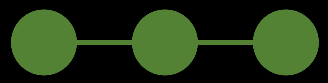
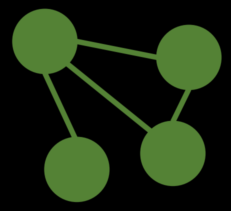
Algorithm Paradigms

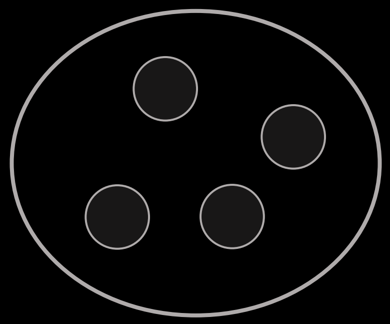
Data Structures & Algorithm

Categories of Data Structures

Linear Ordered	Non-linear Ordered	Not Ordered
Lists	Trees	Sets
Stacks	Graphs	Tables/Maps
Queues		







Categories of Algorithms

Brute Force	Divide & Conquer	Greedy	Dynamic Programming
Selection Sort	Binary Search	Minimum Spanning Tree	Knapsack
Bubble Sort	Merge Sort	Shortest Paths	Fibonacci
Insertion Sort	Quick Sort		
NP Complete Problems			

Algorithmic Paradigms

Algorithmic Paradigms

	Properties	Examples
Brute Force	▪ Generate and Test an Exhaustive Set of all possible combinations ▪ Can be computationally very expensive ▪ Guarantees correct solution	▪ Finding divisors of a number, n by checking if all numbers from 1..n divides n without remainder ▪ Finding duplicates using all combinations ▪ Bubble/Selection Sort
Divide and Conquer	▪ Break the problem into subcomponents typically using recursion ▪ Solve the basic component ▪ Combine the solutions to sub-problems	▪ Quick Sort ▪ Merge Sort ▪ Binary Search ▪ Peak Finding
Dynamic Programming	▪ Optimal substructure: solution to a large problem can be obtained by solution to a smaller optimal problems e.g., Shortest path in a graph (Longest path does not follow optimal substructure) ▪ Overlapping sub-problems: space of sub-problems must be small, that is, any recursive algorithm solving the problem should solve the same sub-problems over and over, rather than generating new sub-problems. ▪ Guarantees optimal solution in terms of correctness and time	▪ Fibonacci Sequence ▪ Assembly Scheduling ▪ Knapsack ▪ Bin packing
Greedy Algorithms	▪ Local optimal solutions at each stage ▪ Does not guarantee optimal solution	▪ Prim’s Algorithm ▪ Dijkstra’s Algorithm ▪ Kruskal’s Algorithm ▪ Bin packing

Optimization problems

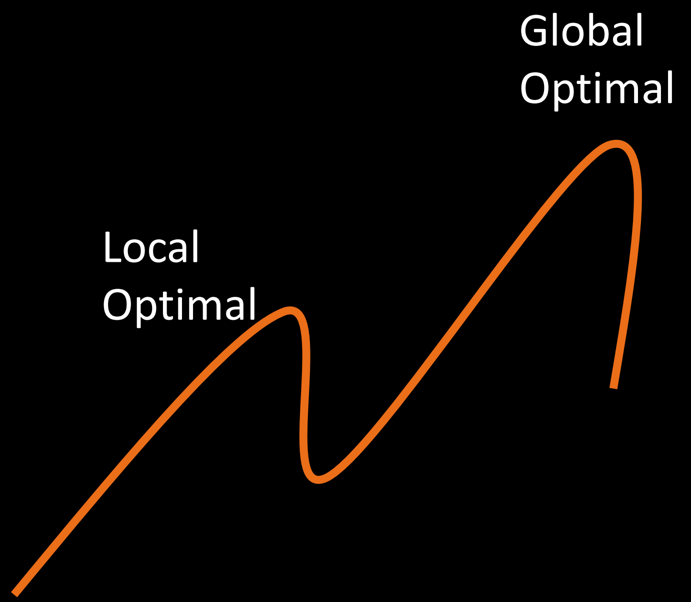
- Optimization Problem: the problem of finding the best solution from all feasible solutions.
- Constraints to a Problem: Minimize or Maximize an Objective function (e.g., go from A->B in 10 hours, Objective function: minimum time). Objective functions define the objective of the optimization and is a single scalar value that is formulated from a set of design responses.
- Feasible Solution: A feasible solution is a set of values for the decision variables that satisfies all of the constraints in an optimization problem.
- Optimal Solution
  - An *optimal solution* is a feasible solution where the objective function reaches its maximum (or minimum) value.
  - A *globally optimal solution* is one where there are no other feasible solutions with better objective function values.
  - A *locally optimal solution* is one where there are no other feasible solutions "in the vicinity" with better objective function values – you can picture this as a point at the top of a "peak" or at the bottom of a "valley" which may be formed by the objective function and/or the constraints.

Optimization problems

- Optimization Problem: find code that runs fast in the course where fast means that it takes less than 1 second to execute and is passing all tests.
- Constraints to a Problem: Minimize an Objective function – minimize time. Constraints: must execute in less than 60 sec.
- Feasible Solution: Lot of student code that passes all tests and executes in less than 1 second.
- Optimal Solution
  - A *globally optimal solution* is the fastest running code.
  - A *locally optimal solution* is one where you pick the submissions which were submitted earlier than later based on a heuristic.

