10 B: Graph Algorithms IV

CS1102S: Data Structures and Algorithms

Martin Henz

March 27, 2009

Generated on Friday 26th March, 2010, 10:59

Maximum Flow

Interpretation of weights

Every weight of an edge (v, w) represents a *capacity* of a flow passing from v to w.

Maximum Flow

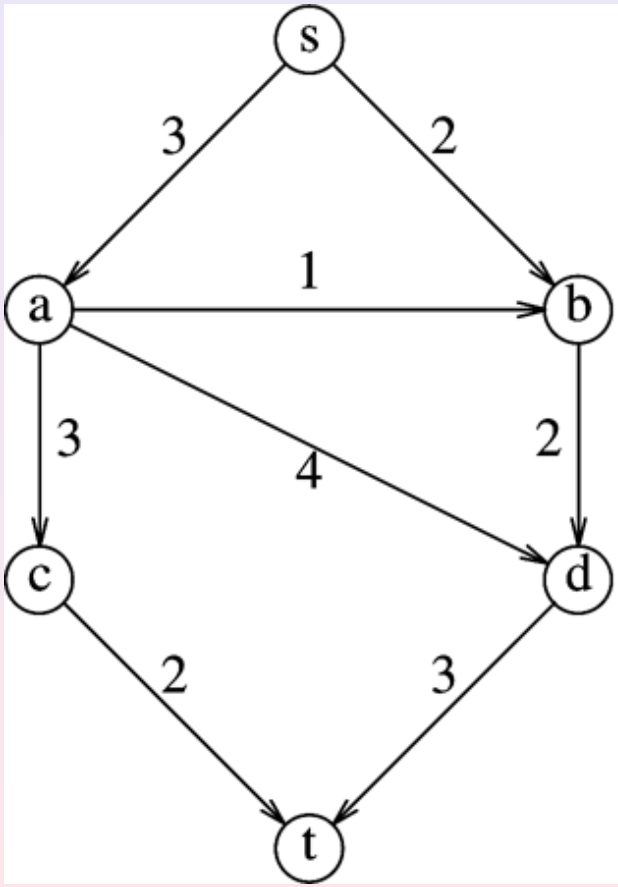
Interpretation of weights

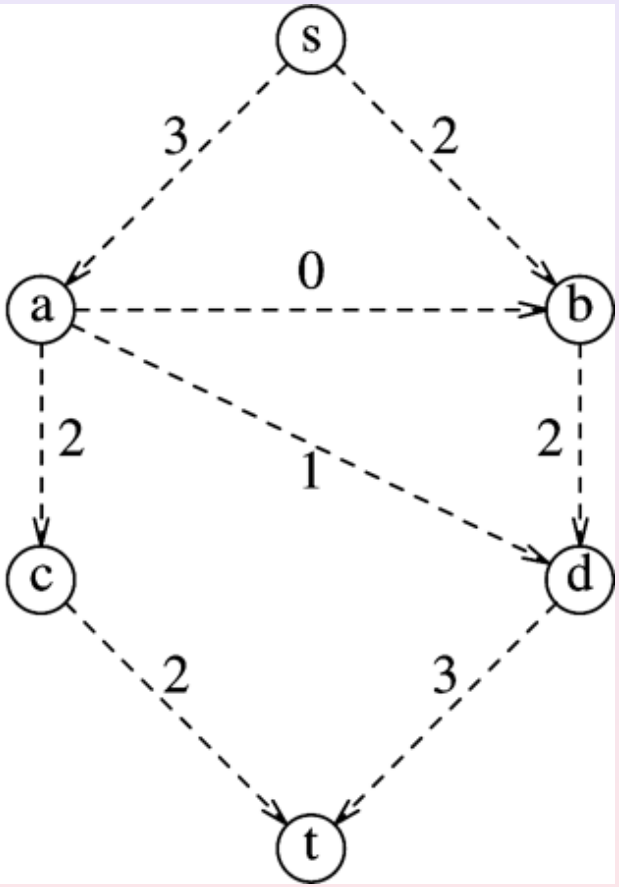
Every weight of an edge (v, w) represents a *capacity* of a flow passing from v to w.

Maximum Flow

Given two vertices s and t, compute the maximal flow achievable from s to t.

