CSE 421

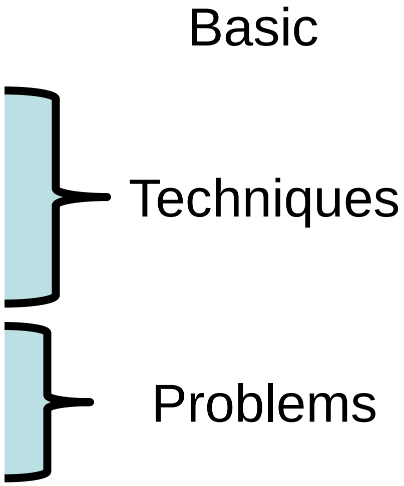
Dynamic Programming

Yin Tat Lee

Roadmap

This course has the following topics:

- Graphs (BFS, DFS)
- Greedy Methods
- Divide and Conquer
- Dynamic Programming
- Network Flow
- NP completeness



Basic

Techniques

Problems

We start network flow on next lecture.

Next lecture is by Sally.

Common Subproblems

- OPT(i) – opt solution using $x_1, \cdots, x_i$ (Longest path)
- OPT(i,j) – opt solution using $x_i, \cdots, x_j$ (RNA)
- OPT(i,j) – opt solution using $x_1, \cdots, x_i$ and $y_1, \cdots, y_i$ (Edit distance)
- OPT(i,W) – opt solution using $x_1, \cdots, x_i$ with budget W (Knapsack problem)
- OPT(i,t) – opt solution at vertex i using t step (negative-weight shortest path)

Today's Goal:

- Give some history of DP
- Give an example that OPT(i,t) with t has no "meaning".

Dynamic Programming

Copy from Bellman's autobiography

I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying I thought, lets kill two birds with one stone. Lets take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

In short, his boss hates math.

He needs a cool term, which sounded less math.

Programming refers to a military schedule.

Richard
Bellman

Example 1:
Dynamic Decision Problem
(original problem for Dynamic Programming)

Dynamic Decision Problem

Given a directed graph G with a starting vertex v_0.

- each vertex represents some state
- each edge e has some reward r_e (can be negative)

We define the reward of a path $p = v_0 v_1 v_2 \cdots$ be

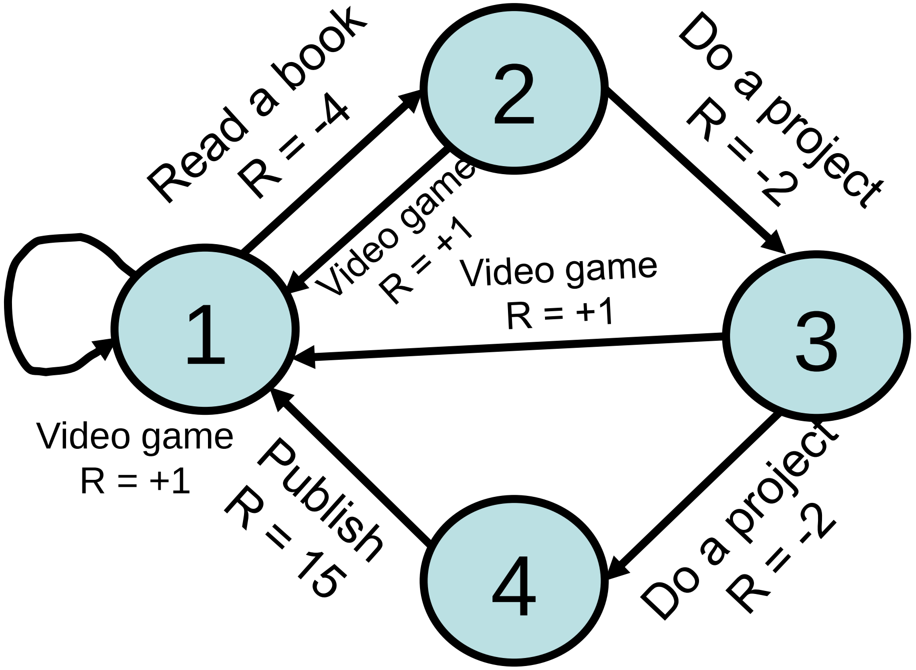
$$R(p) = r_{v_0 v_1} + \gamma \cdot r_{v_1 v_2} + \gamma^2 \cdot r_{v_2 v_3} + \cdots$$

for some given discount factor $0 < \gamma < 1$.

Goal: Find a path p starts at v_0 with maximum reward $R(p)$.

(This is a simplified version of Markov Decision process.)

Example



Start vertex is 1.

Question: What is the best path?