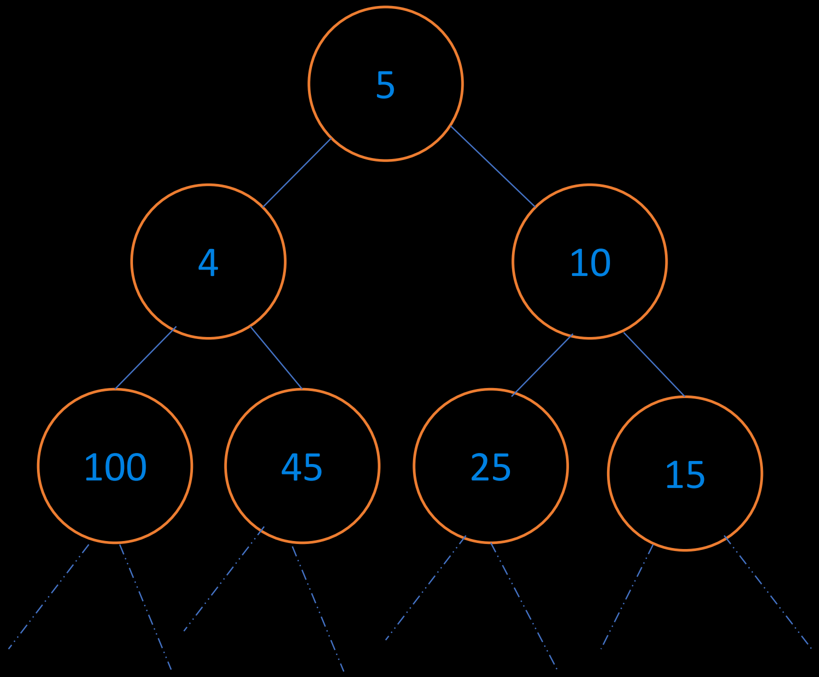
Greedy Algorithms

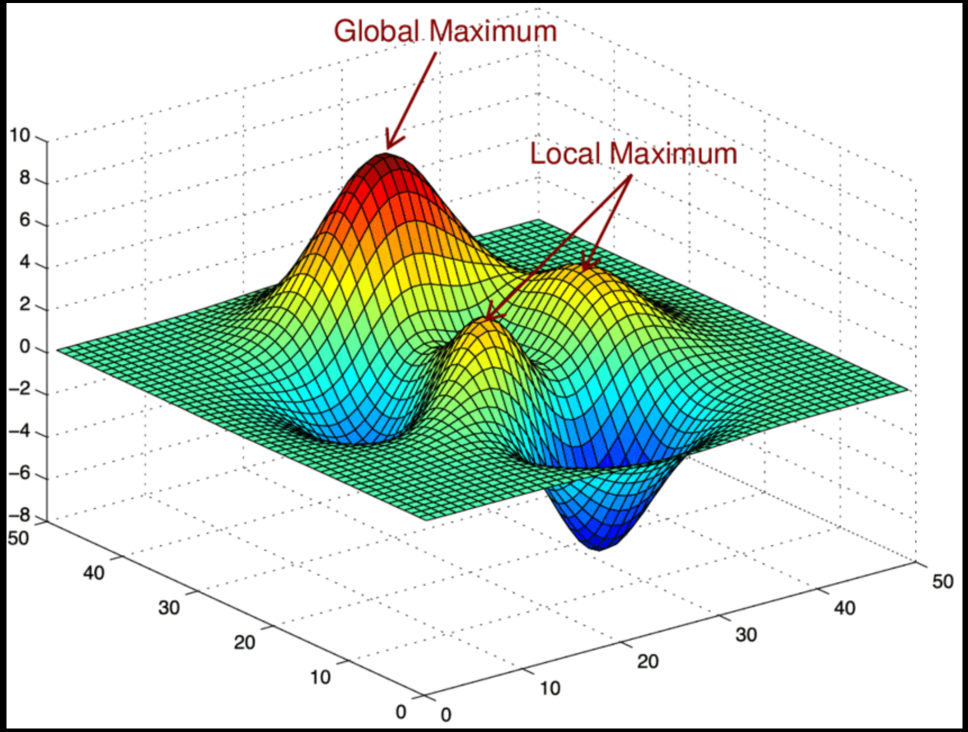
Greedy Algorithms



- Greedy algorithms work in phases.
  - In each phase a decision is made that appears to be good.
  - Generally, this means that a local optimum is chosen.
- At the end, we hope the local optimum found is the global optimum.
  - Prim's algorithm and Dijkstra's algorithm are greedy algorithms.

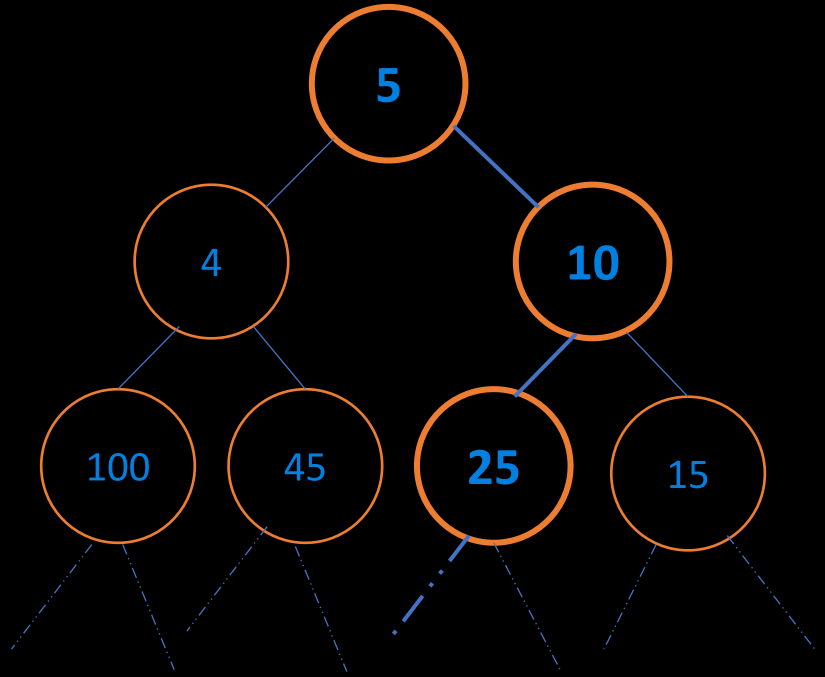
Greedy Algorithms

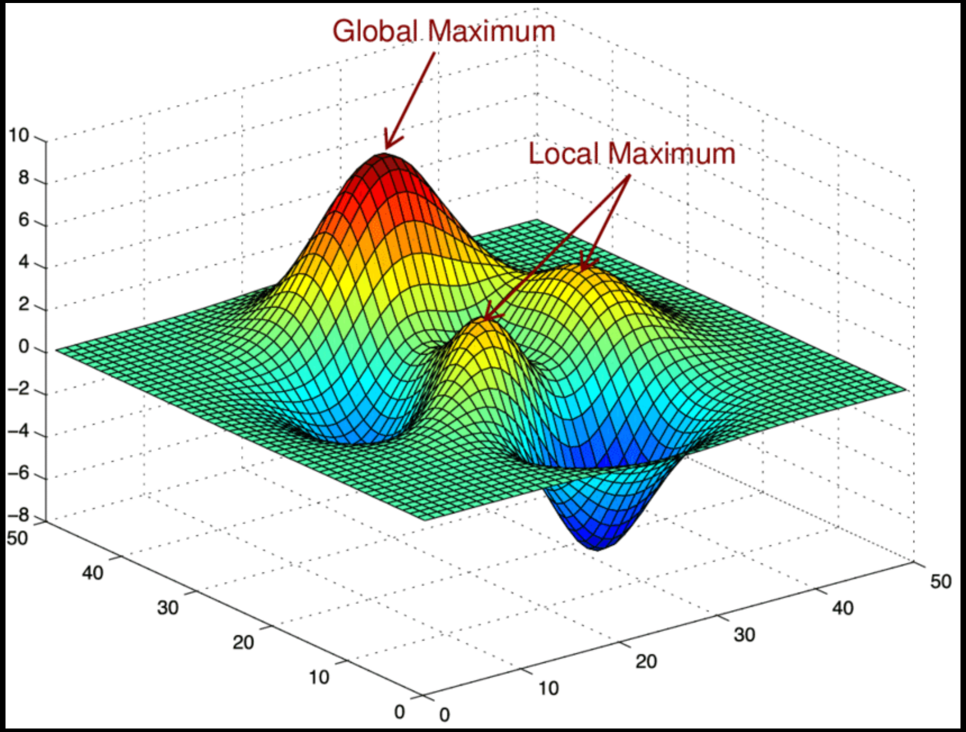




https://images.app.goo.gl/nehZPYDjn3LxjJey8

Greedy Algorithms





https://images.app.goo.gl/nehZPYDjn3LxjJey8

Coin Change

- Make change with the smallest number of coins.
- Start with the largest denomination and dispense as many of those fit.

