Longest Common Subsequence

**Given:** Two sequences \( x = x_1 \, x_2 \, ... \, x_n \) and \( y = y_1 \, y_2 \, ... \, y_m \), over some alphabet \( A \).

**Find:** Longest common subsequence \( z = z_1 \, ... \, z_k \) of \( x \) and \( y \).

**Example:**
\[
\begin{align*}
x &= TGACTA \\
y &= GTGCATG
\end{align*}
\]

LCS \( z = TGCA, \) or TGAT, or TGCT.

**Step 1:** Array of optimal numerical values for sub-problems
\[
A(i, j) = \begin{cases} 
\text{length of LCS}(x_1 \ldots x_i, y_1 \ldots y_j) \\
\end{cases}
\]

**Step 2:** Recurrence
\[
A(i, 0) = A(0, j) = 0 \\
A(i, j) = \begin{cases} 
A(i-1, j-1) + 1 & \text{if } x_i = y_j \\
\max \{ A(i-1, j), A(i, j-1) \} & \text{if } x_i \neq y_j \\
\end{cases}
\]

**Step 3:** Fill in the array

**Step 4:** Recover a solution (LCS) from the array by retracing.
Step 4: Recover a solution (LCS) from the array by retracing.

Longest Increasing Subsequence

Given: a sequence of integers  \( a_1, a_2, \ldots, a_n \)

Find: a longest increasing subsequence

Example: 7, 3, 18, 4, 2, 6

has 3, 4, 6 as a LIS

Step 1: Array

\[
A(i) = \text{length of LIS of } a_1, \ldots, a_i
\]

Final answer = \( \max \{ A(i) \} \)

Step 2: Recurrence

\[
A(i) = 1 + \max \{ A(j) \mid 1 \leq j < i \text{ and } a_j < a_i \}
\]

Step 3: Fill in the array
Step 3: Fill in the array

\[ \text{Time: } O(n^2) \]

Step 4: Use the array to find an actual LIS (by retracing).

Can be made \( O(n \log n) \) with clever data structure.

DP Summary

- "Bottom up" approach, usually using an array of optimal values for sub-problems.

- Efficient recurrence to fill in the array ("Principle of Optimality")

- Can recover not just the optimal values but actual solutions achieving optimal values (by tracing through the array).
- BFS / DFS: templates for graph algs
- Greedy algorithms: algo type
- Divide & Conquer: algo type
- Dynamic Programming: algo type

Next:
- Network Flow Algo: single algo

but can be used to solve many other problems via reductions!
Flow network

- Abstraction for material *flowing* through the edges.
- Digraph $G = (V, E)$ with source $s \in V$ and sink $t \in V$.
- Nonnegative integer capacity $c(e)$ for each $e \in E$.

Minimum cut problem

**Def.** A *st-cut (cut)* is a partition $(A, B)$ of the vertices with $s \in A$ and $t \in B$.

**Def.** Its capacity is the sum of the capacities of the edges from $A$ to $B$.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$
Minimum cut problem

Def. A \textit{st-cut (cut)} is a partition \((A, B)\) of the vertices with \(s \in A\) and \(t \in B\).

Def. Its \textit{capacity} is the sum of the capacities of the edges from \(A\) to \(B\).

\[
cap(A, B) = \sum_{e \text{ out of } A} c(e)
\]

capacity = 10 + 8 + 16 = 34

don’t count edges from \(B\) to \(A\)
**Minimum cut problem**

**Def.** A *st-cut* (cut) is a partition \((A, B)\) of the vertices with \(s \in A\) and \(t \in B\).

**Def.** Its **capacity** is the sum of the capacities of the edges from \(A\) to \(B\).

\[
\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)
\]

**Min-cut problem.** Find a cut of minimum capacity.

**Maximum flow problem**

**Def.** An *st-flow* (flow) \(f\) is a function that satisfies:

- For each \(e \in E\):
  \[0 \leq f(e) \leq c(e)\] [capacity]

- For each \(v \in V - \{s, t\}\):
  \[\sum_{e \text{ in } v} f(e) = \sum_{e \text{ out of } v} f(e)\] [flow conservation]

![](image)
Maximum flow problem

**Def.** An *st*-flow (flow) $f$ is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

**Def.** The value of a flow $f$ is: $val(f) = \sum_{e \text{ out of } s} f(e)$.

![Flow Network Diagram]

$value = 5 + 10 + 10 = 25$
**Maximum flow problem**

**Def.** An *st-flow (flow)* $f$ is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

**Def.** The *value of a flow* $f$ is: $\text{val}(f) = \sum_{e \text{ out of } s} f(e)$.

**Max-flow problem.** Find a flow of maximum value.

---

![Graph with flow values]
Towards a max-flow algorithm

Greedy algorithm.
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path $P$ where each edge has $f(e) < c(e)$.
- Augment flow along path $P$.
- Repeat until you get stuck.

