs.begin(), vs.end());
priority_queue<MyFooStruct> spq;
rep(i,4) spq.push( MyFooStruct(a1[i], a2[i]) );
cout << spq.top().x << "," << spq.top().y << endl; spq.pop();
```

2.4 set, map

```
set<int> s; rep(i, 10) s.insert(i); s.size();
for(set<int>::iterator it=s.begin(); it!=s.end(); it++) cout << *it << " ";
cout << (s.find(10)!=s.end()) << " " << (s.count(9)==1) << endl;
s.erase(100); s.erase(s.begin()); s.erase(--s.end());
set<int> s1,s2, s_union, s_intersect, s_diff; // #include <algorithm>
set_union( s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(s_union, s_union.end()) );
set_intersection( s1.begin(),s1.end(),s2.begin(),s2.end(),inserter(s_intersect,s_intersect.begin()) );
set_difference( s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(s_diff,s_diff.end()) );
map<string,int> m;
rep(i, 3) m.insert( make_pair(wds[i], cnts[i]) );// insert by making <k,v> pair
m["aa"] = 3; // update/insert by assignment
for(map<string,int>::iterator it=m.begin(); it!=m.end(); it++)
```

```
cout << it->first << ":" << it->second << ", ";
```

3 CGAL

3.1 CGAL basic

intersect.cpp, *hello-really-exact.cpp*, *two-kernels.cpp*, *minball.cpp*

- functions

```
typedef  CGAL::Exact_predicates_exact_constructions_kernel K;// or other kernels
K::FT CGAL::squared_distance (Type1<K> obj1, Type2<K> obj2){}//distance^2 between 2 geometric obj
std::sqrt(CGAL::to_double(CGAL::squared_distance(r,l)))//obtain an approximation of the real dist
bool CGAL::left_turn (Point_2 &p, Point_2 &q, Point_2 &r){}
int CGAL::orientation (const Point_2 &p1, const Point_2 &p2, const Point_2 &p3){}
auto CGAL::intersection (Type1< Kernel > obj1, Type2< Kernel > obj2){}
```

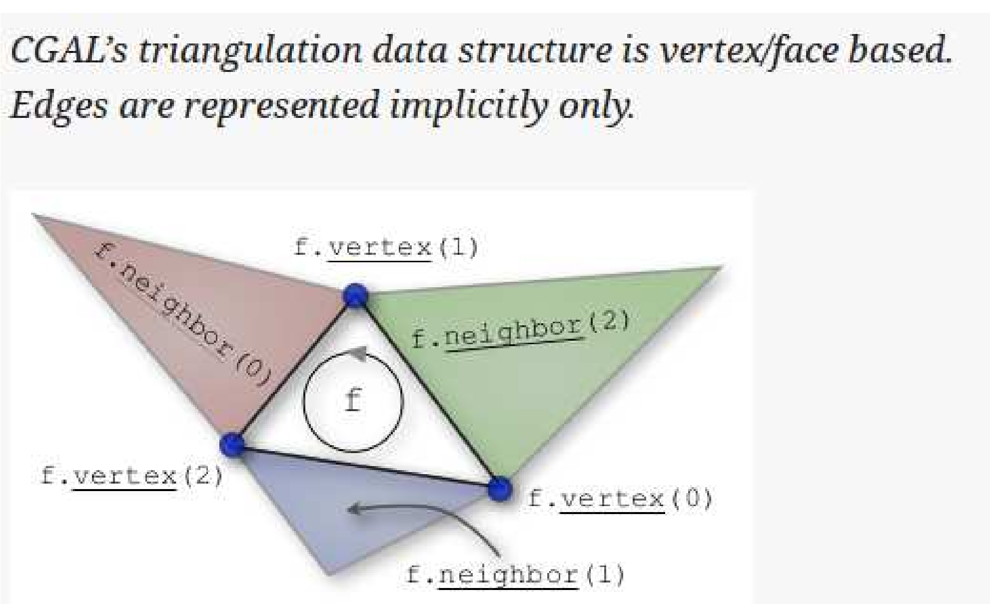
- intersection –*intersect.cpp*
- min circle –*minball.cpp*

