Greedy Algorithms

- ***Compression problem***
- ***Huffman codes***

Definitions

- **Alphabet**: Finite set containing at least one element: $\mathbf{A} = \{a, b, c, d, e\}$
- **Symbol**: Alphabet element: $s \in \mathbf{A}$
- **String (over alphabet)**: Sequence of symbols: ccdabdcaad…
- **Codeword**: Sequence of bits representing coded symbol or string:
  1101010011010101 00…
- p_i: Occurrence probability of symbol s_i in input string

$$\sum_{\forall i \in A} p_i = 1$$

Code types

- **Fixed-length codes** - all codewords have same length (number of bits)

  A – 000, B – 001, C – 010, D – 011, E – 100, F – 101

- **Variable-length codes** - may give different lengths to codewords

  A – 0, B – 00, C – 110, D – 111, E – 1000, F – 1011

Code types (cont.)

- **Prefix code** - No codeword is prefix of any other codeword

  A = 0; B = 10; C = 110; D = 111
- **Uniquely decodable code** - Has only one possible source string producing it
  - Unambigously decoded
  - Examples:
    - Prefix code - end of codeword immediately recognized (without ambiguity) : 010011001110 → 0 | 10 | 0 | 110 | 0 | 111 | 0
    - Fixed-length code

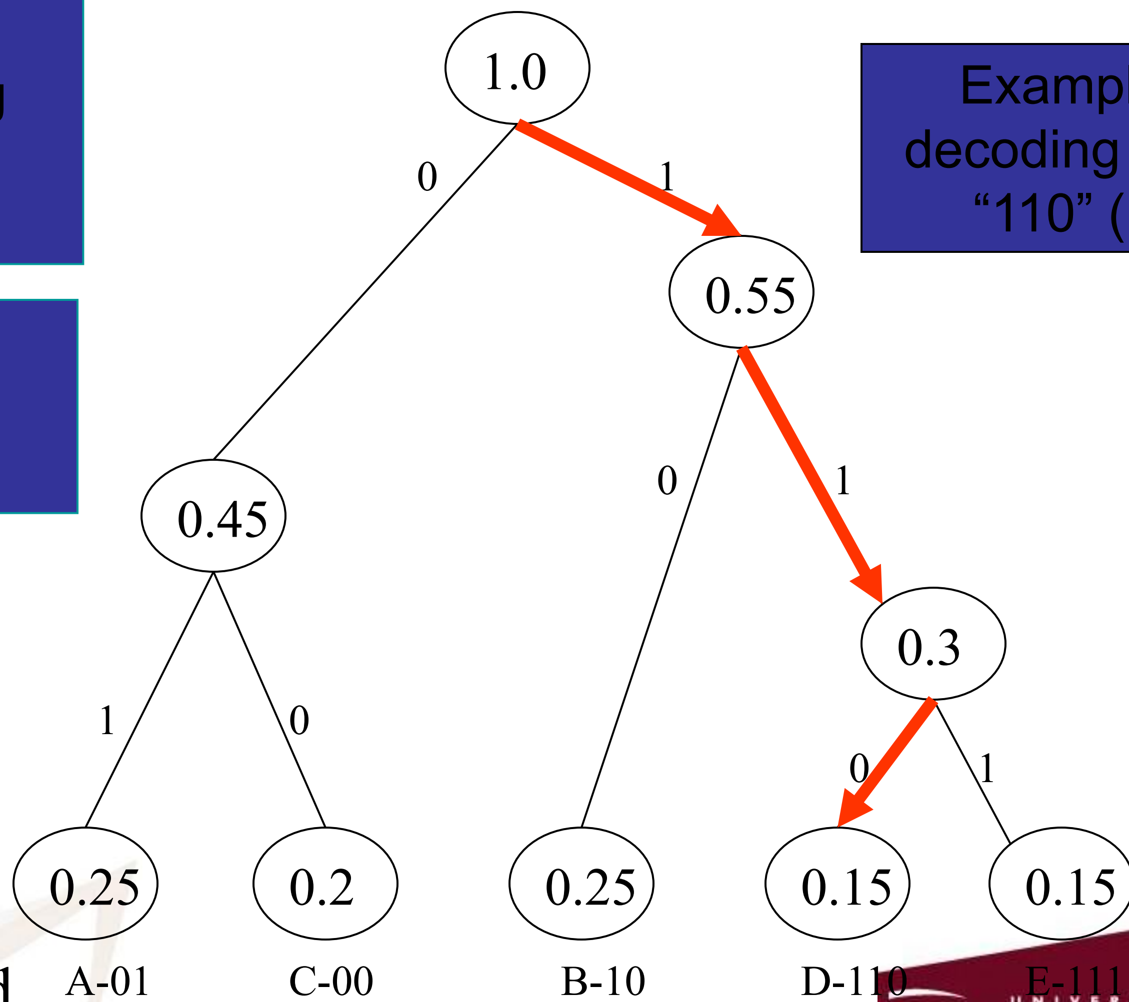
Huffman tree example

Each codeword determined according to path from root to symbol

When decoding tree traverse tree from root.

Example: decoding input "110" (D)

Probabilities



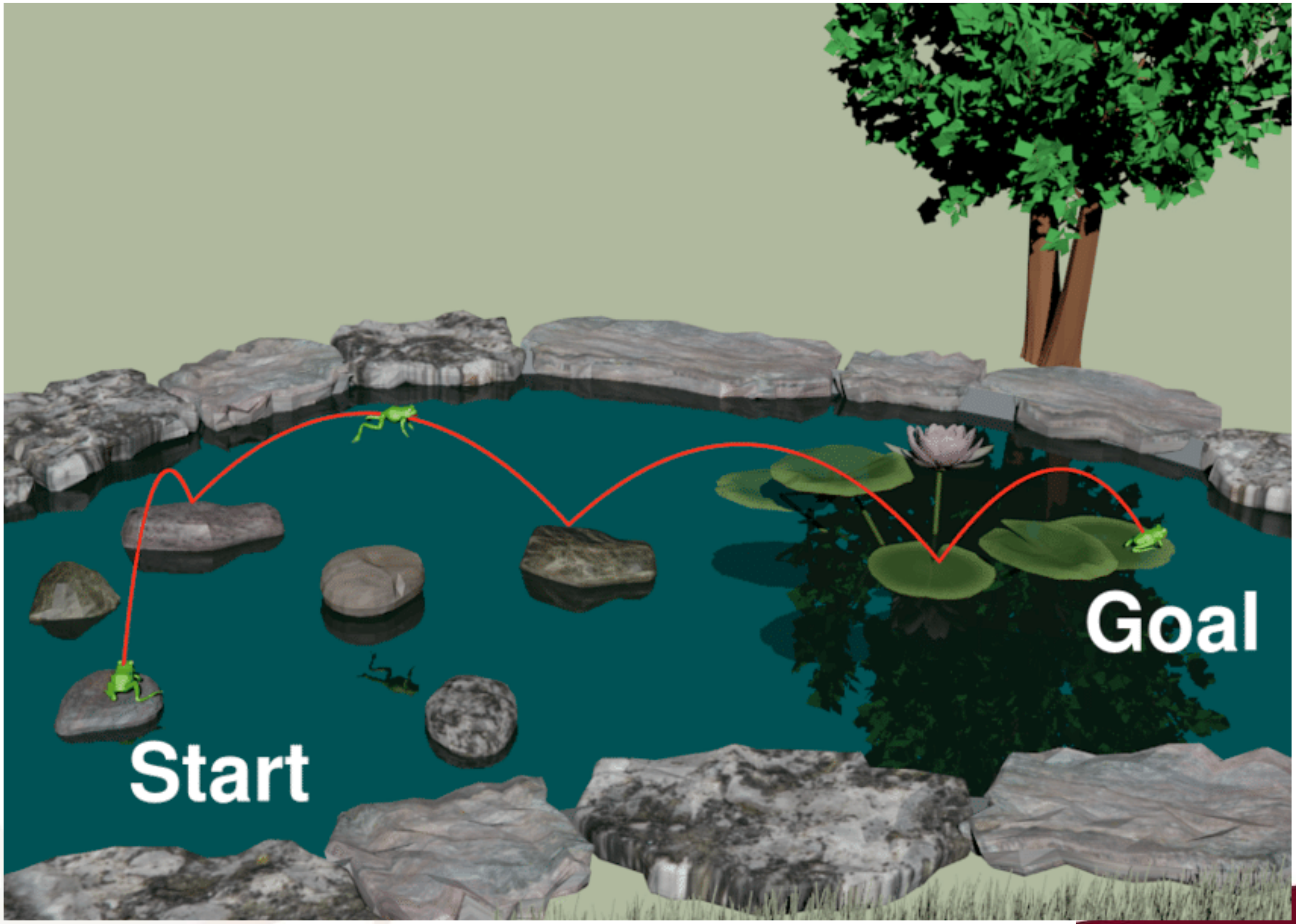
codewords:

Greedy Algorithms

- *Algorithm where the optimal local solution is taken at each time step*
- *Does it lead to a solution?*
- *Does it lead to an optimal solution?*
- *What is an optimal solution?*
- *… sometimes …*
- *How can you tell if they do?*
- *Theory….*

Université Concordia University

Frog jumping



Frog jumping

- Frog needs to go from position 0 to position n by subsequent jumps
- Some positions have rocks some water
- Frog can jump any integer number between 1 and a predefined maximum jump length as long as it lands on a rock and not in the water
- Assume that a solution exists
- Find an optimal sequence: minimizes the number of jumps
- (show on the board)

Frog jumping

- Devise a greedy solution:
- Each iteration perform an optimal step
- What is optimal: few jumps == large jumps
- Take the largest jump that you can do

```
int frog_steps(int n, int max_jump, int m, int stone_positions[]){
// stone_positions assumed in decreasing order – m=#stones
int current_position = 0;
int number_of_jumps = 0;
while(current_position!=n){
    for(int i=0;i<m;++i)
        if(is_stone(current_position + stone_position[i])){
            number_of_jumps++;
            current_position +=stone_position[i];
            break;
        }
}
Return number_of_jumps;}
```

Frog jumping

- Questions:
  - Does the algorithm provides a solution?
  - If yes, is it optimal?

```
int frog_steps(int n, int max_jump, int m, int stone_positions[]){
// stone_positions assumed in decreasing order – m=#stones
int current_position = 0;
int number_of_jumps = 0;
while(current_position!=n){
    for(int i=0;i<m;++i)
        if(is_stone(current_position + stone_position[i]){
            number_of_jumps++;
            current_position +=stone_position[i];
            break;
        }
}
Return number_of_jumps;}
```

Frog jumping

- Greedy algorithm optimality proof
- Let S be our solution.
- S consists of a sequence of steps: $S = \{s_i, i = 1. k\}$
- Let O be an optimal solution $O = \{o_i, i = 1..l\}$
- First observation: k>=l
- Define function position $P(S, i)$ =position of the frog at step I in the sequence S
- Lemma1: $P(S, i) \geq P(O, i) \forall i = 1..l$
- Theorem1: S is optimal

Frog jumping

- Lemma1: $P(S,i) \geq P(O,i) \forall i = 1..l$
- Theorem1: S is optimal
- $P(O,l) = n$ (because the sequence ends when the frog reaches the destination/O
- $P(S,l) \geq P(O,l) \Longrightarrow P(S,l) = n \Longrightarrow l = k$ (from lemma 1 and the fact that all steps are positive)
- $l = k$ means S is optimal q.e.d.