![Diagram of network G with flow and capacity labels and a highlighted path from s to t with flow values]

Towards a max-flow algorithm

Greedy algorithm.
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path $P$ where each edge has $f(e) < c(e)$.
- Augment flow along path $P$.
- Repeat until you get stuck.

![Diagram of network G with flow and capacity labels and a highlighted path from s to t with flow values]
Towards a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s\rightarrow t$ path $P$ where each edge has $f(e) < c(e)$.
- Augment flow along path $P$.
- Repeat until you get stuck.
Towards a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path $P$ where each edge has $f(e) < c(e)$.
- Augment flow along path $P$.
- Repeat until you get stuck.

network G

---

Towards a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path $P$ where each edge has $f(e) < c(e)$.
- Augment flow along path $P$.
- Repeat until you get stuck.

**ending flow value = 16**

network G
Towards a max-flow algorithm

Greedy algorithm.
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path $P$ where each edge has $f(e) < c(e)$.
- Augment flow along path $P$.
- Repeat until you get stuck.
Residual graph

**Original edge:** \( e = (u, v) \in E. \)
- Flow \( f(e). \)
- Capacity \( c(e) \).

**Residual edge.**
- "Undo" flow sent.
- \( e = (u, v) \) and \( e^R = (v, u) \).
- Residual capacity:
  \[
  c_f(e) = \begin{cases} 
  c(e) - f(e) & \text{if } e \in E \\
  f(e) & \text{if } e^R \in E
  \end{cases}
  \]

**Residual graph:** \( G_f = (V, E_f) \).
- Residual edges with positive residual capacity.
- \( E_f = \{ e : f(e) < c(e) \} \cup \{ e^R : f(e) > 0 \} \).
- Key property: \( f' \) is a flow in \( G_f \) iff \( f + f' \) is a flow in \( G \).

Augmenting path

**Def.** An augmenting path is a simple \( s \rightarrow t \) path \( P \) in the residual graph \( G_f \).

**Def.** The bottleneck capacity of an augmenting \( P \) is the minimum residual capacity of any edge in \( P \).

**Key property.** Let \( f \) be a flow and let \( P \) be an augmenting path in \( G_f \). Then \( f' \) is a flow and \( \text{val}(f') = \text{val}(f) + \text{bottleneck}(G_f, P) \).

**AUGMENT** \((f, c, P)\)

\[
\begin{align*}
  b & \leftarrow \text{bottleneck capacity of path } P. \\
  \text{FOREACH edge } e \in P & \\
  \text{IF } (e \in E) & f(e) \leftarrow f(e) + b. \\
  \text{ELSE} & f(e^R) \leftarrow f(e^R) - b. \\
  \text{RETURN } f.
\end{align*}
\]
(1) Is the new flow legal?
(2) Is the new flow better? Does value increase?

$$G^f$$

- Capacity constraints
  - Flow conservation

$$b = \min \text{res. cap.}$$
Cases:

\[ G \]

\[ \begin{aligned}
  &+b &+b \\
  &\downarrow &\downarrow \\
  &+b &-b \\
  &\downarrow &\downarrow \\
  &-b &-b \\
  &\downarrow &\downarrow \\
\end{aligned} \]

both forward edges

1 forward
1 backward

2 backward edges

(2) \[ \text{val}(f') = d_1 + d_2 + d_3 \]

1st edge on s-t path in \( G' \)
is FORWARD edge
Ford-Fulkerson algorithm

Ford-Fulkerson augmenting path algorithm.
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an augmenting path $P$ in the residual graph $G_f$.
- Augment flow along path $P$.
- Repeat until you get stuck.

**FORD-FULKERSON** ($G, s, t, c$)

**FOREACH** edge $e \in E : f(e) ← 0$.

$G_f ←$ residual graph.

**WHILE** (there exists an augmenting path $P$ in $G_f$)

$f ← AUGMENT(f, c, P)$.

Update $G_f$.