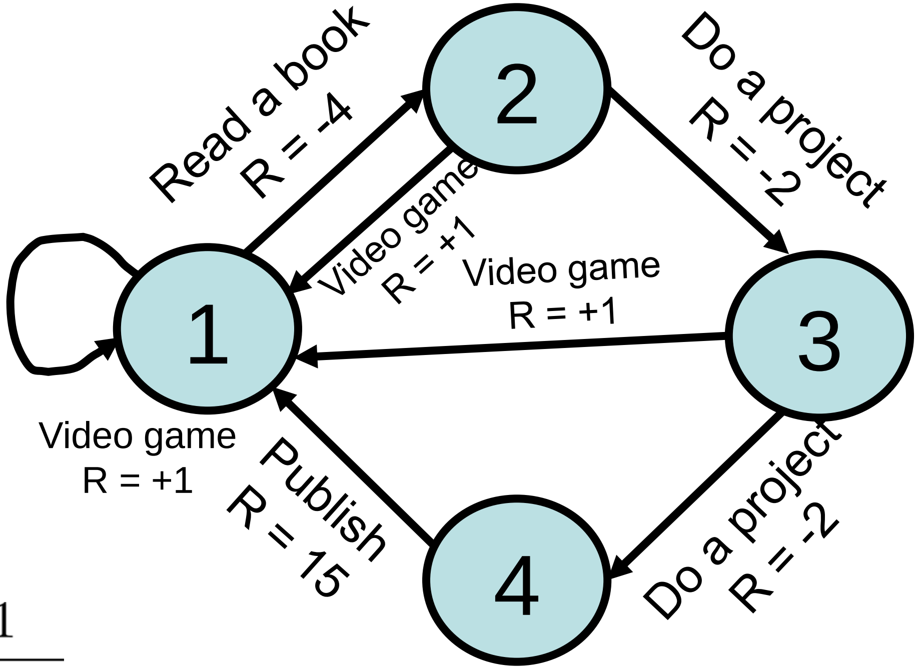
There are two natural choices:

1. Simply play video game every day (1,1,1,1,1,…)
   YOLO!
2. Keep publishing (1,2,3,4,1,2,3,4,…)
   Publish or Perish ☹

So, which is the best path?

It depends on γ. (Or, how many days you have left?)

Example



Start vertex is 1.

Path 1: $1, 1, 1, \cdots$.

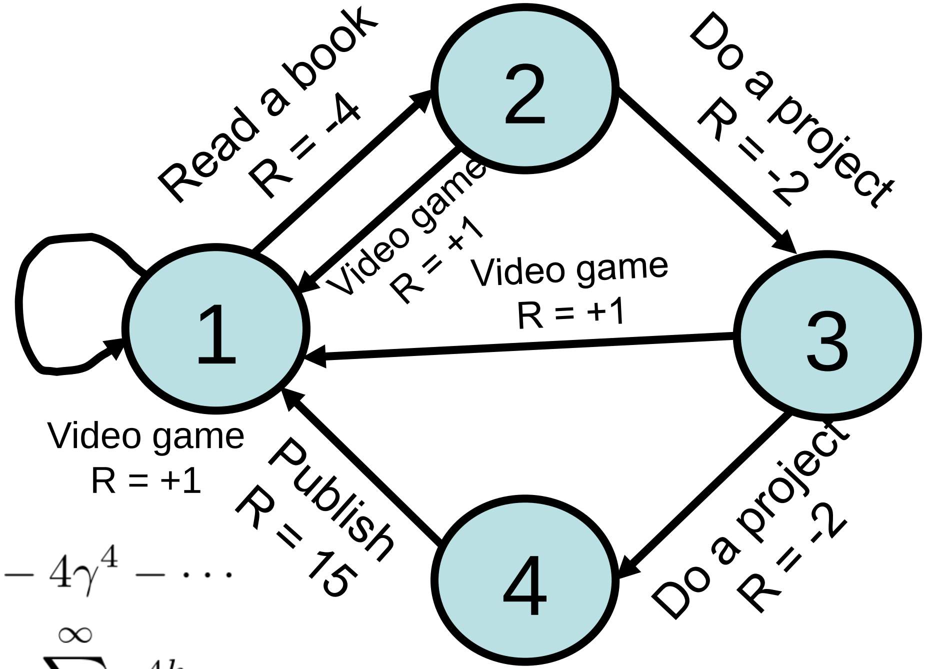
The reward is $1 + \gamma^1 + \gamma^2 + \cdots = \frac{1}{1-\gamma}$

We can view γ is the probability you die in each day.

$1/(1-\gamma)$ is the life expectancy.

Hence, the reward per day is simply 1.

Example



Start vertex is 1.