Frog jumping

Lemma1: $P(S,i) \geq P(O,i) \forall i = 1..l$

Proof by induction: True for i=1 by construction

Assume true for i, prove for i+1, Assume: $P(S,i) \geq P(O,i)$ (1)

Want to prove: $P(S,i+1) \geq P(O,i+1)$

Proof by contradiction: assume $P(S,i+1) < P(O,i+1)$ (2)

$P(S,i+1) = P(S,i) + S(i+1)$ (3) (by definition)

$P(O,i+1) = P(O,i) + O(i+1)$ (4) (by definition)

From (1-4) we have $O(i+1) > S(i+1)$ (5)

Let $S'(i+1) = P(O,i+1)$ - $P(S,i)$ (6)

$S'(i+1) > \mathrm{S(i+1)}$ from (3, 4, 5, 6)

By constuction $S'(i+1) > maxStep$ (7) (otherwise we would select S' instead of S at step I

$O(i+1) = P(O,i+1)$-$P(O,i) > P(O,i+1)$−$P(S,i)$=$S'(i+1) > maxStep$

Contradiction: $O(i+1) \leq maxStep$ q.e.d

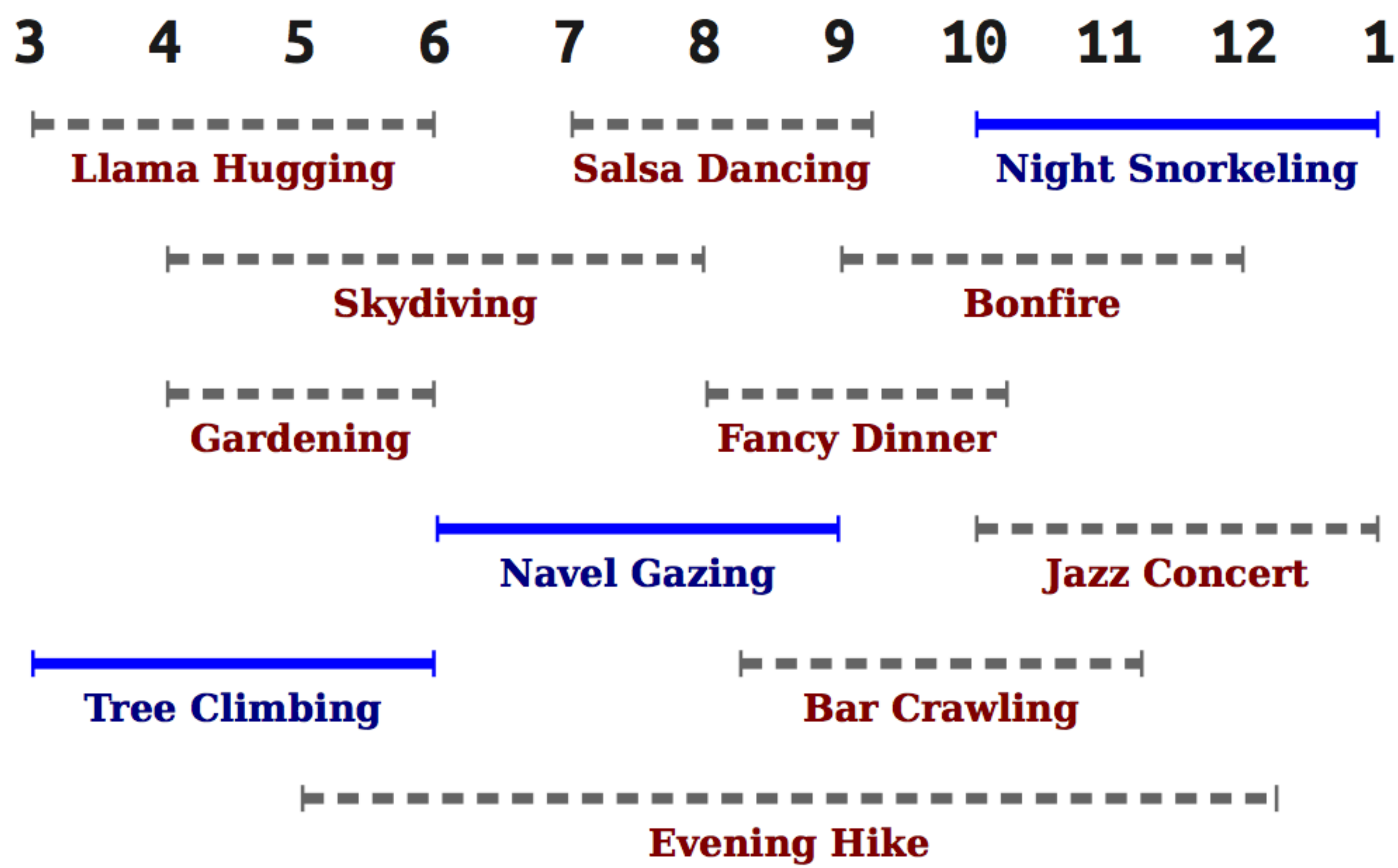
Frog jumping

- Greedy algorithm is optimal
- How did the proof work:
  - At every step of an ideal optimal algorithm – show that greedy does better
  - Called Optimal substructure
  - Proof by induction
  - Basic step is by construction
  - Inductive step show by contradiction

Activity Scheduling

The *activity scheduling* problem is the problem of selecting from a set of scheduled activities the largest subset of activities that do not intersect. More formally, given a set of activities: $S = \{a_i,$ i=0..n-1}, where each activity consists of a start time and end time: $a_i = (s_i, f_i)$ where $0 \le s_i < f_i$, the problem is to select a maximal subset R or non-overlapping activities $(a_i = (s_i, f_i), and\ a_j = (s_j, f_j)$ overlap iff $s_j < f_i or\ s_i < f_j$.

Activity Scheduling



Design a Greedy strategy

- Globally optimal: collect maximum number of activities
- Greedy strategy? How to chose greedily?
- Need to minimize or maximize something
  - Period of the activity (select shortest or longest
  - Start time (start with the first possible)
  - End time (start with the one that finishes first)
- Depends on what is the global goal