Example Graph and Maximum Flow





Idea

Identify an augmenting path

a path that has a feasible flow

Idea

Identify an augmenting path

a path that has a feasible flow

Add path to flow graph

keeping track of a flow that is already achieved

Idea

Identify an augmenting path

a path that has a feasible flow

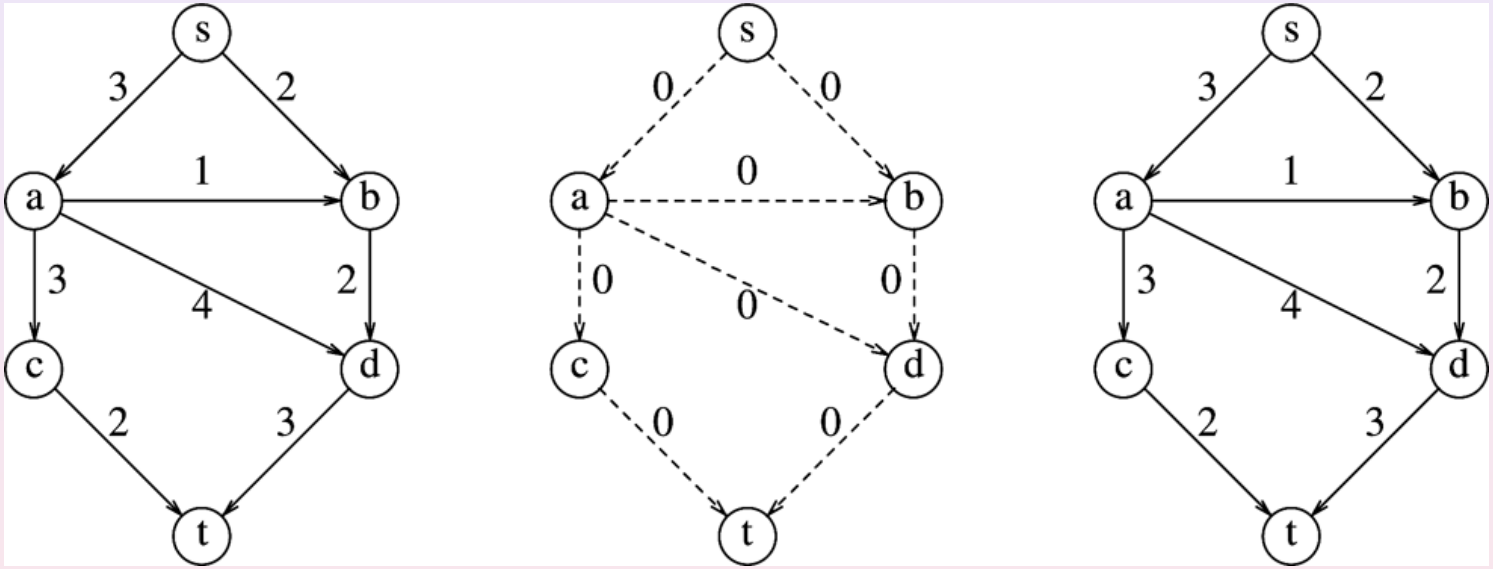
Add path to flow graph

keeping track of a flow that is already achieved

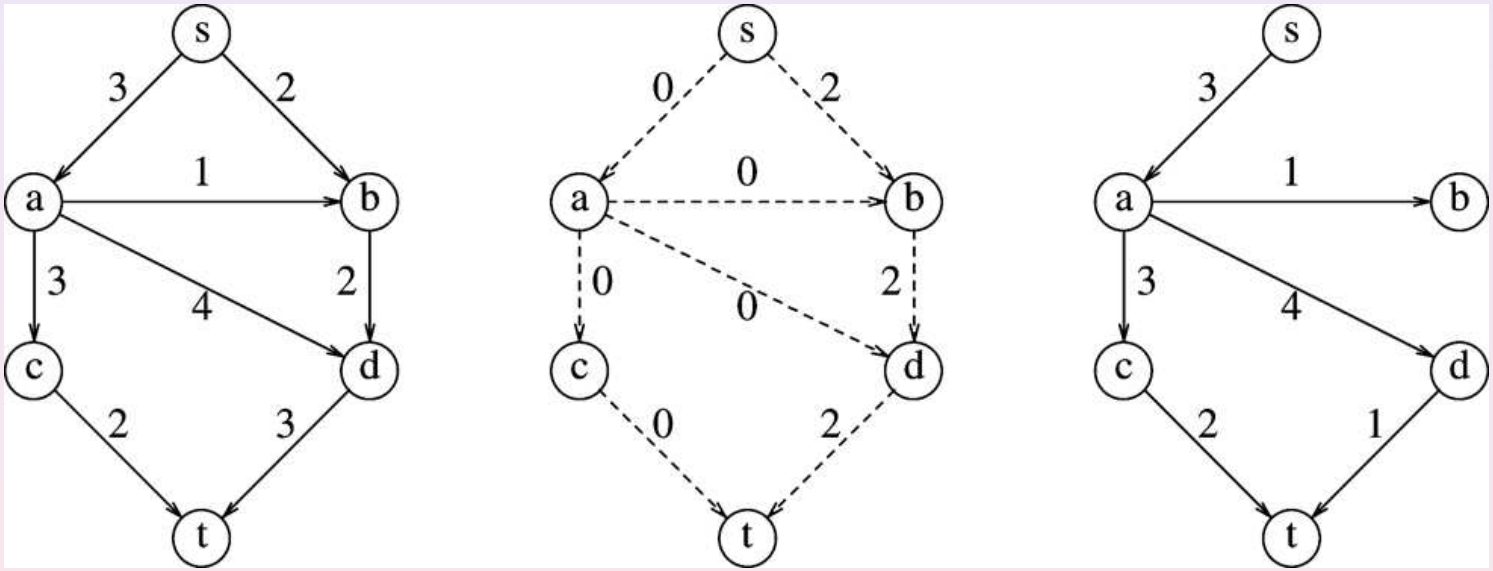
Remove path from residual graph

keeping track of the remaining flow that can still be exploited

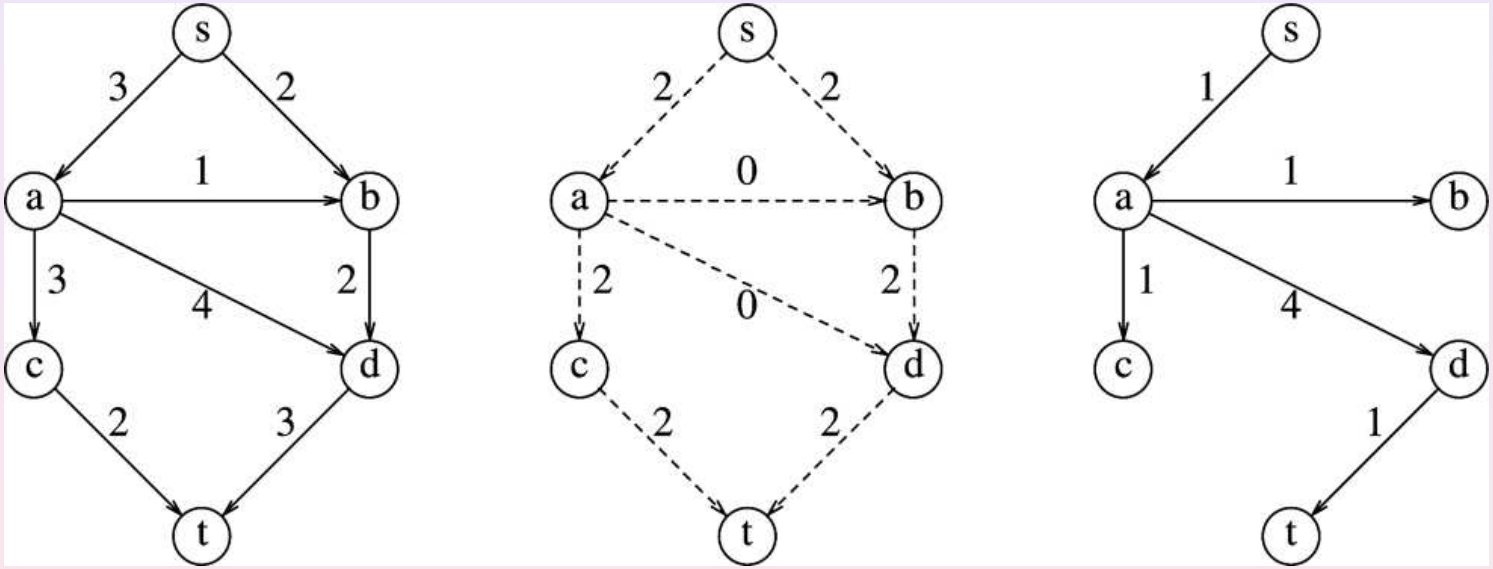
