# Flüsse, Schnitte, bipartite Graphen

Vlad Popa

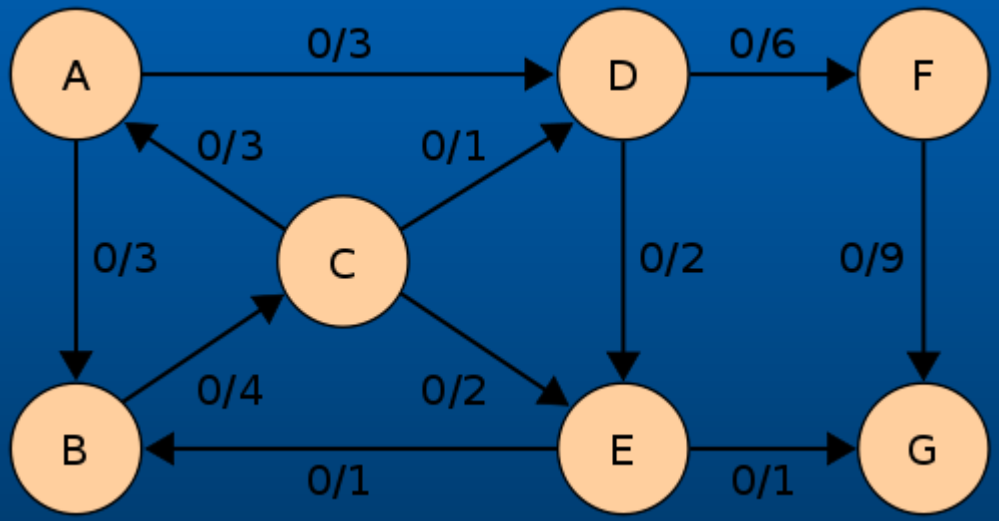
08.06.2010

# Inhaltsverzeihnis

# Flussnetzwerke und Flüsse

- **Definition:** Ein Flussnetzwerk G=(V, E) ist ein gerichteter Graph, in dem jede Kante (u, v) ϵ E eine nichtnegative Kapazität c(u, v) >= 0 hat.
- Zwei Knoten spielen eine besondere Rolle: Quelle s, Senke t.
- Ein Fluss in G ist die Funktion f: VxV -> |R, die folgende Bedingungen erfüllt:
  - Kapazitätsbeschränkung: f(u, v) <= c(u, v)
  - Asymmetrie: f(u, v) = - f(v, u)
  - Flusserhaltung: Bei einem Knoten muss der Zufluss gleich dem Abfluss sein. (Ausnahme: Quelle und Senke)

# Kleines Beispiel



# Algorithmen

1. Ford- Fulkerson

# Ford- Fulkerson Algorithmus

1. Der Algorithmus sucht einen Pfad (Erweiterungspfad) vom Start- zum Zielknoten im Restnetzwerk G.
2. Der Fluss wird mit der kleinsten Kapazität, cmin, entlang des Pfades erhöht.
3. Der Fluss auf einer Kante (u, v) des Pfades wird folgendermaßen aktualisiert:

   f[u, v] = f[u, v] + cmin;

   f[v, u] = -f[u, v];
4. Falls es noch einen Erweiterungspfad gibt springe wieder zu 1.

- Ford-Fulkerson(G; q; s)

```
1 for alle Kanten (u, v) ϵ E
2     do f [u, v]  = 0
3          f [v, u]  = 0
4 while es existiert ein Pfad p von Q nach S im
      Restnetzwerk G
5      do
6              c (p) = min{c (u, v) : (u, v) gehört zu p}
7              for alle Kanten (u, v) von p
8                      do f [u, v] = f [u, v] + c(p)
9                           f [v, u]  = -f [v, u]
```

