Dynamic Programming II

Ethan Arnold, Arnav Sastry

CS 104C

Spring 2017
DP as Transitions

- Last week we talked about dynamic programming in terms of recursive functions.
DP as Transitions

- Last week we talked about dynamic programming in terms of recursive functions.
- We can also view dynamic programming as a way to do transitions between states.
DP as Transitions

- Last week we talked about dynamic programming in terms of recursive functions.
- We can also view dynamic programming as a way to do transitions between states.
- If we add to our state, how do our solutions change?
Knapsack problem

- You are a jewel thief, and you want to steal the most valuable set of jewels possible from a set of \( n \) jewels.
Knapsack problem

- You are a jewel thief, and you want to steal the most valuable set of jewels possible from a set of $n$ jewels.
- Each jewel has a value $v_i$ and a weight $w_i$. 
Knapsack problem

- You are a jewel thief, and you want to steal the most valuable set of jewels possible from a set of $n$ jewels.
- Each jewel has a value $v_i$ and a weight $w_i$.
- You have a sack, which has a weight capacity $C$. 
Knapsack problem

- You are a jewel thief, and you want to steal the most valuable set of jewels possible from a set of $n$ jewels.
- Each jewel has a value $v_i$ and a weight $w_i$.
- You have a sack, which has a weight capacity $C$.
- What is the maximum value you can store in the sack?
Fractional Knapsack

- Say you also have a state-of-the-art laser cutter, and can cut the jewel to any real weight.
Fractional Knapsack

- Say you also have a state-of-the-art laser cutter, and can cut the jewel to any real weight.
- The ratio of the jewel’s value to its weight does not change.
Fractional Knapsack

- Say you also have a state-of-the-art laser cutter, and can cut the jewel to any real weight.
- The ratio of the jewel’s value to it’s weight does not change.
- What is the optimal strategy here?
Fractional knapsack

double fractionalKnapsack(List<Jewel> jewels, double capacity) {
    Collections.sort(jewels, (a, b) -> {
        return Double.compare(
            a.value / a.weight,
            b.value / b.weight
        );
    });
    Collections.reverse(jewels);
    double value = 0.0;
    for (Jewel jewel : jewels) {
        double taken = Math.min(capacity, jewel.weight);
        capacity -= taken;
        value += jewel.value * taken / jewel.weight;
    }
    return value;
}
Fractional knapsack

double fractionalKnapsack(List<Jewel> jewels, double capacity) {
    Collections.sort(jewels, (a, b) -> {
        return Integer.compare(a.value * b.weight, b.value * a.weight);
    });
    Collections.reverse(jewels);
    double value = 0.0;
    for (Jewel jewel : jewels) {
        double taken = Math.min(capacity, jewel.weight);
        capacity -= taken;
        value += taken * jewel.value / jewel.weight;
    }
    return value;
}
Fractional knapsack

do\linebreak \underline{double} fractionalKnapsack(List<Jewel> jewels, double capacity) {

    Collections.sort(jewels, (a, b) -> {
        return Integer.compare(
            a.value * b.weight,
            b.value * a.weight
        );
    });

    Collections.reverse(jewels);

    double value = 0.0;
    for (Jewel jewel : jewels) {
        double taken = Math.min(capacity, jewel.weight);
        capacity -= taken;
        value += taken * jewel.value / jewel.weight;
    }

    return value;
}

▶ Time Complexity?

\[\mathcal{O}(n\log n)\]

Space Complexity? \[\mathcal{O}(n)\]
Fractional knapsack

double fractionalKnapsack(List<Jewel> jewels, double capacity) {
    Collections.sort(jewels, (a, b) -> {
        return Integer.compare(a.value * b.weight, b.value * a.weight);
    });
    Collections.reverse(jewels);
    double value = 0.0;
    for (Jewel jewel : jewels) {
        double taken = Math.min(capacity, jewel.weight);
        capacity -= taken;
        value += taken * jewel.value / jewel.weight;
    }
    return value;
}

▶ Time Complexity? $O(n\log n)$
Fractional knapsack

double fractionalKnapsack(List<Jewel> jewels, double capacity) {
    Collections.sort(jewels, (a, b) -> {
        return Integer.compare(a.value * b.weight, b.value * a.weight);
    });
    Collections.reverse(jewels);
    double value = 0.0;
    for (Jewel jewel : jewels) {
        double taken = Math.min(capacity, jewel.weight);
        capacity -= taken;
        value += taken * jewel.value / jewel.weight;
    }
    return value;
}

▶ Time Complexity? $O(n \log n)$   Space Complexity?
Fractional knapsack

double fractionalKnapsack(List<Jewel> jewels, double capacity) {
    Collections.sort(jewels, (a, b) -> {
        return Integer.compare(
            a.value * b.weight,
            b.value * a.weight
        );
    });
    Collections.reverse(jewels);
    double value = 0.0;
    for (Jewel jewel : jewels) {
        double taken = Math.min(capacity, jewel.weight);
        capacity -= taken;
        value += taken * jewel.value / jewel.weight;
    }
    return value;
}

