Algorithms Lab
BGL Week II

Flows

Maximum Flow
definition

- a flow network is a graph $G = (V, E)$
  with source and target vertices $s, t \in V$
  with capacity $c : V \times V \to \mathbb{N}$
  and flow $f : V \times V \to \mathbb{N}$
- a flow network satisfies

$$\forall u, v \in V : f(u, v) \leq c(u, v)$$

$$\forall u, v \in V : f(u, v) = -f(v, u)$$

$$\forall u \in V - \{s, t\} : \sum_{v \in V} f(u, v) = 0$$

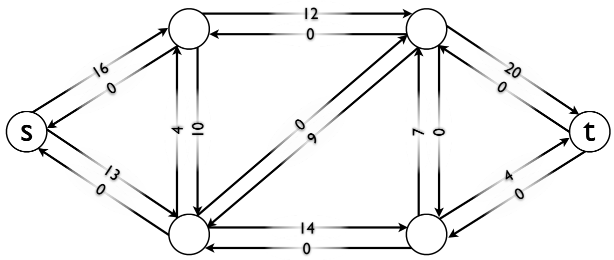
- the flow is $|f| = \sum_{u \in V} f(u, t) = \sum_{v \in V} f(s, v)$

Maximum Flow
Ford-Fulkerson method

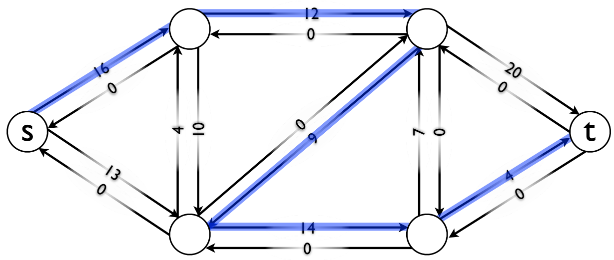
- The maximum flow problem asks for the maximum achievable flow $|f^\star|$, given G, s, t, c
- one method of solving it is due to Ford-Fulkerson:
  1. $\forall u, v \in V : f(u, v) \leftarrow 0$
  2. **while** there is an augmenting path p
  3. augment f along p
  4. **return** $|f^\star|$