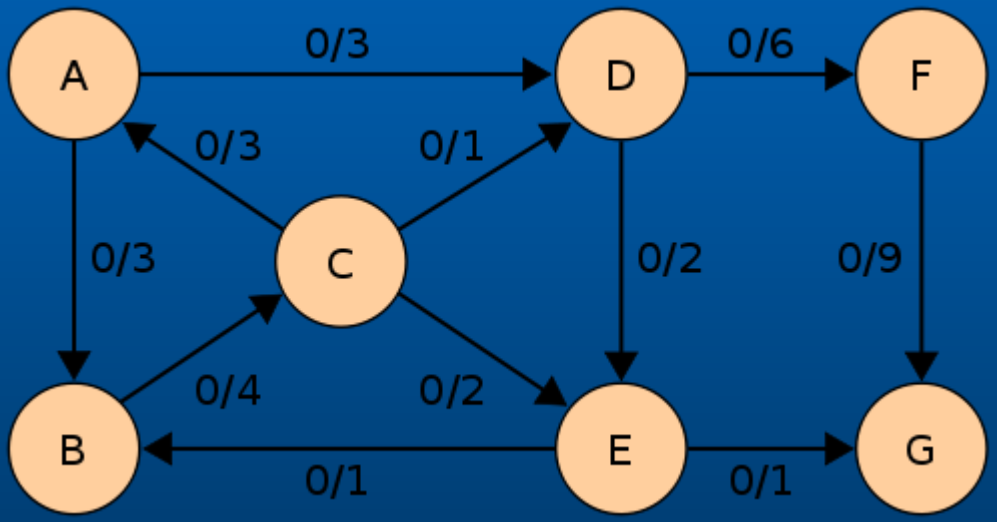
# Algorithmen

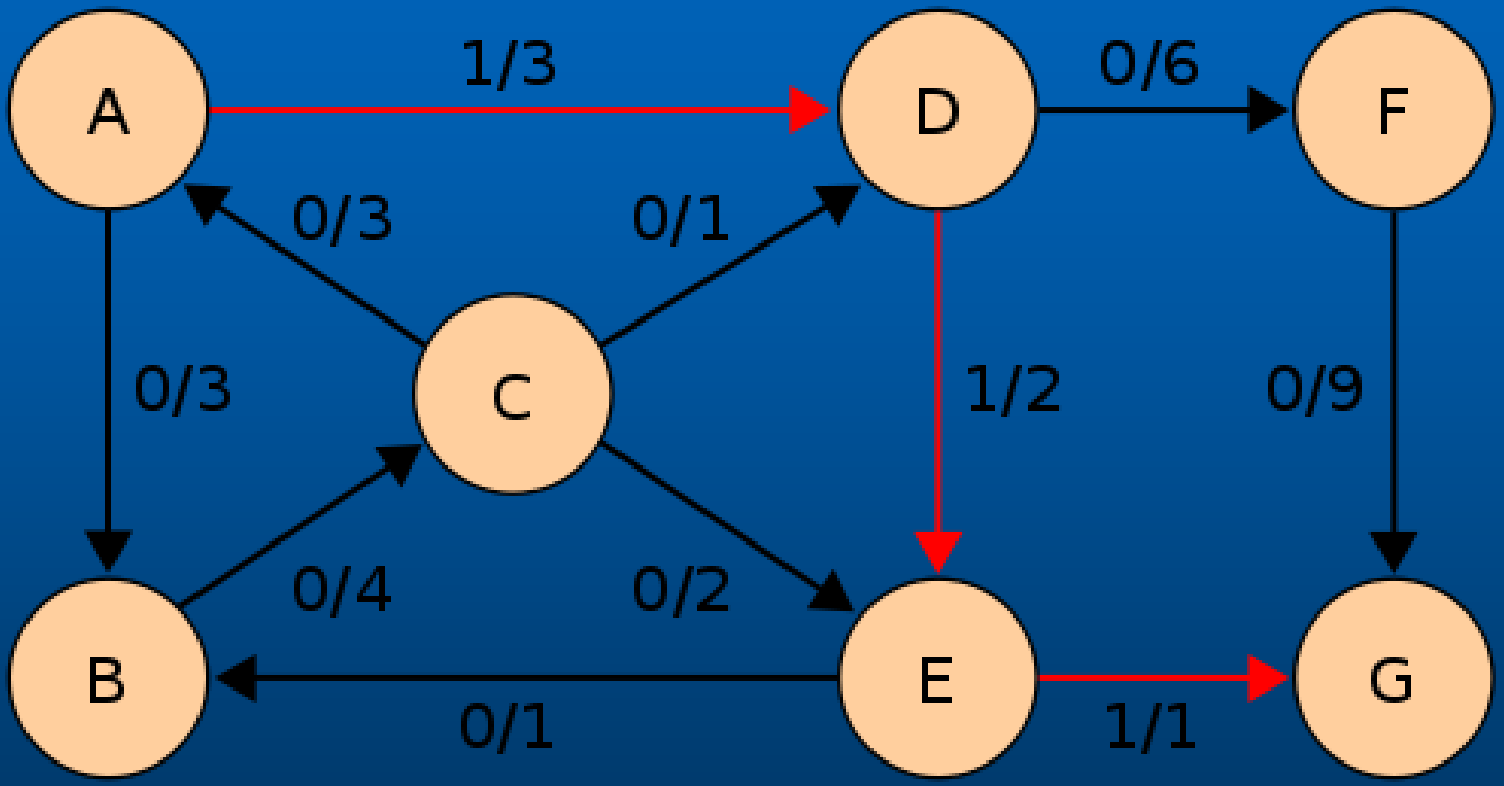
1. Ford- Fulkerson
2. Edmond Karp