- Example: 32 cents
  - Quarters (0.25)
  - Dimes (0.10)
  - Nickels (0.05)
  - Pennies (0.01)

Coin Change

- Make change with the smallest number of coins.
- Start with the largest denomination and dispense as many of those fit.

- Example: 32 cents
  - Quarters (0.25) – 1
  - Dimes (0.10) – 0
  - Nickels (0.05) – 1
  - Pennies (0.01) – 2
- = 4 coins, the minimum possible

Coin Change

- Make change with the smallest number of coins.
- Start with the largest denomination and dispense as many of those fit.

- Example: 32 cents
  - Quarters (0.25) – 1
  - Dimes (0.10) – 0
  - Nickels (0.05) – 1
  - Pennies (0.01) – 2
- = 4 coins, the minimum possible

- Example, alternate universe: 32 cents
  - Gryffindor (0.14)
  - Slytherin (0.08)
  - Hufflepuffs (0.01)

Coin Change

- Make change with the smallest number of coins.
- Start with the largest denomination and dispense as many of those fit.

- Example: 32 cents
  - Quarters (0.25) – 1
  - Dimes (0.10) – 0
  - Nickels (0.05) – 1
  - Pennies (0.01) – 2
- = 4 coins, the minimum possible

- Example, alternate universe: 32 cents
  - Gryffindor (0.14) – 2
  - Slytherin (0.08) – 0
  - Hufflepuffs (0.01) – 4
  - = 6 coins, not the minimum possible as 4 Slytherins is global optimal

Bin Packing

If we have boxes that each require 14 units, 16 units, 4 units and 6 units of space, how many minimum bins are required to store all the four boxes if each bin can take at most 20 units of space using the following Greedy strategies

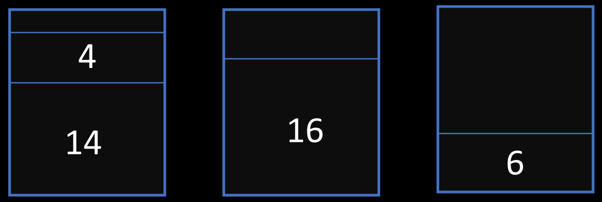
- First Fit: scan the bins and place the new item in the first bin that is large enough.

- Best Fit: scan the bins and place the new item in the bin that finds the spot that creates the smallest empty space

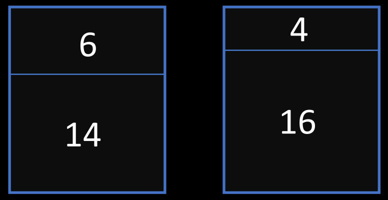
Bin Packing

If we have boxes that each require 14 units, 16 units, 4 units and 6 units of space, how many minimum bins are required to store all the four boxes if each bin can take at most 20 units of space using the following Greedy strategies

- First Fit: scan the bins and place the new item in the first bin that is large enough.



- Best Fit: scan the bins and place the new item in the bin that finds the spot that creates the smallest empty space

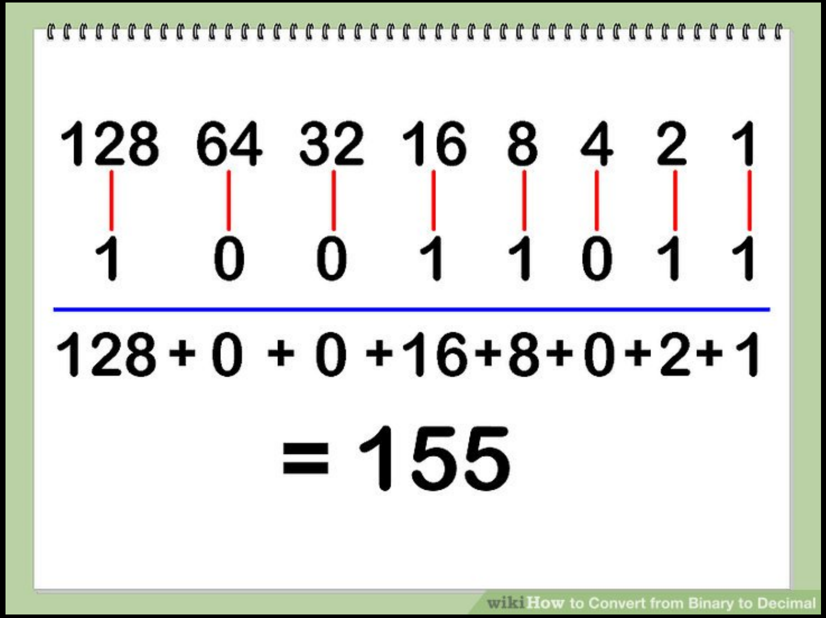


Greedy Algorithm for Converting Decimal to Binary

- Convert decimal to binary.

- Start with the largest power of two that will fit.
- Assign 1 to that place. Subtract that value.
- Repeat.

  - Example: 280
    - 2^8 = 256
    - ninth significant digit = 1
      - 100000000
      - 280-256= 24
    - 2^4 = 16
    - fifth significant digit = 1
      - 100010000
      - 24-16=8
    - 2^3 = 8
    - fourth significant digit = 1
      - 100011000
      - 8-8 = 0, done



Huffman Trees

Rationale

- How is text transmitted?
- How many bits we need to encode a character?

Rationale

- How is text transmitted?
  - 1's and 0's
- How many bits we need to encode a character?
  - $\lceil \log_2 n \rceil$

Rationale

- How is text transmitted? : mississippi

- How many bits we need to encode a character?

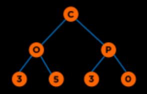
  - $\lceil \log_2 4 \rceil = 2$ bits per character
  - 11 * 2 = 22 bits to transmit "mississippi"

Algorithm for Huffman Encoding

1. Create a table with symbols and their frequencies

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

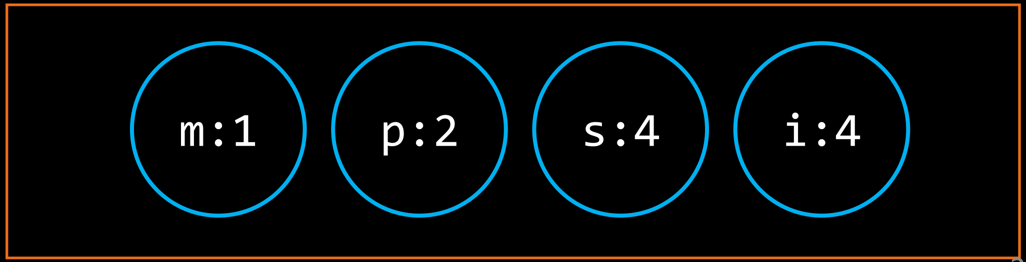


Algorithm for Huffman Encoding

2. Construct a set of trees with root nodes that contain each of the individual symbols and their weight (frequency).
3. Place the set of trees into a priority queue.

