Datastructures and Algorithms

Minimum Spanning Trees, Max Flow in Flow Networks, Traveling Salesperson

Adel Gavranović — ETH Zürich — 2025

Overview

`n.ethz.ch/~agavranovic`

Material

Webpage

Mail

1. Feedback regarding **code** expert

General things regarding code expert

- Many of you wrote something like:
  "Additional comparisons are going to increase time complexity."
- This is not always the case!
  - Some comparisons, if implemented efficiently, **do not necessarily change** the time complexity.
- Time complexity classes are not measured with stopwatches, but through analytical methods.
- The term *"Runtime"* can be ambiguous,
  but the exercise clearly specifies *asymptotic runtime*.

Any questions regarding **code** expert on your part?

2. Learning Objectives

Objectives

- ☐ Understand and be able to perform the following Algorithms by hand:
  - ☐ Kruskal (MST)
  - ☐ Jarnik-Prim-Dijkstra (MST)
  - ☐ Ford-Fulkerson (Max-Flow)
  - ☐ Edmonds-Karp (Max-Flow)
- ☐ Understand how UnionFind works and where one can use it
- ☐ Be able to implement a “2-approximation” of the (metric) TSP problem using MSTs and a DFS

3. Summary

Getting on the same page

- What did you cover in class?
- Is there anything that I need to be aware of?

4. Remark Bonus Exercises

Bonus Exercises

Bonus 1
It was clearly stated in the task description that standard library functions which significantly simplify the task are not allowed. This will be graded with 0 points.

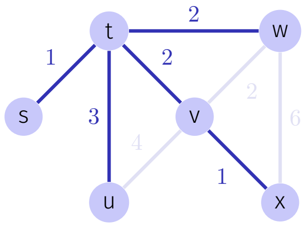
Bonus 2
Here, the use of all standard library functions is explicitly allowed!

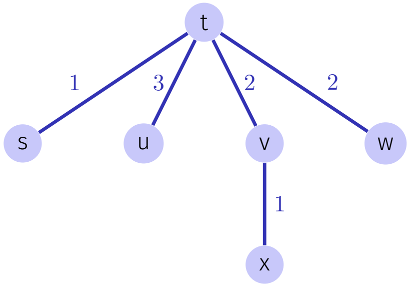
5. Minimum Spanning Trees

5. Minimum Spanning Trees

5.1. Recap Theory

Minimum Spanning Trees





(Solution is not unique.)

Jarnik-Prim-Dijkstra Algorithm

- Finds a minimum spanning tree.
- Starts from a single node and grows.
- Uses a priority queue.
- Evaluates edges, not paths.

Algorithm Jarnik-Prim-Dijkstra(G)

Input: A connected, undirected graph $G = (V, E)$ with weights w
Output: A minimum spanning tree T

```
Initialize $T = \emptyset$
Choose arbitrary vertex $v_0$ from $V$ and add $v_0$ to $T$
while $V \neq \emptyset$ do
    Choose edge $(u, v)$ with smallest weight such that $u$ is in $T$ and $v$ is in $V - T$
    Add $v$ to $T$
return $T$
```

Differences from Dijkstra’s Algorithm

- Jarnik-Prim-Dijkstra evaluates edges. Dijkstra evaluates paths.
- Jarnik-Prim-Dijkstra creates a minimum spanning tree. Dijkstra finds the shortest path.
- Jarnik-Prim-Dijkstra can handle negative weights, in contrast to Dijkstra.[1]

[1]The shortest path algorithm by Dijkstra can only deal with negative edge weights if it is known that no negative cycles can occur. And the algorithm can then still have an exponential runtime. It is therefore not practicable, if negative edges occur.

MakeSet, Union, and Find

- Make-Set(i): create a new set represented by i.
- Find(e): name of the set i that contains e .
- Union(i, j): union of the sets with names i and j.

In MST-Kruskal:

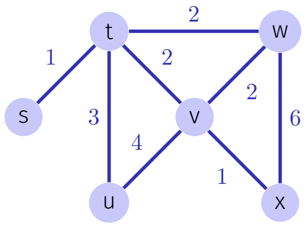
- Make-Set(i): New tree with i as root.
- Find(e): Find root of e
- Union(i, j): Union of the trees i and j.