  - Chose end time

Design a Greedy strategy

```
int maxActivities(vector<activity> activities){
       sort_activities_by_end_time
       int nactivities = 0;
       int time = 0;
       int i=0;
        for(;i<activities.size();++i){
              if(time<=activities[i].begin){
                     time = activities[i].end;
                     nactivities +=1;
              }
       }
    return nactivities;
}
```

Design a Greedy strategy

- Is it correct?
- Is it optimal?
- Yes
- Prove it!!!!!
- How did the proof work:
  - At every step of an ideal optimal algorithm show that greedy does not do worse
  - Called Optimal substructure
  - Proof by induction
  - Basic step is by construction
  - Inductive step show by contradiction

Design a Greedy strategy

- Let S be our greedy schedule and S' an arbitrary optimal schedule
- It follows: $|S| \leq |S'|$
- Let $f(i, S)$ be the time at which the ith activity finishes in schedule S. Similarly: $f(i, S')$
- Lemma1: $f(i, S) \leq f(i, S') \forall\, 1 \leq i \leq |S|$
- Theorem $|S| \geq |S'|$
- Proof by contradiction: assume $|S| < |S'|$ (1)
- Let $\mathrm{k} = |S|$ (k is the last activity in S (2)
- From lemma: $f(k, S) \leq f(k, S')$
- From (1) there is an activity that starts after $f(k, S')$
- contradiction !!! - By construction this activity can be chose by our greedy algorithm $\Longrightarrow |S| \geq |S'|$
- q.e.d.

Design a Greedy strategy

- Lemma1: $f(i,S) \le f(i,S') \forall\, 1 \le i \le |S|$
- By induction:
  - i = 1 (by construction)
  - Assumes it is true for i, prove for i+1
  - Let k be the i+1 activity in S
  - Let k' be the i+1 activity in S'
  - $f(i,S) \le f(i,S')$ (the inductive step) means that k' could be selected as i+1 activity in S
  - This implies that $end(k) \le end(k') \Longrightarrow$
  - $f(i,S) + end(k) \le f(i,S') + end(k') \Longrightarrow$
  - $f(i+1,S) \le f(i+1,S')$ q.e.d.