mississippi

Character	Frequency
m	1
i	4
s	4
p	2



Algorithm for Huffman Encoding

```
4. while the priority queue has more than one item
        Remove the two trees with the smallest weights.
        Combine them into a new binary tree in which the weight of the tree
        root is the sum of the weights of its children.
        Insert the newly created tree back into the priority queue.
```

mississippi

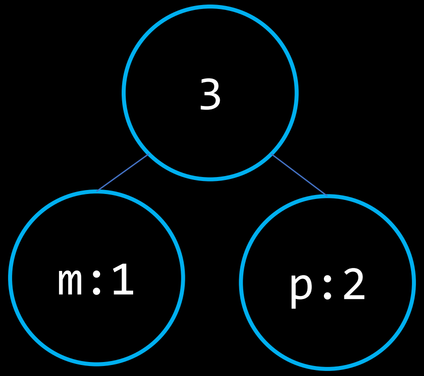
Character	Frequency
m	1
i	4
s	4
p	2

Algorithm for Huffman Encoding

```
4. while the priority queue has more than one item
        Remove the two trees with the smallest weights.
        Combine them into a new binary tree in which the weight of the tree
        root is the sum of the weights of its children.
        Insert the newly created tree back into the priority queue.
```

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

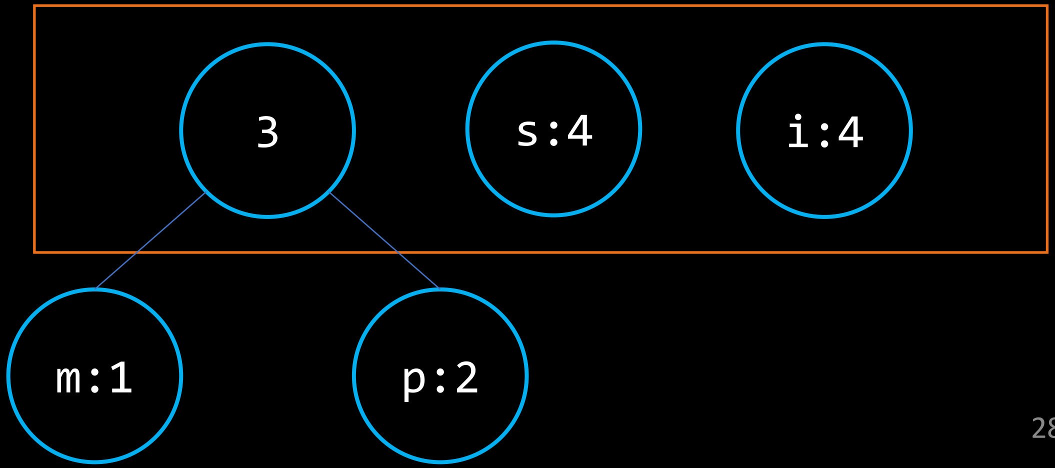


Algorithm for Huffman Encoding

```
4. while the priority queue has more than one item
        Remove the two trees with the smallest weights.
        Combine them into a new binary tree in which the weight of the tree
        root is the sum of the weights of its children.
        Insert the newly created tree back into the priority queue.
```

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

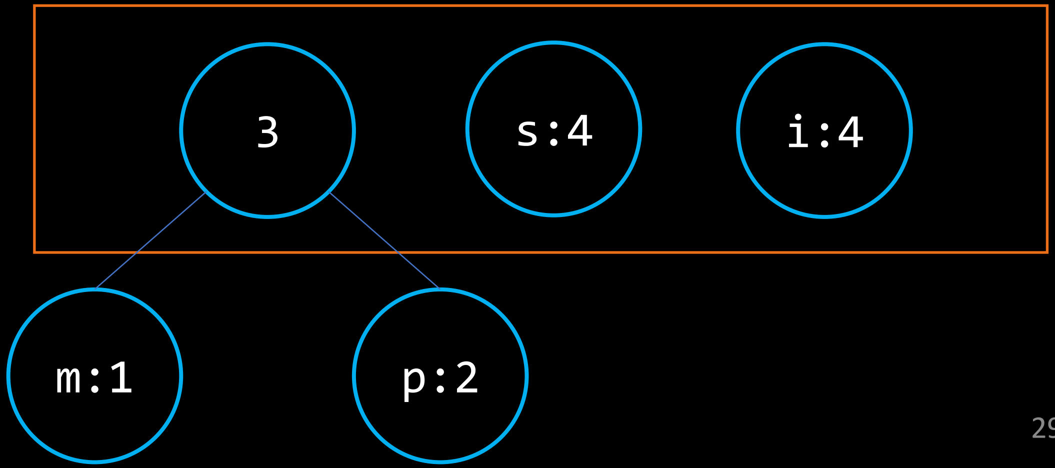


Algorithm for Huffman Encoding

4. while the priority queue has more than one item
   Remove the two trees with the smallest weights.
   Combine them into a new binary tree in which the weight of the tree root is the sum of the weights of its children.
   Insert the newly created tree back into the priority queue.

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

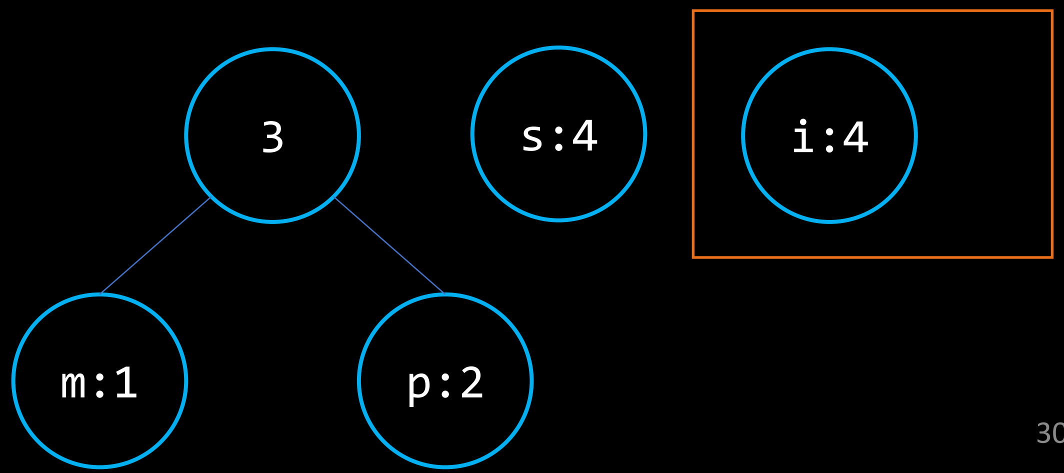


Algorithm for Huffman Encoding

4. while the priority queue has more than one item
   Remove the two trees with the smallest weights.
   Combine them into a new binary tree in which the weight of the tree root is the sum of the weights of its children.
   Insert the newly created tree back into the priority queue.