Path 2: $1,2,3,4,1,2,3,4,\cdots$

The reward is $-4-2\gamma^1-2\gamma^2+15\gamma^3-4\gamma^4-\cdots$

$$=(-4-2\gamma^1-2\gamma^2+15\gamma^3)\cdot\sum_{k=0}^{\infty}\gamma^{4k}$$

$$=\frac{-4-2\gamma^1-2\gamma^2+15\gamma^3}{1-\gamma^4}$$

The reward/day is $\frac{-4-2\gamma^1-2\gamma^2+15\gamma^3}{1-\gamma^4}\Big/\frac{1}{1-\gamma}=\frac{15\gamma^3-2\gamma^2-2\gamma-4}{\gamma^3+\gamma^2+\gamma+1}$

When $\gamma\approx 0$, the reward/day is around -4. (worse than Path 1)

When $\gamma\approx 1$, the reward/day is around $7/4$. (better than Path 1)

Dynamic Decision Problem

Given a directed graph G with a starting vertex v_0.

- each vertex represents some state
- each edge e has some reward r_e (can be negative)

We define the reward of a path $p = v_0 v_1 v_2 \cdots$ be

$$R(p) = r_{v_0 v_1} + \gamma \cdot r_{v_1 v_2} + \gamma^2 \cdot r_{v_2 v_3} + \cdots$$

for some given discount factor $0 < \gamma < 1$.

Goal: Find a path p starts at v_0 with maximum reward $R(p)$.

Bellman shows how to solve it using “Dynamic Programming”.

Hints: Instead of finding the path, find the reward first!

Bellman Equation

Fix any vertex v_0.

Let $p = v_0 v_1 v_2 \cdots$ be a path with maximum reward.

Let $q = v_1 v_2 v_3 \cdots$.

Note that
$$\begin{aligned} R(p) &= r_{v_0 v_1} + \gamma \cdot r_{v_1 v_2} + \gamma^2 \cdot r_{v_2 v_3} + \gamma^3 \cdot r_{v_3 v_4} + \cdots \\ &= r_{v_0 v_1} + \gamma \left(r_{v_1 v_2} + \gamma \cdot r_{v_2 v_3} + \gamma^2 \cdot r_{v_3 v_4} + \cdots \right) \\ &= r_{v_0 v_1} + \gamma \cdot R(q) \end{aligned}$$

Hence, if p maximizes reward at v_0, q maximizes reward at v_1.

Let $R(v)$ be the maximum reward for path starting at v.

We have $R(v) = \max_u [r_{vu} + \gamma \cdot R(u)]$. (Bellman equation)

Like last lecture, unclear how to use.

Value Iteration

Bellman equation: $R(v) = \max_u [r_{vu} + \gamma \cdot R(u)]$.

We define $R^{(0)}(v) = 0$ for all u and

$$R^{(t+1)}(v) = \max_u [r_{vu} + \gamma \cdot R^{(t)}(u)] \text{ for all } v.$$

(Unlike Bellman equation, this can be implemented.)

Lemma: Let $\epsilon_t = \max_u |R(u) - R^{(t)}(u)|$. Then $\epsilon_{t+1} \leq \gamma \cdot \epsilon_t$.

Proof: $R(v) - R^{(t+1)}(v)$

$$= \max_u [r_{vu} + \gamma R(u)] - \max_u \left[r_{vu} + \gamma R^{(t)}(u)\right]$$

$$\leq \max_u \left[r_{vu} + \gamma R(u) - r_{vu} - \gamma R^{(t)}(u)\right]$$

$$= \gamma \cdot \max_u \left[R(u) - R^{(t)}(u)\right] \leq \gamma \cdot \epsilon_t$$

Value Iteration

```
Input: accuracy target $\delta$
R[u,0]=0 for all u
$R_{max}$ = max_e |r_e| / (1-$\gamma$)
T = $\left\lceil \log_\gamma\left(\frac{R_{max}}{\delta}\right) \right\rceil$

for t = 1,2,...,T
   for v = 1 to n
      R[v,t] = $\max_u (r_{vu} + \gamma \cdot R[u,t-1])$
```

R_{max} is some upper bound of R.
Using $\epsilon_{t+1} \leq \gamma \cdot \epsilon_t$ and $\epsilon_0 \leq R_{max}$, we have $\epsilon_T \leq \delta$.

Running Time: $O(mT)$ which is around $m \log(\frac{1}{\delta})/(1-\gamma)$.
So, the runtime is fast when the discount factor not close to 1.

Open Question: How fast you can solve it when γ very close to 1?

