CSE 202: Design and Analysis of Algorithms

Lecture 7

Instructor: Kamalika Chaudhuri
Announcements

- HW2 is up! Due **Mon Apr 25** in class
- Remember: Midterm on **Wed May 4**
- Midterm is **closed book**
- **Syllabus:** Greedy, Divide and Conquer, Dynamic Programming, Flows (upto Ford-Fulkerson)
Main Steps:

1. Divide the problem into **subtasks**

2. Define the subtasks **recursively** (express larger subtasks in terms of smaller ones)

3. Find the **right order** for solving the subtasks (but do not solve them recursively!)
Last Class: Dynamic Programming

- String Reconstruction
- Longest Common Subsequence
- Edit Distance
- Subset Sum
- Independent Set in a Tree
**Independent Set**

**Independent Set:** Given a graph $G = (V, E)$, a subset of vertices $S$ is an independent set if there are no edges between them.

**Max Independent Set Problem:** Given a graph $G = (V, E)$, find the largest independent set in $G$.

**Max Independent Set** is a notoriously hard problem! We will look at a restricted case, when $G$ is a **tree**.
Max. Independent Set in a Tree

A set of nodes is an **independent set** if there are no edges between the nodes.

**Two Cases at node u:**
1. Don’t include u
2. Include u, and don’t include its children
Max. Independent Set in a Tree

A set of nodes is an **independent set** if there are no edges between the nodes

**STEP 1: Define subtask**

$I(u) =$ size of largest independent set in subtree rooted at $u$

We want $I(r)$, where $r =$ root

**Two Cases at node $u$:**

1. Don't include $u$
2. Include $u$, and don't include its children
Max. Independent Set in a Tree

A set of nodes is an **independent set** if there are no edges between the nodes

**STEP 1: Define subtask**

\[ I(u) = \text{size of largest independent set in subtree rooted at } u \]

We want \( I(r) \), where \( r = \text{root} \)

**STEP 2: Express recursively**

\[
I(u) = \max \left\{ \sum \limits_{w \text{ of } u} I(w), \quad 1 + \sum \limits_{\text{grandchildren } w \text{ of } u} I(w) \right\}
\]

Base case: for leaf nodes, \( I(u) = 1 \)

**Two Cases at node u:**

1. Don't include \( u \)
2. Include \( u \), and don't include its children
**Max. Independent Set in a Tree**

A set of nodes is an **independent set** if there are no edges between the nodes.

**STEP 1: Define subtask**

$I(u) = \text{size of largest independent set in subtree rooted at } u$

We want $I(r)$, where $r =$ root.

**STEP 2: Express recursively**

$$l(u) = \max \left\{ \sum_{\text{children} \ w \ of \ u} I(w), 1 + \sum_{\text{grandchildren} \ w \ of \ u} I(w) \right\}$$

Base case: for leaf nodes, $l(u) = 1$

**STEP 3: Order of subtasks**

Reverse order of distance from root; use BFS!

**Two Cases at node $u$:**

1. Don't include $u$
2. Include $u$, and don't include its children
Max. Independent Set in a Tree

A set of nodes is an **independent set** if there are no edges between the nodes

**STEP 1: Define subtask**

\[ l(u) = \text{size of largest independent set in subtree rooted at } u \]

We want \( l(r) \), where \( r = \text{root} \)

**STEP 2: Express recursively**

\[ l(u) = \max \left\{ \sum_{\text{children } w \text{ of } u} I(w), 1 + \sum_{\text{grandchildren } w \text{ of } u} I(w) \right\} \]

Base case: for leaf nodes, \( l(u) = 1 \)

**STEP 3: Order of subtasks**

Reverse order of distance from root; use BFS!

**Running Time: \( O(n) \)**

Edge \( (u, v) \) is examined in Step 2 at most twice:

1. \( v \) is a child of \( u \)
2. \( v \) is a grandchild of \( u \)'s parent

There are \( n-1 \) edges in a tree on \( n \) nodes
Dynamic Programming

• String Reconstruction
• Longest Common Subsequence
• Edit Distance
• Subset Sum
• Independent Set in a Tree
• All Pairs Shortest Paths
All Pairs Shortest Paths

**Problem:** Given \( n \) nodes and distances \( d_{ij} \) (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

Does Dijkstra’s algorithm work?
All Pairs Shortest Paths

Problem: Given n nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

Does Dijkstra’s algorithm work?
An: No! Example: s-v Shortest Paths
All Pairs Shortest Paths (APSP)