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

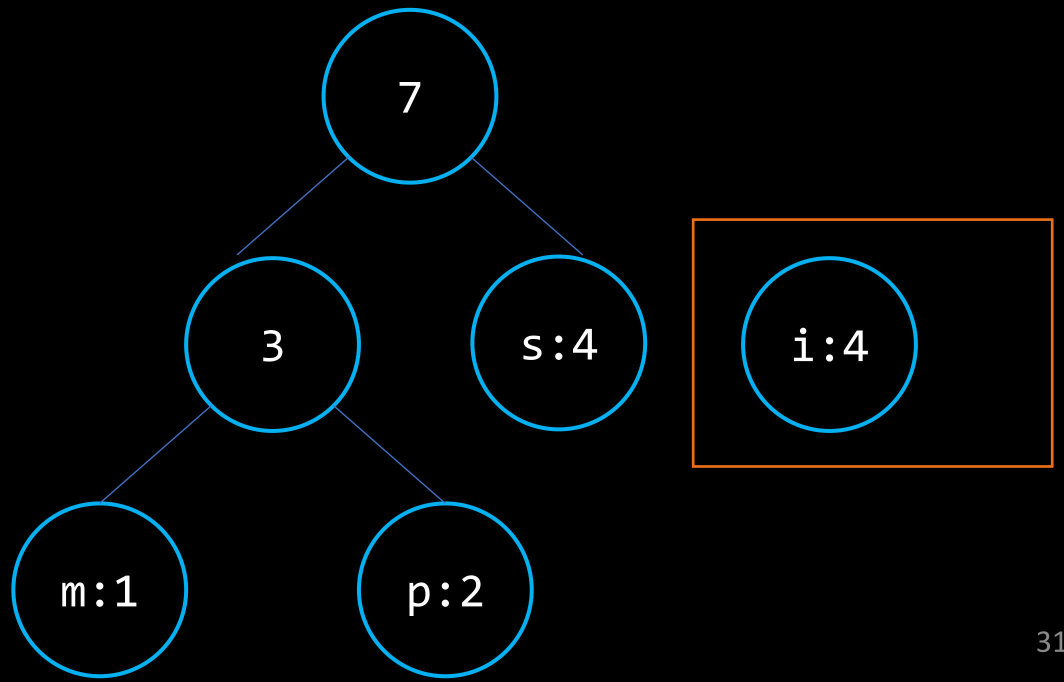


Algorithm for Huffman Encoding

```
4. while the priority queue has more than one item
        Remove the two trees with the smallest weights.
        Combine them into a new binary tree in which the weight of the tree
        root is the sum of the weights of its children.
        Insert the newly created tree back into the priority queue.
```

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

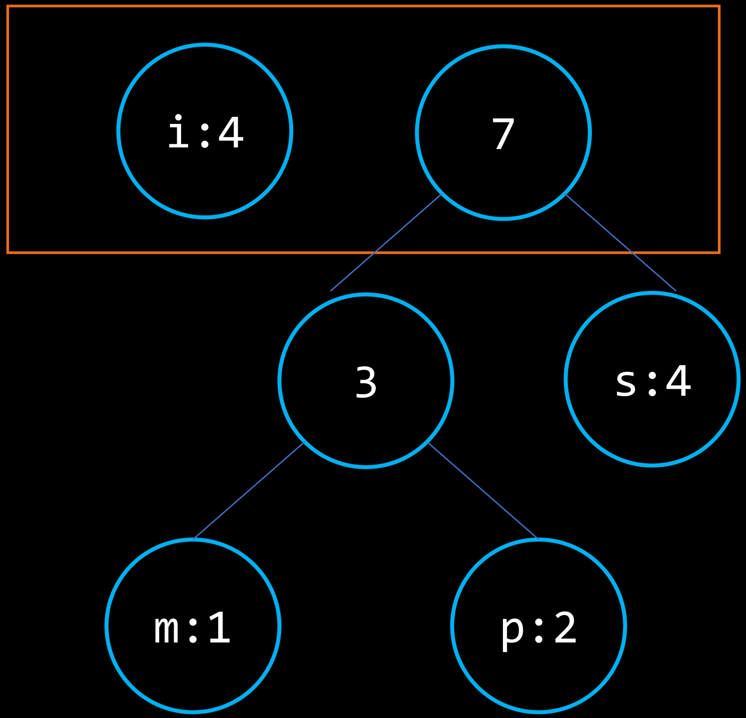


Algorithm for Huffman Encoding

4. while the priority queue has more than one item
   Remove the two trees with the smallest weights.
   Combine them into a new binary tree in which the weight of the tree root is the sum of the weights of its children.
   Insert the newly created tree back into the priority queue.

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

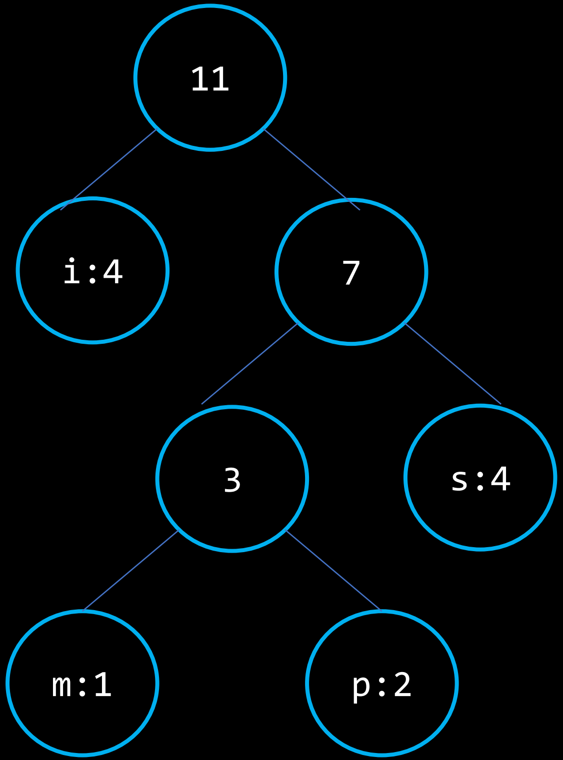


Algorithm for Huffman Encoding

4. while the priority queue has more than one item
   Remove the two trees with the smallest weights.
   Combine them into a new binary tree in which the weight of the tree root is the sum of the weights of its children.
   Insert the newly created tree back into the priority queue.

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

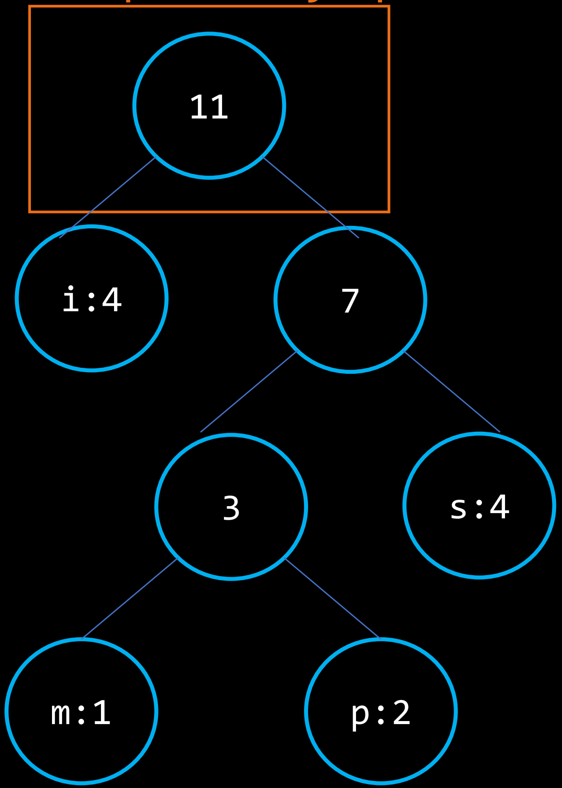


Algorithm for Huffman Encoding

4. while the priority queue has more than one item
    Remove the two trees with the smallest weights.
    Combine them into a new binary tree in which the weight of the tree root is the sum of the weights of its children.
    Insert the newly created tree back into the priority queue.