Example: Initial Setup



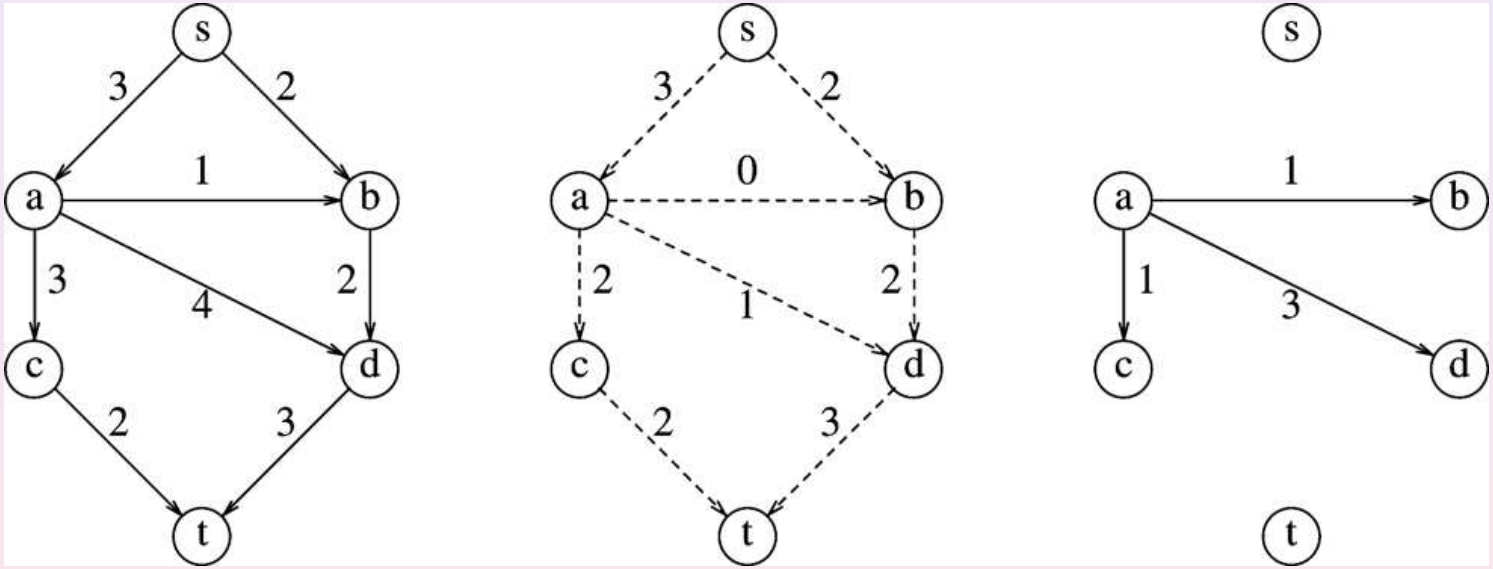
Example Run



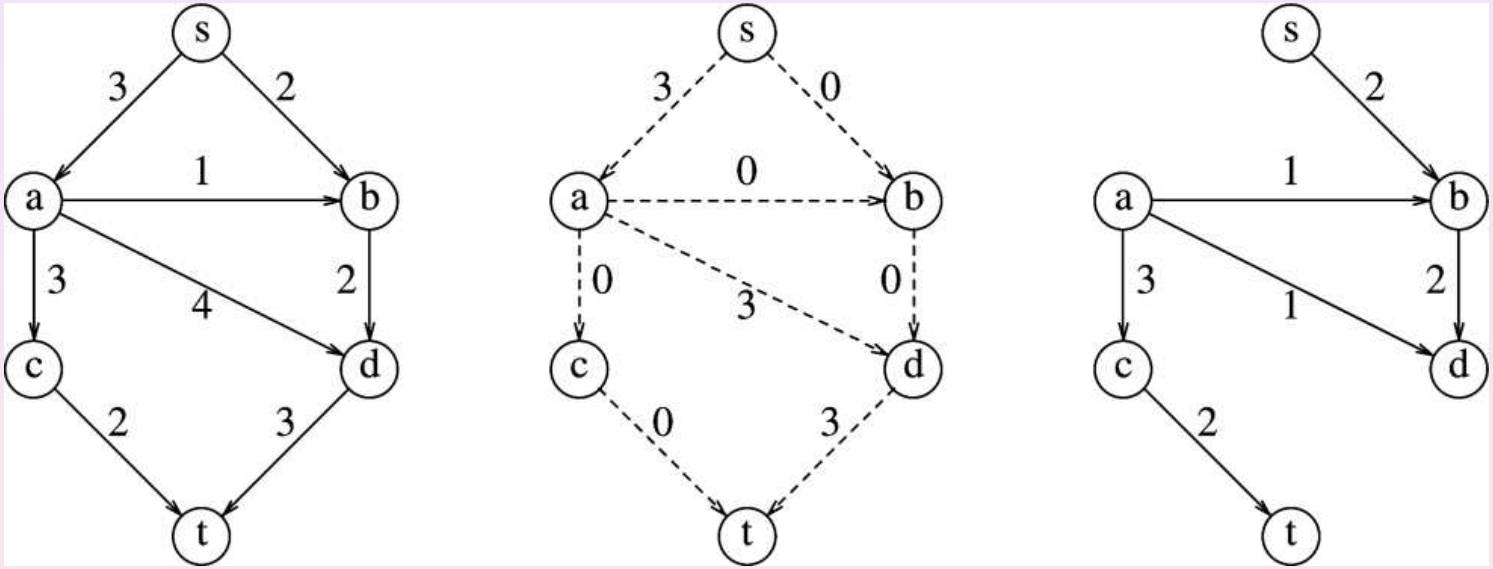
Example Run



Example Run



Counterexample: Suboptimal Solution



Idea

Allow algorithm to change its mind

Add reverse edge to residual graph, allowing a flow back in the opposite direction.