```
typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
typedef CGAL::Min_circle_2_traits_2<K> Traits;
typedef CGAL::Min_circle_2<Traits> Min_circle;
typedef K::Point_2 P;
P points[n];// almost-antenna pb
Min_circle mc1( points, points+n, false); // very slow
Min_circle mc(points, points + n, true);
Traits::Circle c = mc.circle();
K::FT min_r = c.squared_radius();
if (mc.is_degenerate() == false)
  for (int i = 0; i < mc.number_of_support_points(); i++) P sp = mc.support_point(i);
```

3.2 Triangulation

- find nearest vertex
- iterate through all edges
- for a vertex, iterate through its incident edges
- do BFS/DFS on faces
- adding additional information

Proximity.pdf delaunay.cpp O(nlogn)

- in CGAL documentation, see *Triangulation_2* and *Delaunay_triangulation_2* for member functions of a triangulation.
- see *TriangulationDataStructure_2* for info wrt. Edge/Face/Vertex
- 

Edges in `CGAL::Triangulation_data_structure_2` are represented as a `std::pair<Face_handle,int>`: A pair `(f,i)` represents the i-th edge along the boundary of `*f` . The edge connects the *vertices (i+1)%3 and (i+2)%3* of `*f` .

```
//A safe strategy is to let the triangulation choose a suitable insertion order: Instead of
inserting points one by one using t.insert(p) insert a whole (iterator) range [b,e) of points using
t.insert(b,e)
rep(i,n) {
    K::Point_2 p; std::cin >> p;
    pts.push_back(p);}
// construct triangulation
Triangulation t;
t.insert(pts.begin(), pts.end());
for (Edge_iterator e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e){
  //~ if(t.segment(e).vertex(1)!=pts[0] && t.segment(e).vertex(0)!=pts[0]) continue;
  best = min(best, t.segment(e).squared_length() );
}
```

3.2.1 range search, circumcircles

cf. *radiation2*

3.3 LP QP

A model of QuadraticProgram describes a convex quadratic program of the form.

$$\begin{aligned}\text{(QP)}\ \text{minimize}\ & \mathbf{x}^T D\mathbf{x} + \mathbf{c}^T\mathbf{x} + c_0\\ \text{subject to}\ & A\mathbf{x} \gtreqless \mathbf{b},\\ & \mathbf{l} \le \mathbf{x} \le \mathbf{u}\end{aligned}$$

in n real variables $\mathbf{x} = (x_0, \ldots, x_{n-1})$.

Here,

- A is an $m \times n$ matrix (the constraint matrix),
- $\mathbf{b}$ is an m-dimensional vector (the right-hand side),
- $\gtreqless$ is an m-dimensional vector of relations from $\{\le, =, \ge\}$,
- $\mathbf{l}$ is an n-dimensional vector of lower bounds for $\mathbf{x}$, where $l_j \in \mathbb{R} \cup \{-\infty\}$ for all j
- $\mathbf{u}$ is an n-dimensional vector of upper bounds for $\mathbf{x}$, where $u_j \in \mathbb{R} \cup \{\infty\}$ for all j
- D is a symmetric positive-semidefinite $n \times n$ matrix (the quadratic objective function),
- $\mathbf{c}$ is an n-dimensional vector (the linear objective function), and
- c_0 is a constant.

portfolio.cpp LP_QP.pdf