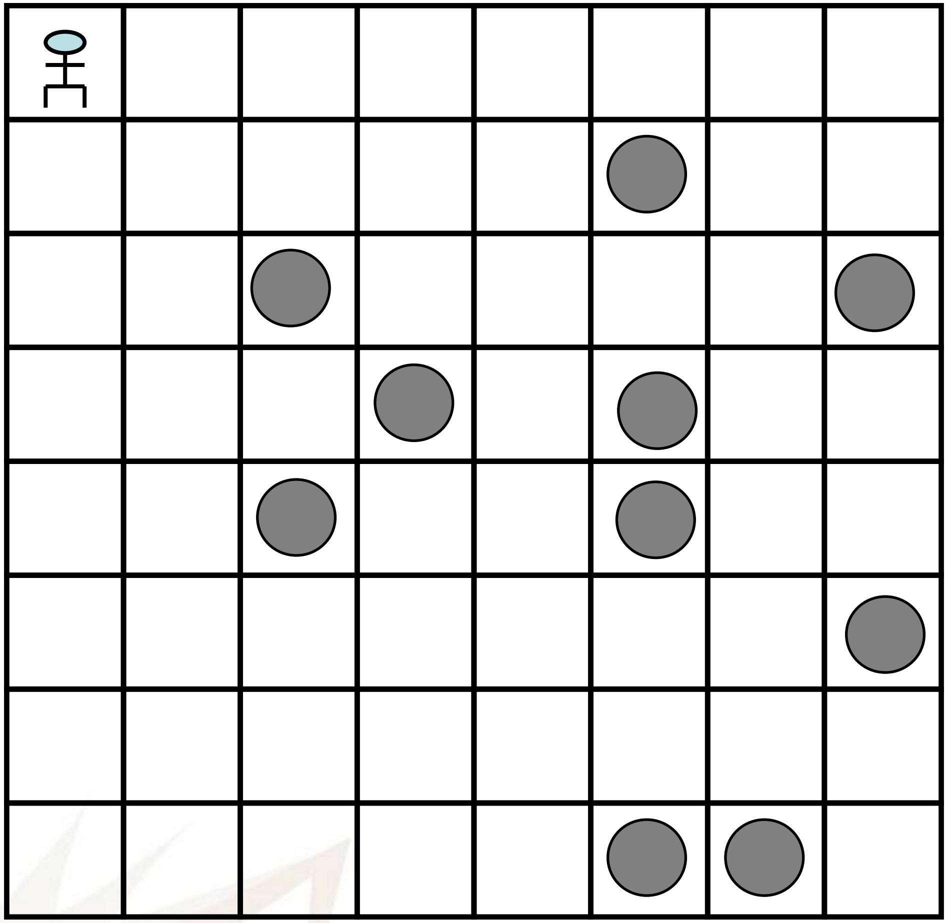
Greedy

- Very Efficient
- Why?
- No turning back!!!!
- No turning back → Greedy → very efficient
- Controlled turning back → Dynamic Programming → fairly efficient
- Unstructured turning backs → very inefficient