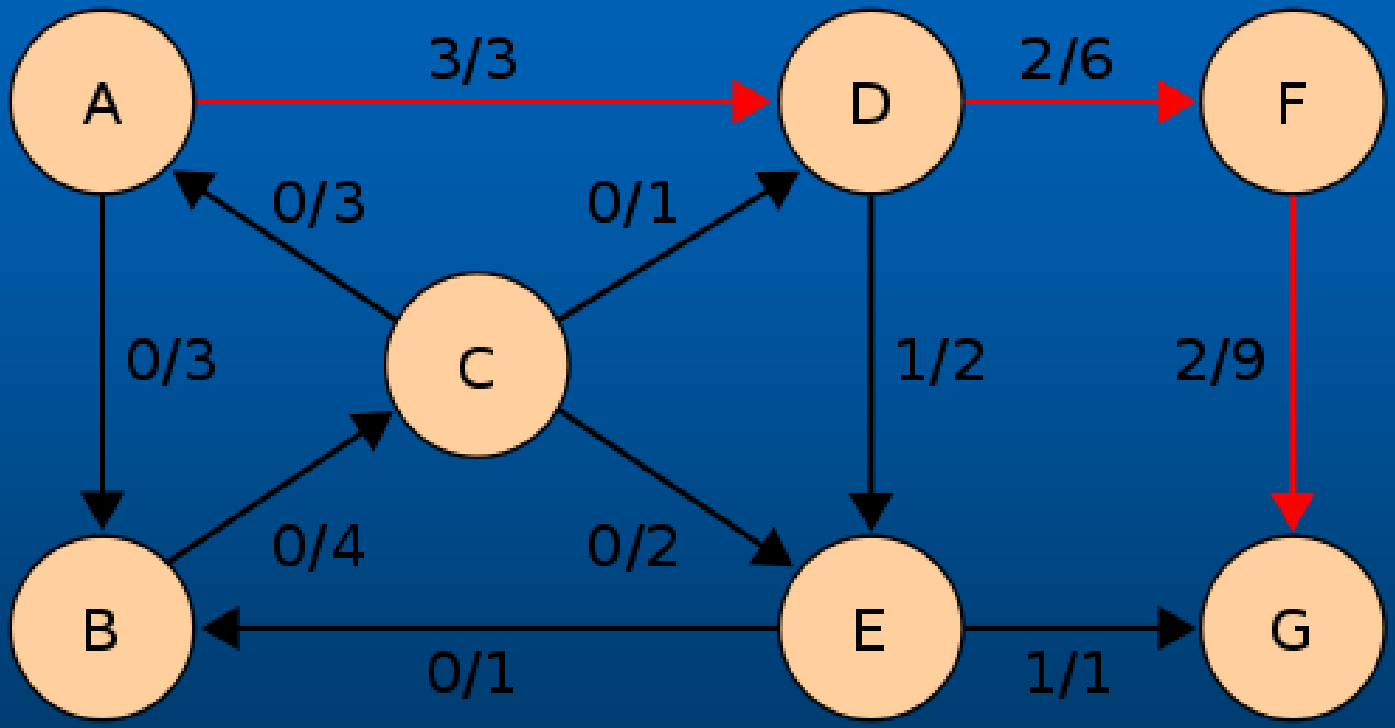
# Edmonds- Karp Algorithmus

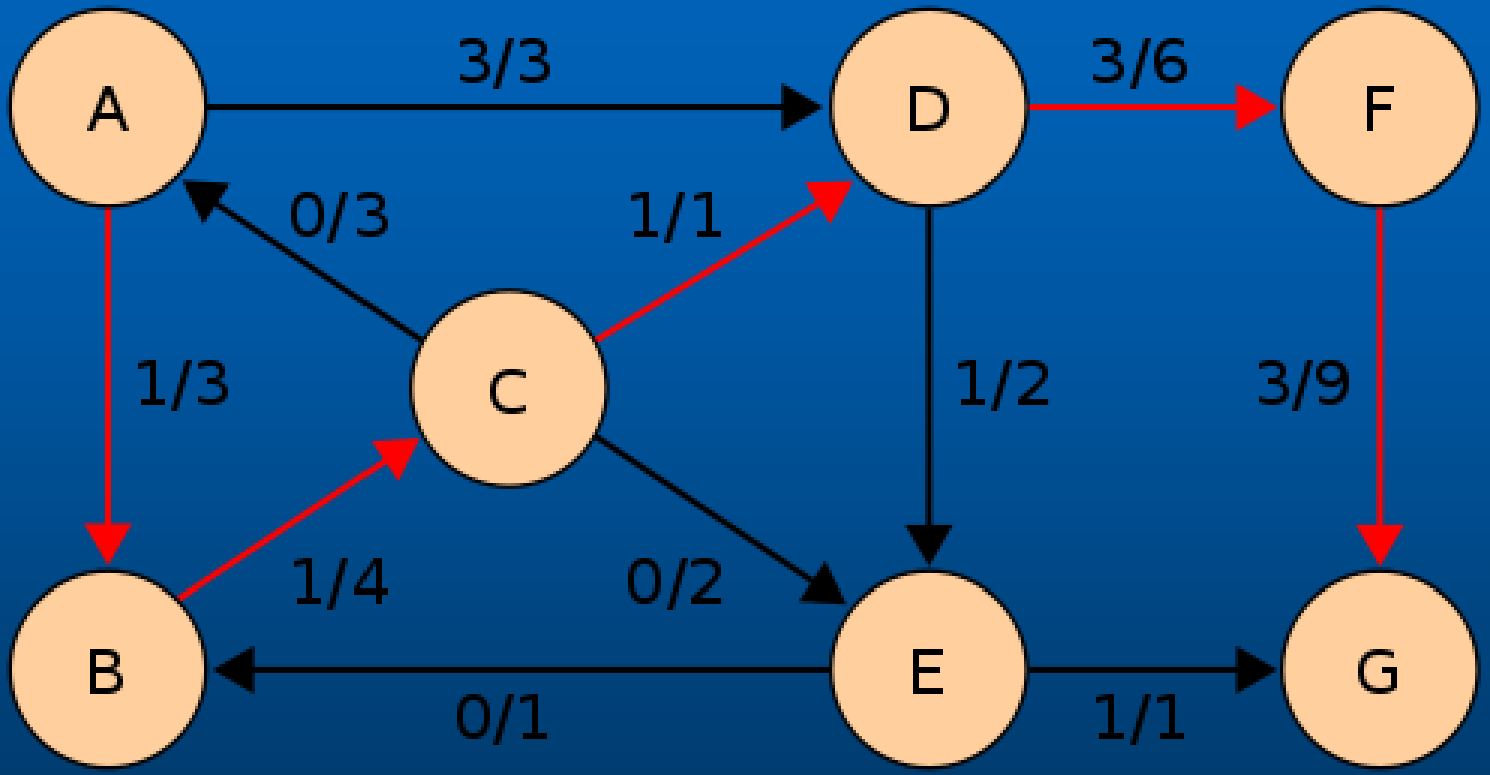
- Benutzt dieselbe Idee wie Ford- Fulkerson.
- Unterschied: Jedes mal wird der Erweiterungspfad durch eine Breitensuche im Restnetzwerk gesucht.