**Problem:** Given \( n \) nodes and distances \( d_{ij} \) (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**Structure:**
For all \( x, y \):
- either \( SP(x, y) = d_{xy} \)
- Or there exists some \( z \) s.t.
  \[ SP(x, y) = SP(x, z) + SP(y, z) \]
All Pairs Shortest Paths (APSP)

**Problem:** Given $n$ nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**Structure:**
For all $x, y$:
- either $SP(x, y) = d_{xy}$
- Or there exists some $z$ s.t $SP(x, y) = SP(x, z) + SP(y, z)$

**Property:** If there is no negative weight cycle, then for all $x, y$, $SP(x, y)$ is simple (that is, includes no cycles)
**All Pairs Shortest Paths**

**Problem:** Given $n$ nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**STEP 1: Define Subtasks**

$D(i,j,k) =$ length of shortest path from $i$ to $j$ with intermediate nodes in \{1,2,...,$k$\}

![Graph Example]

In the graph example, the distances are shown as labels on the edges.
**Problem:** Given n nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**STEP 1: Define Subtasks**

$D(i,j,k) =$ length of shortest path from $i$ to $j$ with intermediate nodes in $\{1,2,...,k\}$

Shortest Path lengths = $D(i,j,n)$
**Problem:** Given \( n \) nodes and distances \( d_{ij} \) (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**STEP 1: Define Subtasks**

\[ D(i,j,k) = \text{length of shortest path from } i \text{ to } j \text{ with intermediate nodes in } \{1,2,...,k\} \]

Shortest Path lengths = \( D(i,j,n) \)

**STEP 2: Express Recursively**

\[ D(i,j,k) = \min\{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)\} \]

Base case: \( D(i,j,0) = d_{ij} \)
All Pairs Shortest Paths

**Problem:** Given $n$ nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**STEP 1: Define Subtasks**

$D(i,j,k) =$ length of shortest path from $i$ to $j$ with intermediate nodes in $\{1,2,\ldots,k\}$

Shortest Path lengths $= D(i,j,n)$

**STEP 2: Express Recursively**

$D(i,j,k) = \min\{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)\}$

Base case: $D(i,j,0) = d_{ij}$

**STEP 3: Order of Subtasks**

By increasing order of $k$
Problem: Given $n$ nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

STEP 1: Define Subtasks
$D(i,j,k) =$ length of shortest path from $i$ to $j$ with intermediate nodes in $\{1,2,\ldots,k\}$
Shortest Path lengths $= D(i,j,n)$

STEP 2: Express Recursively
$D(i,j,k) = \min\{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)\}$
Base case: $D(i,j,0) = d_{ij}$

STEP 3: Order of Subtasks
By increasing order of $k$

Running Time $= O(n^3)$

Exercise:
Reconstruct the shortest paths
Summary: Dynamic Programming

Main Steps:

1. Divide the problem into subtasks

2. Define the subtasks recursively (express larger subtasks in terms of smaller ones)

3. Find the right order for solving the subtasks (but do not solve them recursively!)
## Summary: Dynamic Programming vs Divide and Conquer

<table>
<thead>
<tr>
<th>Divide-and-conquer</th>
<th>Dynamic programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>A problem of size n is decomposed into a few subproblems which are significantly smaller (e.g. n/2, 3n/4,...)</td>
<td>A problem of size n is expressed in terms of subproblems that are not much smaller (e.g. n-1, n-2,...)</td>
</tr>
<tr>
<td>Therefore, size of subproblems decreases geometrically. eg. n, n/2, n/4, n/8, etc</td>
<td>A recursive algorithm would take exp. time.</td>
</tr>
<tr>
<td>Use a recursive algorithm.</td>
<td>Saving grace: in total, there are only polynomially many subproblems.</td>
</tr>
<tr>
<td></td>
<td>Avoid recursion and instead solve the subproblems one-by-one, saving the answers in a table, in a clever explicit order.</td>
</tr>
</tbody>
</table>
Summary: Common Subtasks in DP

Case 1: Input: $x_1, x_2, ..., x_n$ Subproblem: $x_1, ..., x_i$.

Case 2: Input: $x_1, x_2, ..., x_n$ and $y_1, y_2, ..., y_m$ Subproblem: $x_1, ..., x_i$ and $y_1, y_2, ..., y_j$.

Case 3: Input: $x_1, x_2, ..., x_n$. Subproblem: $x_i, ..., x_j$.

Next: Network Flow
Oil Through Pipelines

**Problem:** Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, how much oil can we ship from $s$ to $t$?
Problem: Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, how much oil can we ship from $s$ to $t$?