Idea

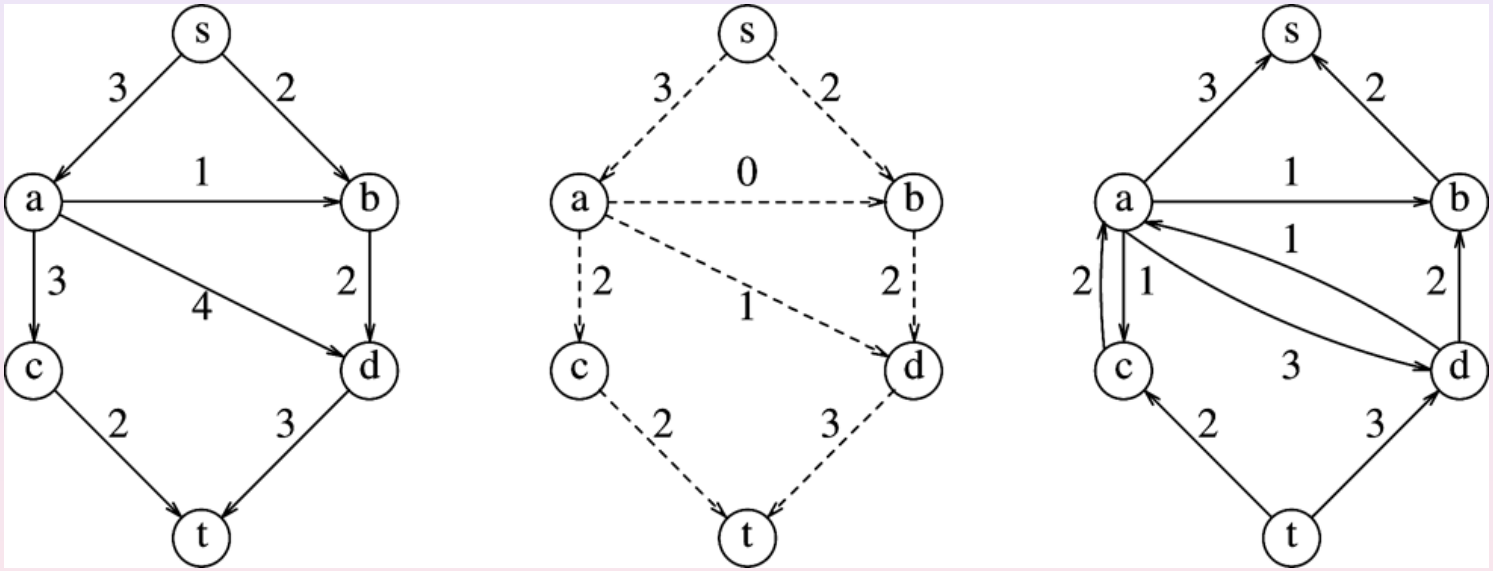
Allow algorithm to change its mind

Add reverse edge to residual graph, allowing a flow back in the opposite direction.

Consequence

This means that later, we can exploit an original flow which has been utilized already in the current flow graph, for a new augmenting path.

Example



Runtime Analysis

Assumption

Edge weights are integers

Runtime Analysis

Assumption

Edge weights are integers

Each augmenting path makes “progress”

adding a flow of at least 1 to the flow graph.

Runtime Analysis

Assumption

Edge weights are integers

Each augmenting path makes "progress"

adding a flow of at least 1 to the flow graph.

Finding augmenting paths

can be done in $O(N)$, using unweighted shortest path algorithm

Runtime Analysis

Assumption

Edge weights are integers

Each augmenting path makes "progress"

adding a flow of at least 1 to the flow graph.