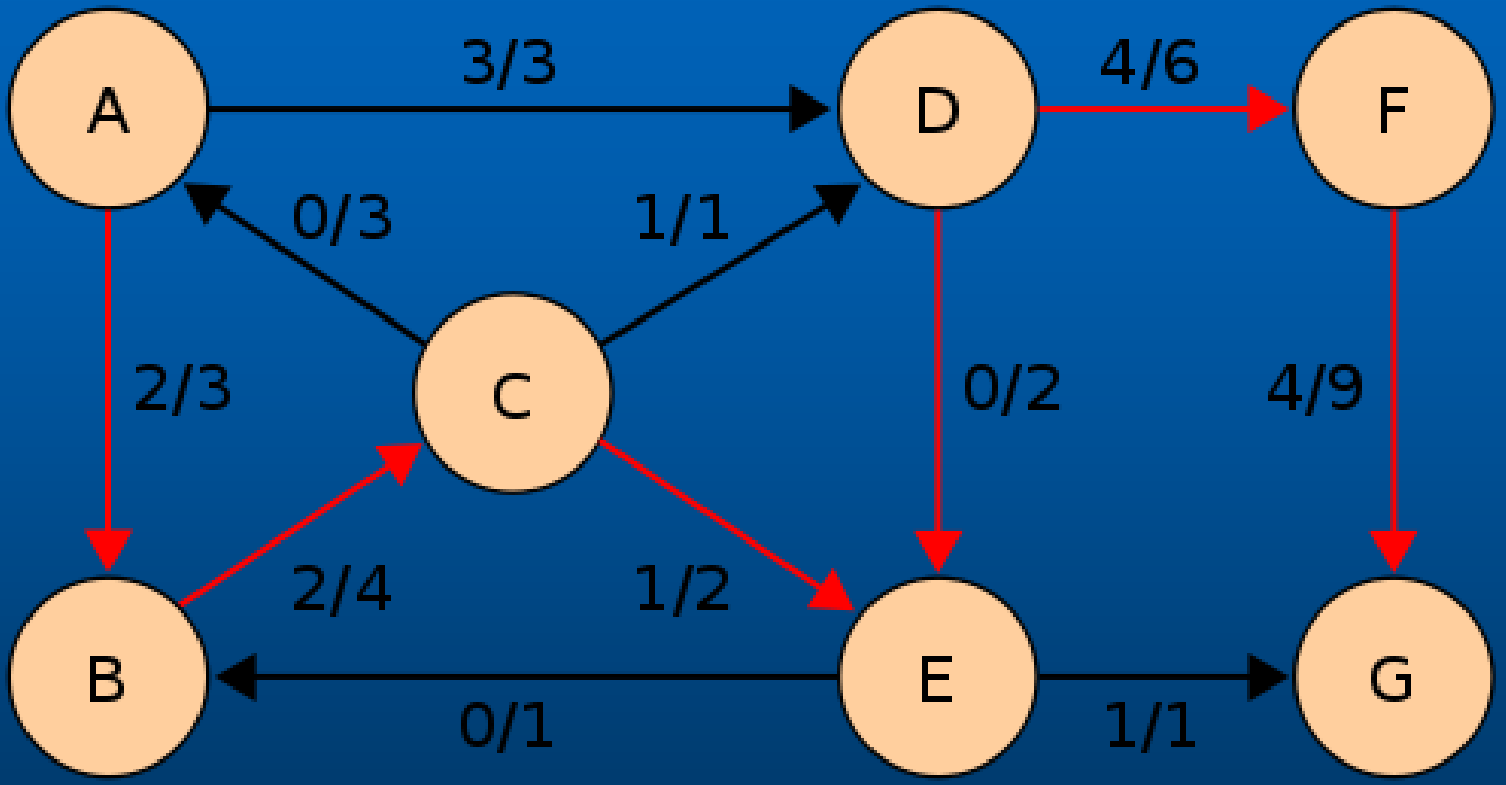
# Beispiel Forts.