An $s$-$t$ flow is a function: $E \rightarrow \mathbb{R}$ such that:
- $0 \leq f(e) \leq c(e)$, for all edges $e$
- flow into node $v = \text{flow out of node } v$, for all nodes $v$ except $s$ and $t$,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Size of flow $f = \text{Total flow out of } s = \text{total flow into } t$

![Graph Diagram]
**Problem:** Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, how much oil can we ship from $s$ to $t$?

An $s$-$t$ flow is a function: $E \rightarrow \mathbb{R}$ such that:
- $0 \leq f(e) \leq c(e)$, for all edges $e$
- flow into node $v$ = flow out of node $v$, for all nodes $v$ except $s$ and $t$,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Size of flow $f =$ Total flow out of $s$ = total flow into $t$

**The Max Flow Problem:** Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size
The Max Flow Problem: Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size

An $s$-$t$ Cut partitions nodes into groups $= (L, R)$ s.t. $s$ in $L$, $t$ in $R$
**Flows and Cuts**

**The Max Flow Problem:** Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size

**An $s$-$t$ Cut** partitions nodes into groups $= (L, R)$ s.t. $s$ in $L$, $t$ in $R$

- Capacity of a cut $(L, R) = \sum_{(u,v) \in E, u \in L, v \in R} c(u, v)$
- Flow across $(L, R) = \sum_{(u,v) \in E, u \in L, v \in R} f(u, v) - \sum_{(v,u) \in E, u \in L, v \in R} f(v, u)$

Size of $f = 3$
The Max Flow Problem: Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size.

An $s$-$t$ Cut partitions nodes into groups $= (L, R)$ s.t. $s$ in $L$, $t$ in $R$.

Capacity of a cut $(L, R) = \sum_{(u,v) \in E, u \in L, v \in R} c(u, v)$

Flow across $(L, R) = \sum_{(u,v) \in E, u \in L, v \in R} f(u, v) - \sum_{(v,u) \in E, u \in L, v \in R} f(v, u)$

Property: For any flow $f$, any $s$-$t$ cut $(L, R)$, $\text{size}(f) \leq \text{capacity}(L, R)$
Flows and Cuts

**The Max Flow Problem:** Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size

An **$s$-$t$ Cut** partitions nodes into groups = $(L, R)$ such that $s$ in $L$, $t$ in $R$

**Capacity of a cut** $(L, R) = \sum_{(u,v) \in E, u \in L, v \in R} c(u, v)$

**Flow across** $(L,R) = \sum_{(u,v) \in E, u \in L, v \in R} f(u, v) - \sum_{(v,u) \in E, u \in L, v \in R} f(v, u)$

**Property:** For any flow $f$, any $s$-$t$ cut $(L, R)$, $\text{size}(f) \leq \text{capacity}(L, R)$

**Proof:** For any cut $(L,R)$, Flow Across $(L,R)$ cannot exceed capacity$(L,R)$

From flow conservation constraints, $\text{size}(f) = \text{flow across}(L,R) \leq \text{capacity}(L,R)$
Flows and Cuts

The Max Flow Problem: Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size.

An $s$-$t$ Cut partitions nodes into groups $=(L, R)$ s.t. $s$ in $L$, $t$ in $R$.

Capacity of a cut $(L, R) = \sum_{(u, v) \in E, u \in L, v \in R} c(u, v)$

Flow across $(L, R) = \sum_{(u, v) \in E, u \in L, v \in R} f(u, v) - \sum_{(v, u) \in E, u \in L, v \in R} f(v, u)$

Size of $f = 3$

Property: For any flow $f$, any $s$-$t$ cut $(L, R)$, $\text{size}(f) \leq \text{capacity}(L, R)$

Proof: For any cut $(L, R)$, Flow Across $(L, R)$ cannot exceed capacity $(L, R)$

From flow conservation constraints, $\text{size}(f) = \text{flow across}(L, R) \leq \text{capacity}(L, R)$

Max-Flow $\leq$ Min-Cut
Flows and Cuts

**The Max Flow Problem:** Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size

An **$s$-$t$ Cut** partitions nodes into groups $=(L, R)$ s.t. $s$ in $L$, $t$ in $R$

Capacity of a cut $(L, R) = \sum_{(u,v) \in E, u \in L, v \in R} c(u, v)$

Flow across $(L,R) = \sum_{(u,v) \in E, u \in L, v \in R} f(u, v) - \sum_{(v,u) \in E, u \in L, v \in R} f(v, u)$

**Property:** For any flow $f$, any $s$-$t$ cut $(L, R)$, $\text{size}(f) \leq \text{capacity}(L, R)$

**Proof:** For any cut $(L,R)$, Flow Across $(L,R)$ cannot exceed capacity$(L,R)$

From flow conservation constraints, $\text{size}(f) = \text{flow across}(L,R) \leq \text{capacity}(L,R)$

**Max-Flow $\leq$ Min-Cut**

In our example: Size of $f = 3$, Capacity of Cut $(s,V - s) = 3$
The Max Flow Problem: Given directed graph $G=(V,E)$, source $s$, sink $t$, edge capacities $c(e)$, find an $s$-$t$ flow of maximum size.