$\Rightarrow$ runtime: $O(E\,|f^\star|)$

Maximum Flow
Ford-Fulkerson method in action

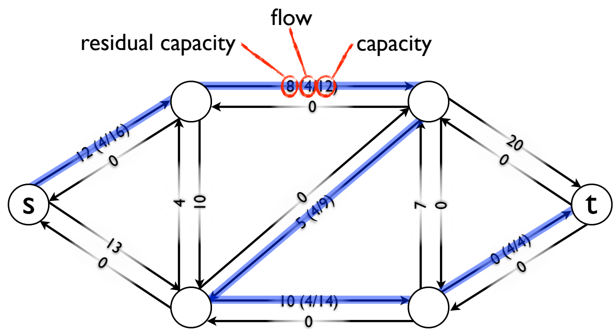


Maximum Flow
Ford-Fulkerson method in action



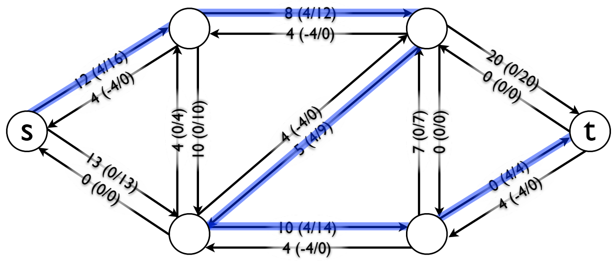
Maximum Flow

Ford-Fulkerson method in action



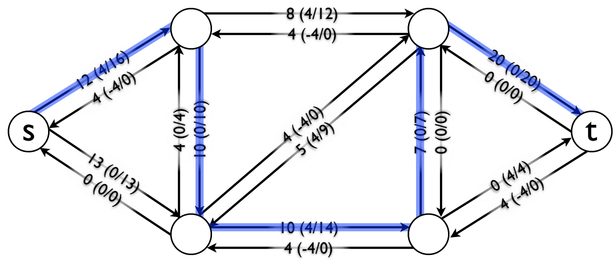
Maximum Flow

Ford-Fulkerson method in action



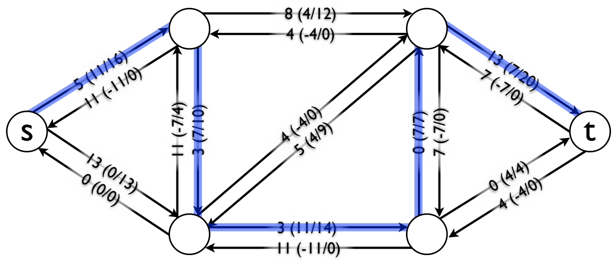
Maximum Flow

Ford-Fulkerson method in action



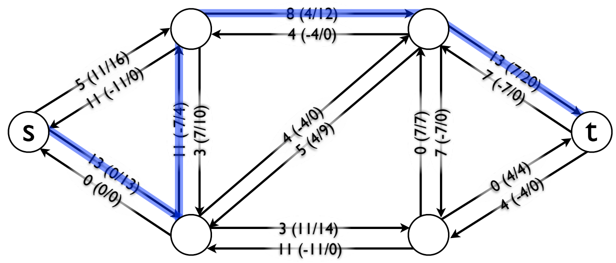
Maximum Flow

Ford-Fulkerson method in action



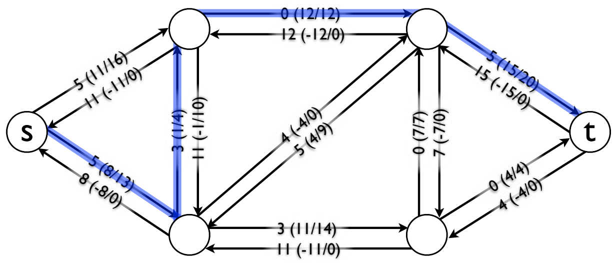
Maximum Flow

Ford-Fulkerson method in action

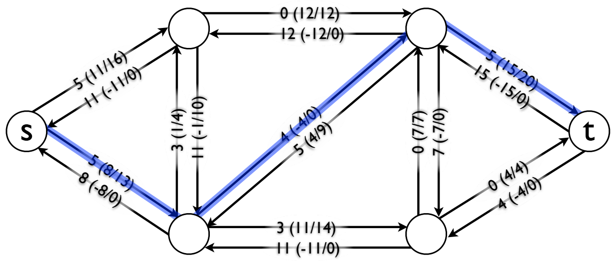


Maximum Flow

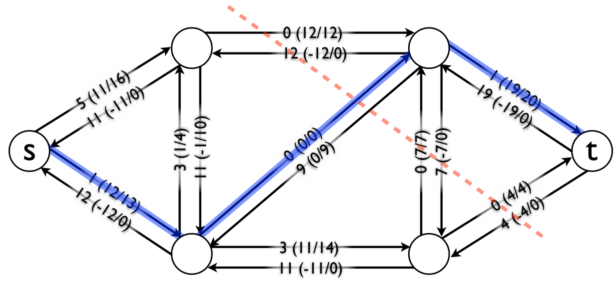
Ford-Fulkerson method in action



Maximum Flow
Ford-Fulkerson method in action



Maximum Flow
Ford-Fulkerson method in action



$\Rightarrow |f^\star| = 23$

Maximum Flow
Edmonds-Karp algorithm

- There are several algorithms that use the Ford-Fulkerson method
- Algorithms differ in how to choose augmenting paths
- Edmonds-Karps algorithm improves the naive bound from $O(E\,|f^\star|)$ to $O(VE^2)$ by using shortest paths

can be really bad...

Maximum Flow
push-relabel algorithms

- Define preflow

$$\forall u, v \in V : f(u,v) \leq c(u,v)$$
$$\forall u, v \in V : f(u,v) = -f(v,u)$$
$$\forall u \in V - \{s,t\} : \textstyle\sum_{v \in V} f(u,v) \leq 0$$

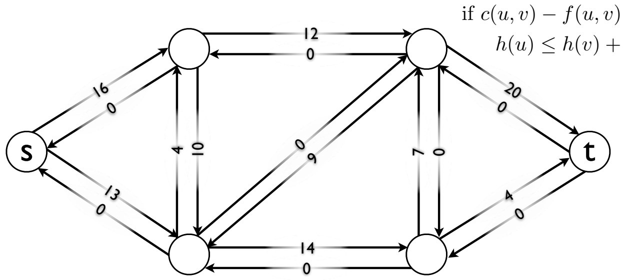
- Define height function

$$h(s) = |V|, h(t) = 0$$
$$h(u) \leq h(v) + 1 \text{ if } c(u,v) - f(u,v) \geq 0$$

- Preflow with height function has no augmenting path
  $\Rightarrow$ Flow with height function is a maximum flow

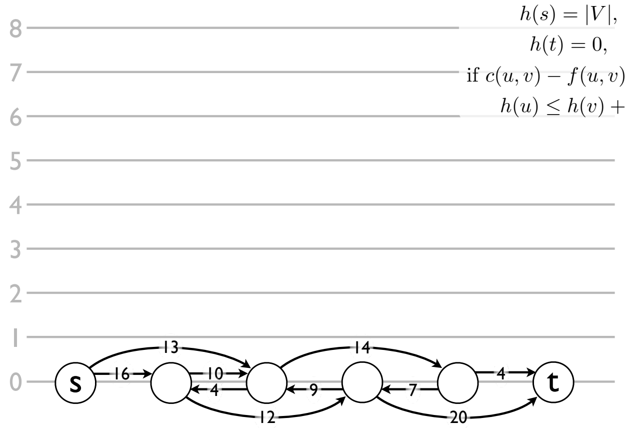
Maximum Flow
push-relabel by example

$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow
push-relabel by example

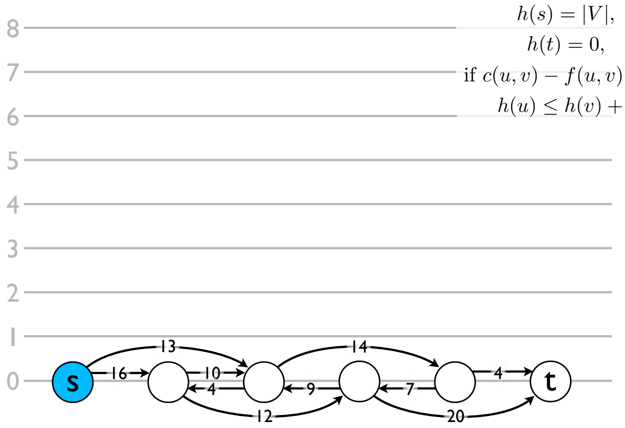
$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

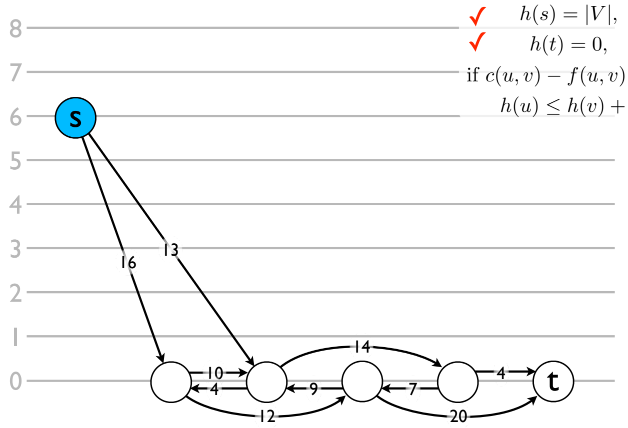
$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