mississippi

Character	Frequency
m	1
i	4
s	4
p	2

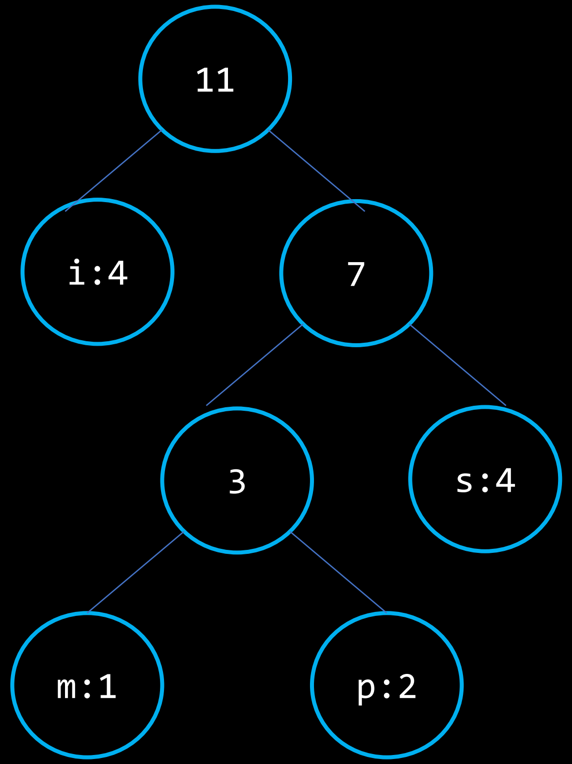


Algorithm for Huffman Encoding

5. Traverse the resulting tree to obtain binary codes for characters: 0 towards left, 1 to towards right

mississippi

Character	Frequency	Code	Bits per Code
m	1	100	3
i	4	0	1
s	4	11	2
p	2	101	3

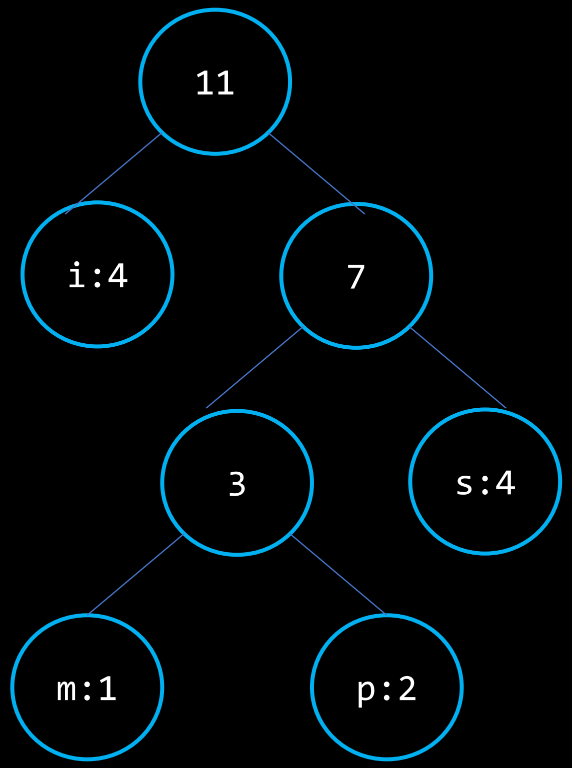


Algorithm for Huffman Encoding

5. Traverse the resulting tree to obtain binary codes for characters: 0 towards left, 1 to towards right

mississippi

Character	Frequency	Code	Bits per Code
m	1	100	3
i	4	0	1
s	4	11	2
p	2	101	3



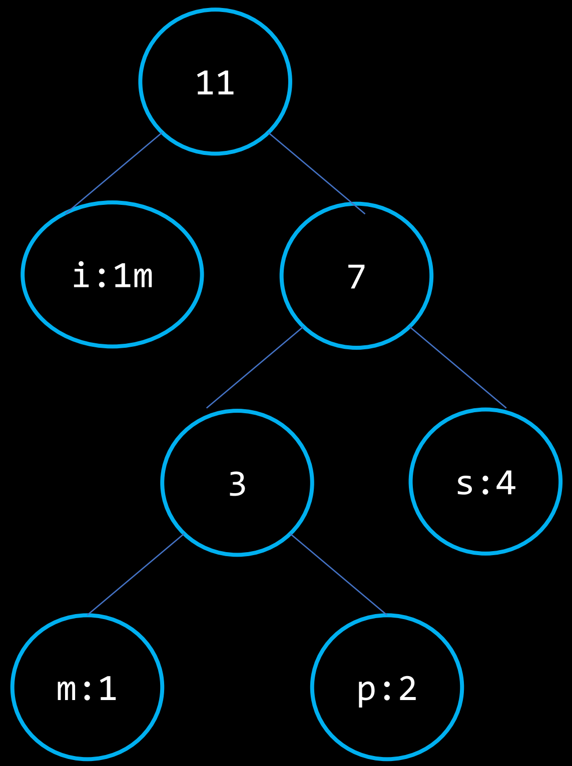
Total bits for "mississippi" =

1*3 + 4*1 + 4*2 + 2*3 = 21 bits

Compression: 21/22 ~ 95.5%

Algorithm for Huffman Encoding

What if there were 1 million succeeding 'i' in "mississippi"?



mississippi...

Character	Frequency	Code	Bits per Code
m	1	100	3
i	1,000,004	0	1
s	4	11	2
p	2	101	3

Total bits for "mississippi…." = 1*3 + 1000004*1 + 4*2 + 2*3 = 1,000,021 bits [Huffman codes]

Total bits for "mississippi…." through regular transmission = 1000011*2 = 2,000,022 bits

Compression: 1000021/2000022 ~ 50%

Algorithm for Huffman Encoding: Interface

```
class huffman_tree
{
    private:
   //add your data structures here

    public:

        /*
        Preconditions: input is a string of characters with ascii values 0-127
        Postconditions: reads the characters of input and constructs a
        huffman tree based on the character frequencies of the file contents
        */
        huffman_tree(const string& input) {}

        /*
        Preconditions: input is a character with ascii value between 0-127
        Postconditions: returns the Huffman code for character if character
                        is in the tree and an empty string otherwise.
        */
        string get_character_code(char character) const { return ""; }

/*
   Preconditions: input is a string of characters with ascii values 0-127
   Postconditions: returns the Huffman encoding for the contents of
                   file_name if file name exists and an empty string
                   otherwise. If the file contains letters not present in the
                   huffman_tree, return an empty string
*/
string encode(const string& input) const { return ""; }

/*
Preconditions: string_to_decode is a string containing Huffman-encoded text
Postconditions: returns the plaintext represented by the string if the string
                is a valid Huffman encoding and an empty string otherwise
*/
   string decode(const string& string_to_decode) const { return ""; }

};
```