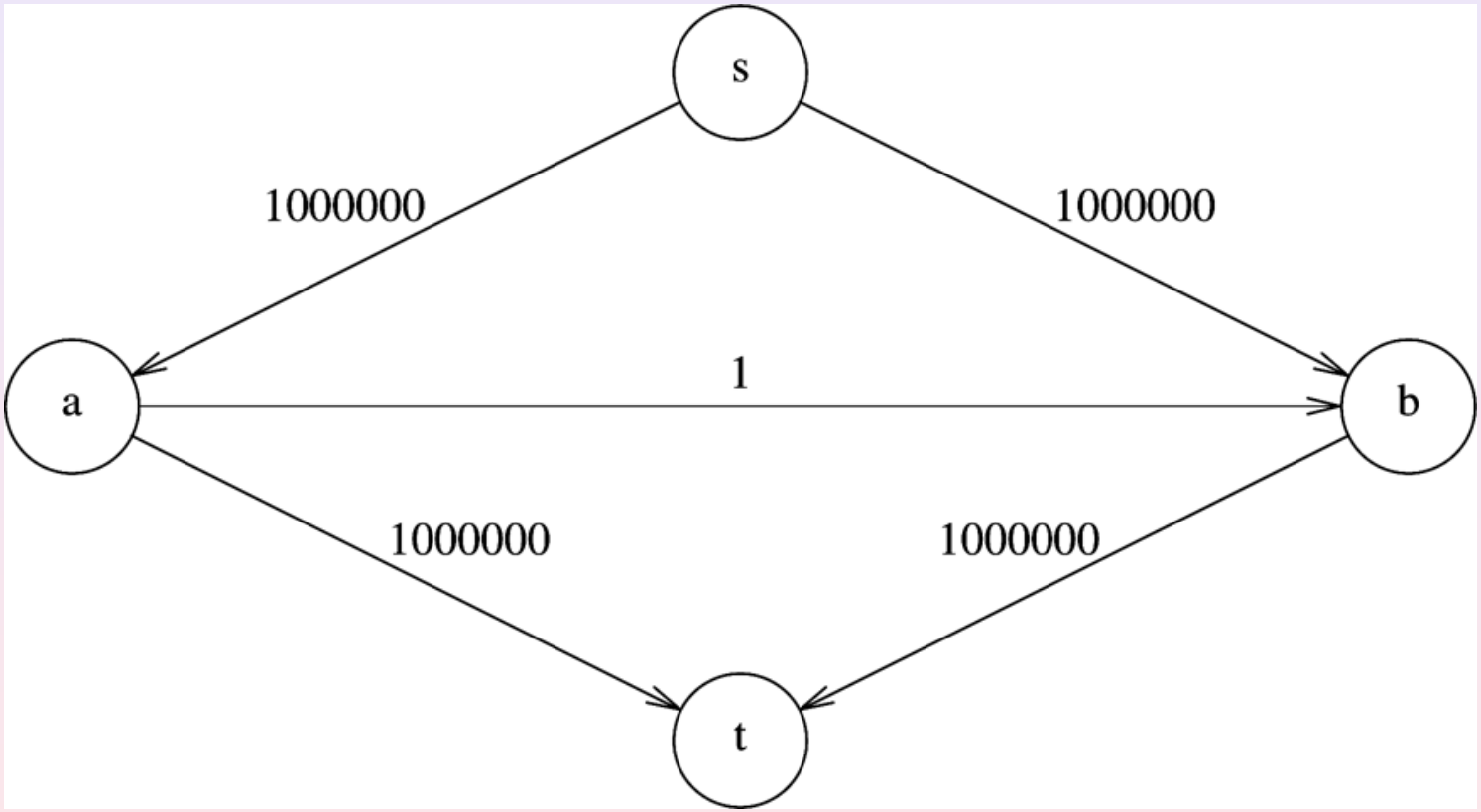
Finding augmenting paths

can be done in $O(N)$, using unweighted shortest path algorithm

Overall

$O(f \cdot |E|)$ where f is the maximum flow

Bad case



Minimum Spanning Tree Problem

Input

Undirected weighted connected graph G

Minimum Spanning Tree Problem

Input

Undirected weighted connected graph G

Spanning tree

Tree that contains all of G's vertices and only edges from G

Minimum Spanning Tree Problem

Input

Undirected weighted connected graph G

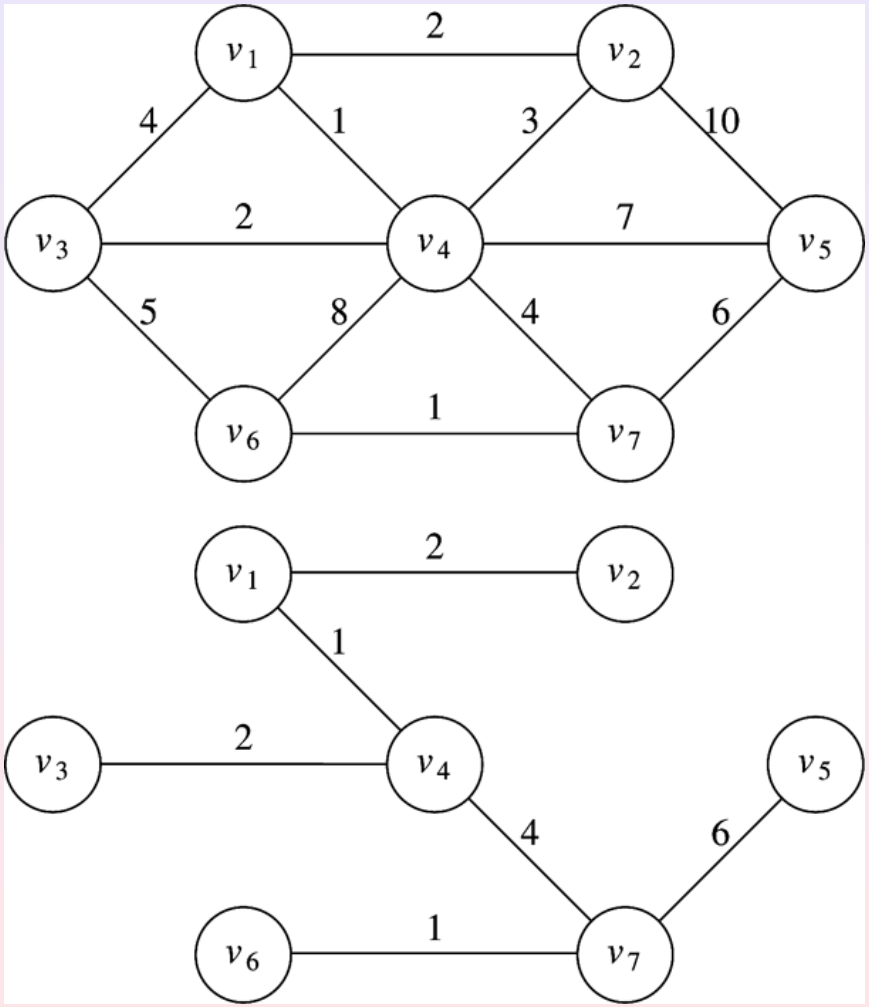
Spanning tree

Tree that contains all of G's vertices and only edges from G

Minimum spanning tree

Spanning tree whose sum of edge weights is minimal

An Example Graph and its MST



Algorithm Idea

Similar to Dijkstra's Algorithm

Start "growing" the MST at arbitrary vertex.