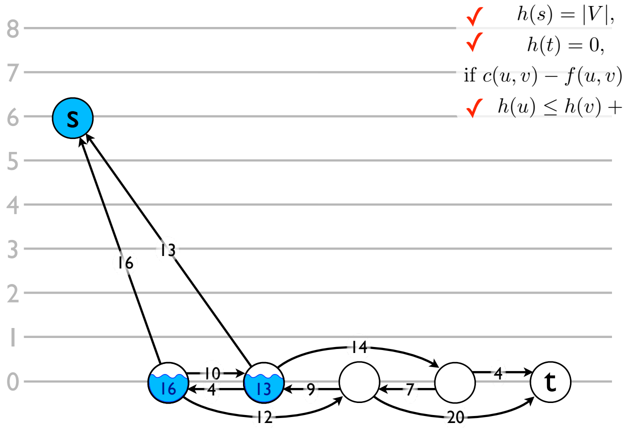
$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

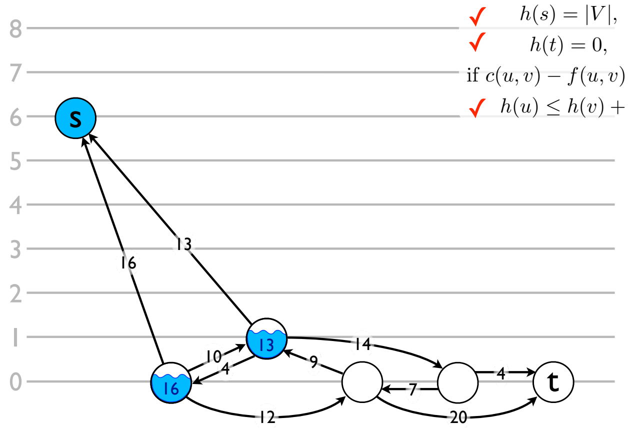
$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

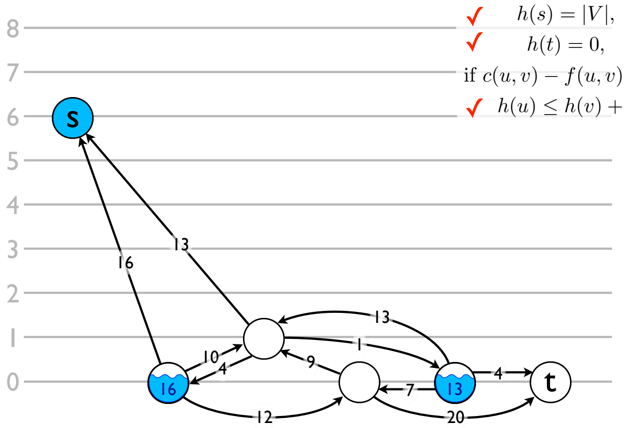
$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

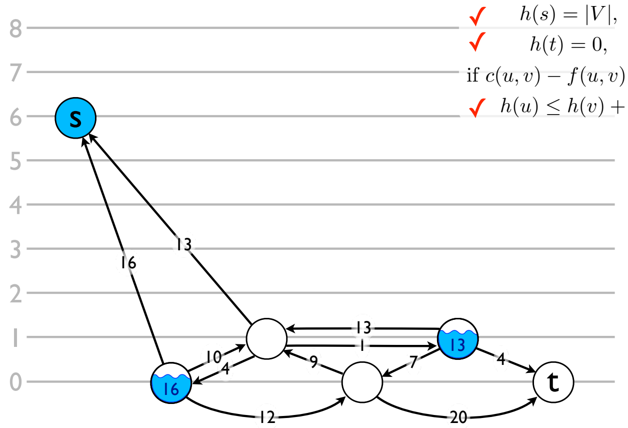
$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

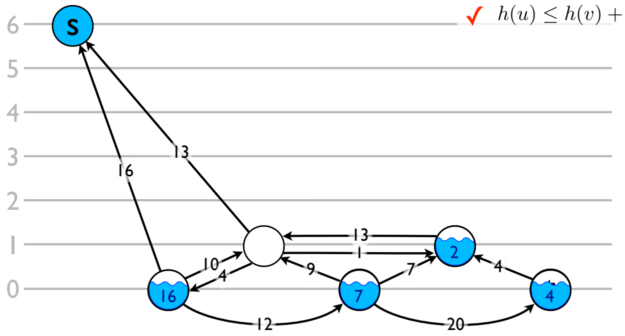
$h(s) = |V|,$
$h(t) = 0,$
$\text{if } c(u,v) - f(u,v) \geq 0:$
$h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

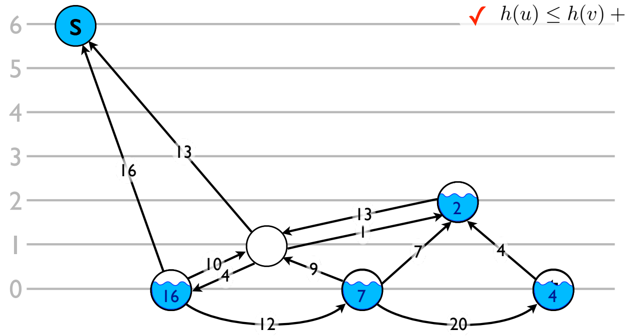
$\checkmark$ $h(s) = |V|$,
$\checkmark$ $h(t) = 0$,
if $c(u,v) - f(u,v) \ge 0$:
$\checkmark$ $h(u) \le h(v) + 1$