Algorithm MST-Kruskal(G)

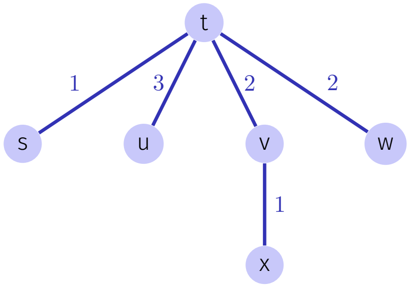
From the Lecture

```
**Input:** Undirected weighted Graph $G = (V, E, c)$
**Output:** MST of $G$
Sort edges by weight $c(e_1) \leq \ldots \leq c(e_m)$
$M \leftarrow \emptyset$
**for** $i = 1$ **to** $|V|$ **do**
    MakeSet($i$)
**for** $i = 1$ **to** $m$ **do**
    $(u, v) \leftarrow e_i$
    **if** Find($u$) $\neq$ Find($v$) **then**
        Union(Find($u$), Find($v$))
        $M \leftarrow M \cup \{e_i\}$
    // conceptually: else $D \leftarrow D \cup \{e_i\}$
**return** $(V, M, c)$
```

Representation as array



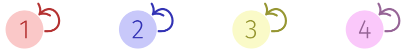
Index	s	t	w	v	u	x



Index	s	t	u	v	w	x
Parent	t	t	t	t	t	v

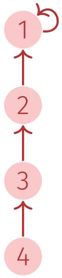
Running Time of Union-Find

From the Lecture



Parent	1	2	3	4
Index	**1**	**2**	**3**	**4**

$\Downarrow$ `Union`$(3,4)$, `Union`$(2,3)$, `Union`$(1,2)$



Parent	1	1	2	3
Index	**1**	2	3	4

Tree may degenerate... Running time of `Find`: $\Theta(n)$

Optimisation of the runtime for `Find`

From the Lecture

`MakeSet`(i)	$p[i] \leftarrow i$; $h[i] \leftarrow 1$; **return** i

`Union`(i, j)	**if** $h[j] > h[i]$ **then** swap(i, j)
	$p[j] \leftarrow i$
	if $h[i] = h[j]$ **then** $h[i] \leftarrow h[i] + 1$

$\Rightarrow$ Tree depth (and worst-case running time for `Find`) in $\Theta(\log n)$

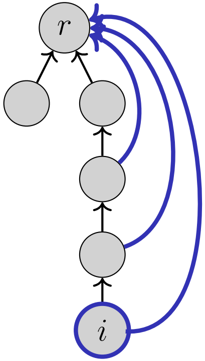
(Proof: lecture notes)

Alternative improvement

From the Lecture

Link all nodes to the root when `Find` is called.

```
ImprovedFind($i$):
$r \leftarrow$ Find($i$)
$v \leftarrow i$
while ($v \neq r$) do
    $p_v \leftarrow p[v]$
    $p[v] \leftarrow r$
    $v \leftarrow p_v$
return $r$
```



Cost: amortised *nearly* constant (inverse of the Ackermann-function).[2]

[2] When combined with union by size, cf. Cormen et al, Kap. 21.4

Running time of Kruskal's Algorithm

- Sorting of the edges: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$. [3]
- Initialisation of the Union-Find data structure $\Theta(|V|)$
- $|E|\times$ Union(Find(x),Find(y)): $\mathcal{O}(|E| \log |V|)$.

Overall $\Theta(|E| \log |V|)$**.**

[3]because G is connected: $|V| \leq |E| \leq |V|^2$

5. Minimum Spanning Trees

5.2. Code-Expert Exercise

Code-Example

’Kruskal MST’ on Code-Expert

6. MaxFlow

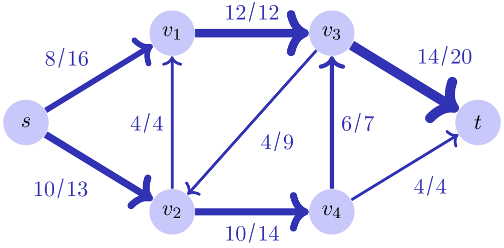
6.1. Recap Theory

Flow

A **Flow** $f : V \times V \to \mathbb{R}$ fulfills the following conditions:

- **Bounded Capacity**:
  For all $u, v \in V$: $f(u,v) \leq c(u,v)$.
- **Skew Symmetry**:
  For all $u, v \in V$:
  $f(u,v) = -f(v,u)$.
- **Conservation of flow**:
  For all $u \in V \setminus \{s,t\}$:

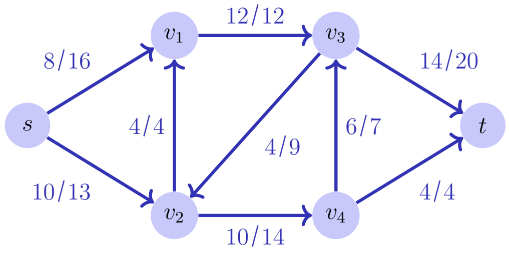
$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v).$$

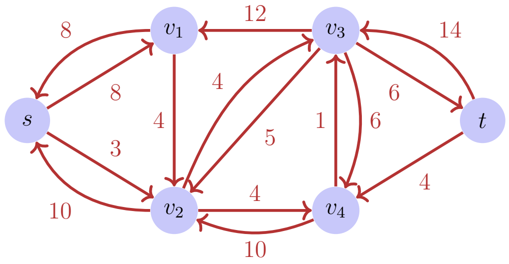


Value of the flow:
$|f| = \sum_{v \in V} f(s,v)$.
Here $|f| = 18$.