Algorithm Idea

Similar to Dijkstra's Algorithm

Start "growing" the MST at arbitrary vertex. At each step, add an edge to MST with smallest weight.

Algorithm Idea

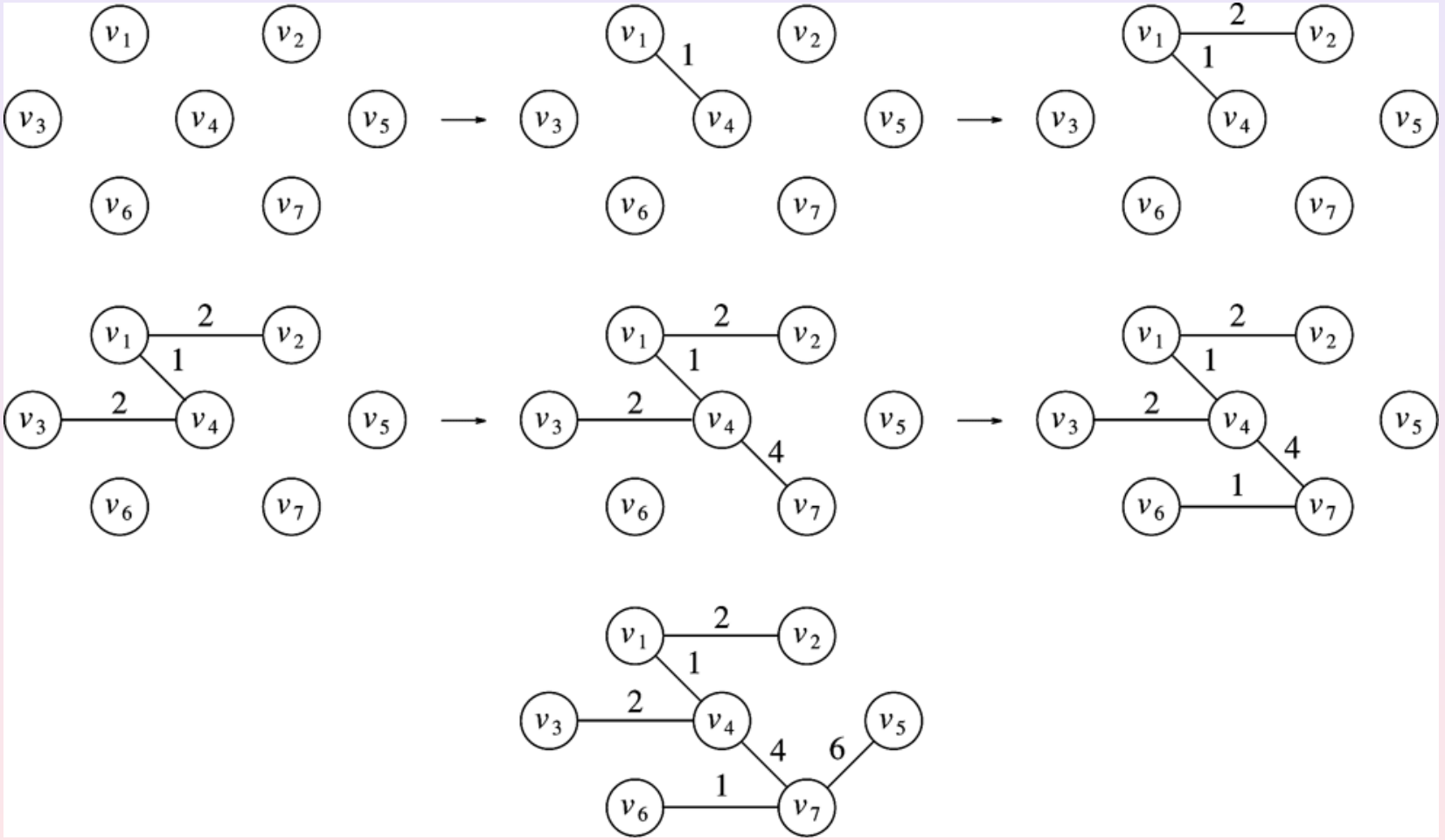
Similar to Dijkstra's Algorithm

Start "growing" the MST at arbitrary vertex. At each step, add an edge to MST with smallest weight.

Implementation

Keep edges from "known" to "unknown" vertices in a priority queue.

Example



Runtime Analysis

Without priority queue

$O(|V|^2)$

Runtime Analysis

Without priority queue

$O(|V|^2)$

With priority queue

$O(|E| \log |V|)$

How Difficult Can Problems Be?