Maximum Flow

push-relabel by example

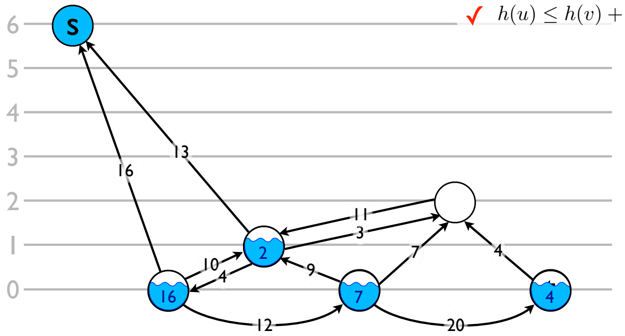
$\checkmark$ $h(s) = |V|$,
$\checkmark$ $h(t) = 0$,
if $c(u,v) - f(u,v) \ge 0$:
$\checkmark$ $h(u) \le h(v) + 1$



Maximum Flow

push-relabel by example

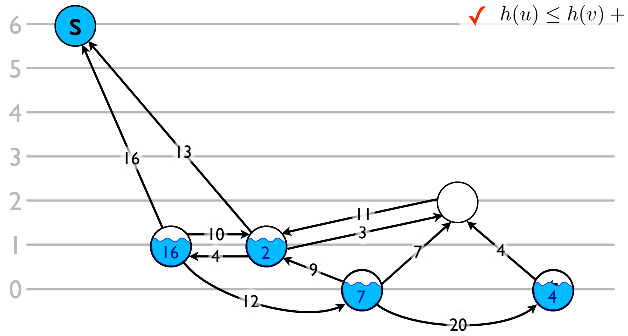
$\checkmark$ $h(s) = |V|$,
$\checkmark$ $h(t) = 0$,
if $c(u,v) - f(u,v) \ge 0$:
$\checkmark$ $h(u) \le h(v) + 1$



Maximum Flow

push-relabel by example

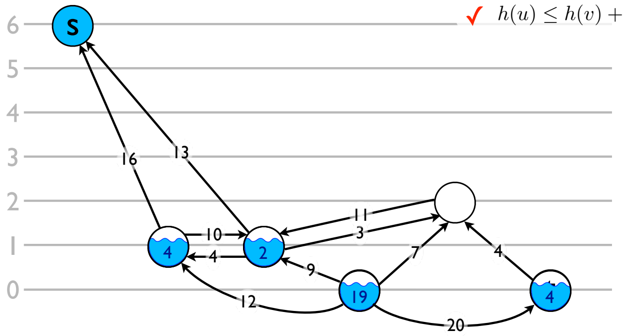
$\checkmark$ $h(s) = |V|$,
$\checkmark$ $h(t) = 0$,
if $c(u,v) - f(u,v) \ge 0$:
$\checkmark$ $h(u) \le h(v) + 1$



Maximum Flow

push-relabel by example

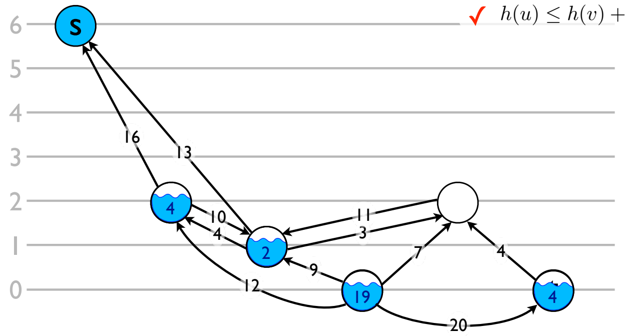
$\checkmark$ $h(s) = |V|$,
$\checkmark$ $h(t) = 0$,
if $c(u,v) - f(u,v) \ge 0$:
$\checkmark$ $h(u) \le h(v) + 1$



Maximum Flow

push-relabel by example

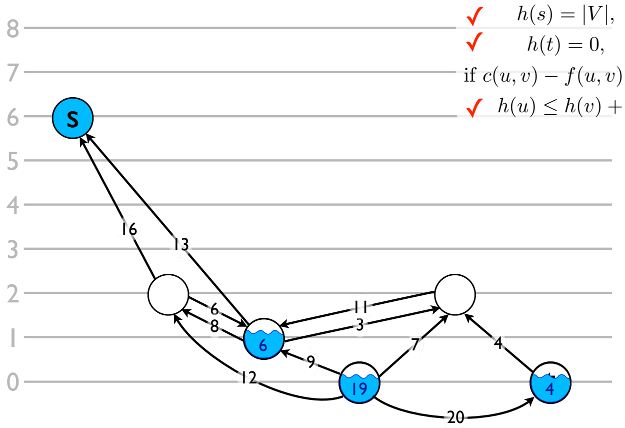
$\checkmark$ $h(s) = |V|$,
$\checkmark$ $h(t) = 0$,
if $c(u,v) - f(u,v) \ge 0$:
$\checkmark$ $h(u) \le h(v) + 1$



Maximum Flow

push-relabel by example

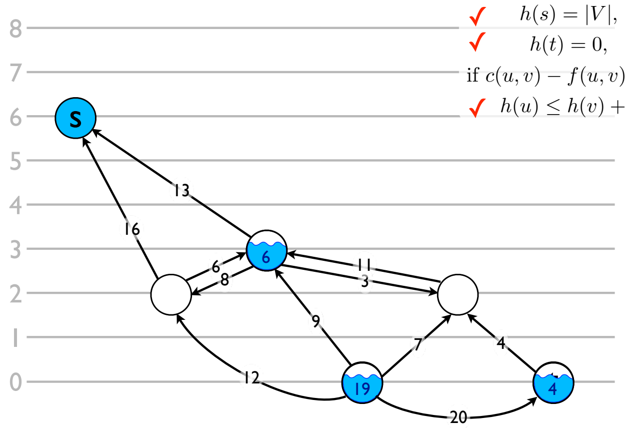
✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