Residual Network

The **Residual network** G_f is provided by the edges with positive residual capacity. What does it look like for this flow network?





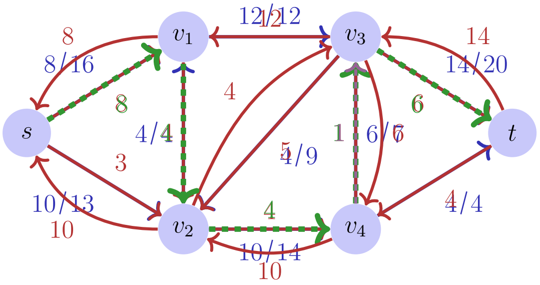
Residual networks provide the same kind of properties as flow networks with the exception of permitting antiparallel edges

Augmenting Paths

Augmenting Path p: simple path from s to t in the residual network G_f.

Residual Capacity $c_f(p)$: the least capacity along the expansion path p

$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ edge in } p\}$$



Note: There can be multiple augmenting paths!

Algorithm Ford-Fulkerson(G, s, t)

Input: Flow network $G = (V, E, c)$
Output: Maximal flow f.

```
for $(u, v) \in E$ do
    $f(u, v) \leftarrow 0$
while Exists path $p : s \rightsquigarrow t$ in residual network $G_f$ do
    $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\}$
    foreach $(u, v) \in p$ do
        if $(u, v) \in E$ then
            $f(u, v) \leftarrow f(u, v) + c_f(p)$
        else
            $f(v, u) \leftarrow f(u, v) - c_f(p)$
```

Edmonds-Karp Algorithm

Choose in the Ford-Fulkerson-Method for finding a path in G_f the expansion path of shortest possible length (e.g. with BFS)

Theorem 1

When the Edmonds-Karp algorithm is applied to some integer valued flow network $G = (V, E)$ with source s and sink t then the number of flow increases applied by the algorithm is in $\mathcal{O}(|V| \cdot |E|)$

$\Rightarrow$ ***Overall asymptotic runtime:*** $\mathcal{O}(|V| \cdot |E|^2)$

Max-Flow Min-Cut Theorem

Theorem 2

Let f be a flow in a flow network $G = (V, E, c)$ with source s and sink t. The following statements are equivalent:

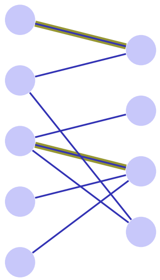
1. *f is a maximal flow in G*
2. *The residual network G_f does not provide any expansion paths*
3. *It holds that $|f| = c(S, T)$ for a cut (S, T) of G.*

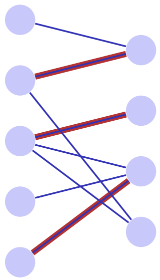
Application: maximal bipartite matching

Given: bipartite undirected graph $G = (V, E)$.

Matching M: $M \subseteq E$ such that $|\{m \in M : v \in m\}| \leq 1$ for all $v \in V$.

Maximal Matching M: Matching M, such that $|M| \geq |M'|$ for each matching M'.