```
// general QP: ** min x'Dx + c'x + c0 st. Ax<=b, l<=x<=u **
Program qp (CGAL::LARGER, false, 0, false, 0);
qp.set_u(X,true);        qp.set_u(Y,true);                        // x,y <= 0
qp.set_l(Z2,true); // z^2>=0
qp.set_a(X,0,1);qp.set_a(Y,0,1);qp.set_b(0,-4); // constraint-0: x + y >= -4
qp.set_a(X,1,4);qp.set_a(Y,1,2);qp.set_a(Z2,1,1);qp.set_b(1,-a*b);// constraint-1: 4x + 2y + z2 >= -ab
qp.set_a(X,2,-1);qp.set_a(Y,2,1);qp.set_b(2,-1); // constraint-2: -x + y => -1
// obj: min ax^2 + by + z^4
qp.set_d(Z2,Z2,2); // z^4
qp.set_d(X,X,a*2); // a*x^2 -- need to *2!
qp.set_c(Y,b); // b*y --need to *2!
Solution s = CGAL::solve_quadratic_program(qp, ET());
assert (s.solves_quadratic_program(qp));
if (s.status() == CGAL::QP_INFEASIBLE) {cout << "no" << endl;continue;}
```

```
else if (s.status() == CGAL::QP_UNBOUNDED) {cout<<"unbounded"<<endl; continue;}
double obj = CGAL::to_double(s.objective_value());
cout.precision(0); cout << fixed << ceil(obj) << endl;
```

4 BGL

4.1 BGL basics

bgl_tutorial_code.cpp, *bgl_handout_1.pdf*

4.1.1 connected component

```
int V = num_vertices(G); vector<int> comp(V); // stores index of the vertices' component
int ncomp = connected_components(G, &comp[0]);
cout << ncomp << " connected components in G. " << endl;
multimap<int, int> mm; // map scc id to vertices
rep(i, V) mm.insert( make_pair(comp[i], i) );
rep(i, ncomp){
  cout << "component-" << i << " have vertices: " ;multimap<int, int>::iterator ibeg, iend;
  for( tie(ibeg,iend) = mm.equal_range(i); ibeg!=iend; ibeg++) cout << ibeg->second << ", "; }
```

4.1.2 strong component –*bgl_tutorial_code.cpp*

cf. *bgl-tutorial monkey island*

```
vector<int> scc(V); // `scc` stores vertices'' strong component id (ie. one PARTITION of all vertices)
int nscc = strong_components(G,
           make_iterator_property_map(scc.begin(),get(vertex_index, G)) ); // nscc = nb of scc in G
```

4.1.3 biconnected components

```
WeightMap wm = get(edge_weight, G); // biconnected component is a partition of *edges*, so we need a
property_map (for edges) to store the component id for each edge
/* we just use the WeightMap (which is just a property_map for edges) to store the components
* else we can use the edge_component_t as above... ( need to redefine the 'Graph' type, see:
http://www.boost.org/doc/libs/1_59_0/libs/graph/example/biconnected_components.cpp )*/
int ncomp = biconnected_components(G, wm); // put wm as argument to store edge's component
cout << ncomp << " biconnected components" << endl;
EdgeIt ei, ei_end;
for (tie(ei, ei_end) = edges(G); ei != ei_end; ++ei)
cout << source(*ei, G) << "-" << target(*ei, G) << ", is in biconnected-component-" <<wm[*ei];
vector<Vertex> art_pts;// get articulation points
articulation_points(G, back_inserter(art_pts));
cout << art_pts.size() << " articulation points, they are: " << endl;
for(vector<Vertex>::iterator it=art_pts.begin(); it!=art_pts.end(); it++) cout << *it << ", ";
```

4.1.4 MST

- Kruskal

```
vector<Edge> mst; // edge vector to store mst: a list of V-1 edges
kruskal_minimum_spanning_tree(G, back_inserter(mst));
int mst_weight = 0;
for (vector<Edge>::iterator ebeg = mst.begin(); ebeg != mst.end(); ++ebeg) {
  int u = source(*ebeg, G), v = target(*ebeg, G); int c = wm[*ebeg];
  cout << u<<'-'<<v<<", weight="<<c<<endl; mst_weight += c;
} cout << "MST total weight is: " << mst_weight << endl;
```

- Prim