- Time Complexity? $O(n \log n)$  
- Space Complexity? $O(n)$
Infinite knapsack

Now say we can’t use our laser cutter, but we’ve stumbled on the motherload of jewels, and effectively have an infinite number of any jewel.
Infinite knapsack

- Now say we can’t use our laser cutter, but we’ve stumbled on the motherload of jewels, and effectively have an infinite number of any jewel.
- Now what’s the best strategy?
Infinite knapsack

- What state do we need to keep track of?
Infinite knapsack

- What state do we need to keep track of?
- How can we change our state? (hint: two ways)
Infinite knapsack

- What state do we need to keep track of?
- How can we change our state? (hint: two ways)
  1. Increase capacity
Infinite knapsack

- What state do we need to keep track of?
- How can we change our state? (hint: two ways)
  1. Increase capacity
  2. Add a jewel
Infinite knapsack

- What state do we need to keep track of?
- How can we change our state? (hint: two ways)
  1. Increase capacity
  2. Add a jewel
- Say we have all of these precomputed for lesser values
Infinite knapsack

- What state do we need to keep track of?
- How can we change our state? (hint: two ways)
  1. Increase capacity
  2. Add a jewel
- Say we have all of these precomputed for lesser values
- How do you update your state with each of these operations?
Infinite knapsack

- What state do we need to keep track of?
- How can we change our state? (hint: two ways)
  1. Increase capacity
  2. Add a jewel
- Say we have all of these precomputed for lesser values
- How do you update your state with each of these operations?
  1. Try taking every jewel first
Infinite knapsack

- What state do we need to keep track of?
- How can we change our state? (hint: two ways)
  1. Increase capacity
  2. Add a jewel

- Say we have all of these precomputed for lesser values
- How do you update your state with each of these operations?
  1. Try taking every jewel first
  2. Recompute all capacities
Infinite knapsack

```java
int infKnapsack(List<Jewel> jewels, int maxCapacity) {
    int[] bestValue = new int[maxCapacity + 1];
    bestValue[0] = 0;
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (Jewel jewel : jewels)
            if (jewel.weight <= cap)
                bestValue[cap] = Math.max(
                    bestValue[cap],
                    jewel.value +
                    bestValue[cap - jewel.weight]
                );
    return bestValue[maxCapacity];
}
```

▶ Time complexity?
\(O(nC)\)

▶ Space complexity?
\(O(n + C)\)
null
Infinite knapsack

```java
int infKnapsack(List<Jewel> jewels, int maxCapacity) {
    int[] bestValue = new int[maxCapacity + 1];
    bestValue[0] = 0;
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (Jewel jewel : jewels)
            if (jewel.weight <= cap)
                bestValue[cap] = Math.max(
                    bestValue[cap],
                    jewel.value +
                    bestValue[cap - jewel.weight]);
    return bestValue[maxCapacity];
}
```

▶ Time complexity? $O(nC)$
Infinite knapsack

```java
int infKnapsack(List<Jewel> jewels, int maxCapacity) {
    int[] bestValue = new int[maxCapacity + 1];
    bestValue[0] = 0;
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (Jewel jewel : jewels)
            if (jewel.weight <= cap)
                bestValue[cap] = Math.max(
                    bestValue[cap],
                    jewel.value +
                    bestValue[cap - jewel.weight]);
    return bestValue[maxCapacity];
}
```

- Time complexity? $O(nC)$
- Space complexity?
Infinite knapsack

```java
int infKnapsack(List<Jewel> jewels, int maxCapacity) {
    int[] bestValue = new int[maxCapacity + 1];
    bestValue[0] = 0;
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (Jewel jewel : jewels)
            if (jewel.weight <= cap)
                bestValue[cap] = Math.max(
                    bestValue[cap],
                    jewel.value +
                    bestValue[cap - jewel.weight]
                );
    return bestValue[maxCapacity];
}
```

- Time complexity? $O(nC)$
- Space complexity? $O(n + C)$
0 / 1 Knapsack

▶ This is the most common variant of knapsack. Each item can be taken at most once.
0 / 1 Knapsack

- This is the most common variant of knapsack. Each item can be taken at most once.
- How can we modify our previous solution to solve this?
This is the most common variant of knapsack. Each item can be taken at most once.

How can we modify our previous solution to solve this?