**RETURN** $f$.

---

Ford-Fulkerson algorithm demo

**network $G$**

```
value of flow
```

**residual graph $G_r$**

```
residual capacity
```

---

307 Lectures Summer 2017 Page 17
Ford-Fulkerson algorithm demo

network G

residual graph $G_r$
Ford-Fulkerson algorithm demo

network $G$

residual graph $G_r$

Ford-Fulkerson algorithm demo

network $G$

residual graph $G_r$
Ford-Fulkerson algorithm demo

network $G$

residual graph $G_r$
Ford-Fulkerson algorithm demo

network G

residual graph $G_r$

Ford-Fulkerson algorithm demo

network G

residual graph $G_r$
**Relationship between flows and cuts**

**Flow value lemma.** Let $f$ be any flow and let $(A, B)$ be any cut. Then, the net flow across $(A, B)$ equals the value of $f$.

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \nu(f)$$

**net flow across cut** \[= 5 + 10 + 10 = 25\]

| Value of flow \[= 25\] |
**Relationship between flows and cuts**

**Flow value lemma.** Let \( f \) be any flow and let \((A, B)\) be any cut. Then, the net flow across \((A, B)\) equals the value of \( f \).

\[
\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \nu(f)
\]

**net flow across cut** \( = 10 + 5 + 10 = 25 \)

---

**Relationship between flows and cuts**

**Flow value lemma.** Let \( f \) be any flow and let \((A, B)\) be any cut. Then, the net flow across \((A, B)\) equals the value of \( f \).

\[
\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \nu(f)
\]

**net flow across cut** \( = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5 + 0 + 0) = 25 \)
Relationship between flows and cuts

**Flow value lemma.** Let \( f \) be any flow and let \((A, B)\) be any cut. Then, the net flow across \((A, B)\) equals the value of \( f \).

\[
\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \nu(f)
\]

**Pf.**

\[
\nu(f) = \sum_{e \text{ out of } s} f(e)
\]

by flow conservation, all terms except \( \nu = s \) are 0

\[
= \sum_{v \in A} \left( \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)
\]

\[
= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).
\]

\[\blacksquare\]
Relationship between flows and cuts

**Weak duality.** Let $f$ be any flow and $(A, B)$ be any cut. Then, $v(f) \leq \text{cap}(A, B)$.

**Pf.**

\[
v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
\leq \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} c(e) \\
= \text{cap}(A, B) \quad \blacksquare
\]

value of flow = 27 \quad \leq \quad \text{capacity of cut} = 30

Max-flow min-cut theorem

**Augmenting path theorem.** A flow $f$ is a max-flow iff no augmenting paths.

**Max-flow min-cut theorem.** Value of the max-flow = capacity of min-cut.

**Pf.** The following three conditions are equivalent for any flow $f$:

i. There exists a cut $(A, B)$ such that $\text{cap}(A, B) = v(f)$.

ii. $f$ is a max-flow.

iii. There is no augmenting path with respect to $f$.

\[ i \Rightarrow ii \]

\[\begin{align*}
\text{Suppose that } (A, B) \text{ is a cut such that } & \text{cap}(A, B) = v(f). \\
\text{Then, for any flow } f' & \quad v(f') \leq \text{cap}(A, B) = v(f). \\
\text{Thus, } & f \text{ is a max-flow.} \\
\end{align*}\]
Max-flow min-cut theorem

**Augmenting path theorem.** A flow \( f \) is a max-flow iff no augmenting paths.

**Max-flow min-cut theorem.** Value of the max-flow = capacity of min-cut.

**Pf.** The following three conditions are equivalent for any flow \( f \):

i. There exists a cut \((A, B)\) such that \( \text{cap}(A, B) = \text{val}(f) \).

ii. \( f \) is a max-flow.

iii. There is no augmenting path with respect to \( f \).

[ \( \text{ii} \iff \text{iii} \) ] We prove contrapositive: \( \sim \text{iii} \iff \sim \text{ii} \).

- Suppose that there is an augmenting path with respect to \( f \).
- Can improve flow \( f \) by sending flow along this path.
- Thus, \( f \) is not a max-flow.  

34
Max-flow min-cut theorem

\[ i \iff i \]

- Let \( f \) be a flow with no augmenting paths.
- Let \( A \) be set of nodes reachable from \( s \) in residual graph \( G_f \).
- By definition of cut \( A, s \in A \).
- By definition of flow \( f, t \notin A \).

\[
\begin{align*}
\nu(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&= \sum_{e \text{ out of } A} c(e) \\
&= \text{cap}(A, B)
\end{align*}
\]

original network \( G \)

edge \( e = (v, w) \) with \( v \in B, w \in A \) must have \( f(e) = 0 \)

edge \( e = (v, w) \) with \( v \in A, w \in B \) must have \( f(e) = c(e) \)