```
int V = num_vertices(G);
vector<int> pred(V); // predecessor vector
prim_minimum_spanning_tree(G, &pred[0]);
rep(j, V){
  Edge e; bool success;
  tie(e, success) = edge(j, pred[j], G);
```

```
    if(success){// because the mst root do not have pred...
      int u = source(e, G), v = target(e, G); int c = wm[e];
      cout << u<<'-'<<v<<", weight="<<c<<endl; }
  }
```

4.1.5 dijkstra –*bgl_tutorial_code.cpp*

```
int V = num_vertices(G);
vector<int> dist(V), pred(V);
dijkstra_shortest_paths(
    G, 0, // dijkstra with source=0
    predecessor_map( make_iterator_property_map(pred.begin(), get(vertex_index, G)) ).
            distance_map( make_iterator_property_map(dist.begin(),get(vertex_index, G)) )
);
rep(i,V) cout << "0-" << i << ", dist = " << dist[i] << endl;
```

4.1.6 topological sort

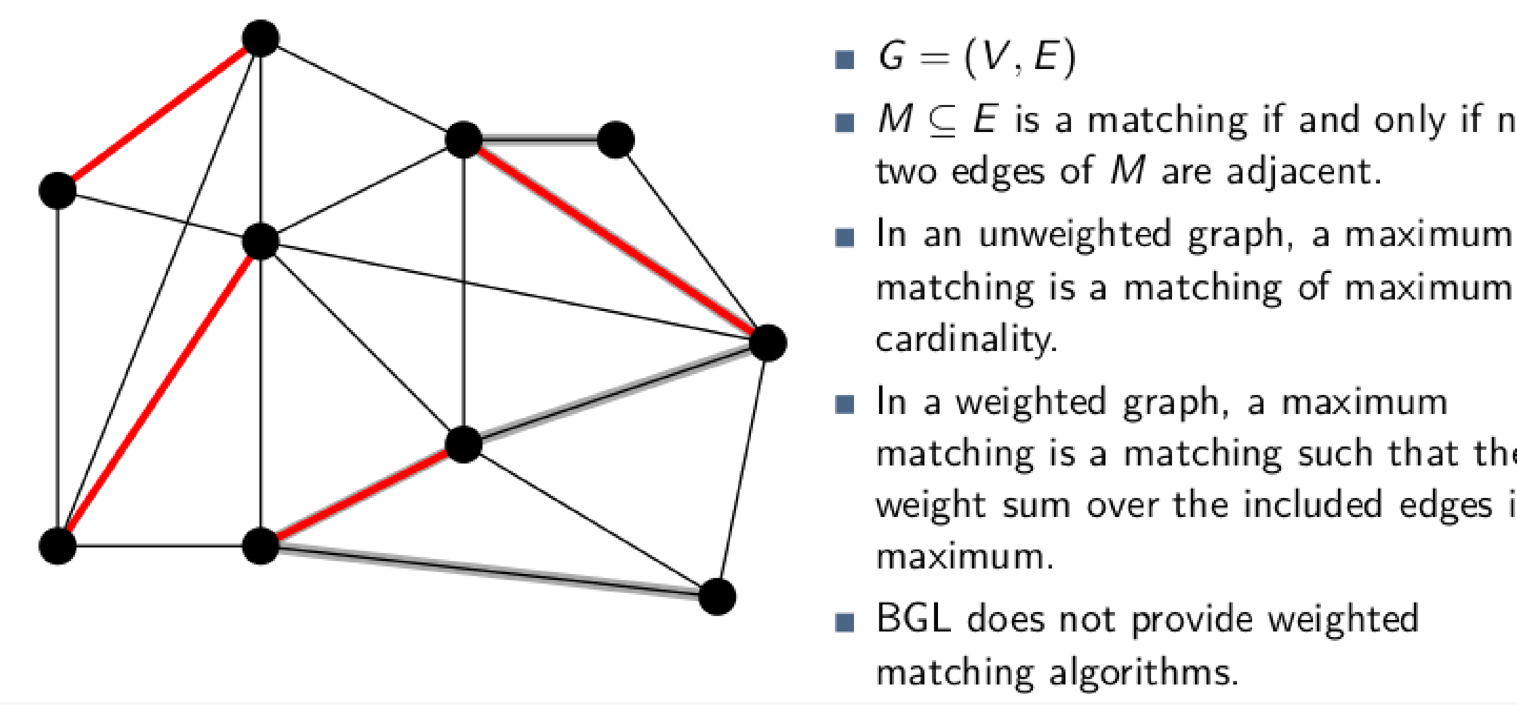
```
vector<Vertex> c; // container
topological_sort(G, back_inserter(c));
cout << "A topological ordering: ";
for ( vector<Vertex>::reverse_iterator ii=c.rbegin(); ii!=c.rend(); ++ii) cout << (*ii) << " ";
```

4.1.7 bfs/dfs