Add another factor to our state. Using the first $i$ jewels, what is the maximum value we can get with capacity $j$?
0 / 1 Knapsack

- This is the most common variant of knapsack. Each item can be taken at most once.
- How can we modify our previous solution to solve this?
- Add another factor to our state. Using the first $i$ jewels, what is the maximum value we can get with capacity $j$?
- How do we transition between states now?
0 / 1 Knapsack

- This is the most common variant of knapsack. Each item can be taken at most once.
- How can we modify our previous solution to solve this?
- Add another factor to our state. Using the first \( i \) jewels, what is the maximum value we can get with capacity \( j \)?
- How do we transition between states now?
- Must think about iteration order. Do we iterate over jewels then capacities, or capacities then jewels?
```java
int knapsack(List<Jewel> jewels, int maxCapacity) {
    int n = jewels.size();
    int[][] bestValue = new int[n][maxCapacity + 1];
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (int i = 1; i < n; i++)
            jewel = jewels.get(i - 1);
            bestValue[i][cap] = bestValue[i - 1][cap];
            if (jewel.weight <= cap)
                bestValue[i][cap] = Math.max(
                    bestValue[i][cap],
                    jewel.value +
                    bestValue[i - 1][cap - jewel.weight]
                );
    return bestValue[maxCapacity];
}
```

▶ Time Complexity?

O(nC)

▶ Space Complexity?

O(nC)
0/1 Knapsack

```java
int knapsack(List<Jewel> jewels, int maxCapacity) {
    int n = jewels.size();
    int[][] bestValue = new int[n][maxCapacity + 1];
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (int i = 1; i < n; i++)
            jewel = jewels.get(i - 1);
            bestValue[i][cap] = bestValue[i - 1][cap];
    if (jewel.weight <= cap)
        bestValue[i][cap] = Math.max(
            bestValue[i][cap],
            jewel.value +
            bestValue[i - 1][cap - jewel.weight]
        );
    return bestValue[maxCapacity];
}
```

► Time Complexity? \(O(nC)\)
0/1 Knapsack

```java
int knapsack(List<Jewel> jewels, int maxCapacity) {
    int n = jewels.size();
    int[][] bestValue = new int[n][maxCapacity + 1];
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (int i = 1; i < n; i++)
            jewel = jewels.get(i - 1);
            bestValue[i][cap] = bestValue[i - 1][cap];
        if (jewel.weight <= cap)
            bestValue[i][cap] = Math.max(
                bestValue[i][cap],
                jewel.value +
                bestValue[i - 1][cap - jewel.weight]
            );
    return bestValue[maxCapacity];
}
```

- Time Complexity? \( O(nC) \)
- Space Complexity?
0/1 Knapsack

```java
int knapsack(List<Jewel> jewels, int maxCapacity) {
    int n = jewels.size();
    int[][] bestValue = new int[n][maxCapacity + 1];
    for (int cap = 1; cap <= maxCapacity; cap++)
        for (int i = 1; i < n; i++)
            jewel = jewels.get(i - 1);
            bestValue[i][cap] = bestValue[i - 1][cap];
            if (jewel.weight <= cap)
                bestValue[i][cap] = Math.max(
                    bestValue[i][cap],
                    jewel.value +
                    bestValue[i - 1][cap - jewel.weight]
                );
    return bestValue[maxCapacity];
}
```

- Time Complexity? $O(nC)$
- Space Complexity? $O(nC)$
Bounded Knapsack

- Now in addition to price $p_i$ and weight $w_i$, each jewel has a count $c_i$.

  - Option 1: Just make $c_i$ copies of each jewel.
  - What is the runtime of this?
    
    $$O\left(\sum_{i} c_i C\right) = O(ncC)$$
Bounded Knapsack

- Now in addition to price $p_i$ and weight $w_i$, each jewel has a count $c_i$.
- How can we modify our previous solution to solve this?
Bounded Knapsack

- Now in addition to price $p_i$ and weight $w_i$, each jewel has a count $c_i$.
- How can we modify our previous solution to solve this?
- Option 1: Just make $c_i$ copies of each jewel.
Bounded Knapsack

- Now in addition to price $p_i$ and weight $w_i$, each jewel has a count $c_i$.
- How can we modify our previous solution to solve this?
- Option 1: Just make $c_i$ copies of each jewel.
- What is the runtime of this?
Bounded Knapsack

- Now in addition to price \( p_i \) and weight \( w_i \), each jewel has a count \( c_i \).
- How can we modify our previous solution to solve this?
- Option 1: Just make \( c_i \) copies of each jewel.
- What is the runtime of this? \( O(\sum_i c_i C) = O(ncC) \)
Bounded Knapsack

- Now in addition to price $p_i$ and weight $w_i$, each jewel has a count $c_i$.
- How can we modify our previous solution to solve this?
- Option 1: Just make $c_i$ copies of each jewel.
- What is the runtime of this? $O(\sum_i c_i C) = O(ncC)$
- Can we do better?
Bounded Knapsack

- For each jewel \((v_i, w_i, c_i)\), create the following jewels and solve 0 / 1 Knapsack:

  1. \((v_i, w_i)\)
  2. \((2v_i, 2w_i)\)
  3. \((4v_i, 4w_i)\)
  4. \((8v_i, 8w_i)\)
  5. \(\ldots\)
  6. \((c_i - \sum 2i, v_i), (c_i - \sum 2i, w_i)\)

- Any amount of a jewel can be represented as a sum of values from this list.