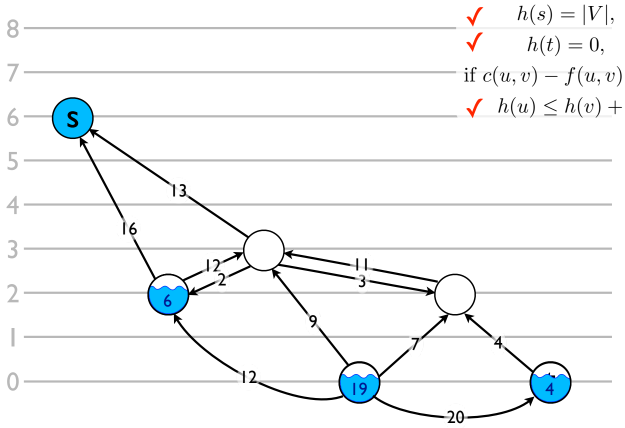
✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

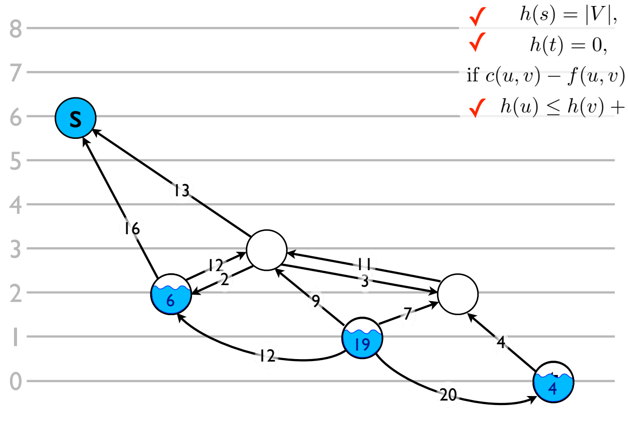
✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

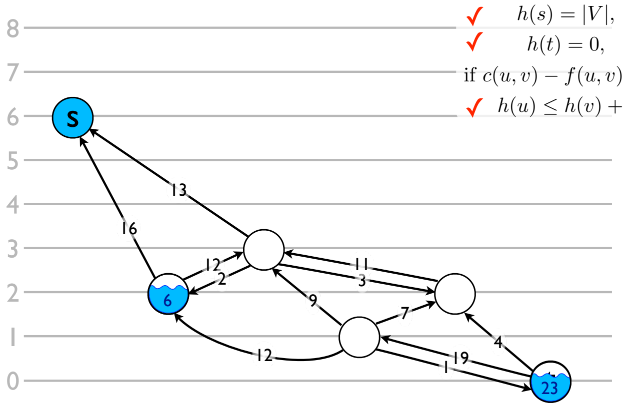
✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



Maximum Flow

push-relabel by example

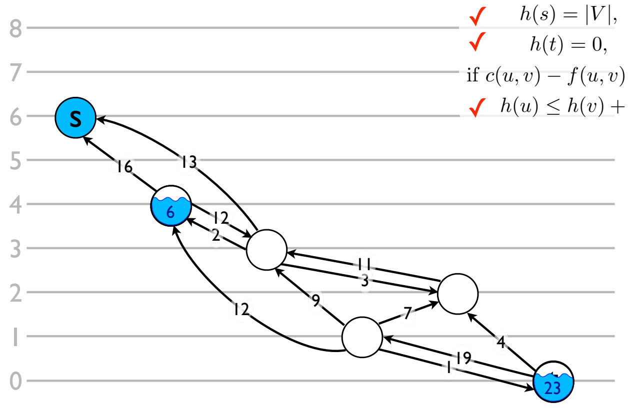
✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