```
class custom_bfs_visitor : public boost::default_bfs_visitor
{ public:
  template < typename Vertex, typename Graph >
  void discover_vertex(Vertex u, const Graph & g) const  { cout << u << ", "; }
};
custom_bfs_visitor vis;
breadth_first_search(G, vertex(0, G), visitor(vis));// almost the same for DFS
```

4.1.8 max (cardinality) matching

cf. *buddy selection*



cf. *knights*

4.2 König's theorem

A graph is bipartite if and only if it does not contain an odd cycle.

In bipartite graphs, the size of minimum vertex cover is equal to the size of the maximum matching; this is König's theorem.[16][17]

An alternative and equivalent form of this theorem is that the size of the maximum independent set plus the size of the maximum matching is equal to the number of vertices.

4.3 Max flow

bgl_flows.cpp residualBFS.cpp mincost_maxflow.cpp

4.3.1 flow modelling

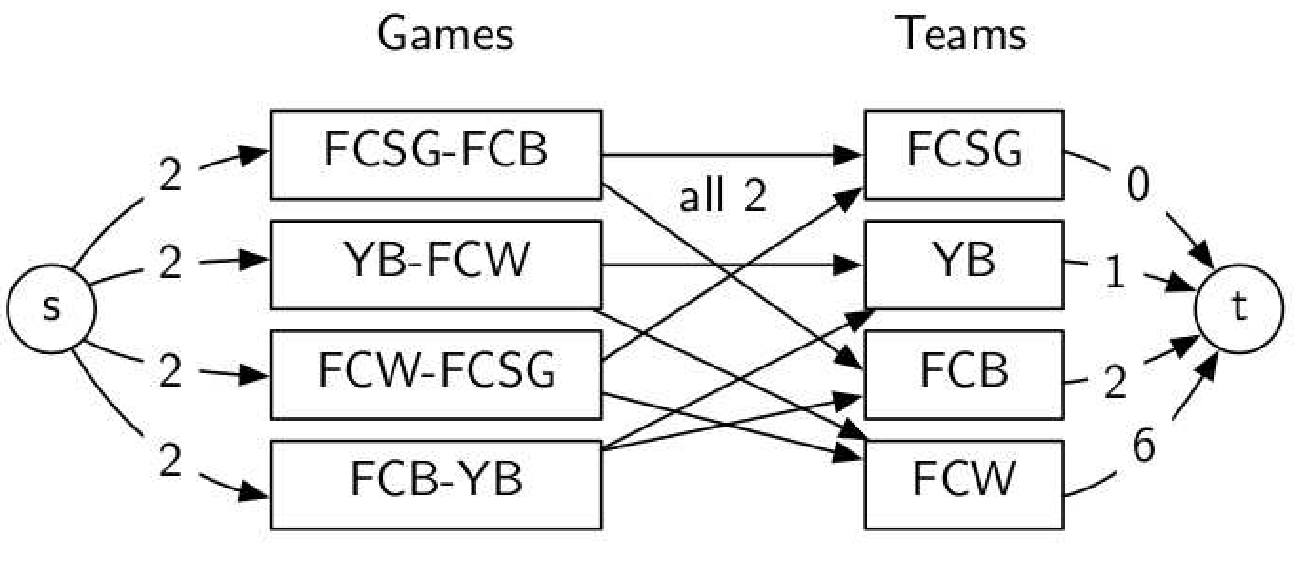


Figure 1.

4.3.2 circulation

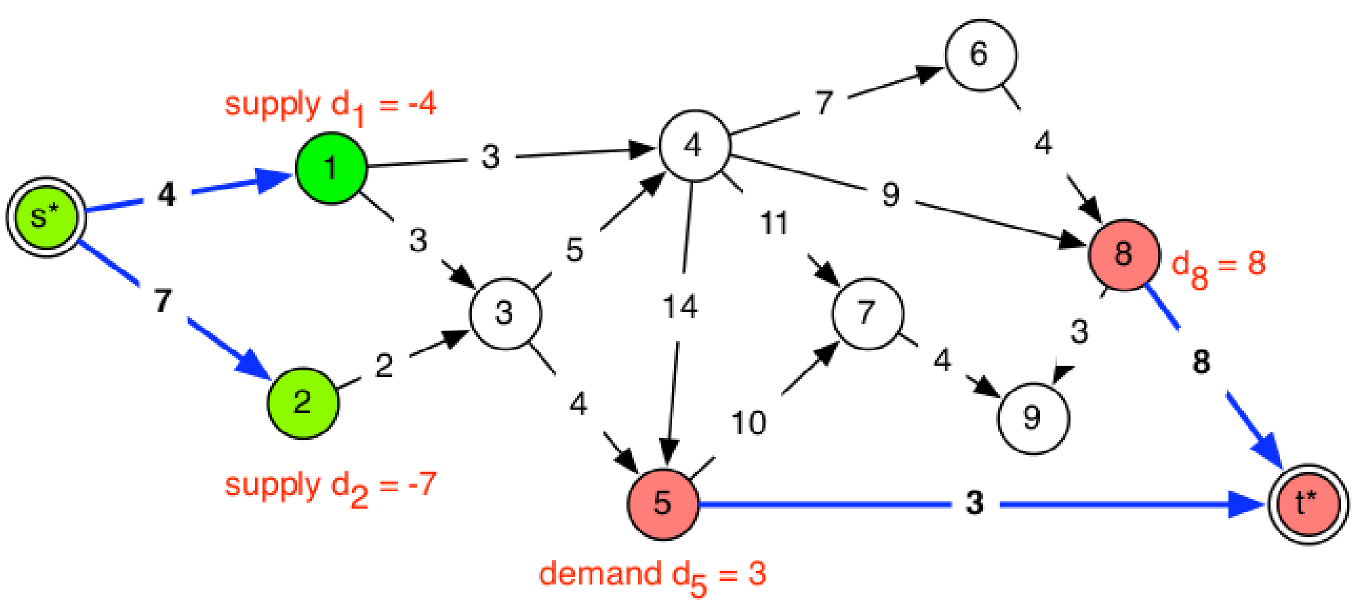


Figure 2.

4.3.3 edge lower bounds

first let lower bound flow, adjust supply/demand of each vertex —> to a circulation pb.

New demand constraints:

$$f^{\text{in}}(v) - f^{\text{out}}(v) = d_v - L_v$$

Also, f_0 uses some of the edge capacities already, so we have:

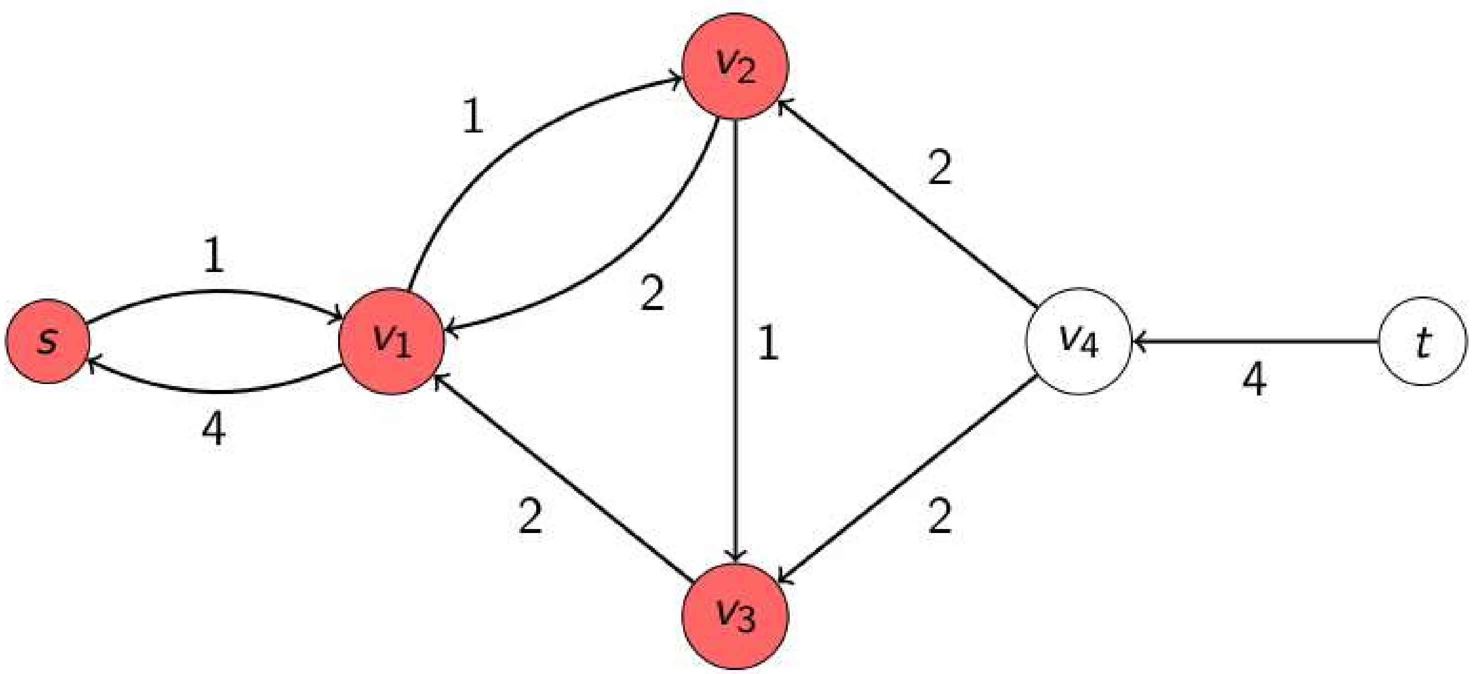
New capacity constraints:

$$0 \leq f(e) \leq c_e - \ell_e$$

These constraints give a standard instance of the circulation problem.

4.3.4 mincut

$V = S \bigcup T$, S=reachable vertices from s in residual graph. *residualBFS.cpp*



4.3.5 edge-disjoint paths

setting each edge of cap=1, maxflow from s to t = nb of edge-disjoint paths from s to t.

cf. *Phantom menace*

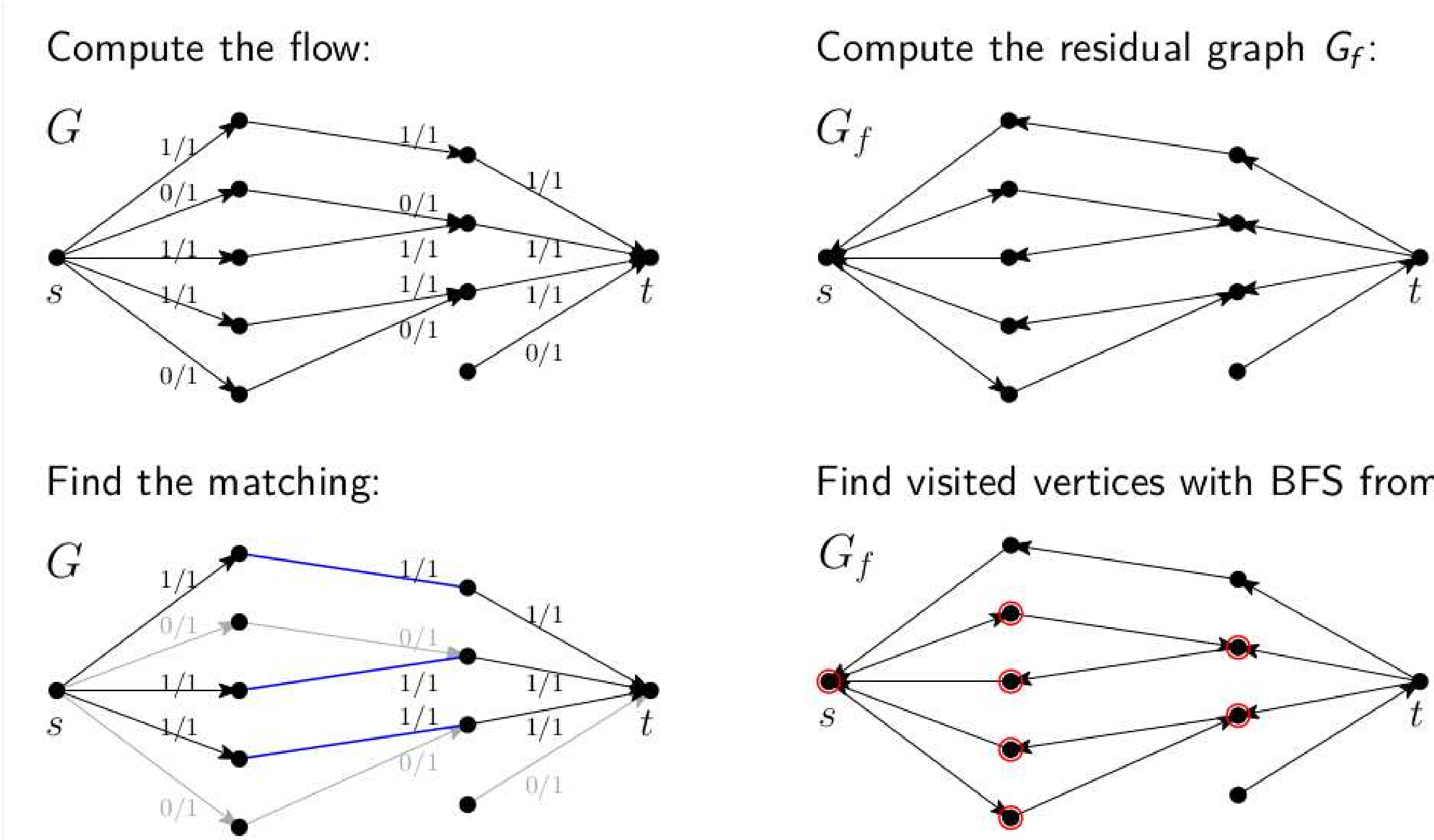
cf. *tetris*

4.3.6 vertex cover (bipartite matching)

residualBFS.cpp, *bgl_ tutorial_ 3_ moreflows.pdf*

- **Maximum independent set**
  Largest $T \subseteq V$, such that
  $\nexists u, v \in T : (u, v) \in E$.

- **Minimum vertex cover**
  Smallest $S \subseteq V$, such that
  $\forall (u, v) \in E : u \in S \lor v \in S$.



⇒ result is the unvisited vertices in L, and visited vertices in R.

Figure 3.

4.3.7 mincost maxflow

canceling negative weights –> cf *canteen*, *carsharing*

5 DP

compressing state!
cf. *bonus level*, *poker chips*