A
D
F
C
B
E
G
1/3
0/6
0/3
0/1
0/3
1/2
0/9
0/4
0/2
0/1
1/1


A
D
F
C
B
E
G
3/3
2/6
0/3
0/1
0/3
1/2
2/9
0/4
0/2
0/1
1/1


A
D
F
C
B
E
G
3/3
3/6
0/3
1/1
1/3
1/2
3/9
1/4
0/2
0/1
1/1


A
D
F
C
B
E
G
3/3
4/6
0/3
1/1
2/3
0/2
4/9
2/4
1/2
0/1
1/1

# Algorithmen

1. Ford- Fulkerson
2. Edmond Karp
3. Dinic

# Dinic Algorithmus

1. Man modifiziert das Netzwerk G, so dass im neuen Netzwerk G' nur diejenigen Kanten bleiben, die auf einem optimalen Weg von s nach t führen. (Durch BFS). Es entsteht ein DAG.
2. Durch eine DFS kann man rekursiv den Fluss auf mehreren Pfaden aktualisieren: Sei

   P1: s -> ...d1...-> n-> .... -> t

   P2: s -> ...d1...-> n -> ... -> t.

   => Wenn in n für P1 B Fluss kommt und mit C kann man insgesamt aufpumpen => für P2: C – B Fluss
3. Wenn 2. den Fluss verändert beginne wieder mit 1.

# Komplexität

- Ford- Fulkerson: Hängt stark von der Wahl des erweiternden Weges ab. Grundsätzlich ist hier die Komplexität O(E * flow)

- Ford-Fulkerson(G; q; s)

```
1 for alle Kanten (u, v) ϵ E
2     do f [u, v]  = 0
3          f [v, u]  = 0
4 while es existiert ein Pfad p von Q nach S im
      Restnetzwerk G                          -> O(flow)
5      do
6              c (p) = min{c (u, v) : (u, v) gehört zu p}
7              for alle Kanten (u, v) von p          -> O(E)
8                      do f [u, v] = f [u, v] + c(p)
9                           f [v, u]  = -f [v, u]
```

# Komplexität

- Ford- Fulkerson: Hängt stark von der Wahl des erweiternden Weges ab. Grundsätzlich ist hier die Komplexität O(E * flow)
- Edmond- Karp: O(V * E^2)

# Komplexität

- Ford- Fulkerson: Hängt stark von der Wahl des erweiternden Weges ab. Grundsätzlich ist hier die Komplexität O(E * flow)
- Edmond- Karp: O(V * E^2)
- Dinic: O(V^2 * E)

# Schnitte

- Definition: ein q- s- Schnitt ist eine Aufteilung des Graphen in zwei Mengen, wobei die Knoten q und s in verschiedenen Mengen liegen.
- Minimaler Schnitt: Schnitt, bei dem die Gesamtkapazität über den Schnitt minimal ist. (c (s, t) - minimal)

# Äquivalenz: Maximaler Fluss – minimaler Schnitt

Wenn f ein Fluss in einem Flussnetzwerk G = (V, E) mit der Quelle s und der Senke t ist, dann sind die folgenden Bedingungen äquivalent:

1. f ist ein maximaler Fluss in G.
2. Das Restnetzwerk Gf enthält keine Erweiterungspfade.
3. Es gilt |f| = c(S, T) (für einen Schnitt (S, T) von G).