- How many items are in this list? \(O(\log(c_i))\).

- What is the runtime of this? \(O(\sum i \log(c_i) C) = O(n \log(c_i) C)\).
Bounded Knapsack

For each jewel \((v_i, w_i, c_i)\), create the following jewels and solve 0 / 1 Knapsack:

1. \((v_i, w_i)\)
2. \((2 \cdot v_i, 2 \cdot w_i)\)
3. \((4 \cdot v_i, 4 \cdot w_i)\)
4. \((8 \cdot v_i, 8 \cdot w_i)\)
5. \(\cdots (2^k \cdot v_i, 2^k \cdot w_i)\)
6. \(((c_i - \sum 2^i) v_i, (c_i - \sum 2^i) w_i)\)
Bounded Knapsack

- For each jewel \((v_i, w_i, c_i)\), create the following jewels and solve 0 / 1 Knapsack:
  1. \((v_i, w_i)\)
  2. \((2 \cdot v_i, 2 \cdot w_i)\)
  3. \((4 \cdot v_i, 4 \cdot w_i)\)
  4. \((8 \cdot v_i, 8 \cdot w_i)\)
  5. \(\cdots (2^k \cdot v_i, 2^k \cdot w_i)\)
  6. \(((c_i - \sum 2^i)v_i, (c_i - \sum 2^i)w_i)\)

- Any amount of a jewel can be represented as a sum of values from this list.
Bounded Knapsack

For each jewel \((v_i, w_i, c_i)\), create the following jewels and solve 0 / 1 Knapsack:

1. \((v_i, w_i)\)
2. \((2 \cdot v_i, 2 \cdot w_i)\)
3. \((4 \cdot v_i, 4 \cdot w_i)\)
4. \((8 \cdot v_i, 8 \cdot w_i)\)
5. \(\cdots \ (2^k \cdot v_i, 2^k \cdot w_i)\)
6. \(((c_i - \sum 2^i) v_i, (c_i - \sum 2^i) w_i)\)

Any amount of a jewel can be represented as a sum of values from this list.

How many items are in this list?
Bounded Knapsack

- For each jewel \((v_i, w_i, c_i)\), create the following jewels and solve 0 / 1 Knapsack:
  1. \((v_i, w_i)\)
  2. \((2 \cdot v_i, 2 \cdot w_i)\)
  3. \((4 \cdot v_i, 4 \cdot w_i)\)
  4. \((8 \cdot v_i, 8 \cdot w_i)\)
  5. \(\cdots (2^k \cdot v_i, 2^k \cdot w_i)\)
  6. \(((c_i - \sum 2^i)v_i, (c_i - \sum 2^i)w_i)\)

- Any amount of a jewel can be represented as a sum of values from this list.

- How many items are in this list? \(O(\log(c_i))\).
Bounded Knapsack

- For each jewel \((v_i, w_i, c_i)\), create the following jewels and solve 0 / 1 Knapsack:
  1. \((v_i, w_i)\)
  2. \((2 \cdot v_i, 2 \cdot w_i)\)
  3. \((4 \cdot v_i, 4 \cdot w_i)\)
  4. \((8 \cdot v_i, 8 \cdot w_i)\)
  5. \(\cdots \ (2^k \cdot v_i, 2^k \cdot w_i)\)
  6. \((c_i - \sum 2^i) v_i, (c_i - \sum 2^i) w_i)\)

- Any amount of a jewel can be represented as a sum of values from this list.

- How many items are in this list? \(O(\log(c_i))\).

- What is the runtime of this?
Bounded Knapsack

- For each jewel \((v_i, w_i, c_i)\), create the following jewels and solve 0 / 1 Knapsack:
  1. \((v_i, w_i)\)
  2. \((2 \cdot v_i, 2 \cdot w_i)\)
  3. \((4 \cdot v_i, 4 \cdot w_i)\)
  4. \((8 \cdot v_i, 8 \cdot w_i)\)
  5. \((2^k \cdot v_i, 2^k \cdot w_i)\)
  6. \(((c_i - \sum 2^i)v_i, (c_i - \sum 2^i)w_i)\)

- Any amount of a jewel can be represented as a sum of values from this list.

- How many items are in this list? \(O(\log(c_i))\).

- What is the runtime of this? \(O(\sum_i \log(c_i)C) = O(n \log(c)C)\)