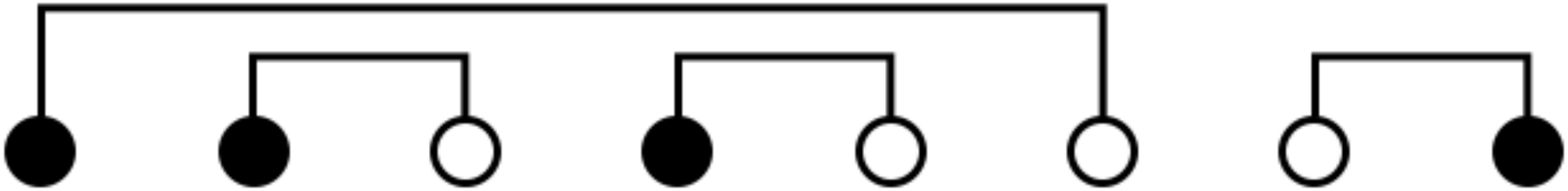
Greedy

- Efficient, Simple and Elegant
- Does not always provide an optimal solution
- Important to be able to prove one way or another
- Trade-off optimality for speed
  - Not optimal solution but still ok
  - Can prove for some algorithms how far we can be from optimal solution
- Famous Greedy Algorithms
  - Huffman codes
  - Spanning trees: Prim's and Kruskal's algorithm
  - Activity selection
  - Dijkstra Shortest Path algorithm (foundation of your GPS)

Greedy

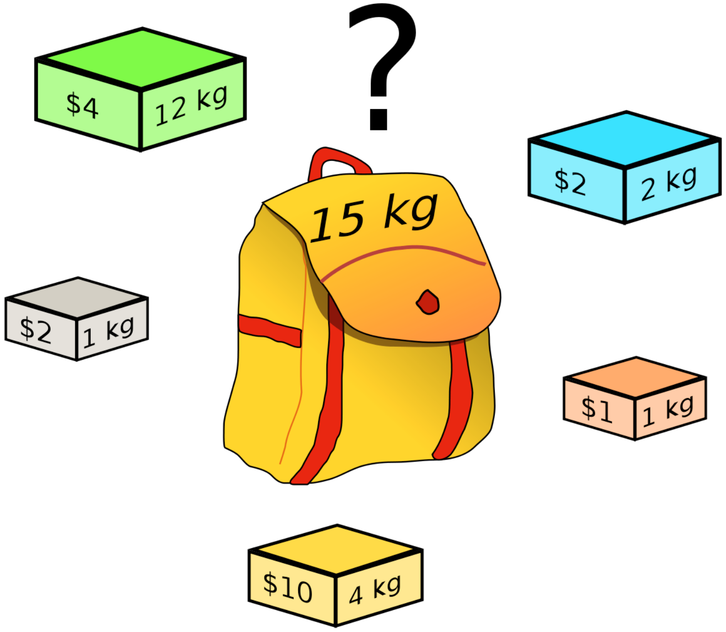


Greedy



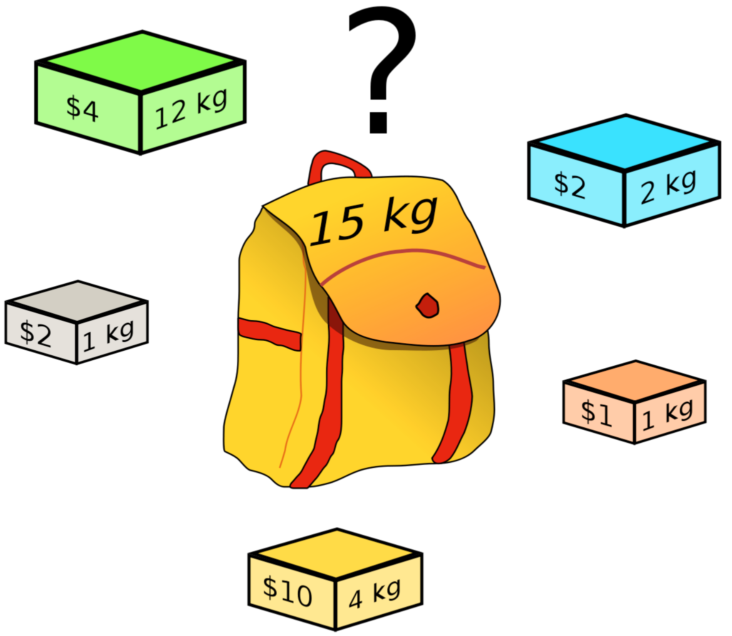
Knapsack Problem

- I have a collection of n objects
- Each object has a weight and a value
- Pack objects such that we maximize the value
- Given a maximum weight
- Assume I have infinitely many objects of each type