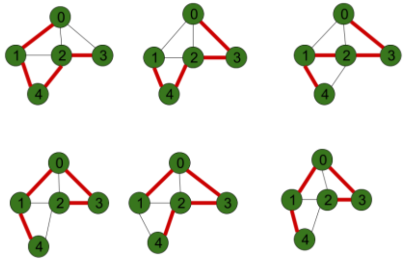
Example 2:
Traveling Salesperson Problem
(DP can be used for hard problems)

Traveling Salesperson Problem

Given: n cities and the pairwise distance d_{ij}

Goal: Find shortest path that visit every city exactly once time

(For simplicity, not required to return to starting point)



Brute force: $n! \sim 2^{O(n \log n)}$ time

Subproblem: $T(v, S)$ – length of shortest cycle that visit all cities in S exactly once and ends at v.

Observation: $T(v, S) = \min_{u \in S - v} T(u, S - v) + d_{uv}$

They made a breakthrough in 2020 on getting best approximation of metric TSP

Traveling Salesperson Problem

Brute force: $n! \sim 2^{O(n \log n)}$ time

Subproblem: $T(v, S)$ – length of shortest cycle that visit all cities in S exactly once and ends at v.

Algorithm:

```
Set T(v,{v}) = 0 for all v

for k = 2,...,n
   for all sets S of size k
      for all v in S
         T(v,S) = $\min_{u \in S-v} T(u, S-v) + d_{uv}$
```

Runtime: $O(n2^n)$

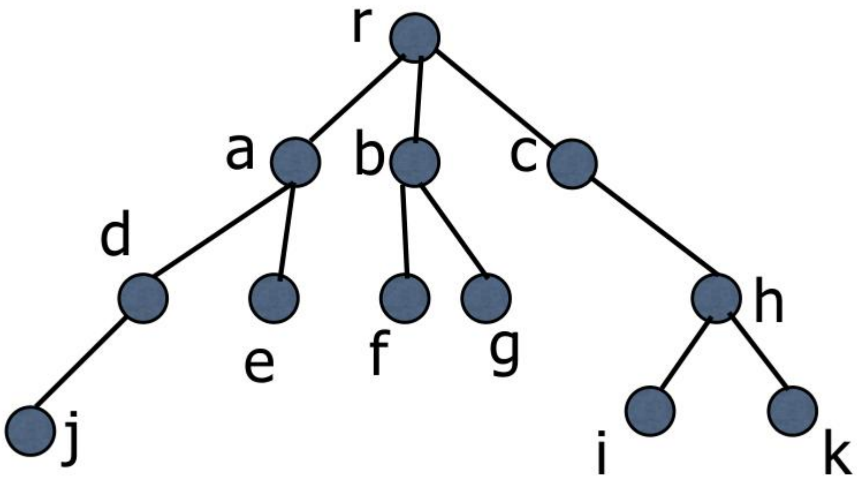
Exercise:
Minimum Vertex Cover for Tree

Minimum Vertex Cover for Tree

Given an undirected tree $T = (V, E)$.

We call $S \subset V$ is a vertex cover if every edge touches some vertex in S.

Give a linear time algorithm to find the minimum vertex cover of tree.



Minimum Vertex Cover for Tree

Answer:

Let $V(r)$ be the size of minimum vertex cover of the subtree at r.

Case 1: The optimal cover does not contain r.

Then, it must contain children(r).

Hence, $V(r) = \#children(r) + \sum_{g:\, grandchild\ of\ r} V(g)$

Case 2: The optimal cover does contain r.

Then, $V(r) = 1 + \sum_{c:\, child\ of\ r} V(c)$

Combining both cases, we have

$$V(r) = \min(\#children(r) + \sum_{g:\, grandchild\ of\ r} V(g)\,, 1 + \sum_{c:\, child\ of\ r} V(c))$$

Exercise:
Chain Matrix Multiplication

Chain Matrix Multiplication

Given: n matrices $M_1, M_2, \cdots, M_n$

Goal: Find the cheapest order to compute $M_1 M_2 \cdots M_n$

Example: To compute $VWXZY$, we could multiply

$V((WX)(YZ))$ or $\left(V\left(W(XY)\right)\right)Z$

Assumption: Multiply $a \times b$ matrix with $b \times c$ matrix takes abc time.

Subproblems: $C(i, j)$ – time to compute $M_i M_{i+1} \cdots M_j$

Chain Matrix Multiplication

Assumption: Multiply $a \times b$ matrix with $b \times c$ matrix takes abc time.

Subproblems: $C(i, j)$ – time to compute $M_i M_{i+1} \cdots M_j$

Observation: If the last multiplication in optimal solution is

$$(M_i \cdots M_k)(M_{k+1} \cdots M_j)$$

Then, $C(i, j) = C(i, k) + C(k + 1, j) + m_i m_{k+1} m_j$

Algorithm:

```
Set C(i,i) = 0 for all I = 1,2,...,n

for s = 1,...,n-1
   for i = 1,…,n-1
      C(i,i+s) = $\min_{i<k<i+s} C(i,k) + C(k+1, i+s) + m_i m_{k+1} m_{i+s}$
```