# Pseudocode

MinCut(G; q; s)

- Berechne maximalen Fluss
- Starte DFS von q im Restnetzwerk
- A = von q erreichbare Knoten
- return A, G \ A

# Maximaler Fluss bei minimalen Kosten

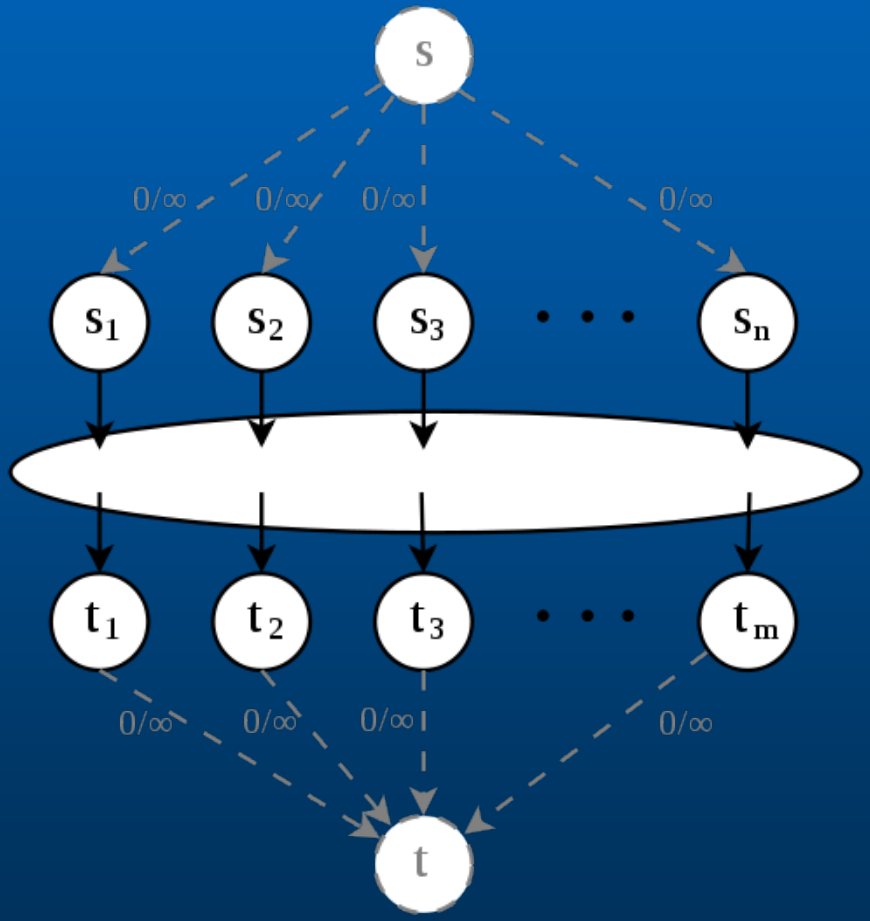
- Jede Kante besitzt zusätzlich Kosten, die angeben, wie teuer es ist, eine Einheit Fluss über diese Kante zu leiten.
- f[u, v] < 0 => cost[u, v] = -cost[v, u].
- Endziel: $\sum_{(u, v) \in E}$ f[u, v] * cost[u, v] soll minimal sein.

# Standardaufgabe

Es gibt n Arbeiter und n Aufgaben, die gelöst werden müssen. Jeder Arbeiter hat eine Liste, in welcher für jede Aufgabe, die Zeit für ihre Bearbeitung angegeben wird.

Gesucht wird die minimale Endzeit nachdem alle Aufgaben bearbeitet wurden.