Maximum Flow

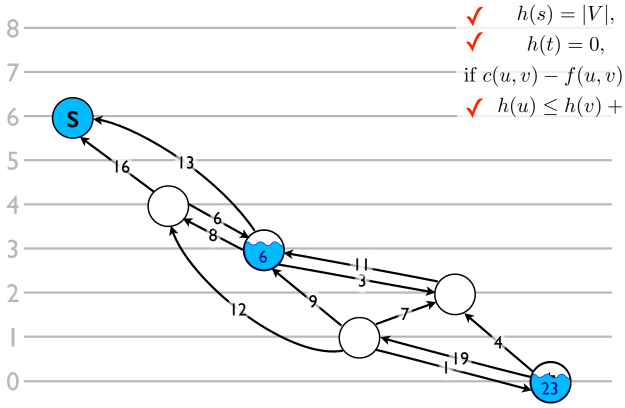
push-relabel by example

✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



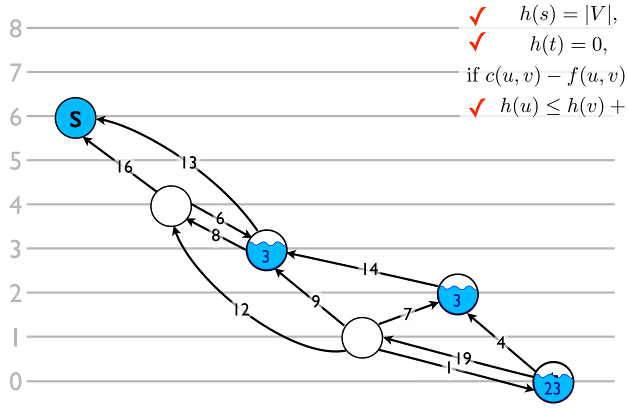
Maximum Flow
push-relabel by example

✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



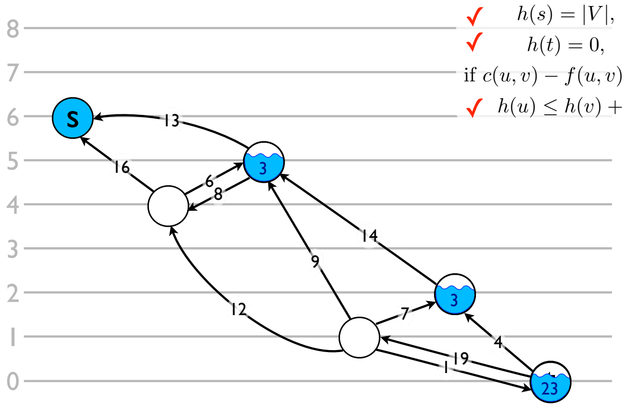
Maximum Flow
push-relabel by example

✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



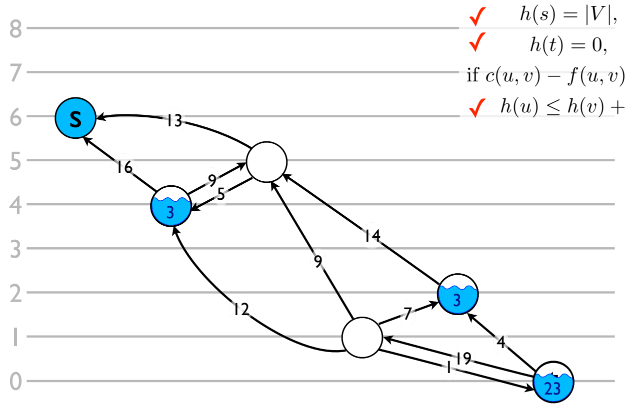
Maximum Flow
push-relabel by example

✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



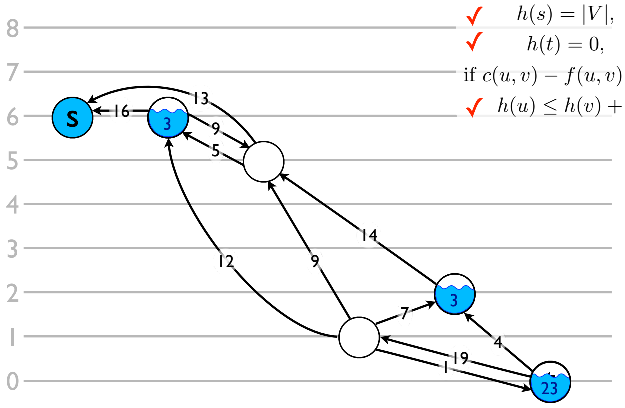
Maximum Flow
push-relabel by example

✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



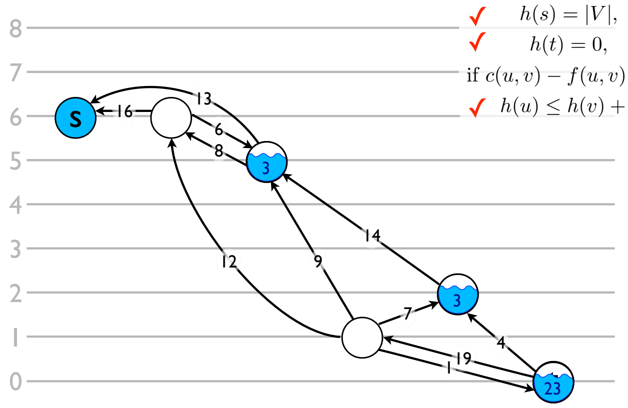
Maximum Flow
push-relabel by example

✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



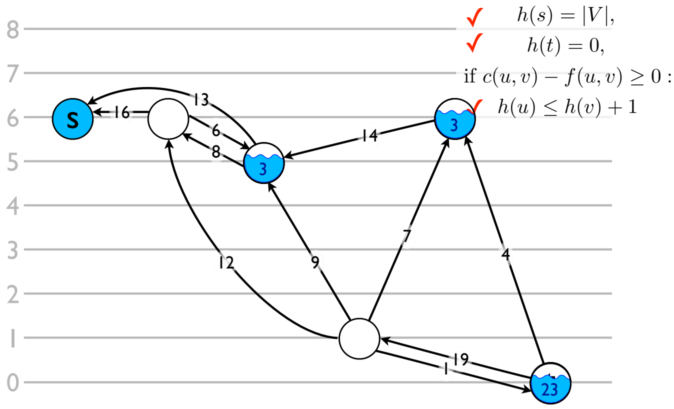
Maximum Flow
push-relabel by example

✓ $h(s) = |V|,$
✓ $h(t) = 0,$
if $c(u,v) - f(u,v) \geq 0:$
✓ $h(u) \leq h(v) + 1$