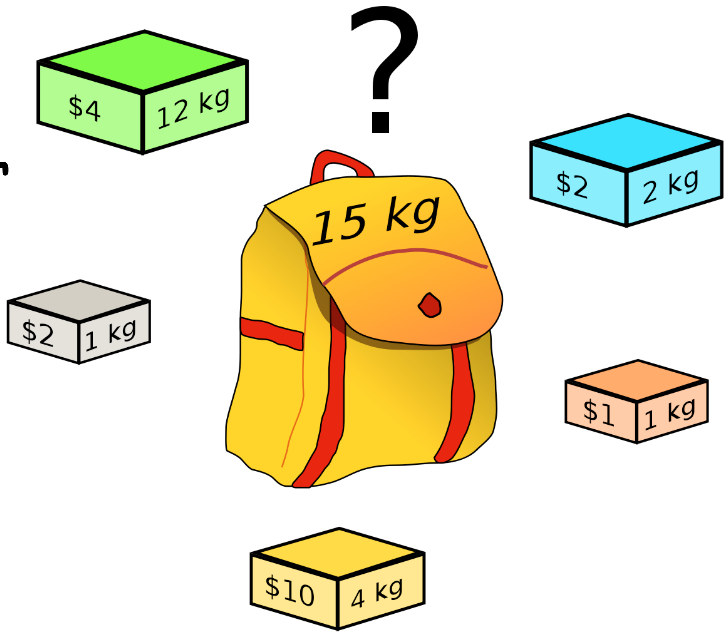
Knapsack Problem

- I have a collection of n objects
- Each object has a weight and a value
- Pack objects such that we maximize the value
- Given a maximum weight
- Assume I have infinitely many objects of each type

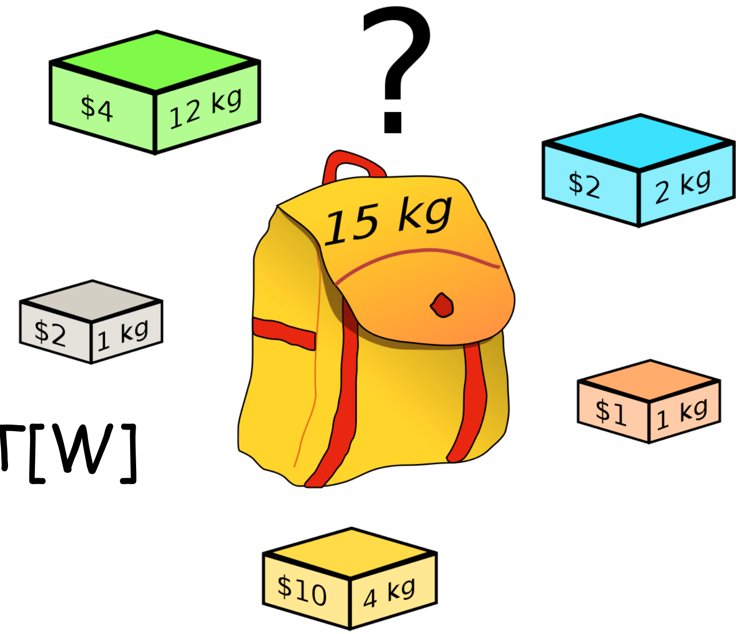


Knapsack Problem

- Greedy strategy:
- Pick the elements most valuable per unit of weights
- Example:
  - Values = {11/3, 2, 2}
  - Weights = {5, 3, 3}
  - W = 6
  - Ratios = {11/15, 10/15, 10/15
  - Greedy solution: Select first element
  - Optimal solution: select the second and third



Knapsack Problem



- Dynamic Programming strategy:
- Values v[i], i=1..n
- Weights w[i], i=1..n
- Maximum weight W
- Want maximum value of elements: T[W]
- What can we say about T:
- T[0] = 0
- T[negative number]=- inf (convention for invalid case)
- Greedy not optimal, so we need to look back
- In a smart way
- $T[W] = max_i(v[i] + T[W - w[i]])$