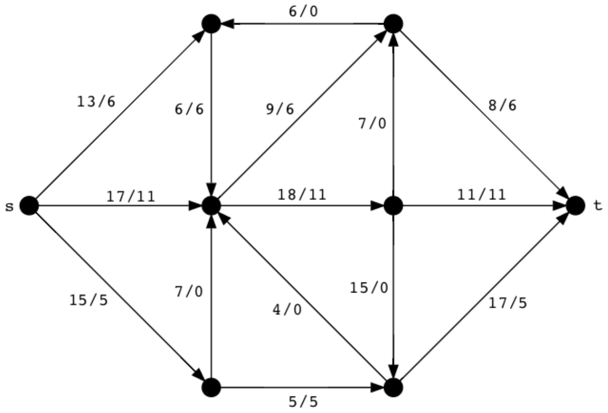




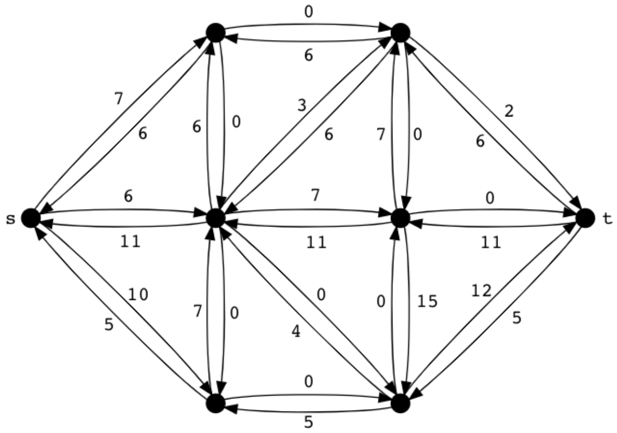
6.2. Quiz

Manual Max Flow Exercise

This graph shows a flow chart that is not at maximum capacity. Run one iteration of the Ford-Fulkerson algorithm to find the max flow.

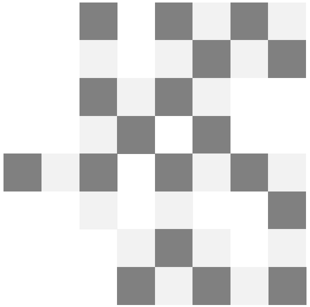


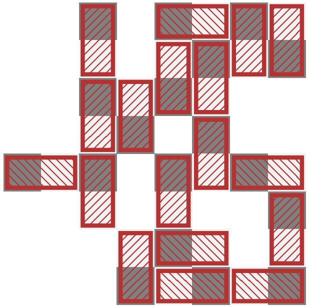
Manual Max Flow Solution



update not shown since it is not unique!

Max Flow Question



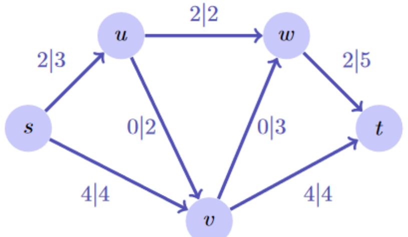


Let an $n \times n$ chessboard be given without some squares. Describe an efficient algorithm to find out if the board can be completely covered with dominoes. Then model the problem as a flow problem.

6.3. Old Exam Questions

Exam Question Example

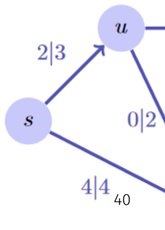
Gegeben ist das folgende Flussnetzwerk G mit Quelle s und Senke t. Die einzelnen Kapazitäten c_i und Flüsse ϕ_i sind an den Kanten angegeben als $\phi_i|c_i$. Geben Sie den Wert des Flusses f an.



Provided in the following is a flow network G with source s and sink t. Capacities c_i and flows ϕ_i are provided at the edges as $\phi_i|c_i$. Provide the value of the flow f.

$|f| = \boxed{\quad}$

Gegeben ist da
mit Quelle s un
pazitäten c_i und
ten angegeben a
des Flusses f a



7. TSP

Travelling Salesperson Problem

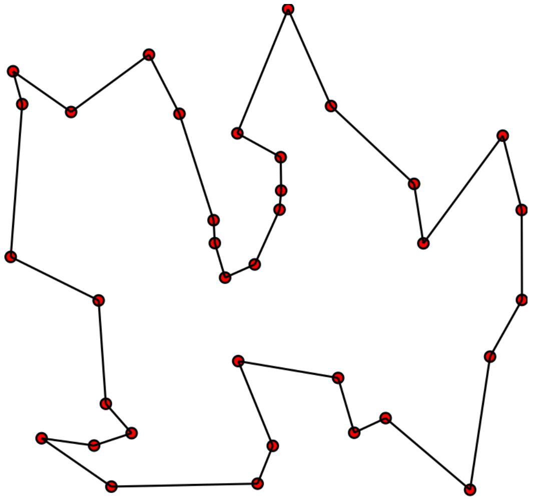
Problem

Given a map and list of cities, what is the shortest possible route that visits each city once and returns at the original city?

Mathematical model

On an undirected, weighted graph G, which cycle containing all of G's vertices has the lowest weight sum?

Travelling Salesperson Problem



Travelling Salesperson Problem

- The problem has no known polynomial-time solution.
- Many heuristic algorithms exists. They do not always return the optimal solution.

Travelling Salesperson Problem

- The heuristic algorithm that you are asked to implement on CodeExpert (*The Travelling Student*) on CodeExpert uses an MST:

  1. Compute the minimum spanning tree M
  2. Make a depth first search on M

- The algorithm is 2-approximate, meaning that the solution it generates has at most twice the cost of the optimal solution.
- The algorithm assumes a complete graph $G = (V, E, c)$ that satisfies the triangle inequality: $\forall v, w, x \in V : c(v, w) \leq c(v, x) + c(x, w)$

8. Outro

General Questions?

See you next time!

Have a nice week!