Questions

Mentimeter

Code

7058 2268

Dynamic Programming

Dynamic Programming

- Top-down DP: Memoization
- Bottom-up DP: Tabulation

Fibonacci Sequence

```
int standardFibonacci(int n)
{
        if (n <= 1)
                return n;
        return standardFibonacci(n-1) + standardFibonacci(n-2);
}
```

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence

```
int standardFibonacci(int n)
{
        if (n <= 1)
                return n;
        return standardFibonacci(n-1) + standardFibonacci(n-2);
}
```

```
Time Complexity:
Space Complexity:
```

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence

```
int standardFibonacci(int n)
{
        if (n <= 1)
                return n;
        return standardFibonacci(n-1) + standardFibonacci(n-2);
}
```

Time Complexity: $O(2^n)$
Space Complexity: $O(n)$

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence: Tabulation

```
int bottomUpDPFibonacci(int n)
{
    vector<int> dp(n+1);
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++)
        dp[i] = dp[i - 1] + dp[i - 2];
    return dp[n];
}
```

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence: Tabulation

```
int bottomUpDPFibonacci(int n)
{
    vector<int> dp(n+1);
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++)
        dp[i] = dp[i - 1] + dp[i - 2];
    return dp[n];
}
```

```
Time Complexity:
Space Complexity:
```

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence: Tabulation

```
int bottomUpDPFibonacci(int n)
{
    vector<int> dp(n+1);
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++)
        dp[i] = dp[i - 1] + dp[i - 2];
    return dp[n];
}
```

```
Time Complexity: O(n)
Space Complexity: O(n)
```

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence: Memoization

```
int fib(vector<int> &dp, int n)
{
    if (n <= 1)
        return n;
    if (dp.at(n) != 0)
        return dp.at(n);
    dp.at(n) = fib(dp, n - 1) + fib(dp, n - 2);
    return dp.at(n);
}
```

```
int topDownDPFibonacci(int n)
{
    vector<int> dp(n + 1, 0);
    return fib(dp, n);
}
```

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence: Memoization

```
int fib(vector<int> &dp, int n)
{
    if (n <= 1)
        return n;
    if (dp.at(n) != 0)
        return dp.at(n);
    dp.at(n) = fib(dp, n - 1) + fib(dp, n - 2);
    return dp.at(n);
}
```

```
int topDownDPFibonacci(int n)
{
    vector<int> dp(n + 1, 0);
    return fib(dp, n);
}
```

```
Time Complexity:
Space Complexity:
```

https://onlinegdb.com/S1VSAMjUu

Fibonacci Sequence: Memoization

```
int fib(vector<int> &dp, int n)
{
    if (n <= 1)
        return n;
    if (dp.at(n) != 0)
        return dp.at(n);
    dp.at(n) = fib(dp, n - 1) + fib(dp, n - 2);
    return dp.at(n);
}
```

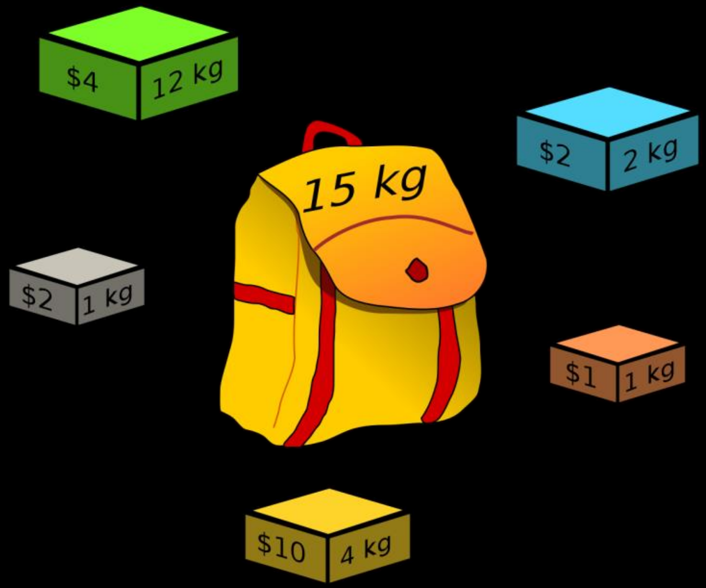
```
int topDownDPFibonacci(int n)
{
    vector<int> dp(n + 1, 0);
    return fib(dp, n);
}
```

Time Complexity: O(n)
Space Complexity: O(n)

https://onlinegdb.com/S1VSAMjUu

Knapsack Problem

- One or more Constraints
- Bounded/Unbounded/Fractional Items
- One or more knapsacks
- 2^N combinations of sets for N-items in a Set

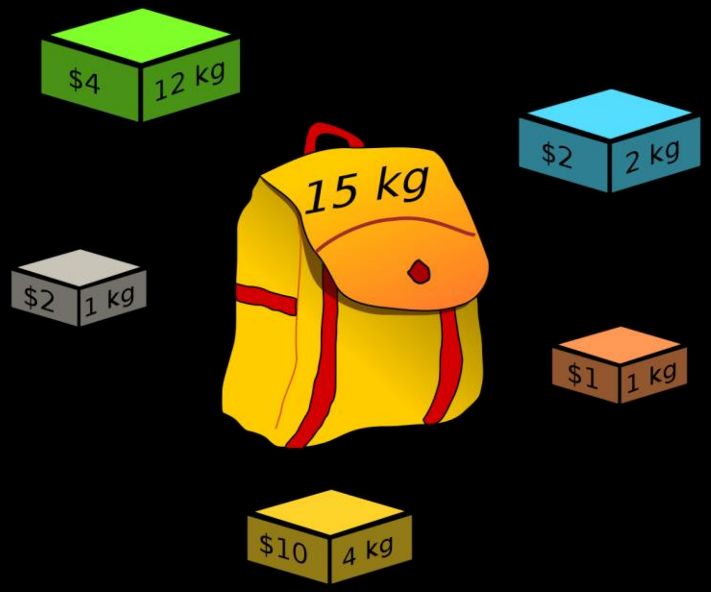


https://images.app.goo.gl/vQejKPdsUynyp3As6

0-1 Knapsack Problem

- Items are bounded, non-fractional and only one knapsack allowed.

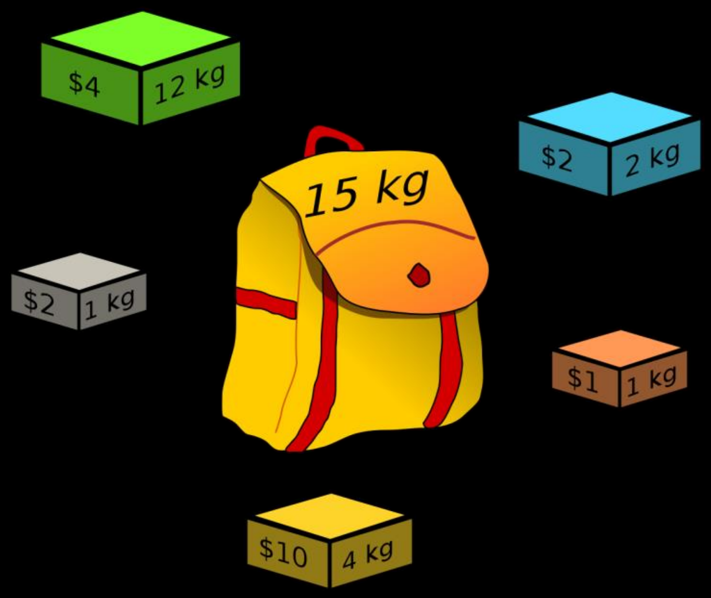
- Maximize profit/value



https://images.app.goo.gl/vQejKPdsUynyp3As6

Knapsack Problem

- OPT(i, W) = optimal value of max weight subset that uses items 1, …, i with weight limit W
- Greedy:
  - Repeatedly add item with maximum vi/wi ratio ….
  - Capacity M=7, Number of objects n = 3
  - w = [5, 4, 3]
  - v = [10, 7, 5]   (ordered by vi/wi ratio)



https://images.app.goo.gl/vQejKPdsUynyp3As6

Knapsack Problem

- $OPT(i, W)$ = optimal value of max weight subset that uses items $1, \ldots, i$ with weight limit W
  (If we have i items and a Knapsack of capacity W, what is the optimal value?)