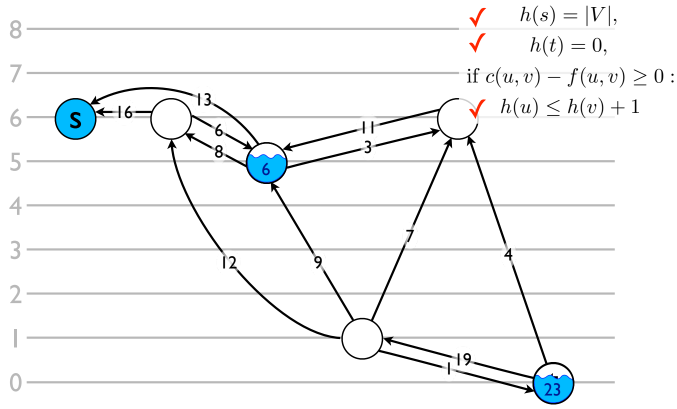
Maximum Flow

push-relabel by example



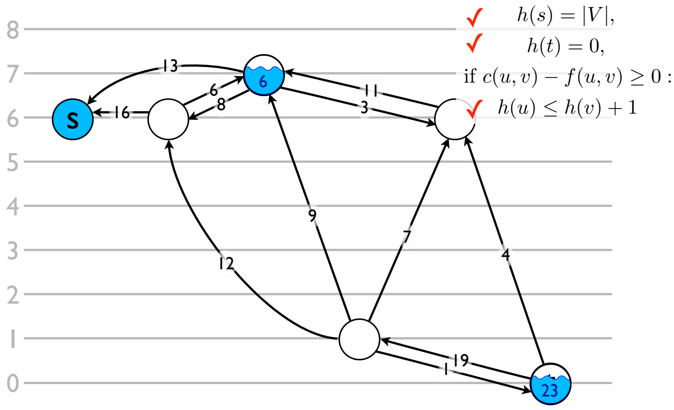
Maximum Flow

push-relabel by example



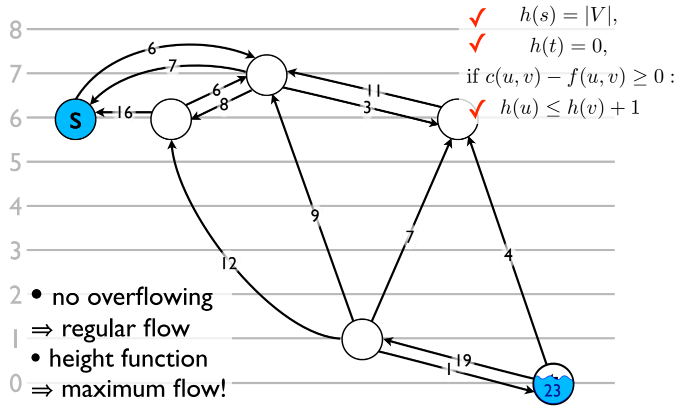
Maximum Flow

push-relabel by example



Maximum Flow

push-relabel by example



Maximum Flow

relabel-to-front algorithm

- Push-relabel algorithm has runtime $O(V^2E)$
  ⇒ Better than Ford-Fulkerson method!
- The order in which we "push" and "relabel" is arbitrary
- We can do better if we choose a good order
- Relabel-to-front achieves runtime $O(V^3)$
  ⇒ with "dynamic trees" even $O(VE\log(V^2E^{-1}))$

Maximum Flow

overview

method	algorithm	runtime
Ford-Fulkerson	naive	$O(E\,\lvert f^\star \rvert)$
Ford-Fulkerson	Edmonds-Karp	$O(VE^2)$
push-relabel	naive	$O(V^2E)$
push-relabel	relabel-to-front	$O(V^3)$
push-relabel	+ dynamic trees	$O(VE\log\frac{V^2}{E})$

Flows in BGL

Flows in BGL
configuring the types

```
typedef adjacency_list_traits<vecS, vecS, directedS> Traits;

typedef adjacency_list<vecS, vecS, directedS, no_property,
  property<edge_capacity_t, long,
  property<edge_residual_capacity_t, long,
  property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;

typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
typedef graph_traits<Graph>::edge_descriptor EdgeDescriptor;
```

Flows in BGL
adding edges

```
EdgeCapacityMap capacity = get(edge_capacity, g);
ReverseEdgeMap rev_edge = get(edge_reverse, g);
ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);

bool edgeDidNotExist;
EdgeDescriptor e, reverseE;
tie(e, edgeDidNotExist) = add_edge(a, b, g);
tie(reverseE, edgeDidNotExist) = add_edge(b, a, g);
capacity[e] = c;
capacity[reverseE] = 0;
rev_edge[e] = reverseE;
rev_edge[reverseE] = e;
```

Flows in BGL
invoking algorithms

```
#include <boost/graph/push_relabel_max_flow.hpp>
long flow = push_relabel_max_flow(g, source, sink);
#include <boost/graph/edmonds_karp_max_flow.hpp>
long flow = edmonds_karp_max_flow(g, source, sink);
```

How to read in a graph?

How to read in a graph?

```
typedef adjacency_list<vecS, vecS, undirectedS,
        no_property, property<edge_weight_t, int> > Graph;

int main
{
  int n,m; cin >> n >> m;

  vector< pair<int, int> > edges(m);
  vector<int> weights(m);

  for(int i = 0; i < m; ++i)
    cin >> edges[i].first >> edges[i].second >> weights[i];

  Graph g(edges.begin(), edges.end(), weights.begin(), n);
}
```

Named Parameters

Named Parameters

would it not be nice?

```
void printName(const string& title,
               const string& firstName,
               const string& middleName,
               const string& lastName)
{
  cout << title << (title != “” ? “. ” : “Mr./Mrs. ”)
       << firstName << (firstName != “” ? “ ” : “”)
       << (middleName != “” ? middleName[0] : “”)
       << (middleName != “” ? “. ” : “”)
       << lastName;
}

int main()
{
  printName(“”, “Peter”, “”, “Müller”);
  printName(“Dr”, “Franz”, “”, “Möller”);
  printName(“”, “Beat”, “Sepp”, “Wolf”);
}
```

