Discrete Mathematics

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Course preparation group:
Nguyễn Khánh Phương
Đỗ Phan Thuận
Phạm Quang Dũng
Huỳnh Thanh Bình
Trần Vĩnh Đức
Bùi Quốc Trung
Đinh Viết Sang
Ban Hà Bằng

Content of Part 2

PART 1
COMBINATORIAL THEORY
(Lý thuyết tổ hợp)

PART 2
GRAPH THEORY
(Lý thuyết đồ thị)

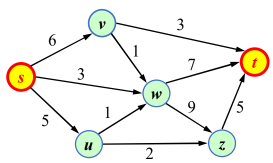
Contents

Network

Network is a directed graph G = (V, E):

- There is only one vertex *s* without any incoming arcs, called source vertex and only vertex *t* without any outgoing arcs called target vertex.
- Each edge *e* of *G* is assigned a nonnegative value *c*(*e*) which is called capacity of *e*.

Example:



Flow in network

Definition. Flow f in network $G=(V,E)$ is to assign value $f(e)$ on each edge e ($f(e)$ is flow on edge e) so that following conditions are satisfied:

1) Capacity rule:

For each edge e, $0 \le f(e) \le c(e)$

2) Conservation Rule (Điều kiện cân bằng luồng): Each $v \neq s, t$

$$\sum_{e \in E^-(v)} f(e) = \sum_{e \in E^+(v)} f(e)$$

Definition. Value of flow f is

$$val(f) = \boxed{\sum_{e \in E^+(s)} f(e)} \overset{(*)}{=} \boxed{\sum_{e \in E^-(t)} f(e)}$$

Edges going out of *s* Edges going into *t*

(Equation (*) is obtained by summing up all the conservation rules.)

Flow in network

Definition. Flow f in network $G=(V,E)$ is to assign value $f(e)$ on each edge e ($f(e)$ is flow on edge e) so that following conditions are satisfied:

1) Capacity rule:

For each edge e, $0 \le f(e) \le c(e)$

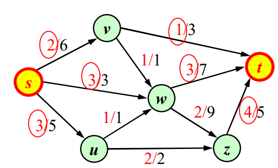
2) Conservation Rule (Điều kiện cân bằng luồng): Each $v \neq s, t$

$$\sum_{e \in E^-(v)} f(e) = \sum_{e \in E^+(v)} f(e)$$

$E^-(v)$ and $E^+(v)$ are sets of arcs entering and leaving vertex *v*, respectively.

Flow in network: Example

Example:



- 2 values on each edge: value of flow on edge is in red, the other is capacity of edge.
- Conditions 1) and 2) are satisfied => f is flow on the network.
- Value of flow:

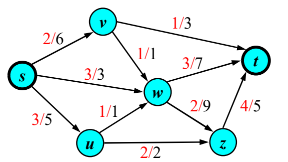
$$8 = f(s,v) + f(s,u) + f(s,w) = f(v,t) + f(w,t) + f(z,t)$$

$$val(f) = \sum_{e \in E^+(s)} f(e) \overset{(*)}{=} \sum_{e \in E^-(t)} f(e)$$

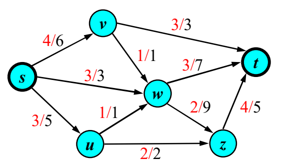
Maximum Flow problem

A flow in network G is called maximum flow if among all flows in G, it is the one having maximum value.

Maximum flow problem is the problem aiming to determine a feasible flow through the network with maximum value



Flow with value 8 = 2 + 3 + 3 = 1 + 3 + 4



Maximum flow with value 10 = 4 + 3 + 3 = 3 + 3 + 4

Contents

Some applications

Network	Vertex	Arc	Flow
Network	transaction stations, computers, satellites	cables	voice, video, packets
Electrical network	gates, registers, processors	conductors	Power circuit
Mechanical	joints	rods, beams, springs	heat, energy
Irrigation	pumping stations, water source	pipelines	Liquid, water
Finance	bank	transitions	money
Traffic	airport, station	highways, flights	Goods, customers
Chemistry	sites	bonds	energy

Cut

Cut is a partition vertex set of the network into 2 disjoint sets S and T with $s \in S$, $t \in T$.

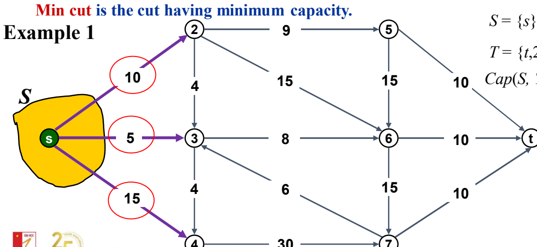
- Capacity $cap(S,T)$ of the cut (S, T) is:

$$cap(S,T) = \sum_{e \in S \to T} c(e),$$

where $S \to T := \{(v,w) \in E : v \in S, w \in T\}$

Min cut is the cut having minimum capacity.