- Dynamic Programming

  - Case 1: item i is not included:
    - Take best of $\{1, 2, \ldots., i\text{-}1\}$ using weight limit W: $OPT(i\text{-}1, W)$

  - Case 2: item i with weight w_i and value v_i is included:
    - only possible if $W >= w_i$
    - new weight limit = $W - w_i$
    - Take best of $\{1, 2, \ldots.., i\text{-}1\}$ using weight limit $W - w_i$ and add v_i:
      - $OPT(i\text{-}1, W\text{-}w_i) + v_i$

$$
OPT(i, W): \begin{cases} 0 & \text{if } i = 0 \\ OPT(i - 1, W) & \text{if } w_i > W \\ \max(OPT(i - 1, W), v_i + OPT(i - 1, W - w_i)) & \text{otherwise} \end{cases}
$$

Knapsack Problem

Weight of Knapsack, $W = 2$

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

$$
OPT(i, W) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, W) & \text{if } w_i > W \\ \max(OPT(i-1, W),\ v_i + OPT(i-1, W-w_i)) & \text{otherwise} \end{cases}
$$

$OPT(i, W)$: optimal value of max weight subset that uses items 1, …, i with weight limit W

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size
\# rows: No. of items + 1
\# columns: W + 1

	0	1	2
{0}			
{0, 1:1}			
{0, 1, 2:1}			
{0, 1, 2, 3:1}			

OPT(i, W):

- 0 — if i = 0
- OPT(i - 1, W) — if w_i > W
- max(OPT(i - 1, W), v_i + OPT(i - 1, W - w_i)) — otherwise

OPT(i, W): optimal value of max weight subset that uses items 1, …, i with weight limit W

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size
rows: No. of items + 1
columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0		
{0, 1, 2:1}	0		
{0, 1, 2, 3:1}	0		

$$
OPT(i, W) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, W) & \text{if } w_i > W \\ \max(OPT(i-1, W),\ v_i + OPT(i-1, W-w_i)) & \text{otherwise} \end{cases}
$$

OPT(i, W): optimal value of max weight subset that uses items 1, …, i with weight limit W

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size
rows: No. of items + 1
columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0	10	10
{0, 1, 2:1}	0		
{0, 1, 2, 3:1}	0		

OPT(i, W):

- 0 — if i = 0
- OPT(i - 1, W) — if w_i > W
- max(OPT(i - 1, W), v_i + OPT(i - 1, W - w_i)) — otherwise

OPT(i, W): optimal value of max weight subset that uses items 1, …, i with weight limit W

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size
\# rows: No. of items + 1
\# columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0	10	10
{0, 1, 2:1}	0	20	30
{0, 1, 2, 3:1}	0		

OPT(i, W):

–	0	if i = 0
–	OPT(i - 1, W)	if w_i > W
–	max(OPT(i - 1, W), v_i + OPT(i - 1, W - w_i))	otherwise

OPT(i, W): optimal value of max weight subset that uses items 1, …, i with weight limit W

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size

rows: No. of items + 1

columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0	10	10
{0, 1, 2:1}	0	20	30
{0, 1, 2, 3:1}	0	30	50

```
OPT(i, W):
        – 0                                           if i = 0
        – OPT(i - 1, W)                               if w_i > W
        – max(OPT(i - 1, W), v_i + OPT(i - 1, W - w_i))   otherwise
```

OPT(i, W): optimal value of max weight subset that uses items 1, …, i with weight limit W

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size
rows: No. of items + 1
columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0	10	10
{0, 1, 2:1}	0	20	30
{0, 1, 2, 3:1}	0	30	50

Which items we selected?

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size

\# rows: No. of items + 1

\# columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0	10	10
{0, 1, 2:1}	0	20	30
{0, 1, 2, 3:1}	0	30	50

Which items we selected?

Item 3 is in the bag as value of bag with item 3 in it is different from the value with Item {1, 2}.

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size
rows: No. of items + 1
columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0	10	10
{0, 1, 2:1}	0	20	30
{0, 1, 2, 3:1}	0	30	50

Which items we selected?

Item 2 is in the bag as value of bag with item 2 in it is different from the value with Item {1}.

Knapsack Problem

Weight of Knapsack, W = 2

Value (v_i)	Weight (w_i)
10	1
20	1
30	1

Grid Size
rows: No. of items + 1
columns: W + 1

	0	1	2
{0}	0	0	0
{0, 1:1}	0	10	10
{0, 1, 2:1}	0	20	30
{0, 1, 2, 3:1}	0	30	50

Which items we selected?

Item 2 and 3

Knapsack Problem – 11.2 Stepik

```
int backpack(int W, vector<int> weights, vector<int> values)
{
    int m = weights.size();   // Number of items
    int dp[m + 1][W + 1];     // dp[i][j] is the max value for the first i items with knapsack of capacity j

    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= W; j++)
        {
            if (i == 0 || j == 0) //If there are no items (i = 0) or the capacity of knapsack is zero (j = 0)
                dp[i][j] = 0;
            else if (weights[i - 1] > j) //If weight is bigger than the capacity
                dp[i][j] = dp[i - 1][j];
            else
                dp[i][j] = max(dp[i - 1][j], values[i - 1] + dp[i - 1][j - weights[i - 1]]);
        }
    }

    return dp[m][W];
}
```

https://stepik.org/lesson/390641/step/2?thread=solutions&unit=379741

Use cases of Data Structures and Algorithms

- Priority Queues / heaps
  - Prim’s algorithm – choosing next edge to add
  - Dijkstra’s algorithm – choosing next vertex to set distance
  - Huffman compression algorithm
  - Operating systems - load balancing and interrupt handling
- Binary search trees / hash tables
  - Anything where data is stored with a key / any database
  - Customer list with email address as a key
  - Students with ID number as key
- Minimum Spanning Tree / Graphs
  - Network design
  - Cluster analysis
- Shortest paths / Graphs
  - Delivery planning/scheduling
- Bin packing
  - scheduling problems – scheduling conferences
- Knapsack
  - choosing investment portfolio
  - resource allocation
  - https://www.vice.com/en_us/article/4x385b/the-world-is-knapsack-problem-and-were-just-living-in-it

Questions