Examples

- Finding the smallest element in a linked list:

Examples

- Finding the smallest element in a linked list: $O(N)$

Examples

- Finding the smallest element in a linked list: $O(N)$
- Inserting an element into a heap:

Examples

- Finding the smallest element in a linked list: $O(N)$
- Inserting an element into a heap: $O(\log N)$

Examples

- Finding the smallest element in a linked list: $O(N)$
- Inserting an element into a heap: $O(\log N)$
- Sorting an array of items using comparisons:

Examples

- Finding the smallest element in a linked list: $O(N)$
- Inserting an element into a heap: $O(\log N)$
- Sorting an array of items using comparisons: $O(N \log N)$

Examples

- Finding the smallest element in a linked list: $O(N)$
- Inserting an element into a heap: $O(\log N)$
- Sorting an array of items using comparisons: $O(N \log N)$
- Printing all sequences of N binary digits:

Examples

- Finding the smallest element in a linked list: $O(N)$
- Inserting an element into a heap: $O(\log N)$
- Sorting an array of items using comparisons: $O(N \log N)$
- Printing all sequences of N binary digits: $O(2^N)$

Algorithmic Problems

Given Input

Clear description of input data

Algorithmic Problems

Given Input

Clear description of input data

Required Output

Description of what a solution constitutes

Algorithmic Problems

Given Input

Clear description of input data

Required Output

Description of what a solution constitutes

Algorithmic Solution

Description of a process how to arrive at the required output, given any legal input

Questions

- Are all algorithmic problems solvable?

Questions

- Are all algorithmic problems solvable?
- Are all algorithmic problems solvable in polynomial time?

Questions

- Are all algorithmic problems solvable?
- Are all algorithmic problems solvable in polynomial time? Is there a k such that there is an $O(N^k)$ algorithm for the problem?

Questions

- Are all algorithmic problems solvable?
- Are all algorithmic problems solvable in polynomial time? Is there a k such that there is an $O(N^k)$ algorithm for the problem?
  Clearly no! The output can have exponential size!

Questions

- Are all algorithmic problems solvable?
- Are all algorithmic problems solvable in polynomial time?
  Is there a k such that there is an $O(N^k)$ algorithm for the problem?
  Clearly no! The output can have exponential size!
- If we do not have a polynomial algorithm, can we always prove that there is none?

Halting Problem

Definition (Halting Problem)

Given a description of a program and a finite input, decide whether the program finishes running or will run forever, given that input.

Undecidability of the Halting Problem

Theorem (Undecidability of the Halting Problem)

The Halting Problem is undecidable; there cannot exist a program that returns `true`*, if and only if a given function* `f` *terminates when applied to a given value* `x`*.*

Proof of the Undecidability of the Halting Problem

Assume that there is a Java function `halts` that when applied to a representation `f'` of a Java function `f`, and a Java object `x` returns `true` if `f(x)` terminates, and `false` if `f(x)` does not terminate. Such a function `halts` exists if and only if the Halting Problem is decidable.

Proof of the Undecidability of the Halting Problem

With the assumption of the existence of `halts`, let us construct a Java function `strange` as follows:

```
boolean strange(w') {
  if (halts(w',w')
    while (true);
  return true;
}
```

Such a function `strange` can surely be written, if `halts` exists.

Proof of the Undecidability of the Halting Problem

```
boolean strange(w') {
  if (halts(w',w')
    while (true);
  return 0;
}
```

What will `strange(strange')` return?

Proof of the Undecidability of the Halting Problem

Conclusion: `strange` cannot exist, and therefore `halt` cannot exist.

The Halting Problem is undecidable.

Remember Lecture 2 A: Parameter Passing

Java uses pass-by-value parameter passing.

```
public static void tryChanging(int a) {
    a = 1;
    return;
}
...
int b = 2;
tryChanging(b);
System.out.println(b);
```

Remember Lecture 2 A: Parameter Passing with Objects

```
public static void tryChanging(SomeObject obj) {
    obj.someField = 1;
    obj = new SomeObject();
    obj.someField = 2;
    return;
}
...
SomeObject someObj = new SomeObject();
tryChanging(someObj);
System.out.println(someObj.someField);
```

Remember Lecture 7 A: Sorting

Input

Unsorted array of elements

Behavior

Rearrange elements of array such that the smallest appears first, followed by the second smallest etc, finally followed by the largest element

Will This Work?

```
public static <AnyType extends Comparable<? super A
  void mergeSort( AnyType [] a) {
    AnyType[] ret = ....; // declare helper array
     .... // here goes a program that places
     .... // the element of "a" into "ret" so
     .... // that "ret" is sorted
    a = ret;
    return;
}
...
Integer[] myArray = ...;
IterativeMergeSort.mergeSort(myArray);
```

Will This Work?

```
public static <AnyType extends Comparable<? super A
  void mergeSort( AnyType [] a) {
    AnyType[] ret = ....; // declare helper array
    .... // here goes a program that places
    .... // the element of "a" into "ret" so
    .... // that "ret" is sorted
    a = ret;
    return;
}
...
Integer[] myArray = ...;
IterativeMergeSort.mergeSort(myArray);
```

Answer: No! The assignment a = ret; has no effect on myArray!

Next Week

- NP-Completeness
- Algorithm design techniques