Example 1



$S = \{s\}$

$T = \{t,2,3,4,5,6,7\}$

$Cap(S, T) = 10+5+15=30$

Flow and cut

Lemma 1. Assume f is flow, and (S, T) is a cut. Then

Flow through this cut is equal to the value of flow f:

$$\boxed{\sum_{e \in S \to T} f(e) - \sum_{e \in T \to S} f(e)} = \boxed{\sum_{e \in E^+(s)} f(e) = \sum_{e \in E^-(t)} f(e) = val(f)}$$

Max Flow and Min Cut

Corollary. Assume f is a flow, (S, T) is a cut. If $\text{val}(f) = \text{cap}(S, T)$, then f is maximum flow and (S, T) is minimum cut.

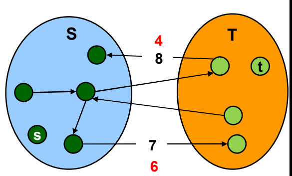
Flow and cut

Lemma 2. Assume f is a flow, (S, T) is a cut. Then

$$\text{val}(f) \le \text{cap}(S, T).$$

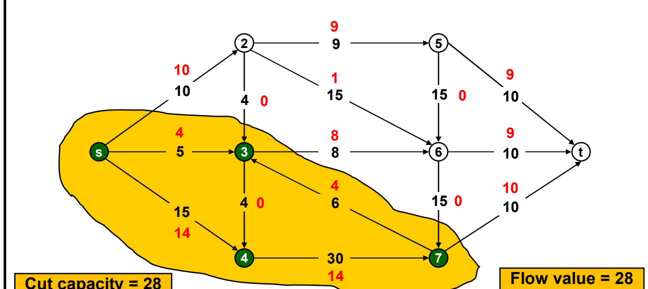
Proof

$$\begin{aligned} val(f) &= \sum_{e \in S \to T} f(e) - \sum_{e \in T \to S} f(e) \\ &\le \sum_{e \in S \to T} f(e) \\ &\le \sum_{e \in S \to T} c(e) \\ &= \text{cap}(S, T) \end{aligned}$$



Max-Flow Min-Cut Theorem

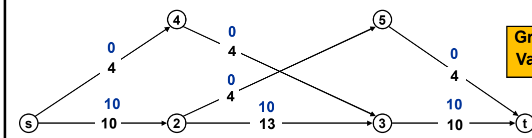
Theorem (Ford-Fulkerson, 1956): In a network, value of max flow is equal to capacity of min cut.



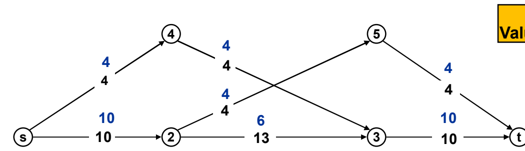
Contents

Greedy algorithm

Greedy algorithm does not provide optimal solution.



Greedy algorithm: Value of flow = 10



Optimal: Value of flow = 14

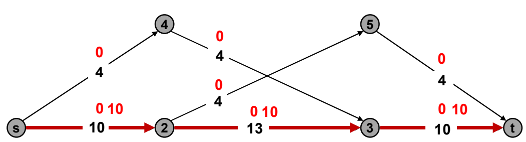
Greedy algorithm

Greedy algorithm:

- Starting with flow 0 (value of flow = 0).
- Find path P from *s* to *t* such that each edge e satisfies *f* (*e*)<*c*(*e*).
- Augument flow along P.
- Repeat until can not find P

s->4->3->t ???

s->2->5->t ???

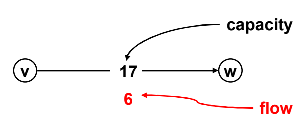


Value of flow = 10

Residual graph

Network G = (V, E).

- Arc $e = (v, w) \in E$
- Flow $f(e)$
- Capacity $c(e)$

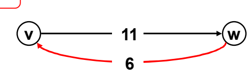


$e = (u,v) \Rightarrow e^R = (v,u)$

Residual graph: $G_f = (V, E_f)$.

- $E_f = \{e: f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$
- Capacity of each arc e

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



Theorem about maximum flow and minimum cut

Augmenting path Theorem (Ford-Fulkerson, 1956): Flow is maximum if and only if there does not exist augmenting path on network.

Max flow and min cut Theorem (Ford-Fulkerson, 1956): the maximum possible flow in a network (from source to sink) is exactly equal to the minimum capacity of all possible cuts.