An $s$-$t$ Cut partitions nodes into groups = $(L, R)$ s.t. $s$ in $L$, $t$ in $R$.

Capacity of a cut $(L, R) = \sum_{(u,v) \in E, u \in L, v \in R} c(u,v)$

Flow across $(L,R) = \sum_{(u,v) \in E, u \in L, v \in R} f(u,v) - \sum_{(v,u) \in E, u \in L, v \in R} f(v,u)$

Property: For any flow $f$, any $s$-$t$ cut $(L, R)$, size($f$) <= capacity($L, R$)

Proof: For any cut $(L,R)$, Flow Across $(L,R)$ cannot exceed capacity($L,R$). From flow conservation constraints, size($f$) = flow across($L,R$) <= capacity($L,R$)

Max-Flow <= Min-Cut

In our example: Size of $f = 3$, Capacity of Cut $(s,V - s) = 3$. Thus, a Min Cut is a certificate of optimality for a flow.
**Ford-Fulkerson algorithm**

**FF Algorithm:** Start with zero flow

Repeat:
- Find a path from s to t along which flow can be increased
- Increase the flow along that path

Example

First choose:

Next choose:

But what if we first chose:

Then we’d have to allow:

cancels out existing flow
Ford-Fulkerson, continued

**FF Algorithm:** Start with zero flow

Repeat:
- Find a path from s to t along which flow can be increased
- Increase the flow along that path

In any iteration, we have some flow f and we are trying to improve it. How to do this?

1: Construct a residual graph $G_f$ (“what’s left to take?”)
   - $G_f = (V, E_f)$ where $E_f \subseteq E \cup E^R$
   - For any $(u,v)$ in $E$ or $E^R$, $c_f(u,v) = c(u,v) - f(u,v) + f(v,u)$
     - [ignore edges with zero $c_f$: don’t put them in $E_f$]

2: Find a path from s to t in $G_f$
3: Increase flow along this path, as much as possible

Example

![Graph](image)
Example: Round 1

Construct residual graph $G_f = (V, E_f)$

$E_f \subseteq E \cup E^R$

For any $(u,v)$ in $E$ or $E^R$,

$c_f(u,v) = c(u,v) - f(u,v) + f(v,u)$

Find a path from $s$ to $t$ in $G_f$

Augment $f$ along this path
Example: Round 2

Construct residual graph $G_f = (V, E_f)$

$$E_f \subseteq E \cup E^R$$

For any $(u, v)$ in $E$ or $E^R$,

$$c_f(u, v) = c(u, v) - f(u, v) + f(v, u)$$

Find a path from $s$ to $t$ in $G_f$

Augment $f$ along this path
Example: Round 3

Construct residual graph $G_f = (V, E_f)$

$E_f \subseteq E \cup E^R$

For any $(u,v)$ in $E$ or $E^R$,

$c_f(u,v) = c(u,v) - f(u,v) + f(v,u)$

Find a path from $s$ to $t$ in $G_f$

Augment $f$ along this path
Example: Round 3

Construct residual graph \( G_f = (V, E_f) \)

\[ E_f \subseteq E \cup E^R \]

For any \((u,v)\) in \(E\) or \(E^R\),

\[ c_f(u,v) = c(u,v) - f(u,v) + f(v,u) \]

Find a path from \(s\) to \(t\) in \(G_f\)

Augment \(f\) along this path

\[ \begin{align*}
G_f \text{ (after)}
\end{align*} \]
Analysis: Correctness

FF algorithm gives us a valid flow. But is it the maximum possible flow?

Consider final residual graph $G_f = (V, E_f)$
Let $L = \text{nodes reachable from } s \text{ in } G_f$ and let $R = \text{rest of nodes } = V - L$
So $s \in L$ and $t \in R$

![Diagram](image)

Edges from $L$ to $R$ must be at full capacity
Edges from $R$ to $L$ must be empty
Therefore, flow across cut $(L,R)$ is

$$\sum_{(u,v) \in E, u \in L, v \in R} c(u, v)$$

Thus, $\text{size}(f) = \text{capacity}(L,R)$

Recall: for any flow and any cut, $\text{size}(\text{flow}) \leq \text{capacity(\text{cut})}$

Therefore $f$ is the max flow and $(L,R)$ is the min cut!

Thus, Max Flow = Min Cut