# Graphmodellierung



# Bipartite Graphen

- Definition:

  Ein Graph G = (V, E) heißt bipartit, genau dann wenn V=V1 U V2 und V1 ∩ V2 = { } und für jede Kante e existieren zwei Knoten v1 ϵ V1, v2 ϵ V2, so dass e={v1,v2}.

# Matchings

- Definitionen:

Sei G = (V, E) ein Graph.

- M heißt Matching in G, falls M eine Teilmenge von E ist und alle Kanten in M paarweise disjunkt sind (kein Knoten ist zu mehr als einer Kante inzident)

- M heißt perfektes Matching, falls jeder Knoten durch genau eine Kante aus M überdeckt ist. (|M|=|V|/2)

- |M| ist die Größe des Matchings.

# Das Heiratsproblem

- Gegeben seien heiratswillige Damen und Herren. Jede Dame gibt an, mit welchem der Herren sie sich eventuell vermählen würde.
- Das Problem besteht nun darin, möglichst viele Damen so zu verheiraten, dass jede Dame einen Herren ihrer Wahl erhält, und dass selbstverständlich keine zwei Damen mit demselben Herrn verheiratet sind.

# Minimal vertex cover

- Definition: Ein “Vertex Cover” ist eine Menge von Knoten in einem beliebigen Graph, wobei jede Kante zu wenigstens einem Knoten inzident ist.
- Ein Vertex  Cover ist minimal genau dann wenn es die minimale Anzahl von Knoten hat.
- Normalerweise ist das ein NP- hartes Problem.
- Ist das auch für bipartite Graphen so?

# Minimal vertex cover

- Definition: Ein “Vertex Cover” ist eine Menge von Knoten in einem beliebigen Graph, wobei jede Kante zu wenigstens einem Knoten inzident ist.
- Ein Vertex  Cover ist minimal genau dann wenn es die minimale Anzahl von Knoten hat.
- Normalerweise ist das ein NP- hartes Problem.
- Ist das auch für bipartite Graphen so?

**Nein!**

# Minimal edge cover

- Definition: Ein “Edge Cover” ist eine Menge von Kanten in einem beliebigen Graphen, wobei jeder Knoten wenigstens eine Kante hat die zu ihm inzident ist.
- Ein Edge Cover ist minimal genau dann wenn es die minimale Anzahl von Kanten hat.
- Für ein bipartiter Graph entspricht er einem Matching.

# Maximum independent Set

- Definition: Ein “Independent Set“ ist eine Menge von Knoten in einem Graph, wo keine zwei Knoten durch eine Kante verbunden sind.
- Ein “Independent Set” ist genau dann maximal, wenn beim Einfügen eines beliebigen Knotens die Definition verletzt wird.
- Der Komplement eines solcher Menge ist immer ein vertex cover => Komplement eines Maximum Independent Sets ist ein Minimal Vertex Cover

# Eigenschaften bipartiter Graphen

- Ein Graph ist bipartit wenn er keinen Zyklus ungerader Länge enthält. (er enthält keine Clique >= 3)
- Größe des minimum vertex covers = Größe des maximalen Matchings. (König´s Theorem)
- Maximum independent set + maximaler Matching = |V|
- Größe des minimum edge covers = Größe des maximalen independent sets.
- Jeder bipartite Graph ist mit zwei Farben färbbar.

# Algorithmus für maximalen Matching

```
int match (int n) {
    mark[n] = 1;

    for (int i = 0; i < G[n].size(); ++ i)
        if (!d[G[n][i]]) {
            d[G[n][i]] = n;          p[n] = G[n][i];
            return 1;
        }

    for (int i = 0; i < G[n].size(); ++ i)
        if (mark[d[G[n][i]]] != 1 && match(d[G[n][i]])) {
            d[G[n][i]] = n;          p[nod] = G[n][i];
            return 1;
        }

    return 0;
}
```

# Aufgaben

- Joc4
- Felinare
- Strazi
- Critice
- Two Shortest
- Dijkstra, Dijkstra
- Dominoes
- Ghizi
- Paznici
- Algola