Named Parameters

would it not be nice?

```
void printName(const string& title = “”,
               const string& firstName,
               const string& middleName = “”,
               const string& lastName)
{
  cout << title << (title != “” ? “. ” : “Mr./Mrs. ”)
       << firstName << (firstName != “” ? “ ” : “”)
       << (middleName != “” ? middleName[0] : “”)
       << (middleName != “” ? “. ” : “”)
       << lastName;
}

int main()
{
  printName(“Peter”, “Müller”);
  printName(“Dr”, “Franz”, “Möller”);
  printName(“Beat”, “Sepp”, “Wolf”);
}
```

Named Parameters

would it not be nice?

```
void printName(const string& title = “”,
               const string& firstName = “”,
               const string& middleName = “”,
               const string& lastName = “”)
{
  cout << title << (title != “” ? “. ” : “Mr./Mrs. ”)
       << firstName << (firstName != “” ? “ ” : “”)
       << (middleName != “” ? middleName[0] : “”)
       << (middleName != “” ? “. ” : “”)
       << lastName;
}

int main()
{
  printName(“Peter”, “Müller”);          //“P. Müller”
  printName(“Dr”, “Franz”, “Möller”);    //“Dr. Franz M. ”
  printName(“Beat”, “Sepp”, “Wolf”);     //“Beat. Sepp W.”
}
```

Named Parameters

would it not be nice?

```
void printName(const string& title = “”,
               const string& firstName = “”,
               const string& middleName = “”,
               const string& lastName = “”)
{
  cout << title << (title != “” ? “. ” : “Mr./Mrs. ”)
       << firstName << (firstName != “” ? “ ” : “”)
       << (middleName != “” ? middleName[0] : “”)
       << (middleName != “” ? “. ” : “”)
       << lastName;
}

int main()
{
  printName(first = “Peter”, last = “Müller”);
  printName(title = “Dr”, first = “Franz”, last = “Möller”);
  printName(first = “Beat”, middle = “Sepp”, last = “Wolf”);
}
```

Named Parameters

would it not be nice?

```
int main()
{
  ...
  Graph g(n);
  vector<int> p(n), d(n);

  dijkstra_shortest_paths(g, 0, predecessor_map = &p[0],
                                distance_map = &d[0]);
}
```

Named Parameters
BGL syntax

```
int main()
{
  ...
  Graph g(n);
  vector<int> p(n), d(n);

  dijkstra_shortest_paths
          (g, 0, predecessor_map(&p[0]).distance_map(&d[0]));
}
```

function method

Named Parameters
BGL Documentation

dijkstra_shortest_paths

```
// named parameter version
template <typename Graph, typename P, typename T, typename R>
void
dijkstra_shortest_paths(Graph& g,
  typename graph_traits<Graph>::vertex_descriptor s,
  const bgl_named_params<P, T, R>& params);

// non-named parameter version
template <typename Graph, typename DijkstraVisitor,
          typename PredecessorMap, typename DistanceMap,
          typename WeightMap, typename VertexIndexMap, typename CompareFunction, typename CombineFunction,
          typename DistInf, typename DistZero, typename ColorMap = default>
void dijkstra_shortest_paths
  (const Graph& g,
   typename graph_traits<Graph>::vertex_descriptor s,
   PredecessorMap predecessor, DistanceMap distance, WeightMap weight,
   VertexIndexMap index_map,
   CompareFunction compare, CombineFunction combine, DistInf inf, DistZero zero,
   DijkstraVisitor vis, ColorMap color = default)

// version that does not initialize the property maps (except for the default color map)
template <class Graph, class DijkstraVisitor,
          class PredecessorMap, class DistanceMap,
          class WeightMap, class IndexMap, class Compare, class Combine,
          class DistZero, class ColorMap>
void
dijkstra_shortest_paths_no_init
  (const Graph& g,
   typename graph_traits<Graph>::vertex_descriptor s,
   PredecessorMap predecessor, DistanceMap distance, WeightMap weight,
   IndexMap index_map,
   Compare compare, Combine combine, DistZero zero,
   DijkstraVisitor vis, ColorMap color = default);
```

Named Parameters
BGL syntax

```
int main()
{
  ...
  Graph g(n);
  vector<int> p(n), d(n);

  dijkstra_shortest_paths
          (g, 0, predecessor_map(&p[0]).distance_map(&d[0]));
                                   bgl_named_params
  //or?

  dijkstra_shortest_paths
          (g, 0, distance_map(&d[0]).predecessor_map(&p[0]));
}
```

Named Parameters
BGL Documentation

Parameters

IN: `const Graph& g`

The graph object on which the algorithm will be applied. The type `Graph` must be a model of Vertex List Graph and Incidence Graph.
Python: The parameter is named `graph`.

IN: `vertex_descriptor s`

The source vertex. All distance will be calculated from this vertex, and the shortest paths tree will be rooted at this vertex.
Python: The parameter is named `root_vertex`.

Named Parameters

IN: `weight_map(WeightMap w_map)`

The weight or ``length" of each edge in the graph. The weights must all be non-negative, and the algorithm will throw a `negative_edge` exception is one of the edges is negative. The type `WeightMap` must be a model of Readable Property Map. The edge descriptor type of the graph needs to be usable as the key type for the weight map. The value type for this map must be the same as the value type of the distance map.
Default: `get(edge_weight, g)`
Python: Must be an `edge_double_map` for the graph.
Python default: `graph.get_edge_double_map("weight")`

IN: `vertex_index_map(VertexIndexMap i_map)`

This maps each vertex to an integer in the range `[0, num_vertices(g))`. This is necessary for efficient updates of the heap data structure [61] when an edge is relaxed. The type `VertexIndexMap` must be a model of Readable Property Map. The value type of the map must be an integer type. The vertex descriptor type of the graph needs to be usable as the key type of the map.