LECTURE 13

DYNAMIC PROGRAMMING END

INTRODUCTION TO GRAPHS.
O/1 Knapsack

Problem: n items \( \{0_1, 0_2, \ldots, 0_n\} \)
values \( \{v_1, \ldots, v_n\} \)
weights \( \{w_1, \ldots, w_n\} \)
Output: subset \( S \subseteq \{0_1, \ldots, 0_n\} \)
s.t.
\[
\max \sum_{i \in S} v_i \text{ s.t. } \sum_{i \in S} w_i \leq W
\]

Example:

\[
\begin{array}{cc}
V_1: 2.2 & V_2: 4 \\
W_1: 1.5 & W_2: 3 \\
\end{array}
\]

\[
\downarrow \quad W = 7.8
\]

Possible greedy alg 1: Highest value first.
On our example ex: \( \{0_2, 0_4\} \)
\[
v = 4.5 \geq 7 \not\rightarrow \text{opt.}
\]
\[
w = 9 + 4.6 \leq 7.8
\]

Possible greedy alg 2: Lightest first
- Return opt in our running ex

\[
\begin{array}{ccc}
v & w \\
5 & 4 \\
0.5 & 1 \\
\end{array}
\]

\[
\text{opt} \in \{0_2\} \text{ and in } S0_2
\]

Possible greedy alg 3: highest value weight.
Let's change the problem: Assume $W$ and $w_i$ are all positive ints.

Thought Experiment: Consider $S^*$ and the object $w_i$ the last index $n$.

Tautology: case 1: $0_n \in S^*$

case 2: $0_n \notin S^*$

Opt: \[ \min \{ v_i, w_i, \} \]

Consider $v_i = 2$, $w_i = 2$

\[ \text{greedy: } \frac{v_i}{w_i} = 2 \]

Not surprising, since the LCS is intractable.
Suppose case 1: \( \emptyset \not\subset S^* \)

Claim: Then \( S^* \) is optimal for the subproblem that contains \( \{0, \ldots, v_{n-1}\} \) and capacity \( = W \)

(prove by a simple proof by contradiction)

Suppose case 2: \( \emptyset \not\subset S^* \)

Claim: Then \( S = S^* - \{v_0\} \) is optimal for the subproblem that contains \( \{0, \ldots, v_{n-1}\} \) and capacity \( = W - w_n \) (by o.w. we can contradict \( S^* \) is optimal).

So \( S^* = \max \{ \text{opt for } \{0, \ldots, v_{n-1}\}, W \}

\text{opt for } \{0, \ldots, v_{n-1}\}, W - w_n + v_n \)
Subproblems: \(OPT_i, c = \) optimal solution for the subproblem that contains \(S_0, \ldots, S_i, \) capacity \(c.\)

\[
OPT_i, c = \max \left\{ \begin{array}{l}
OPT_{i-1}, c \\
OPT_{i-1}, c-w_i + v_i
\end{array} \right. \]

DP-Knapsack: \(\{S_0, \ldots, S_n\}, W)\)

\(A[i][w]\) initially 0.

for \(i = 1 \ldots n\)

for \(c = 1 \ldots W\)

\(A[i][c] = \max \left\{ \begin{array}{l}
A[i-1][c] \\
A[i-1][c-w] + v_i \text{ iif } c-w_i > 0
\end{array} \right. \)

return \(A[n][W]\)
(1) Did we prove that Knapsack with integer weights is tractable (i.e., poly-time input size)?

A: No: Why is \( O(n\log W) \) not poly-time in input size, i.e., # bits to represent the input.

represent \( n \) values: \( u_1, \ldots, u_n \geq \log(\text{max value}) \)

weight: \( w_1, \ldots, w_n \geq \log(\text{max weight}) \)

\( n \cdot \text{weight} \cdot w_n - \log(\text{max weight}) \)

\( W = \log(W) \) so do represent so \( W \) is exponential \( n \cdot \log \log(W) \)

so \( O(\log W) \) is not poly-time!
Graphs: A graph \( G(V, E) \) is a pair of sets,

\[ V \text{; a set of nodes} \]
\[ E \text{; a set of edges}. \]

Visually:

\[ \text{A natural way to represent connected data (which is the case for many applications).} \]
Ex: Social networks (FB, Twitter), or the web.

V: people
E: friendships | who follows whom

Other Ex: Scientific networks, telecommunication networks.
(e.g. protein interaction networks)

Terminology:

**Directed** vs **Undirected**
- In edges have direction (e.g. Twitter or FB friendship)
- Edges do not have direction

Simple vs Multigraph
- At most 1 edge between any pair of vertices u & v
- Parallel edges allowed
Cyclic vs Acyclic
- Cyclic: Has cycles
- Acyclic: No cycles

Connected vs Unconnected
- Connected: "In one piece"
- Unconnected: "Multiple disconnected pieces"

Conventions:
- $|V| = n$
- $|E| = m$

Q: If $G$ is undirected and has $n$ vertices, what's the max number of edges?
A: $\binom{n}{2} = O(n^2)$

Q: What if $G$ is directed?
A: $2\binom{n}{2}$

$m \leq n^2$
Q: If \( G \) is undirected and connected, what's \( \min \) in terms of edges you need?

\[ m \geq n - 1 \]

So for a problem that takes as input connected, simple graphs:

\[ n - 1 \leq m \leq n^2 \]

So \( \log(n) \leq \log(m) \leq 2 \log(n) \)

\[ \log(m) = \Theta(\log(m)) \]

Conversely, use \( \log(m) \) instead of \( \log(m) \)

\[ \log(m) \leq \Theta(\log(m)) \]
E.g. we'll say Dijkstra's algorithm runs in $O(\log n)$ time instead of $O(\log \log m)$ time.

**Def:** Degree of $v$:

- **Out-deg**($v$) = # outgoing edges from $v$, i.e.
  $$(v, w) \in E$$

- **In-deg**($v$) = # incoming edges to $v$, i.e.
  $$w \rightarrow v$$

For undirected graph, we simply use deg.
Graph Storage Formats:

1. Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Storage: $O(n^2)$

Ops: Look up if $(u, v) \in E \geq 1$ op.

Iterate over $u$'s outlier edges: $O(n)$
(2) Adj list format

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

Storage $O(n+m)$

Ops: Lookup $(u, v)$ $\rightarrow$ deg $u$

Iterating over $u$, out/in edges $\rightarrow$ deg $u$

If your application needs primarily edges too, you need a 2nd structure.
An important quantity to know:

\[ \sum_{v=1}^{n} \text{out-deg}(v) = m \]

If the graph is undirected:

\[ \sum_{v=1}^{n} \text{deg}(v) = 2m \]

\((uv)\) will be counted twice.