We will prove the following Theorem:

Theorem. Assume f is a flow in network. The following three statements are equivalent

- (i) Can find the cut (S, T) such that $\mathrm{val}(f) = \mathrm{cap}(S, T)$.
- (ii) f is maximum flow.
- (iii) Could not find augmenting path to augment value flow f.

Ford – Fulkerson algorithm

Augmenting flow *f* along path P

```
float Augment(f,P)
{
   b ← cf(P)
   FOR e ∈ P
     IF (e ∈ E)
        f(e) ← f(e) + b
     ELSE
        f(eR) ← f(e) - b
   RETURN f
}
```

Ford-Fulkerson algorithm

```
float Ford_Fulkerson(G,c,s,t)
{
   FOR e ∈ E   // initialize flow f to 0
      f(e) ← 0
   Gf ← residual graph for f

   WHILE (there exists an augmenting path P)
   {
      f ← augment(f, P)
      Update residual graph Gf
   }
   RETURN f
}
```


Contents

Computation time

Question: Is the Ford-Fulkerson algorithm a polynomial algorithm? (algorithm with computation time is bounded by a fixed degree polynomial of the input length)

- Answer: Not at all. If the maximum capacity is C then the algorithm may have to do C iterations.

The following example shows how the algorithm can take a lot of iteration

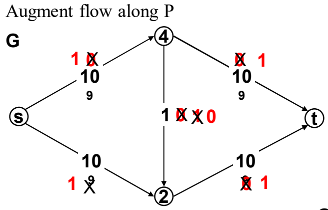
Ford – Fulkerson algorithm is not polynomial algoirthm

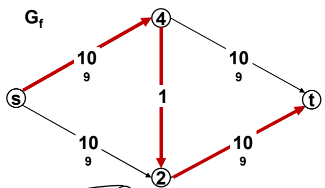
Augmenting path P: *s*, 4, 2, *t*

Augment flow along P

Augmenting path P': *s*, 2, 4, *t*

Augment flow along P'

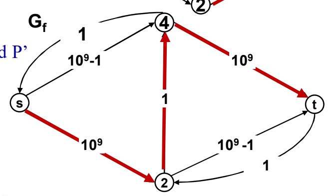




Algorithm executes alternating augmenting flow along two paths P and P'

(*s*, 4)
(*s*, 2)
(4, 2)
(2, *t*)
(4, *t*)

2×10^9 iterations



Contents

1. Problem description and applications
2. Cut
3. Residual graph and Augmenting path
4. Ford-Fulkerson algorithm
5. **Edmond-Karp algorithm**
6. Some applications

Computation time

Question: Is the Ford-Fulkerson algorithm a polynomial algorithm? (algorithm with computation time is bounded by a fixed degree polynomial of the input length)

- Answer: Not at all. If the maximum capacity is C then the algorithm may have to do C iterations.

If the capacities of arcs on network is real, there is an example that the Ford-Fulkerson algorithm does not stop.

Zwick constructs examples showing that the algorithm may not stop, if the capacity of arcs is irrational

How to select the augmenting path?

Be careful when select augmenting path, because

- Several options may lead to an exponential algorithm.
- Smart choice leads to polynomial algorithm.
- If capacity is irrational number, the algorithm may not stop

The goal: select augmenting path such that:

- Can find the augmenting path effectively.
- The algorithm requires as few iterations as possible..

Select augmenting path with

- Maximum capacity. (fat path)
- Capacity is large enough. (capacity scaling)
- The number of edges along the way is the least. (shortest path)

Edmond-Karp algorithm **Applying BFS**

5. Edmonds – Karp Algorithm

Edmonds and Karp, *JACM* 1972

- If the augmenting path is the shortest path from s to t, then the computation time of the algorithm is $O(|E|^2 |V|)$.

Applications

- Some general flow problems:
  - The problem with multiple sources and sinks
  - The problem with limited capacities at nodes
- Airline scheduling
- Image segmentation
- Bipartite matching
- Data mining
- Project selection
- Network connectivity
- …

Ford-Fulkerson algorithm

```
float Ford_Fulkerson(G,c,s,t)
{
   FOR e ∈ E   // initialize flow to 0
      f(e) ← 0
   Gf ← residual graph for flow f

   WHILE (there exists an augmenting path P)
   {  f ← augment(f, P)
      update Gf
   }
   RETURN f
}
```

Find augmenting path by BFS:

- Easy to implement
- Augmenting path has the least edges

$O(|E|^2 |V|)$

Edmonds – Karp algorithm

```
FOR e ∈ E
   f(e) ← 0
Gf ← residual graph for flow f

WHILE (there exists an augmenting path)
{
   find augmenting P by BFS
   f ← augment(f, P)
   update Gf
}
RETURN f
```